# Similarity Search Over Time Series and Trajectory Data

by

Lei Chen

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2005

I hereby declare that I am the sole author of this thesis.
I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Lei Chen

I further authorize the University of Waterloo to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Lei Chen

# Abstract

Time series data have been used in many applications, such as financial data analysis and weather forecasting. Similarly, trajectories of moving objects are often used to perform movement pattern analysis in surveillance video and sensor monitoring systems. All these applications are closely related to similarity-based time series or trajectory data retrieval.

In this dissertation, various similarity models are proposed to capture the similarities among time series and trajectory data under various circumstances and requirements, such as the appearance of noise and local time shifting. A novel representation, called *multi-scale time series histograms*, is proposed to answer pattern existence queries and shape match queries. Earlier proposals generally address one or the other; multi-scale time series histograms can answer both types, which offers users more flexibility. A metric distance function, called *Edit distance with Real Penalty* (ERP), is proposed that can support local time shifting in time series and trajectory data. A second distance function, *Edit Distance on Real sequence* (EDR) is proposed to measure the similarity between time series or trajectories with local time shifting and noise.

Since the proposed similarity models are computationally expensive, several indexing and pruning methods are proposed to improve the retrieval efficiency. For multi-scale time series histograms, A multi-step filtering process is introduced to improve the retrieval efficiency without introducing false dismissals. For ERP, a framework is developed to index time series or trajectory data under a metric distance function, which exploits the pruning power of lower bounding and triangle inequality. For EDR, three pruning techniques – mean value Q-grams, near triangle inequality, and trajectory histograms – are developed to improve the retrieval efficiency.

**Acknowledgements**

First of all, I gratefully acknowledge the persistent support and encouragement from my advisor, Professor M. Tamer Özsu. During my five years Ph.D. study, Tamer not only provided constant academic guidance to my research, he also gave me suggestions on how to overcome the difficulties that I met in my life. There is a famous Chinese saying "One day's teacher is your father for your whole life". To me, Tamer is a great supervisor and my second father in my life.

I wish to express my deep gratitude to Professor Raymond Ng, who gave me valuable suggestions on my research. The two weeks that I worked together with Raymond gave me an unforgettable research experience. I also want to thank Professor Vincent Oria, whom I discussed and worked with on various research topics, which makes the research enjoyable. Vincent is a great colleague and friend.

I want to thank Professor Dennis Shasha for serving on my dissertation committee. His comments on my thesis are precious. I also thank the other members of my dissertation committee, Professor Grant Weddell, Professor Ihab Francis Ilyas, and Professor Ajit Singh, for their interest in this dissertation and for their feedback.

Finally and most importantly, I would like to thank my wife, Yu, for her love and support during my Ph.D. study and for her braveness to give birth to our son, Evan, on January 4, 2005, which makes our life full of happiness. I also want to thank my parents for their efforts to provide me with the best possible eduction and help me to take care of my wife and my baby when I need time to prepare for my thesis defense, my mother-in-law who helped us a lot during last few years, my friends, Jie Lian and Xiaoyun Wu, who made time to accompany my wife at the hospital when I need time to work on my thesis.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Background

Similarity-based retrieval of time series and trajectory data has attracted increasing interest in database and knowledge discovery communities because of its wide use in various applications. For example, using a remote sensing system, and by mining the trajectories of animals in a large farming area, it is possible to determine migration patterns of certain groups of animals. In sports videos, it is quite useful for coaches or sports researchers to know the movement patterns of top players. In a store surveillance video monitoring system, finding the movement patterns of customers may help in the arrangement of merchandise. In a financial data analysis application, finding the sales volume data which have some specific patterns may help marketing people predict sale cycle patterns. All these applications are closely related to finding *similar* time series or trajectory data from a database. For example, the following query is quite common in a stock analysis system: "Give me all stock data from last month that is similar to IBM's stock data from the same period". As shown in Figure 1.1, the query is looking for the stock data within the boundaries that are described by two dashed curves.

**Definition 1.1** *A time series or trajectory $R$ is defined as a sequence of pairs:*

$$R = [(t_1, r_1) \ldots, (t_N, r_N)] \tag{1.1}$$

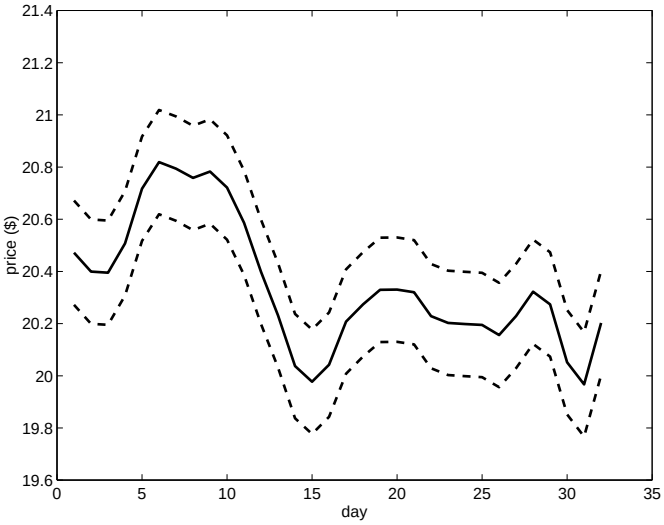



Figure 1.1: A query example on stock time series data

*where $N$, the number of sample timestamps in $R$, is defined as the length of $R$ and $r_i$ is a vector of arity $d$ that was sampled at timestamp $t_i$.*

The difference between time series and trajectory data is that vectors of most time series are one dimensional ($d = 1$), while vectors of trajectory data are often two or more dimensional ($d \geq 2$). A second difference is that in some applications, the time dimension ($t_i$) of trajectories is important. For example, in spatio-temporal databases, time components of trajectories are important to answer *time slice* or *time interval* queries [115, 73, 103, 81, 101]. The following are examples of time slice queries and time interval queries, respectively [115]:

- Search the objects within a rectangle $R$ at time $t_i$ (time slice);
- Search the objects within a rectangle $R$ from $t_i$ to $t_j$ (time interval).

However, in the case of similarity-based retrieval, the focus is on shapes of time series or trajectories. Sequences of sampled vectors are important in measuring the similarity between two time series or trajectories. Thus, in this thesis, time components of time

series and trajectory data can be ignored in measuring the similarity of time series or trajectory data. In the rest of the thesis, the terms time sequence and time series are used interchangeably.

As demonstrated by the above example query, a major problem of similarity-based retrieval is the definition of "similarity" of two data items. This is based on a *distance function*.

**Definition 1.2** *Given a data space $\mathcal{D}$ defined on time series or trajectory data and any two data $x, y \in \mathcal{D}$, a* distance function*, dist, on $\mathcal{D}$ is defined as:*

$$dist : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{R} \tag{1.2}$$

*where $\mathcal{R}$ is the real data set and dist has the following properties:*

- $dist(x, y) \geq 0$ *(nonnegativity);*
- $dist(x, y) = 0 \Leftrightarrow x = y$ *(reflexivity);*
- $d(x, y) = d(y, x)$ *(symmetry).*

**Definition 1.3** *Given a data space $\mathcal{D}$, and $x, y, z \in \mathcal{D}$, along with a distance function dist on $\mathcal{D}$, $x$ is* similar *to $y$ if $dist(x, y) \leq \epsilon$, where $\epsilon$ is a predefined threshold.*

A distance function directly affects the matching quality of the retrieved results, such as the accuracy of classification and clustering. The distance function is application and data dependent, and needs to be carefully designed to meet application requirements.

A second problem related to similarity-based retrieval is how to improve retrieval efficiency. The total execution time of a query can be expressed as:

$$T = \text{number of distance computations} \times \text{computation cost of } dist() + \text{I/O time} \tag{1.3}$$

The I/O time cost in the above equation is the sequential (linear scan) or random (searching indexes) disk access time. With the growth in database size, sequential scan is not practical, and it is necessary to design efficient indexing algorithms to reduce both the number of distance evaluations and the I/O cost.

## 1.2 Problem Statement

Based on the search criteria, similarity-based queries over time series and trajectory data are classified into two types:

1. *Pattern existence queries*: Given a time series or trajectory database $\mathcal{D}$ and a query pattern $QP$, find all the time series or trajectories in $\mathcal{D}$ that contain the specified pattern $QP$.

2. *Shape match queries*: Given a time series or trajectory database $\mathcal{D}$ and a query time series or trajectory $Q$, find all the time series or trajectories in $\mathcal{D}$ that have a movement shape similar to that of $Q$.

In pattern existence queries, as long as the time series or trajectory data have the specified pattern, they will be retrieved, no matter when the pattern appears and how it appears. For example,

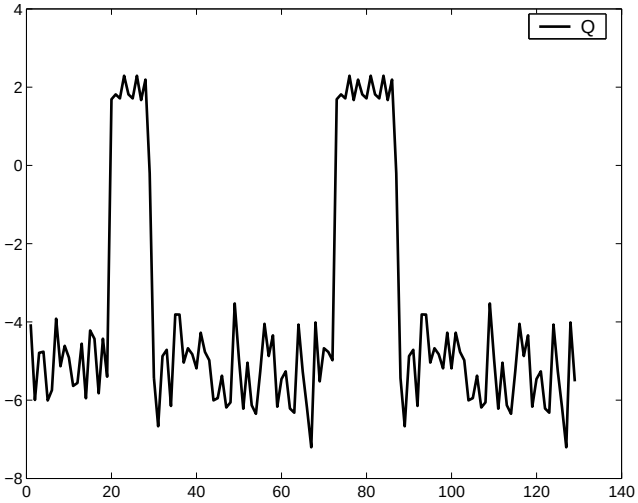
- "Give me all the stock data of last month that have a *head and shoulder pattern*"; or

- "Give me all the temperature data of patients in the last 24 hours that have *two peaks*".

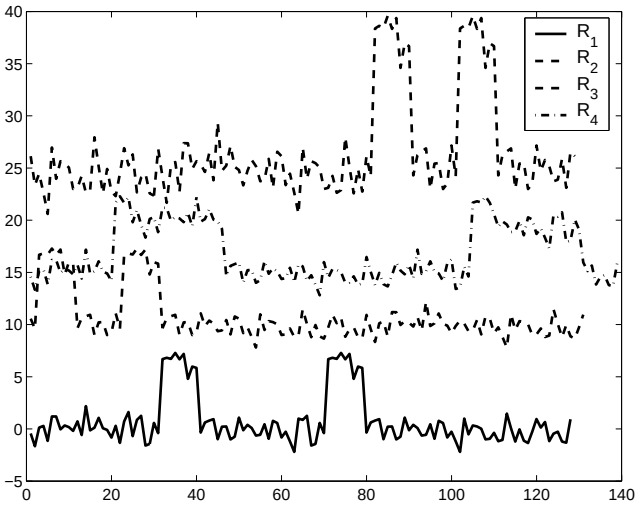
The second query, for example, is used to detect "goalpost fever", one of the symptoms of Hodgkin's disease, in the temperature time series data that contain peaks exactly twice within 24 hours [95]. Figure 1.2(a) shows an example time series with *two peaks*. Using this as query pattern, a *two peaks* existence query should retrieve various time series that contain *two peaks* as shown in Figure 1.2(b). Therefore, for pattern existence queries, the important thing is the existence of the specified pattern in time series data.

Different from pattern existence queries, shape match queries look for the time series data that have similar movement shapes. For example,

- "Give me all stock data of last month that is similar to the IBM's stock data of last month".




(a) An example query time series with two peaks




(b) Various time series data containing two peaks

Figure 1.2: A pattern existence query on two peaks

As shown in Figure 1.1, the stock data within the boundaries (described by two dashed curves) of IBM stock data are searched. Most of the previous work focus on solving shape match queries [3, 31, 37, 88, 118, 57, 52, 14, 119].

In this dissertation, both types of queries are studied with respect to the two key issues, namely the design of a distance function and the development of techniques to improve the retrieval efficiency.

Several assumptions are made about time series or trajectory data that are considered.

First, time series and trajectory data are not required to be sampled at the same rate. For each time series or trajectory, the width between a pair of time points $t_i$ and $t_{i+1}$ is not necessarily the same as the width between any other pair [19]. Interpolation is not necessary for the proposed distance functions or retrieval techniques, unless applications require it.

Second, there is a strong requirement that pruning or indexing methods should reduce (if possible, eliminate) *false candidates*, but should not lead to *false dismissals*. The false dismissals are defined as data that are missed by the search method, but should be in the result of a query. False candidates are defined as data that should not be retrieved as answers to a query, but are retrieved by the search method.

Third, the techniques should be able to deal with various types of disturbances, such as time shifting and noise. Time shifting may be caused by different sampling rate, and noise may be introduced by sensor failures, errors in detection techniques, and disturbance signals. Data cleaning is not always possible, since this would generally require domain knowledge that may not be readily available.

## 1.3 Background and Motivation

Significant research has been carried out on similarity-based search over time series and trajectory data, both in distance functions and their efficient evaluation. These works are discussed in Chapter 2 in detail. Nevertheless, there is a need for further investigation of this topic.

To answer pattern existence queries, several approximation approaches have been proposed to transform time series data to character strings over a discrete alphabet and apply string matching techniques to find the patterns [4, 95, 85]. In these approaches, best line fitting algorithms have been used to convert a time series to a sequence of linear approximations (as shown in Figure 1.3, the query time series $Q$ is approximated by a sequences

of line segments). Then, the slope of each line segment is mapped into a symbol according to a predefined mapping table (e.g. Table 1.1) and consecutive symbols are connected together to form a word[1]. Finally, these words are checked to see whether they match query pattern $QP$, which is specified by a regular expression. According to the example of converted line segments and the mapping table, the regular expression of the query example for retrieving time series that contain two peaks can be $F^*U^+D^+F^*U^+D^+F^*$.

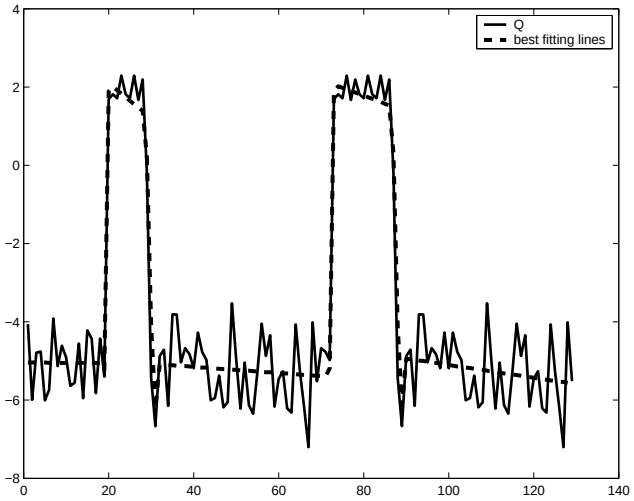



Figure 1.3: An example of converting $Q$ into line segments

| Slope symbol | Slope partiton |
|---|---|
| U(Up) | $(tan(\pi/6), \infty)$ |
| F(flat) | $[tan(-\pi/6), tan(\pi/6)]$ |
| D(Down) | $(-\infty, tan(-\pi/6))$ |

Table 1.1: An example of mapping quantized slope subspaces into symbols

However, the above transformation process is sensitive to noise. Furthermore, because of the quantization of the value space for transforming time series data into strings, data

[1]There are many other approaches to convert time series to their symbolic representation; they will be discussed in Section 2.2.2.

points located near the boundaries of two quantized subspaces may be assigned to different symbols (e.g. given an arbitrary small value $\epsilon$, $tan(\pi/6) - \epsilon$ and $tan(\pi/6) + \epsilon$ are in two different value ranges according to Table 1.1). As a consequence, they are considered to be different by string comparison techniques.

Most of the previous research on similarity search over time series and trajectory data focus on answering shape match queries. The pioneering work by Agrawal et al. [3] used Euclidean distance to measure similarity. Discrete Fourier Transform (DFT) was used as a dimensionality reduction technique for time series data, and an R-tree [38] was used as the index structure. Faloutsos et al. [31] extended this work to allow subsequence matching and proposed the GEMINI framework for indexing time series. GEMINI reduces the dimensionality of the original space and uses a lower bound on the true distance of the original space to guarantee no false dismissals when an index is used as a filter. Subsequent work have focused on two main aspects: new dimensionality reduction techniques (assuming that Euclidean distance is the similarity measure); and new approaches for measuring the similarity between two time series.

Examples of dimensionality reduction techniques include Single Value Decomposition (SVD) [53, 58], Discrete Wavelet Transform (DWT) [59, 82], Piecewise Aggregate Approximation (PAA) [53, 117], Adaptive Piecewise Constant Approximation (APAC) [52], and Chebyshev Coefficients [19]. However, these techniques can admit a high percentage of false positives due to their loose lower bounds [23]. If the distance measure is a metric distance function, then existing indexing structures proposed for metric distance functions may be applicable. Examples include the MVP-tree [15], the M-tree [26], the SA-tree [75], and the OMNI-family of access methods [32]. Therefore, developing a tighter lower bound for an existing or new distance function and combining the virtues of pruning by lower bounding and triangle inequality are still challenging topics.

As mentioned before, the design of a distance function is application and data dependent. Several distance functions have been proposed for different application, such as Euclidean distance [3, 31, 87, 53], Dynamic Time Warping [12, 118, 57, 52], and Longest Common Subsequences [13, 108]. Euclidean distance requires the time series to have the same length, which restricts its applications. Dynamic Time Warping (DTW) can handle sequences with different lengths and local time shifting, but it does not follow *triangle*

*inequality*, which is one of important properties of a metric distance function (discussed in detail in Chapter 2). Since most of the indexing structures assume that the underlying distance functions follow triangle inequality, DTW is non-indexable with current techniques. Furthermore, Euclidean distance and DTW are sensitive to noise, thus, they are not suitable for data that contain noise. Longest Common Subsequence (LCSS) is proposed to handle possible noise that may appear in data; however, it ignores the various gaps in between similar subsequences, which leads to inaccuracies. Therefore, a robust and accurate metric distance function is needed.

Finally, there are no approaches that have been proposed to answer both pattern existence and shape match queries. To satisfy applications that need answers from both types of queries, different representations, distance functions and retrieval methods have to be combined, which may cause storing redundant data and reducing the retrieval accuracy and efficiency.

In this thesis, the above problems are studied and the following solutions are proposed:

1. A novel representation of time series data called *multi-scale time series histograms* to answer both types of queries over time series data;

2. A metric distance function that can handle local time shifting;

3. A distance function that can handle data that contain local time shifting and noise; and

4. Indexing and pruning methods for efficient retrieval of time series and trajectory data.

## 1.4 Contributions

The major contributions of this dissertation are four-fold:

1. A novel representation of time series data, called *multi-scale time series histograms* (MSTSH), is proposed to answer two types of queries over time series data: pattern existence queries and shape match queries. As indicated in the previous subsection,

most of the earlier works have addressed either one of these query types, but not both. The presence of multiple scales (levels) enables querying at multiple precision. Most importantly, the distances of time series histograms at lower scales are lower bounds of the distances at higher scales, which guarantees that no false dismissals will be introduced when a multi-step filtering process is used to answer shape match queries. The evaluation algorithm uses averages of time series histograms to reduce the dimensionality and avoid computing the distances of full time series histograms.

2. A metric distance function, called *Edit distance with Real Penalty* (ERP), is designed that can support local time shifting. In many applications, such as speech recognition [3, 31] and music retrieval [119], sequences are required to do a flexible shifting of the time axis to accommodate sequences that are similar but out of phase. In these applications, Euclidean distance cannot be applied. Even though DTW can be used in these cases, it does not follow triangle inequality, which makes it non-indexable by traditional access methods. ERP is proposed to handle local time shifting issues and it is proved to be a metric distance function, which makes it indexable. Benchmark results show that ERP is natural for time series data.

3. A framework is proposed to index time series or trajectory data under a metric distance function. Given a metric distance function, besides following the GEMINI framework that applies dimensionality reduction techniques and defines lower bounding distances, triangle inequality can also be used to prune the search space. The indexing framework applies both lower bounding and triangle inequality, and performs better than applying only lower bounding or triangle inequality.

4. A second distance function, called *Edit Distance on Real sequence* (EDR), is defined to measure the similarity between two time series or trajectory data that contain local time shifting and noise. EDR is based on edit distance on strings. Through a set of objective tests on benchmark data, it is shown that EDR is more robust than Euclidean distance, DTW, and ERP, and is more accurate than LCSS when it is used to measure the similarity between time series or trajectories that contain noise and local time shifting. Three pruning techniques are developed – mean value Q-grams, near triangle inequality, and histograms – to improve the retrieval efficiency of

EDR. Unlike the pruning methods proposed for LCSS [108] or DTW [52], these three pruning methods do not require setting constraints on warping length (or matching region) between two trajectories and, therefore, offer users more flexibility.

## 1.5 Organization of Thesis

The rest of the thesis is arranged as follows. A survey of related work with particular focus on distance functions and efficient retrieval mechanism is provided in Chapter two. Applying multi-resolution time series histograms to answer both pattern existence queries and shape match queries is discussed in Chapter three. In Chapter four, the *Edit distance with Real Penalty* (ERP) is introduced to handle data with local time shifting and performance results are provided that demonstrate superiority of combined indexing structures. In Chapter five, the other distance function, *Edit Distance on Real sequence* (EDR), is defined that handles the cases where data contain both local time shifting and noise. Finally, Chapter six presents conclusions and proposes future research topics.

# Chapter 2

# Related Work

As indicated in Chapter 1, two key issues have to be addressed for similarity search over time series and trajectory data: selection or design of a distance function and efficient retrieval techniques. This chapter focuses on related work on these two topics. Before discussing the work on these two topics, a brief overview of different classification schemes on similarity-based search over time series and trajectory data is given in Section 2.1. Section 2.2 presents distance functions that have been proposed to measure the similarity between time series or trajectory data. Dimension reduction techniques and metric space indexing techniques which have been used to improve the retrieval efficiency are reviewed in Section 2.3. An excellent survey of metric space indexing techniques has recently been published [22]; therefore, detailed comparison of metric space indexing structures are not discussed in this chapter.

## 2.1 Classification of Similarity-based Search

Figure 2.1 summarizes the main symbols used in the following discussions and in the rest of this thesis.

In general, similarity-based search can be classified into two types: *range queries* and *k-nearest neighbour queries*. This classification is not limited to time series and trajectory data, it is applicable to other data types, such as image, video, audio, and text.

| Symbols | Meaning |
|---|---|
| $\mathcal{D}$ | a set of time series of trajectory data |
| $S, R, Q, T$ | time series or trajectories, e.g. $S = [(t_1, s_1), \ldots, (t_N, s_N)]$ |
| $\|R\|$ | the length of time series data $R$ |
| $M, N$ | the length of a time series data |
| $(t_i, s_i)$ | the $i^{th}$ element of $S$ |
| $dist$ | a distance function on $\mathcal{D}$ |
| $s_i$ | i$^{th}$ element vector of $S$ |
| $s_{i,x}$ | the x coordinate of i$^{th}$ element vector of $S$ |
| $r$ | a constant real number for query range |
| $x, y, z$ | data belong to a defined data space $\mathcal{D}$ |
| $\tau$ | number of histogram bins |
| $\delta$ | scale level of histograms |
| $\epsilon$ | a real value for matching threshold |
| $wr$ | the warping rang in dynamic time warping |
| $v$ | staring shifting position |
| $g$ | a gap introduced in Edit Distance definition |
| $Rest(S)$ | the subsequence of S without the first element: $[(t_2, s_2), \ldots, (t_N, s_N)]$ |
| $first(S)$ | the first element of sequence of $S$ |
| $last(S)$ | the last element of sequence $S$ |
| $\widetilde{S}$ | $S$ after alignment with another series |
| $DLB$ | a lower bound of the distance |
| $H_R, H_S, H_T$ | histograms of time series or trajectories, e.g. $H_S$ histograms for sequence $S$ |
| $L$ | the $\mathcal{D}$ size or the database size |
| $W$ | a vector of weights for different elements of a sequence |

Figure 2.1: Meanings of symbols used

1. Range query: Given a data space $\mathcal{D}$, a distance function *dist* defined on $\mathcal{D}$, a query data $Q$, and a range $r \geq 0$, retrieve all data in $\mathcal{D}$ that are within distance $r$ to the query data $Q$.
$$Range(Q, r)_{dist} = \{R \in \mathcal{D} | dist(Q, R) \leq r\} \tag{2.1}$$

2. $k$-nearest neighbor query ($k - NN$): Given a data space $\mathcal{D}$, a distance function *dist* defined on $\mathcal{D}$, a query data $Q$ and $k \geq 1$, retrieve the $k$ closest data to $Q$ in $\mathcal{D}$.

$$k - NN(Q)_{dist} = \{A \subseteq \mathcal{D} | \forall R \in A, S \in \mathcal{D} - A, dist(R, Q) \leq dist(S, Q) \wedge |A| = k\} \tag{2.2}$$

In chapter 1, similarity-based search over time series and trajectory data is classified into pattern existence queries and shape match queries. This classification is orthogonal to the $k$-NN and range query classification given above. Pattern existence and shape match queries can be range queries or $k$-NN queries.

There is another classification of similarity-based queries based on search length. In this classification, the queries are classified into two types: *whole matching* and *subsequence matching* [31].

- **Whole matching queries:** Given a data set $\mathcal{D}$ of time series or trajectories of length $N$, a distance function *dist*, a query time series or trajectory $Q$ of the same length, and a real value $r \geq 0$, find all the time series or trajectories in $\mathcal{D}$ that are within distance $r$ to $Q$. In whole matching queries, query sequence and sequences in $\mathcal{D}$ must have the same length $N$.

  $$Wholematch(Q, r)_{dist} = \{R \in \mathcal{D} | dist(R, Q) \leq r \text{ and } |R| = |Q|\} \tag{2.3}$$

  where $|R|$ returns the length of sequence $R$.

- **Subsequence matching queries:** Given a data set $\mathcal{D}$ of time series or trajectories of arbitrary lengths, a query time or trajectory $Q$ whose length is less than the minimum sequence length in $\mathcal{D}$, a distance function *dist*, and a real value $r \geq 0$,

find all the time series or trajectories, one or more subsequences of which are within distance $r$ to $Q$.

$$Subsequencematch(Q, r)_{dist} = \{R \in \mathcal{D} | dist(R[v : (v + |Q| - 1)], Q) \leq r\} \quad (2.4)$$

where $|Q| \leq |R|$, $v$ is a starting position of the subsequence, and $R[v : (v + |Q| - 1)]$ is the subsequence of R that starts from $v$ and ends at $v + |Q| - 1$.

By putting a sliding window of length $|Q|$ at every possible offset of each data sequence $R$ and using this subsequence to compare with $Q$, subsequence matching can be converted into whole matching. However, this conversion will generate nearly $|R|$ subsequences for each $R$, which will increase the storage requirement and the height of the R*-tree that is used to index the subsequences. Several attempts have been made to solve this problem, such as using *minimum bounding rectangles* (MBRs) to store the subsequences that are adjacent to each other [31], *dural match* [70], which divides each data sequence $R$ into disjoint windows and query sequence $Q$ into sliding windows, and *general match* [69], which divides each data sequence $R$ into $J$ sliding windows (the sliding step is $J$) and dividing query sequence $Q$ into $J$ disjoint windows (number of disjoint window sequences is $J$).

Compared to the classification of whole matching and subsequence matching, classifying queries into pattern existence and shape match queries is more general. This is because whole matching and subsequence matching assume that the underlying distance function is $L_p$-norm (as will be defined in next section) and require sequences (or subsequences) having the same length with that of query data, which limits their usability. In many applications, such as music retrieval [119], video and image sequence analysis [78, 14] and speech recognition [12], time series or trajectory data usually have similar shapes but different lengths due to different sampling rates. Therefore, in this thesis, pattern existence and shape match queries are studied. Regardless of the types of queries, the two key issues, namely distance function selection and efficient retrieval techniques, remain the same. Existing work on these two issues will be reviewed in next two sections.

## 2.2 Data Representation and Distance Functions

Data representation formats and distance functions are closely related to each other, therefore, current techniques on these two issues are reviewed together. Generally, these techniques can be classified into two main categories: distance functions on raw representations and distance functions on other representations.

### 2.2.1 Distance Functions on Raw Representation

As defined in Chapter 1, a time series or trajectory data can be represented as a sequence of pairs, $R = [(r_1, t_1), (r_2, t_2) \ldots, (r_N, t_N)]$. $R$ is called *raw representation* of the time series data. Many distance functions have been proposed based on raw representation. The definitions of distance functions presented in this section use one dimensional time series data ($d = 1$) as an example, but they can be easily extended to multi-dimensional trajectory data.

**Definition 2.1** *$L_p$-norm: Given two time series $R$ and $S$ of the same length $N$, the $L_p$-norm distance between $R$ and $S$ is:*

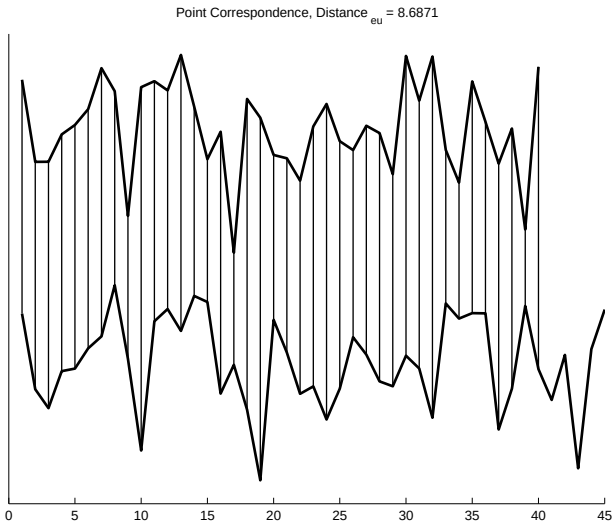
$$L_p - norm(R, S) = \sqrt[p]{\sum_{i=1}^{N} (r_i - s_i)^p} \tag{2.5}$$

$L_1$-Norm ($p = 1$) is named the *Manhattan distance* or *city block distance*, $L_2$-Norm ($p = 2$) is known as the famous *Euclidean distance*, while in the extreme case when $p = \infty$, it is called the *maximum norm* and can be redefined as follows:

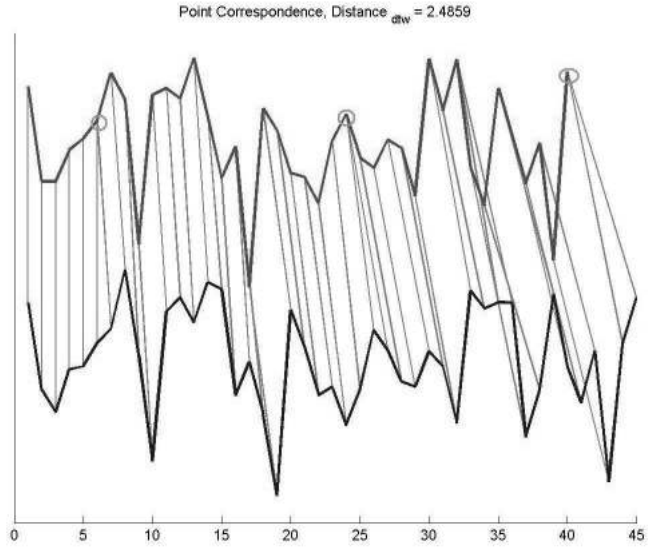
$$L_\infty - norm(R, S) = max_{i=1}^{N}(|r_i - s_i|) \tag{2.6}$$

A variant of $L_p$-norm, called *weighted $L_p$-norm* is defined as:

$$L_p - norm(R, S, W) = \sqrt[p]{\sum_{i=1}^{N} w_i (r_i - s_i)^p} \tag{2.7}$$




(a) Using Euclidean distance




(b) Using DTW

Figure 2.2: Point correspondence when two similar time series contain local time shifting

where $W$ is a vector (a matrix for multi-dimensional trajectory data) of weights for different elements of time series.

Euclidean distance is the first distance function that was introduced for similarity search of time series data in database literature [3, 31]. It has the advantage of being easy to compute and the computation cost is linear in terms of sequence length. However, Euclidean distance requires that the two time series be of the same length and it does not support *local time shifting*. Local time shifting occurs when one element of one sequence is shifted along the time axis to match an element of another time sequence (even when the two matched elements appear in different positions in the sequences). This is useful when the sequences have similar shape but are out of phase. It is called "local", because not all of the elements of the query sequence need to be shifted and the shifted elements do not have the same shifting factor. By contrast, in "global" time shifting, all the elements are shifted along the time axis by a fixed shifting factor. Generally, local time shifting cannot be handled by $L_p$-norm, because $L_p$-norm requires the $i^{th}$ element of query sequence be aligned with the $i^{th}$ element of the data sequence. Figure 2.2(a) shows an example on how elements from two time series could be matched when Euclidean distance is computed in the presence of local time shifting.

*Dynamic Time Warping* (DTW), which is widely used in speech recognition [45, 60, 72, 86, 92, 102], is introduced to handle local time shifting [12, 118, 57, 52] and time sequences with different lengths.

**Definition 2.2** *The DTW distance between two time series $R$ and $S$ of lengths $M$ and $N$, respectively, is defined as:*

$$DTW(R,S) = \begin{cases} \infty & \text{if } M = 0 \text{ or } N = 0 \\ dist(r_1, s_1) + min\{DTW(Rest(R), Rest(S)), & \\ DTW(Rest(R), S), DTW(R, Rest(S))\} & \text{otherwise} \end{cases} \tag{2.8}$$

*where $dist(r_1, s_1) = |(r_1 - s_1)|$ and $Rest(R)$ means the subsequence $R$ without the first element, $[(t_2, r_2), \ldots, (t_M, r_M)]$.*

The distance function *dist* that measures the distance between two elements can be one of $L_p$-norms. $L_1$-norm is used here as an example.

DTW does not require that the two time series data have the same length, and it can handle local time shifting by duplicating the previous element of the time sequence.

Matching examples are shown in Figure 2.2(b) for DTW. In Figure 2.2(b), two time series data are similar to each other in terms of overall shape; DTW allows some elements to be be replicated (e.g. the data points marked out by small circles) to accommodate the cases where elements are similar but out of phrase (time-wise). Comparing Figures 2.2(a) and 2.2(b) reveals how DTW can handle local time shifting whereas Euclidean distance cannot.

However, in the best case, the computation cost of DTW is quadratic, $O(M * N)$ ($M$ and $N$ are the lengths of two compared time series, respectively), with dynamic programming; therefore, when the database size increases, enormous time requires to be spent on computing DTW to answer a query unless some indexing methods can be used to save the number of computations. Unfortunately, DTW does not follow *triangle inequality*, which makes many indexing structures based on triangle inequality, such as VP-trees [105], M-trees [26], Sa-tree [75], and OMNI-family of access methods [32], not applicable to DTW.

Before giving the proof that DTW does not follow triangle inequality, the idea of using triangle inequality to remove false candidates is illustrated. A distance function $dist$ defined on a data space $\mathcal{D}$ satisfies triangle inequality, if and only if

$$\forall Q, R, S \in \mathcal{D}, dist(Q, S) + dist(R, S) \geq dist(Q, R) \tag{2.9}$$

By rewriting inequality 2.9, $dist(Q, S) \geq dist(Q, R) - dist(R, S)$ can be derived. If $dist(Q, R)$ and $dist(R, S)$ are known, $dist(Q, R) - dist(R, S)$ can be treated as a lower bound distance of $dist(Q, S)$. If the current query range is $r$ and $dist(Q, R) - dist(S, R) > r$, the computation of $dist(Q, S)$ can be saved since $S$ does not belong to the answer set. This is the basic principle that most of distance-based indexing structures follow.

**Theorem 2.1** *DTW does not follow triangle inequality.*

**Proof.** By a counter example. Given three time series data, $Q = [0], R = [1, 2]$ and $S = [2, 3, 3]$, then $DTW(Q, R) = 3$, $DTW(R, S) = 3$, and $DTW(Q, S) = 8 > DTW(Q, R) + DTW(R, S) = 3 + 3 = 6$. □

*Longest Common SubSequences* (LCSS) is proposed to handle time series data that may contain possible noise [13, 108]. The noise could be introduced by hardware failures, disturbance signals, and transmission errors. The intuition of LCSS is to remove the noise effects by only counting the number of *matched* elements between two time sequences.

**Definition 2.3** *The LCSS score of two time series $R$ and $S$ of lengths $M$ and $N$, respectively, is defined as:*

$$LCSS(R,S) = \begin{cases} 0 & \text{if } M = 0 \text{ or } N = 0 \\ LCSS(Rest(R), Rest(S)) + 1 & \text{if } dist(r_1, s_1) \leq \epsilon \\ max\{LCSS(Rest(R), S), & \\ \quad LCSS(R, Rest(S))\} & \text{otherwise} \end{cases} \tag{2.10}$$

*where* $dist(r_1, s_1) = |(r_1 - s_1)|$ .

In the definition of LCSS, instead of using distance, "score" is used. Thus, the higher the score, the more similar are two time series. The LCSS score can be converted into distance using the following formula:

$$LCSS_{dist}(R,S) = 1 - \frac{LCSS(R,S)}{min(|R|, |S|)} \tag{2.11}$$

In LCSS, a matching threshold $\epsilon$ has to be set up to determine whether or not two elements *match*. LCSS can handle noise, because the matching threshold quantizes the distance between two elements to two values, 0 and 1, which removes the larger distance effects caused by noise. The effect is shown by the following example.

Consider a query time series $Q = [1, 2, 3, 4]$ that will be matched with three other time sequences:

- $R = [10, 9, 8, 7]$,
- $S = [1, 100, 2, 3, 4]$,
- $T = [1, 100, 101, 2, 3, 4]$.

The second element of $S$ as well as the second and third elements of $T$ are noise (their values are "abnormal" compared to the values near them). The correct ranking in terms of similarity to $Q$ is: $S, T, R$, since, except noise, the rest of the elements of $S$ and $T$ match the elements of $Q$ perfectly. The ranking results with Euclidean distance and DTW are the same, which is: $R, S, T$. However, even from general movement trends (subsequent values increase or decrease) of the two trajectories, $Q$ and $R$ are quite different time series compared to $Q$ and $S$. Euclidean distance and DTW are sensitive to noise because:

- Euclidean distance and DTW require each element of the query sequence have a corresponding element in the matched sequence, even for noise.
- $L_p$-norm is used to measure the distance between two individual elements and noise may have large effect on $L_p$-norm of two time sequences.

The above example shows that noise can cause similar time sequences to be treated as dissimilar when noise-sensitive distance functions are used. Assuming $\epsilon = 1$, LCSS ranks three time sequences in terms of their similarities to $Q$ as $S = T, R$ ("=" means that $S$ and $T$ have the same distance to $Q$). LCSS works better than Euclidean distance and DTW on this example because the large distance difference introduced by "noise", such as 100 to 2 for Euclidean distance, is quantized to 0 by the threshold $\epsilon$.

Matching examples are shown in Figure 2.3 for DTW and LCSS. Comparing Figures 2.3(a) and 2.3(b) shows that DTW requires all the elements match each other while LCSS only counts the "common" elements, making it robust to noise.

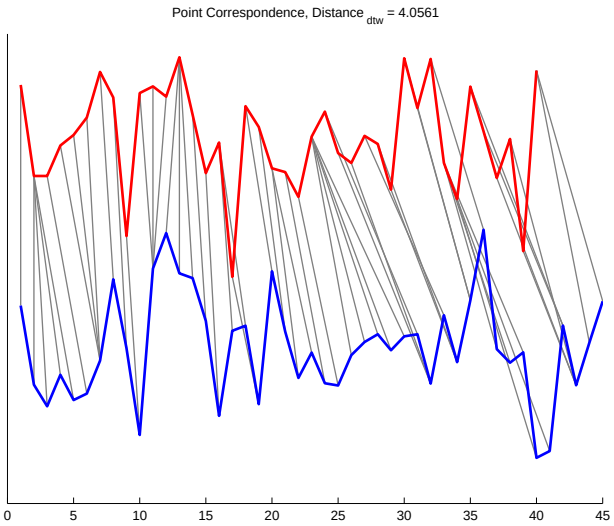
However, LCSS does not differentiate time series with similar subsequences but various *gap* differences between similar subsequences, which leads to inaccuracies. Here the gap differences refer to subsequences in between two identified similar components of two time series. As shown in Figure 2.4, two time series data may have exactly the same LCSS distance to the query sequences, but, they may have quite different sizes of gap in between the similar subsequences. Therefore, even though LCSS can handle noise, it sacrifices too much accuracy for that and it is definitely not a good solution for handling noise [25].

Similar to DTW, in the best case, the computation cost of LCSS is quadratic as well. Furthermore, LCSS does not follow triangle inequality, which also causes it to be non-indexable by most of the distance-based indexing methods.

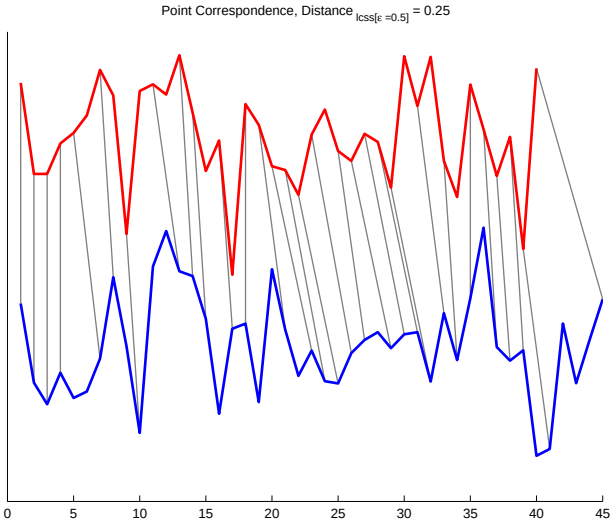
**Theorem 2.2** *LCSS does not follow triangle inequality.*

**Proof.** By a counter example. Given three time series data, $Q = [0], R = [1, 2]$ and $S = [2, 3, 3]$, and $\epsilon = 1$, then $LCSS(Q, R) = 1$, $LCSS(R, S) = 2$ and $LCSS(Q, S) = 0$. Therefore, $LCSS(Q, R) + LCSS(Q, S) = 1 + 0 = 1 < LCSS(R, S) = 2$. □

Table 2.1 is a comparison of the three distance functions based on six criteria: ability to handle sequences with different lengths, ability to handle sequences with local time




(a) Using DTW

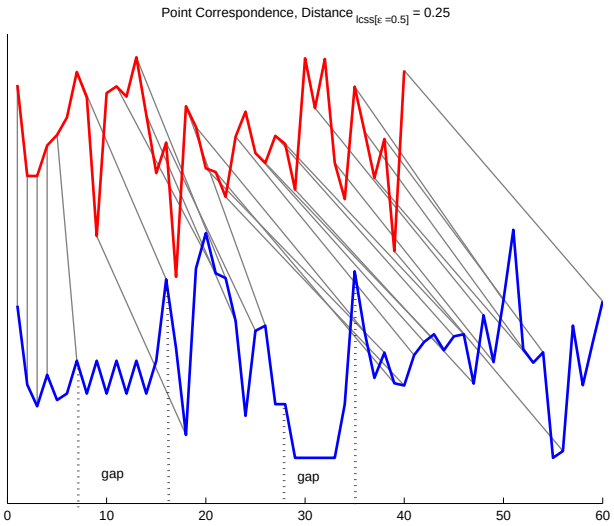



(b) Using LCSS

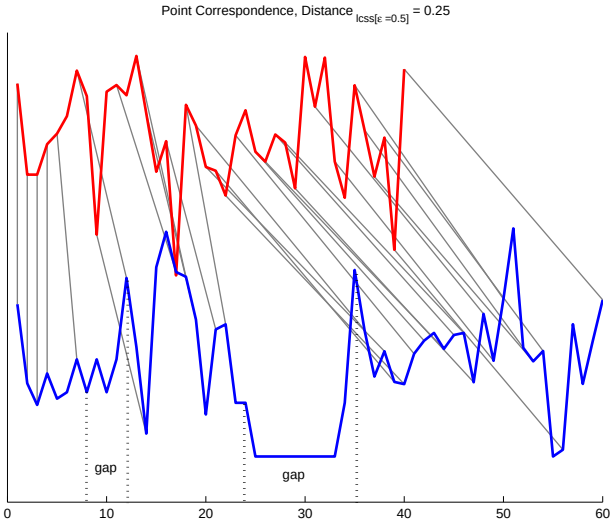
Figure 2.3: Point correspondence when two similar time series contain noise

shifting, ability to handle sequences that contain noise, whether a matching threshold is needed, computation cost, and whether the distance function is a metric.

From Table 2.1, the following observations can be made:




(a) Using LCSS




(b) Using LCSS

Figure 2.4: Point correspondence for LCSS on sequences with different sizes of gap in between

- The computation cost of $L_p$-norm is linear and it is a metric, but it cannot handle time series data with different lengths, local time shifting, or noise.

| Distance Function | Different Lengths | Local Time Shifting | Noise | Setting Matching Thresh-old | Computation Cost | Metric |
|---|---|---|---|---|---|---|
| $L_p$-norm | No | No | No | No | $O(N)$ | Yes |
| DTW | Yes | Yes | No | No | $O(N^2)$ | No |
| LCSS | Yes | Yes | Yes | Yes | $O(N^2)$ | No |

Table 2.1: Comparison among three distance functions on raw representations of time series data

- The computation cost of DTW and LCSS are quadratic, and they are not metric distance functions; it is not possible to improve the retrieval efficiency with distance-based access methods.
- DTW can handle time sequences with different lengths and local time shifting, but it is sensitive to noise.
- LCSS needs a predefined matching threshold to handle noise; it can also handle time sequences with different lengths and local time shifting.

### 2.2.2 Distance Functions on Other Representations

Besides raw format representation, there are several other representations, together with distance functions, to measure the similarity between time series. These representations are proposed either for particular applications, such as financial data analysis [110], or for the purpose of dimensionality reduction [54].

#### Piecewise Liner Approximation and Distance Functions

Piecewise linear approximation (PLA) has been proposed as a useful form of data compression and noise filtering [79]. It approximates the time series data as a sequence of linear functions (lines) as shown in Figure 2.5. One of the key issues with PLA is how to

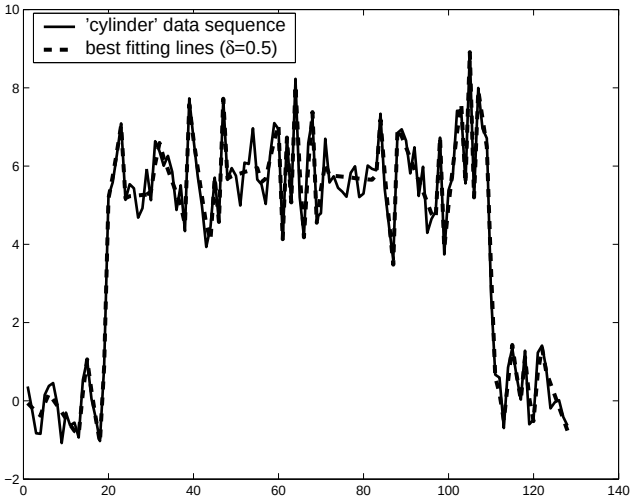
optimally segment the time series into subsequences which can be approximated by linear functions. Many algorithms have been proposed [95, 85, 56], but, since the solution involves a tradeoff between accuracy and data compression ratio, it still remains as an open problem. As shown in Figure 2.5, with different allowance errors, the same time series data can be approximated by different number of line segments. There are several distance functions that have been proposed for PLA.

1. Probabilistic Distance.

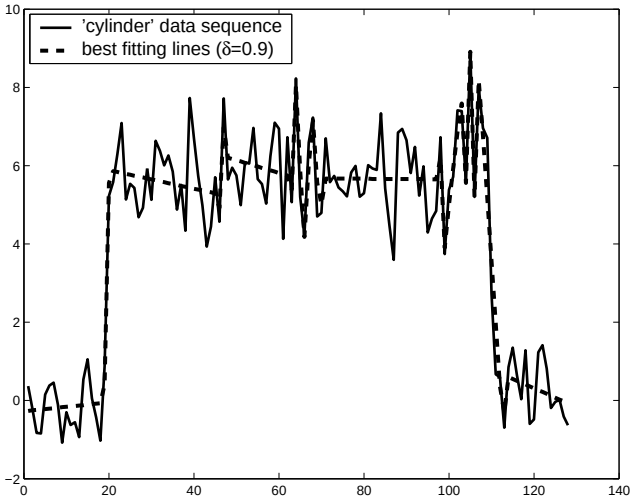
   Keogh and Smyth [56] propose a probabilistic model which is based on PLA representation of time series data. First, the time series data (including query time series) are segmented and PLA is used to represent the data. Then, a set of local features (such as peaks, troughs, and plateaus) are extracted from the time series data to describe the characteristics of the data. Finally, the overall distance is computed based on the local features and relative positions of individual features. The computation cost of proposed distance function is $O((\frac{N}{s})^{Q_f})$, where $N$ is the number of data points in the data sequence, $s$ is the scaling factor, which records number of data points in each sub-segment, and $Q_f$ is the number of local features in each query sub-segment. The computation cost of the distance function is exponential. Even with some heuristic search methods [56], the computation cost is still quadratic.

2. Modified Lp-Norm distance

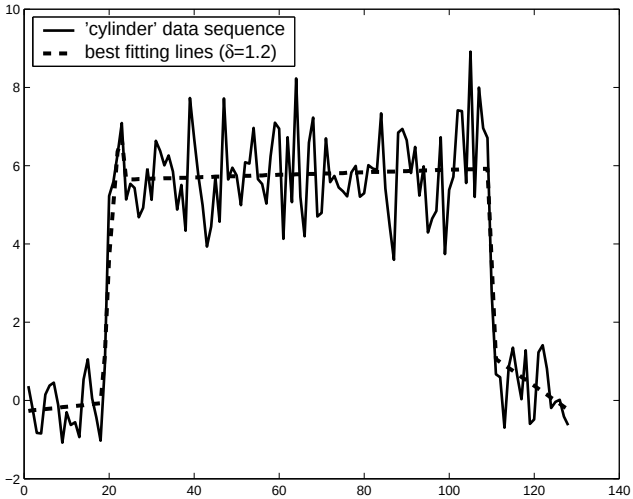
   Morinaka et al. [71] propose a modified $L_p$-norm distance on PLA representation in order to quickly find the approximate answers without false dismissals. In this context, PLA works not only as a representation, but also as a dimensionality reduction technique (this will be discussed in detail in Section 2.3.1). In order to avoid false dismissals, the modified $L_p$-norm distance is compensated by the value of potential approximation error deviation of a line segment. The computation cost of the distance function is linear. However, no indexing structures have been designed for this distance function.




(a) best fitting lines with allowance error $\delta = 0.5$




(b) best fitting lines with allowance error $\delta = 0.9$




(c) best fitting lines with allowance error $\delta = 1.2$

Figure 2.5: Piecewise linear approximation lines for Cylinder data with different allowance error

### Frequency Transformation Representation and Distance Functions

Most of the frequency transformation representations, such as *Discrete Fourier Transformation* (DFT), and *Discrete Wavelet Transformation* (DWT), are mainly treated as

dimensionality reduction techniques, and $L_p$-norm is applied to the transformed representations to measure their similarity. The details of these transformations will be discussed in Section 2.3.1 under dimensionality reduction techniques. In this section, the distance functions on these transformed representations are reviewed. According to Parseval's Theorem [77], which states that DFT preserves the energy of a signal, Euclidean distance on the first few DFT coefficients is a lower bound of the distance in the original time sequence space [31]. Therefore, no false dismissals are introduced when answering queries with an index structure (e.g. R-tree [38]) that is created on the first few DFT coefficients.

Compared to DFT, DWT coefficients describe properties of time sequence both at various locations and at various time granularity. Each time granularity here refers to the level of detail that can be captured by DWT (the whole sequence or the subsequences). Therefore, DWT is more often used as a representation scheme rather than as as a pure dimensionality reduction technique. Besides computing Euclidean distances on transformed DWT coefficients to measure the similarity [82, 59], several other distance functions are proposed on DWT coefficients.

Huhtala et al [41] propose a probability-based measure derived from the cosine similarity. The *inverse angle probability* of two Haar wavelet coefficient vectors [42] $x = \langle x_1, \ldots, x_n \rangle$ and $y = \langle y_1, \ldots, y_n \rangle$ is defined as:

$$iap(x, y) = -ln(Pr(sim(x, \hat{x}) \leq sim(x, y))) \tag{2.12}$$

where $sim(x, y) = |x_1 y_1 + \ldots + x_n y_n|$ and $\hat{x}$ is a random vector generated by choosing each value independently from the same normal distribution with zero mean. The inverse angle probability overcomes the problem that cosine similarity cannot reflect the significance of a particular value when the significance depends on dimensionality of the vectors.

Struzik and Siebes [98] propose a weighted correlation distance function on transformed Haar wavelet coefficients of time series data which has been experimentally demonstrated to be closely related to the subjective "feeling" of the similarity of time series. Given two Haar wavelet coefficient vectors $x = \langle x_1, \ldots, x_n \rangle$ and $y = \langle y_1, \ldots, y_n \rangle$ of two time series data, the weighted correlation between $x$ and $y$ is defined as:

$$C(x, y) = \sum_{i,j=0}^{n} w_i x_i w_j y_j \delta_{i,j} \tag{2.13}$$

where $\delta_{i,j} = 1$ iff $i = j$, and $w_i$ and $w_j$ are weights which depend on the respective scales $i$ and $j$.

The distance function between $x$ and $y$ is defined as:

$$dist(x, y) = -ln(|\frac{C(x, y)}{\sqrt{C(x, x)C(y, y)}}|) \tag{2.14}$$

**Landmark and Important Points Representation and Distance Function**

Researchers in psychology and cognitive science have observed that humans, to some extent, consider two charts similar if their landmarks (turning points) are similar. Perng et. al [80] define a landmark model for time series data, where points (times, events) of *greatest importance* are captured and stored. Depending on the application, important points of time series can range from simple predicates (e.g. local maxima or local minima) to more sophisticated formats (e.g. a point is $n^{th}$ order landmark of a curve if the $n^{th}$ derivative is 0 on that point). These important points are used to measure similarity between two time series.

Given two sequences of landmarks $L^R = [L_1^R, \ldots, L_n^R]$ and $L^S = [L_1^S, \ldots, L_n^S]$ of time series $R$ and $S$, respectively, where $L_i^R = (t_i^R, x_i^R)$ and $L_i^S = (t_i^S, x_i^S)$, the distance between the $k^{th}$ landmarks is

$$dist_k(L^R, L^S) = (\delta_k^{time}(L^R, L^S), \delta_k^{amp}(L^R, L^S)) \tag{2.15}$$

where

$$\delta_k^{time}(L^R, L^S) = \begin{cases} \frac{\|(t_k^R - t_{k-1}^R) - (t_k^S - t_{k-1}^S)\|}{(\|t_k^R - t_{k-1}^R\| + \|t_k^S - t_{k-1}^S\|)} & \text{if } 1 < k \leq n \\ 0 & \text{otherwise} \end{cases} \tag{2.16}$$

$$\delta_k^{amp}(L^R, L^S) = \begin{cases} 0 & \text{if } x_k^R = x_k^S \\ \frac{\|x_k^R - x_k^S\|}{(\|x_k^R\| + \|x_k^S\|)} & \text{otherwise} \end{cases} \tag{2.17}$$

The distance between the two landmark sequences is

$$dist(L^R, L^S) = (\|\delta^{time}(L^R, L^S)\|, \|\delta^{amp}(L^R, L^S)\|) \tag{2.18}$$

where $\|\cdot\|$ is a vector norm, $\delta^{time}(L^R, L^S) = \langle \delta_1^{time}(L^R, L^S), \ldots, \delta_n^{time}(L^R, L^S) \rangle$, and $\delta^{time}(L^R, L^S) = \langle \delta_1^{time}(L^R, L^S), \ldots, \delta_n^{time}(L^R, L^S) \rangle$. However, the important point and feature (format of value representation, e.g. consecutive value difference, consecutive time difference, and consecutive value difference ratio) selection is subjective, which directly affects the effectiveness of matching quality.

### ARIMA Model and LPC Cepstral Coefficient Representation and Distance Function

In statistical analysis of time series data, a time series is considered to be a sequence of observations of a particular variable. It consists of four components: a trend, a cycle, a stochastic persistence component, and a random element [90]. The four components can be captured by an Auto-Regressive Integrated Moving Average (ARIMA) model [113]. Therefore, the similarity between time series data can be modelled by the similarity between two corresponding ARIMA models. The advantage of using this model is that it requires a few fixed number of parameters compare regardless of the lengths of time series.

Autocorrelation functions (ACFs) and partial autocorrelation functions (PACFs) are usually used to determine models to which time series data belong. Thus, time series with similar ACFs or PACFs are likely to belong to the same statistical model. Wang et al. [110] propose to use ACFs and PACFs to represent time series data and measure similarity between two ACFs or PACFs using Euclidean distance. If the distance is smaller than a predefined threshold, two ACFs or PACFs will have the similar shapes or movement patterns, and the two time series represented by ACFs or PACFs are similar.

Kalpakis et al. [50] propose using Euclidean distance between Linear Predictive Coding (LPC) centrums of two ARIMA models to measure the similarity between two time series data. They prove experimentally that using LPC centrum is more effective and efficient than applying DFT, DWT, and PCA (Principle Component Analysis) on ARIMA models in tasks that involve clustering of time series data.

### Symbolic Representation and Distance Function

Since many distance functions, algorithms, and data structures have been developed for strings, it is intuitive to consider the possibility of converting real valued time series data

into symbolic representation and applying string matching techniques to time series retrieval. Agrawal et al. [4] propose $\mathcal{SDL}$, which is a language for describing and retrieving the "shape" of one dimensional time series. The "shape" is defined based on the difference of every pair of consecutive values, which is quantized and represented by a distinct symbol of a predefined alphabet, such as the one in Table 1.1. Thus, a similarity-based search over time series data is converted to a matching problem between the regular expression derived from a query time series and words transformed from time series data in the database. Huang and Yu [43] propose an Interactive Matching of Patterns with Advanced Constraints in Time-series (IMPACT) method to transform time series data into symbolic representation using change ratios between consecutive values. A general suffix tree is used to index the strings. IMPACTS can handle dynamic query constraints with different degrees of accuracy and dynamically specified combinational patterns. Lin et al. [64] propose a symbolic representation of one dimensional time series data by first transforming it into *piecewise aggregate approximation* (PAA) (PAA will be discussed in detail in dimensionality reduction techniques). Then, the values of PAA are quantized and each quantized region is mapped to a symbol. Finally, by connecting the mapped symbols of a PAA sequentially, a time series is converted into a string.

Instead of mapping values, average values, and differences between values of time series data to symbols, two other approaches are proposed to convert movement slopes of time series data into symbols. Time series data are first segmented using the best line fitting algorithms [95, 85], and then the movement slope of each line segment is quantized into a symbol from a predefined alphabet. By connecting each symbol together according to its appearance order, a string is derived from the time series data.

The distance measures for the above symbolic representation are either exact symbol equality matching [4, 95, 85, 43] or modified Euclidean distance [64].

### Signature Representation and Distance Function

The approaches that convert time series data into strings offer opportunities to apply text indexing methods for similarity search over time series data. Andŕe-Jönsson and Badal [7] propose using signature files to represent time series data. Time series data are first converted into strings by quantizing amplitude differences into discrete symbols from a

predefined alphabet [4]. Then, signature files are generated by sliding a window along each string and mapping the text in the window into a number of signature bits. Similarly, the query signature is generated from the query time series data. Finally, a linear scan is carried out to search signature files in the database. The advantage of the approach is that a signature file is very compact and searching of signature files can be done in linear time. However, since signature files cannot avoid false candidates, the obtained results need to be verified by conducting a similarity search on the original time series data, and because of the loss in accuracy during the conversion from real value to a symbol, a lot of false candidates will be introduced.

To summarize, besides raw representation, many other representations have been proposed for time series data. These representations, together with distance functions proposed on them, are developed for specific applications. It is hard (or unfair) to compare them using a uniform benchmark data set in terms of their efficacy in capturing the characteristics of time series or trajectory data, and efficiency in answering range queries or $k$-NN queries. Nevertheless, these representation and distance functions provide hints for possible solutions to generic applications.

## 2.3 Indexing Techniques

Most of the existing work on indexing time series data follows the GEMINI framework [31]. The key point of GEMINI is the use of a lower bound in a lower dimensional space of the true distance in the original space to guarantee no false dismissals when the index is used as a filter. Several dimensionality reduction techniques have been proposed for this purpose, such as DFT [3, 87], DWT [59, 82, 114], *Single Value Decomposition* (SVD) [58], *Piece-Wise Aggregate Approximation* (PAA) [117, 53], *Adaptive Piece-wise Constant Approximation* (APCA) [54] and *Chebyshev Polynomials* [19]. If the distance measure is a metric, the metric space indexing structures that are proposed in the literature, such as MVP-tree [15], M-tree [26], and Sa-tree [75], can be used to improve the search efficiency. In the next two subsections, the dimensionality reduction techniques and the metric space indexing structures for time series data are reviewed. A complete review of metric space indexing is given in [22].

### 2.3.1 Dimensionality Reduction Techniques

Since time components are not involved in the computation of distance functions defined on raw representation of time series data, such as $L_p$-norm, DTW and LCSS, a one dimensional time series data of length $N$ can be simply treated as a sequence of $N$ real values. As a consequence, these $N$ values can be considered as a data point in a $N$-dimensional space. After this conversion, range queries and $k$-NN queries over time series data can be executed efficiently by using spatial access methods (SAM), such as R-tree [38] and K-d tree [10]. However, for time series data, $N$ is usually in the hundreds or thousands, which leads to the *dimensionality curse* problem [112], so that applying SAM to answer $k$-NN or range queries will cost more than answering queries using linear scan. The common solution to this problem is to apply *dimensionality reduction* techniques to reduce the dimensionality to a range (less than 20) which can be handled by SAM. The dimensionality reduction techniques can be divided into two categories: *exact* and *approximate* dimensionality reduction techniques. The exact dimensionality reduction techniques guarantee that no false dismissals occur when queries are executed on the reduced dimensional space. In order to achieve this, the distance function defined in the reduced dimensional space must be the lower bound of the distance in the original space. The approximate dimensionality reduction techniques do not guarantee no false dismissals, therefore, no lower bounding distance function needs to be defined. Obviously, all the exact dimensionality reduction techniques can be considered as approximate methods by removing the lower bounding constraint. Within the context of this thesis only exact dimensionality reduction techniques are relevant, since there is a strong requirement that no false dismissals occur.

All exact dimensionality reduction techniques should follow the following theorem in order not to introduce false dismissals.

**Theorem 2.3** *(Lower Bounding Lemma [31]) To guarantee no false dismissals, a dimensionality reduction technique $F$ must satisfy:*

$$dist_{reduced-dimensional-space}(F(O_1), F(O_2)) \leq dist_{original-space}(O_1, O_2) \tag{2.19}$$

*where $O_1$, and $O_2$ are the data points in the original space and dist is a distance function that measures the distance between $O_1$ and $O_2$.*

Theorem 2.3 also indicates an efficiency measure for dimensionality reduction techniques: the tightness of the lower bound distance. The tighter the lower bound distance to the original distance, the fewer false alarms are introduced by the dimensionality reduction techniques. Thus, the tightness of the lower bounding distance is an implementation-free measure of dimensionality reduction techniques, since it is does not relate to any implementation details (such as whether or not the code is optimized). Since dimensionality reduction techniques only deal with sampled vector values, time series data are simply represented as sequences of values in the rest of this chapter.

### Discrete Fourier Transform (DFT)

The *Discrete Fourier Transform* (DFT) is the first dimensionality reduction technique that was introduced for indexing time series data [3]. DFT describes a time series by a set of sine/cosine waves. A complex number, called a *Fourier Coefficient*, is used to represent each wave. A time series data of length $N$ can be decomposed into $N$ sine/cosine waves that can be combined into the original time series data. Since many of the waves have very low amplitudes, DFT coefficients can be used to approximate the original time series by discarding coefficients that have very low amplitudes.

Given a time series $R = \langle r_0, \ldots, r_{N-1} \rangle$, its DFT is a sequence $\overline{R} = \langle \overline{r_0}, \ldots, \overline{r_{N-1}} \rangle$, where

$$\overline{r_F} = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} r_i e^{-j\frac{2\pi F i}{N}} \tag{2.20}$$

$F = 0, \ldots, N-1$, and $j = \sqrt{-1}$.

The time sequence $R$ can be recovered from $\overline{R}$ by *Inverse Discrete Fourier Transform* (IDFT)

$$x_i = \frac{1}{\sqrt{N}} \sum_{F=0}^{N-1} \overline{r_F} e^{j\frac{2\pi F i}{N}} \tag{2.21}$$

where $i = 0, \ldots, N-1$.

One of the properties of DFT is that it follows Parseval's Theorem [79], which states that DFT preserves the *energy* of the original sequence. The energy of a sequence is defined as follows.

**Definition 2.4** *Given a time sequence R, its energy $E(R)$ is defined as the sum of energies (square of the amplitude $r_i$) at each point of the R:*

$$E(R) = \sum_{0}^{N-1} \|r_i\|^2 \tag{2.22}$$

**Theorem 2.4** *(Parseval's Theorem)*

$$\sum_{i=0}^{N-1} |r_i|^2 = \sum_{F=0}^{N-1} |\overline{r_F}|^2 \tag{2.23}$$

Since DFT is a linear transformation [77], given two time sequences $R$ and $S$, Parseval's Theorem also holds on Euclidean distance between two sequences.

$$L_2 - norm(R, S) = L_2 - norm(\overline{R}, \overline{S}) \tag{2.24}$$

Thus, for dimensionality reduction purposes, only the first few $n$ ($n \ll N$) DFT coefficients are used to computed Euclidean distance, which is lower bound to Euclidean distance on $R$ and $S$. In other words, DFT satisfies the Lower Bounding Lemma, and no false dismissals will be introduced.

### Discrete Wavelet Transform

*Discrete Wavelet Transform* (DWT) has also been used as a dimensionality reduction technique [59, 82, 114]. Compared to DFT, DWT is a multi-resolution representation of time series data, and it can approximate time series from global sequences to local subsequences. Furthermore, unlike DFT that only offers the frequency information, DWT offers time-frequency location information.

In DWT, the basis of wavelets is a set of functions defined on real numbers $R$.

$$\psi_{j,k}(t) = 2^{j/2}\psi(2^j t - k) \tag{2.25}$$

where $2^j$ is the scaling (stretching) of $t$, and $k$ is the translation (shift) on $t$.

Any signal (time series) $f(t) \in L_2 - norm$ space can be uniquely represented by the following series, which is called wavelet transform of function $f(t)$.

$$f(t) = \sum_{j,k \in Z} a_{j,k} \psi_{j,k}(t) \tag{2.26}$$

where $a_{j,k}$ is called the *DWT coefficient* of $f(t)$ and it can be calculated by the inner products $\langle \psi_{j,k}(t), f(t) \rangle$.

The Haar wavelets are the most widely used wavelets in image [96], speech [1], and signal [5] processing because they can be computed quickly (linear time cost) and they can be easily used to illustrate the main features of wavelets. Haar wavelet transform is used here as an example to show how the time series can be transformed to wavelets.

Haar wavelet transform can be considered a series of averaging and differentiating operations on a discrete time function (a time series data). In these operations, the average and difference between every two adjacent values of a time series is computed. For example, given a time series $R = [8, 6, 2, 4]$, the Haar wavelet transform is listed as follows:

| Resolution | Averages | Coefficients |
|---|---|---|
| 4 | $(\underbrace{8, 6}, \underbrace{2, 4})$ | |
| 2 | $(\underbrace{7, 3})$ | (1, -1) |
| 1 | (5) | 2 |

Haar wavelet transform of $R$ is $Harr(R) = [c, d_0^0, d_0^1, d_1^1] = [5, 2, 1, -1]$, which is composed of the last average value 5 and the coefficients. The rest of the coefficients (including the original time series data) can be reconstructed by adding difference values to or subtracting difference values from averages, recursively.

A lower bound distance has been defined on Haar wavelet transformed space, so Harr wavelet transform follows the lowering bounding Lemma [59].

**Theorem 2.5** *Let $R$ and $S$ be two time series with the same length $N$ ($N$ is a power of 2) and $Harr(R)$ and $Harr(S)$ be the Haar Transforms of $R$ and $S$, respectively. Let $Harr(R) - Harr(S) = [C, D_1, \ldots, D_{N-1}]$. The Euclidean distance $L_2 - norm(R, S) = W_{log_2 N}$ can be recursively computed from $[C, D_1, \ldots, D_{N-1}]$ as follows:*

$$W_0 = C \tag{2.27}$$

$$W_{i+1} = \sqrt{2(W_i^2 + D_{2^i}^2 + D_{2^i+1}^2 + \ldots + D_{2^{i+1}-1}^2)} \quad (2.28)$$
$$for\ 0 \leq i \leq log_2 N - 1$$

**Corollary 2.1** *If the first n* $(1 \leq n \leq N)$ *dimensions of Haar transform are used, no false dismissals will occur.*

The above theorem and corollary only prove that Haar wavelet transform will not introduce false dismissals. Popivanov and Miller [82] have proven that other wavelet transforms, such as Daubechies wavelet transform [29], will not introduce false dismissals, either. However, there exist a discrepancy regarding the performance study of DWT. Chan and Wu [59] claim that Haar wavelets outperform other wavelets such as Daubechies and Coiflet wavelets, while Popivanov and Miller [82] find that Daubechies wavelets work better than Haar wavelets. Thus, more work is required to test the performance of different types of wavelets.

### Single Value Decomposition

As a dimensionality reduction technique, *Single Value Decomposition* (SVD) has been widely used in content-based retrieval of text documents [40], time series data [58] , and image databases [76, 51]. The intuition behind SVD is that given a set of $N$ dimensional points, these points will only "concentrate" on a few dimensions, on which they can be discriminated from each other very well. SVD is used to find these dimensions and map the points to these dimensions. Figure 2.6 shows that by rotating the $x - y$ axis to $x' - y'$ axis and only projecting the 2D points to $x'$ axis, all the points can be well discriminated on the new axis.

A set of $N$ dimensional time series data can be represented as $L \times N$ matrix $A$, where $L$ is the number of time series data in the set. $A$ can be transformed using SVD as follows.

$$A = U\Sigma V^T \quad (2.29)$$

where $U$ is a column-orthogonal $L \times N$ matrix, $\Sigma$ is a diagonal $N \times N$ matrix of the *eigenvalues* of the matrix $A$, and $V$ is a column-orthogonal $N \times N$ matrix, which is called

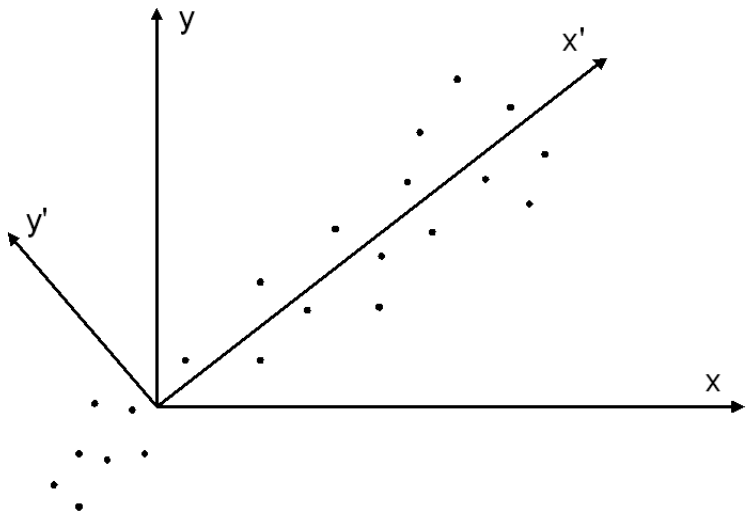



Figure 2.6: Illustration of SVD projection of data on the axis x'

*basis matrix* of SVD. Since $U$ and $V$ are orthogonal:

$$UU^T = I_L \tag{2.30}$$
$$U^TU = I_N \tag{2.31}$$
$$VV^T = V^TV = I_N \tag{2.32}$$

where $I$ is the identity matrix.

The SVD transformed data is given by $U \times \Sigma$ that can be computed by $A \times V$. In fact, $U \times \Sigma$ is a rotation of the original matrix $A$, the dimensionality of transformed data sequences does not change. By only taking the first $n$ largest eigenvalues of $\Sigma$ and corresponding entries in $A$, $U$, and $V$, the dimensionality can be reduced from $N$ to $n$ in the transformed space.

SVD is better than other transforms such as DFT or DWT in terms of minimizing the reconstruction error. However, SVD needs the whole data set to do the transform. Thus, the data set should not be updated frequently. Furthermore, the computation and space costs of SVD is high: $O(LN^2)$ and $O(LN)$, respectively. Finally, SVD transformation is based on all the data in the database, and it assumes the query data follows the same data

distribution as the data in the database. However, for a similarity-based search, the query data is usually not in the database. Therefore, SVD may introduce a loose lower bound of the distance between query data and data in the database, which may leads many false candidates.

### Piecewise Aggregate Approximation

*Piecewise Aggregate Approximation* (PAA) was introduced by Yi and Faloutsos [117] and Keogh et al. [53] independently. The idea of PAA is quite straightforward. Given a time series data of length $N$, PAA divides it into $n$ segments of equal length and represents each segment using the average of the segment. Thus, a $N$-dimensional point is mapped to a $n$-dimensional point where $n < N$. Consider a time series $R = \langle r_1, \ldots, r_N \rangle$, the PAA representation of $R$ is $\overline{R} = \langle \overline{r_1}, \ldots \overline{r_n} \rangle$ where:

$$\overline{r_i} = \frac{n}{N} \sum_{j=\frac{N}{n}(i-1)+1}^{\frac{N}{n}i} r_j \tag{2.33}$$

The distance in the reduced dimension is defined as:

$$dist_{PAA}(\overline{Q}, \overline{R}) = \sqrt{\frac{N}{n}} \sqrt{\sum_{i=1}^{n} (\overline{q_i} - \overline{r_i})^2} \tag{2.34}$$

Yi and Faloutsos [117] and Keogh et al. [53] also prove that PAA follows lower bounding lemma for Euclidean distance. Compared to other transforms (such as DFT or SVD), PAA is easy to understand and requires less time on transformation. Furthermore, PAA can handle general $L_p$-norm and weighted $L_p$-norm without introducing false dismissals.

### Adaptive Piece Constant Approximation

*Adaptive Piece Constant Approximation* (APCA) relaxes the constraints of PAA that requires each segment be the same length and allows segments with different lengths [54]. The intuition of APCA is that regions with great fluctuations (high activity) can be represented with several short length segments, and flat regions (low activity) can be represented

with fewer long segments. Given a time series $R = [r_1, \ldots, r_N]$, its APCA representation is

$$C = [\langle cv_1, cr_1 \rangle, \ldots, \langle cv_n, cr_n \rangle] \tag{2.35}$$

where $cv_i$ is the mean value of the data points in i$^{th}$ segment, $cr_i$ is the right endpoint of i$^{th}$ segment, and $n$ is the number of segments.

In order to define a distance function on APCA to lower bound Euclidean distance on time sequences, Keogh et al. [54] propose a special version of the APCA for query time sequences. Given a query time series $Q$ and a time series $S$ with its APCA representation $C$, the APCA representation $Q'$ of $Q$ is obtained by projecting the endpoints of $C$ onto $Q$ and computing the mean value of the sections of $Q$ that fall within the projected interval.

$$Q' = [\langle qv_1, qr_1 \rangle, \ldots, \langle qv_n, qr_n \rangle] \tag{2.36}$$

where $qr_i = cr_i$ and $qv_i = mean(Q_{cr_{i-1}+1}, \ldots, Q_{cr_i})$.

The lower bounding distance is defined as:

$$D_{LB}(Q', C) = \sqrt{\sum_{i=1}^{n} (cr_i - cr_{i-1})(qv_i - cv_i)^2} \tag{2.37}$$

Keogh et al. [54] also point out that APCA is indexable by mapping each APCA representation to a $2n$-dimensional point, and define a lower bound distance between $Q'$ to the indexing node (a bounding rectangle). The experimental results shows that APCA outperforms other indexable representations such as DFT, DWT, and PAA in terms of efficiency.

### Piecewise Linear Approximation

*Piecewise Linear Approximation* (PLA) has been widely used in time series analysis [79] and answering approximate queries [95]. As introduced before, PLA is a method to approximate a time series by a sequence of line segments. It has been widely used as a representation format for time series data. After converting time series data to PLA, it is easy to compute the approximate distance between converted line segments. However, so far there is only one proposal to use PLA to answer queries without introducing false dismissals. Morinaka

et al. [71] propose a complex distance function on the converted PLA space and prove that the distance on PLA is a lower bound of the real distance between two time sequences. They first apply *least square approximation* (LSA) method to segment a time series data to lines. Each line is the longest possible segment whose accumulated error does not exceed a predefined deviation threshold $\Delta$. Then, an approximate distance is defined on the two segmented line sequences based on linear functions that describe each line segment. Two different distances are defined, one to account for the case when line segments intersect each other and the other when they do not. However, because the line segment may over- or under-estimate the time sequence, the approximate distance of two line segment sequences may be larger than the real distance between the two time sequences. Therefore, a bound *wed* (*worst error deviation*), is proposed to bound the approximate error deviation of a line segment. For each line segment, *wed* is computed based on difference between values of original subsequence that are above and below the line segment. By adding the *wed* to the approximate distance, it is possible to guarantee that modified approximate distance is the lower bound of the real distance in the original time series space.

Even though PLA offers very promising pruning ability, no indexing schemes for it have so far been proposed.

### Chebyshev Polynomials Approximation

Recently, Cai and Raymond proposed an effective dimensionality reduction and indexing technique for time series data, called Chevyshev Polynomials Approximation [19]. They model approximate time series data as a min-max polynomial problem that minimizes the maximum deviation from the true value through lower order continuous polynomials. Chebyshev Polynomials is selected because Chebyshev approximation is almost identical to the optimal min-max polynomial. A function $f(t)$ can be approximated by a set of Chebyshev polynomials as follows:

$$f(t) \simeq c_0 P_0 + \ldots + c_m P_m \tag{2.38}$$

where $P_i$ is Chebyshev polynomial with degree $i$ and $c_i$ is the coefficient.

Thus, for a collection of time series of length $N$, Chebyshev polynomials of degree $m$ ($m \ll N$) can be used to approximate them and reduce the dimensionality to $n = m + 1$.

Since Chebyshev polynomial coefficients can only be computed for interval functions, a time series which is a *discrete* function has to be converted to an interval function by the following steps:

1. Convert a time series $R = [r_1, \ldots, r_N]$ to $\overline{R} = [(r_1, s_1), \ldots, (r_N, s_N)]$, where $s_i$ is the derived timestamp according to the appearance position of $r_i$ and $s_i$ is normalized to $[-1, 1]$.

2. Divide the interval $[-1, 1]$ to $N$ disjoint subintervals, $I1$, ..., $I_N$:

$$I_i = \begin{cases} [-1, \frac{s_1+s_2}{2}) & if\ i = 1 \\ [\frac{s_{i-1}+s_i}{2}), \frac{s_i+s_{i+1}}{2}) & if\ 2 \leq i \leq N-1 \\ [\frac{s_{N-1}+s_N}{2}, 1]) & if\ i = N \end{cases} \tag{2.39}$$

3. Create an interval function $f(r)$ of $R$:

$$f(r) = \frac{r_i}{\sqrt{w(r)|I_i|}}\ if\ r \in I_i (for 1 \leq i \leq N) \tag{2.40}$$

   where $|I_i|$ is the length of subinterval $I_i$ and $w(r)$ is a weight function:

$$w(r) = \frac{1}{\sqrt{(1-r^2)}} \tag{2.41}$$

After converting a time series data to an interval function $f(r)$, the $n(= m+1)$ Chebyshev coefficients can be computed as follows:

$$c_0 = \frac{1}{N}\sum_{j=1}^{N} f(r_j)P_0(r_j) = \frac{1}{N}\sum_{j=1}^{N} f(r_j) \tag{2.42}$$

$$c_i = \frac{2}{N}\sum_{j=1}^{N} f(r_j)P_i(r_j) \tag{2.43}$$

where $P_i(r_j)$ is Chebyshev polynomial of degree $i$.

The distance between two sequences of Chebyshev coefficients $\overline{Q} = [a_0, \ldots, a_m]$ and $\overline{R} = [b_0, \ldots, b_m]$ that are computed from two time series $S$ and $T$ is defined as:

$$dist_{cby}(\overline{Q}, \overline{R}) = \sqrt{\frac{\pi}{2} \sum_{i=0}^{i=m} (a_i - b_i)^2} \tag{2.44}$$

Cai and Ng prove that $dist_{cby}(\overline{Q}, \overline{R})$ is a lower bound of $L_2$-Norm ($Q$, $R$). Thus, using Chebyshev Polynomials as a dimensionality reduction technique will not introduce false dismissals.

### Comparison of Dimensionality Reduction Techniques

Table 2.2 is a comparison of dimensionality reduction techniques discussed in this section according to whether they have a limitation on lengths, the computation cost, their computation space cost, whether they are indexable, and whether they support weighted $L_p$-norm.

| Dim Reduction Tech. | Length Limitation | Computation Cost | Space cost | Indexable | Support Weighted $L_p$-norm |
|---|---|---|---|---|---|
| PLA | No | $O(N)$ | $O(N)$ | No | Yes |
| DFT | No | $O(Nlog(N))$ | $O(N)$ | Yes | No |
| DWT | Yes ($2^n$) | $O(N)$ | $O(N)$ | Yes | No |
| SVD | No | $O(LN^2)$ | $O(LN)$ | Yes | No |
| PCA | No | $O(N)$ | $O(N)$ | Yes | Yes |
| APCA | No | $O(N)$ | $O(N)$ | Yes | Yes |
| Chebyshev | No | $O(N)$ | $O(N)$ | Yes | Yes |

Table 2.2: Comparison of dimensionality reduction techniques of time series data

From Table 2.2, the following are observed:

- In terms of the computation and space cost, SVD is the worst method among the dimensionality reduction techniques.

- PLA, PCA, APCA, and Chebyshev are all promising techniques in terms of supporting distance functions other than Euclidean distance.
- Only DWT has limitation on lengths of time series.
- All except PLA are indexable.

In terms of efficiency of these dimensionality reduction techniques (i.e., the tightness of lower bound distances), experimental results show that APCA is better than DFT, DWT, SVD, and PAA [54] and Chebyshev is tighter than APCA [19]. Therefore, Chebyshev is the most effective dimensionality reduction technique.

### 2.3.2 Metric Space Indexing

As discussed above, one way to deal with the retrieval of high-dimensional data is to reduce data dimensionality and then plug in a spatial access method. Another alternative is to use metric space indexing methods to improve the retrieval efficiency as long as the defined distance function is a metric. An extensive survey of these techniques recently been published [22], so only a short overview is provided here.

Given a data space $\mathcal{D}$ of objects and a nonnegative distance function $dist : \mathcal{D} \times \mathcal{D} \rightarrow R^+$ defined over $\mathcal{D}$, $dist$ is a metric distance function, and $\mathcal{D}$ is referred to a metric space, if $dist$ has the following properties:

- $dist(x, y) = 0 \Leftrightarrow x = y$
- $dist(x, y) = d(y, x)$
- $dist(x, z) \leq d(x, y) + d(y, z)$

The last condition is called triangle inequality property, and it is used extensively by most metric indexing structures for similarity search to reduce the query time.

The metric space indexing structures can be divided into two categories:

- *pivot-based* indexing structures;
- *clustering-based* indexing structures.

In *pivot-based indexing*, a set of pivots (reference points) $p_1, \ldots, p_k$ are selected from the database. For each data $x$ in the database, its distance to $k$ pivots $(dist(x, p_1), \ldots, dist(x, p_k))$ are computed and stored. Given a query data $Q$ and query range $r$, the distance from $Q$ to each pivot $p_i$, $dist(Q, p_i)$, is computed first. If it is known that $|dist(Q, p_i) - dist(x, p_i)| > r$ holds for some $i$, then it is not necessary to compute the distance between $x$ and $Q$ because, by triangle inequality, $dist(Q, x) > r$. Many indexing structures and their variants follow this idea, such as AESA [94], LAESA [68], spaghetti [20], fa-tress [9], fq-array [21] and OMNI-family of access methods [32]. There are a number of tree-like indexing structures, such as bk-trees [17, 93], metric tress [105], tlasea [67] and vp-tree with its variants [105, 14], which use selected pivots as the roots of a tree or subtree. The distance range between each root to all its children are computed and stored. The similar pruning steps as the above example can be used to traverse the tree.

OMNI-family access methods are not included in the review of [22], which are briefly reviewed here.

Given a metric data space $\mathcal{D}$, OMNI-family selects a set of representative points, called *foci* (pivots), which is defined as *foci-base*.

$$foci - base = \{f_1, f_2, \ldots, f_c | f_k \in \mathcal{D}, f_k \neq f_j, k \neq j\} \tag{2.45}$$

where $c$ is the number of foci selected from $\mathcal{D}$.

For each data $x \in \mathcal{D}$ and $x \neq f_i$, the *OMNI-coordinate* $OC_x$ is the set of distances from $x$ to each foci in foci-base:

$$OC_x = \{dist(x, f_1), dist(x, f_2), \ldots, dist(x, f_c)\} \tag{2.46}$$

Given a query $Q$, its OMNI-coordinate is computed and used to search in the OMNI-coordinates of the data set to prune false candidates. Several access methods are proposed for this pruning step, such as OMNI-sequential, OMNIB-tree, and OMNIR-tree, which make up the OMNI-family.

Since the selected foci greatly affect the pruning ability of OMNI-family, Filho et al. propose a Hull Foci (HF) algorithm to choose them. HF first randomly selects a data point $x$ from $\mathcal{D}$, the data point with the farthest distance to $x$ is selected as the first foci $f_1$. Then, the data point that has the farthest distance to $f_1$ is selected as the second foci

$f_2$. After that, HF recursively selects the rest of the foci by choosing a data point that has the most similar distance to the previously selected foci. In order to measure different foci selection algorithms, Bustos et al. [18] propose an efficiency measure to compare two selected foci bases.

*Clustering-based indexing* structures recursively divide the data space into separate subspaces and select a representative point (usually "centroid" of the subspace) together with several data points for each subspace. There are many methods to eliminate a subspace during search. For example, M-trees [26] store the covering radius $cr(c_i)$ for each centroid point $c_i$ of the subspace, which is the maximum distance between any data point in the subspace to $c_i$. Given a range query $Range(Q, r)$, if $d(Q, c_i) - r > cr(c_i)$, the subspace with $c_i$ as centroid can be eliminated. Voronoi trees [30] assign each data point to the subspace which has the smallest distance to its centroid. Given a range query $Range(Q, r)$, the distance between $Q$ to each centroid point $c_i$ is computed; and the $c_i$ that has the smallest distance to $Q$ is selected and denoted as $c$. The subspace will be eliminated if its centroid $c_i$ satisfies $d(Q, c_i) > d(Q, c) + 2r$, which means that the subspace will not intersect with the query ball $Range(Q, r)$. GNAT tree combines the elimination methods of M-tress and Voronoi-trees [16].

# Chapter 3

# Multi-Scale Time Series Histograms

## 3.1 Introduction

As discussed in earlier chapters, similarity-based time series retrieval can be divided into pattern existence queries [4, 84, 95] and shape match queries [3, 14, 31, 54, 88]. For pattern existence queries, users are interested in the existence of the specified pattern, and do not care when and how the pattern appears. Therefore, retrieval techniques for pattern existence queries should be invariant to the following:

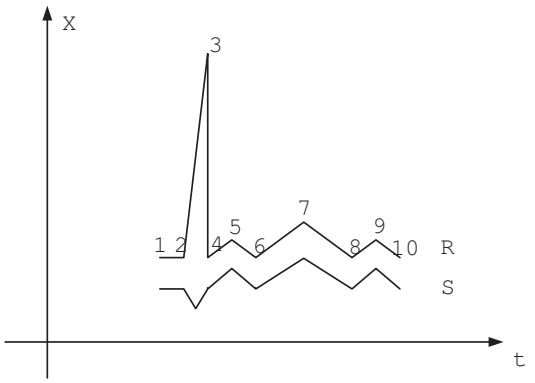
- **Noise.** In Figure 3.1(a), two similar time series $R$ and $S$ are shown. Point 3, which is likely a noise data point, will introduce a large difference value when Euclidean distance is computed between two time series, possibly causing them to be treated as dissimilar.

- **Amplitude scaling and shifting.** In Figure 3.1(b), the two time series $R$ and $S$ are similar in terms of the pattern that they contain (they both have the "two bottom" pattern), but their amplitudes are different (in fact, $R = aS + b$ where $a$ and $b$ are scaling factor and shifting factor, respectively).

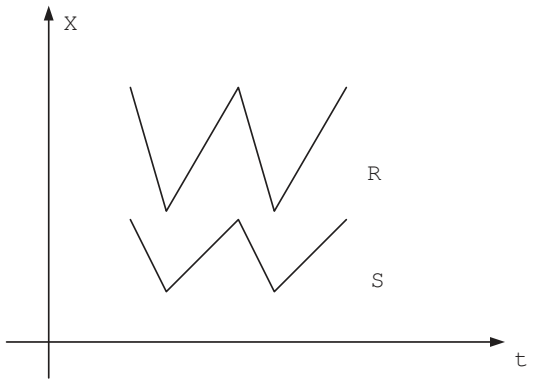
- **Time shifting.** In Figure 3.1(c), time series $R$ and $S$ record the same event (e.g. temperature data) but from different starting times; shifting $R$ to the left on the time axis can align the shape with $S$. $R$ and $S$ are considered dissimilar if they are simply

compared by the respective positions of the data points. However, it can be argued that $R$ and $S$ are similar because they contain the same pattern ("two peaks").

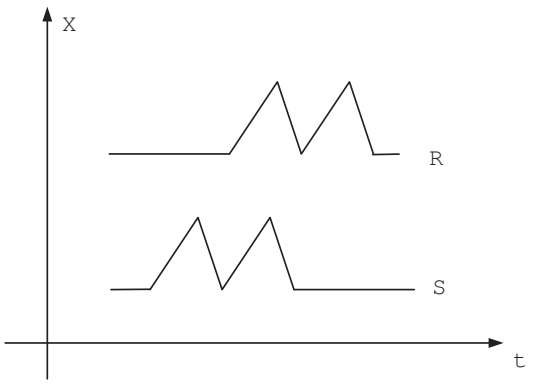
- **Time scaling.** In Figure 3.1(d) time series $R$ and $S$ have different sampling rate. However, both $R$ and $S$ contain "two peaks", therefore, they should be treated as similar.



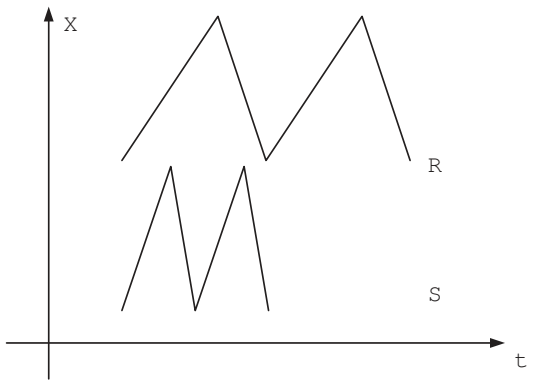

(a) Two similar time series with noise




(b) Two similar time series with amplitude scaling and shifting




(c) Two similar time series with time shifting




(d) Two similar time series with time scaling

Figure 3.1: The different factors that may affect the similarity between two time series

Several approximation approaches have been proposed to transform time series to character strings over a discrete alphabet and apply string matching techniques to find the patterns [4, 84, 95]. However, the transformation process is sensitive to noise. Furthermore, because of the quantization of the value space for transforming time series into strings, data points located near the boundaries of two quantized subspaces may be assigned to different alphabets. As a consequence, they are falsely considered to be different by string comparison techniques.

For shape match queries, users are interested in retrieving the time series that have similar shapes to the query data. Most of the previous work focused on solving these type of queries, such as applying Euclidean distance [3, 31, 54, 88], DTW [12, 118, 57, 52] and LCSS [14, 108] to compute the distance between two data sequences.

There exist applications that require answers from both types of queries. An example is an interactive analysis of time series. Users may be initially interested in retrieving all the time series that have some specific pattern that can be quickly answered by pattern existence queries. They can then apply shape match queries to these results to retrieve those that are similar to a time series that is of interest. However, there are no techniques that have been developed to answer both pattern existence queries and shape match queries. Of course, two different techniques used to answer pattern existence queries and shape match queries can be applied to these applications together; however, different types of representation (e.g. symbolic representation and raw representation), distance functions (e.g. string matching and Euclidean distance), and indexing structures (e.g. suffix tree and R*-tree) will lead to storing redundant information and raise difficulties on improving the retrieval efficiency.

In this chapter, based on the observation that data distributions can capture patterns of time series, a multi-scale histogram-based representation is proposed to approximate time series that is invariant to noise, amplitude shifting and scaling, and time shifting and scaling. The multi-scale representation can answer both pattern existence queries and shape match queries and offers users flexibility to search time series at different precision levels (scales). The cumulative histogram distance [97], which is closer to perceptual similarity than $L_1$-norm, $L_2$-norm, or weighted Euclidean distance, is used to measure the similarity between two time series histograms. Furthermore, distances of lower scale time series

histograms are proven to be lower bounds of higher scale distances, which guarantees that using the multi-step filtering process in answering shape match queries will not introduce false dismissals. In order to reduce the computation cost in computing the distances of time series histograms, distances between averages of time series histograms are used as the first step of the filtering process, since the distances between averages are also lower bounds of distances of time series histograms. Two different approaches in constructing histograms, *equal size bin* and *equal area bin*, are also investigated. The experimental results show that the histogram-based representation, together with the cumulative histogram distance measure, can effectively capture the patterns of time series as well as the shape of time series. Multi-scale time series histograms are compared with another multi-scale representation, wavelet, and the results show that wavelet representation is not suitable to answer pattern existence queries and it is not robust to time shifting when it used to answer shape match queries.

The rest of the chapter is organized as follows: Section 3.2 presents concepts of time series histograms and two variations of them. Multi-scale time series histograms are discussed in Section 3.3. In Section 3.4, experimental results using multi-scale time series histograms on finding patterns and answering shape match queries are presented. Section 3.5 presents the comparison results between multi-scale histograms and wavelet representation in answering pattern existence and shape matching queries.

## 3.2 Time Series Histograms

Given a set of time series $\mathcal{D} = \{R_1, R_2, \ldots, R_L\}$, each time series $R_i$ is normalized into its *normal form* using its mean ($\mu$) and variance ($\sigma$) [37]:

$$Norm(R) = [(t_1, \frac{r_1 - \mu}{\sigma}), \ldots, (t_N, \frac{r_N - \mu}{\sigma})] \tag{3.1}$$

The similarity measures computed from time series normal form are invariant to amplitude scaling and shifting.

Time series histograms are developed in the following way. Given the maximum ($max_{\mathcal{D}}$) and minimum ($min_{\mathcal{D}}$) values of normalized time series, the range $[min_{\mathcal{D}}, max_{\mathcal{D}}]$ is divided into $\tau$ disjoint equal size sub-regions, called *histogram bins*. Given a time series $R$, its

histogram $H_R$ can be computed by counting the number of data points $h_i$ ($1 \leq i \leq \tau$) that are located in each histogram bin $i$: $H_R = [h_1, \ldots, h_\tau]$. Algorithm 3.1 describes how to construct time series histograms for data set $\mathcal{D}$.

**Algorithm 3.1** An algorithm for constructing histograms for data set $\mathcal{D}$

**Input:** a time series set $\mathcal{D}$ and the number of histogram bins $\tau$

**Output:** a histogram data set $H_\mathcal{D}$

```
1: find max_D and min_D of the data set D
2: divide [min_D, max_D] into τ disjoint equal size histogram bins h_i
3: for all each time series R_i of D do
4:    for each data point r_i of R_i do
5:       for each histogram bin h_i do
6:          if h_{i,lowerbound} ≤ r_i < h_{i,upperbound} then
7:             h_i = h_i + 1 ;
8:             break;
9:          end if
10:      end for
11:   end for
12:   insert generated H_{S_i} to the result data set H_D
13: end for
14: return the result data set H_D
```

The time series histogram is normalized by dividing the value of each histogram bin by the total number of data points in the time series. Since a time series histogram is computed from the normal form of a time series, the distance that is computed from two time series histograms are invariant to amplitude scaling and shifting. Furthermore, because time series histograms ignore the temporal information, they are also robust to time shifting and scaling. For example, in Figures 3.1(c) and 3.1(d), the histogram of normalized $R$ is similar to that of $S$. Moreover, since time series histograms show the whole distribution of the data, and noise only makes up a very small portion, comparisons based on histograms can remove the disturbance caused by noise. Therefore, time series histograms are ideal representations for answering pattern existence queries.

$L_1$-norm or $L_2$-norm [100] can be used to measure the similarity between two histograms. However, these do not take the similarity between time series histogram bins into consideration, which may lead to poor comparison results. Consider three histograms $H_R$, $H_S$ and $H_T$ representing three time series of equal length. Assume that $H_R$ and $H_S$ have the same value on consecutive bins and $H_T$ has the same value in a bin which is quite far away from the bins of $H_R$ and $H_S$. $L_1$-norm and $L_2$-norm distances between any two of these three histograms are equal. However, for answering shape match queries, $H_R$ is closer to $H_S$ than it is to $H_T$. Even for pattern existence queries, data points which are located near the boundary of two histogram bins should be treated differently compared to those points that are far apart, which is not considered by $L_1$-norm and $L_2$-norm.

A weighted Euclidean distance can be used to compute the distance between two histograms [39].

**Definition 3.1** *Given two time series $R$ and $S$, the weighted Euclidean distance (WED) between their time series histograms $H_R$ and $H_S$ is:*

$$WED(H_R, H_S) = Z^T A Z \tag{3.2}$$

*where $Z = (H_R - H_S)$, $Z^T$ is the transpose of $Z$, and $A = [a_{ij}]$ is a similarity matrix whose element $a_{ij} = 1 - |j - i|/\tau$ (where $\tau$ is the number of bins) denotes similarity between two time series histogram bins $i$ and $j$. As $a_{ij}$ gets larger, bins $i$ and $j$ become more similar.*

Compared to $L_1$-norm and $L_2$-norm, WED underestimates distances because it tends to estimate the similarity of data distribution without a pronounced mode [97].

Cumulative histogram distances [97] overcome the shortcomings of $L_1$-norm, $L_2$-norm and WED, the similarity that is measured by cumulative histogram distance is closer to the perceptual similarity of histograms. Therefore, this distance function is used to measure the similarity between two time series histograms.

**Definition 3.2** *Given a time series $R$ and its time series histogram $H_R = [h_1, h_2, \ldots, h_\tau]$, where $\tau$ is the number of bins, the cumulative histogram of $R$ is:*

$$\hat{H}_R = [\hat{h}_1, \hat{h}_2, \ldots, \hat{h}_\tau] \tag{3.3}$$

*where $\hat{h}_i = \sum_{j \leq i} h_j$.*

**Definition 3.3** *Given two cumulative histograms $\hat{H}_R$ and $\hat{H}_S$ of two time series $R$ and $S$, respectively, the cumulative histogram distance (CHD) is defined as:*

$$CHD(\hat{H}_R, \hat{H}_S) = \sqrt{\hat{Z}^T \hat{Z}} \tag{3.4}$$

*where $\hat{Z} = (\hat{H}_R - \hat{H}_S)$.*

**Algorithm 3.2** An algorithm for answering pattern existence queries

**Input:** An query time series $Q$ ($Q$ contains the specified patterns), its cumulative histogram $\hat{H}_Q$, a matching threshold $\epsilon$

**Output:** A list of time series that contain the specified patterns

```
1: for all time series histograms Ĥ_i of R_i in the database do
2:   if CHD(Ĥ_i, Ĥ_Q) ≤ ε then
3:     insert the time series ID ID_i into the result list
4:   end if
5: end for
6: return the result list
```

The algorithm for answering pattern existence queries using CHD is given in Algorithm 3.2. For each $\hat{H}_i$ of $R_i$, if $CHD(\hat{H}_i, \hat{H}_Q) \leq \epsilon$, then $R_i$ is in the result set ($\epsilon$ is a matching threshold). Later, in the experiments, the effectiveness of WED and CHD are compared in answering pattern existence queries. In Algorithm 3.2, a matching threshold ($\epsilon$) has to be set to determine whether the examined time series contains a pattern similar to that of the query time series. In the experiments, a suitable threshold is found based on the histogram of the query time series. Algorithm 3.2 applies sequential scan of all the time series in the database, however, it does not mean that sequential scan is the only method to answer pattern existence queries. Spatial access methods, such as R-tree [38], Kd-tree [10], and X-tree [11] can be applied to improve the retrieval efficiency. Furthermore, to avoid the dimensionality curse problem [112], a method is proposed later to reduce the dimensionality of time series histograms to one, then a simple $B^+$-tree will work. The reason that the sequential scan is included is for a fair efficacy comparison on CHD with other methods, such as WED and regular expression matching, in answering pattern existence queries, which is reported in Section 3.4.

So far, time series histograms are defined with equal size bins. However, values of time series normally are not uniformly distributed, leaving a lot of bins empty. In such cases, computing the distances between two time series histograms that contain many zeros is not helpful in differentiating the corresponding time series.

It has been claimed [64] that distributions of most time series follow the normal distribution, which has also been verified on the data sets that are used in the experiments reported in this chapter. Consequently, instead of segmenting the value space into $\tau$ equal size sub-regions, the value space is segmented into sub-regions (called *sub-spaces*) such that each sub-region has the same area under the normal distribution curve of that data. The boundary of each subspace can be computed as follows, assuming that $h_{i,l}$ is the lower bound of subspace $i$ and $h_{i,u}$ is the upper bound:

$$\int_{min_{\mathcal{D}}}^{max_{\mathcal{D}}} p(x)dx = \sum_{i=1}^{\tau} \int_{h_{i,l}}^{h_{i,u}} p(x)dx \tag{3.5}$$

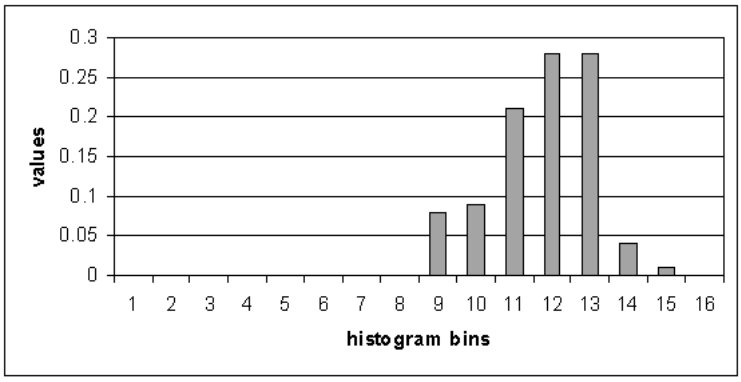
where $p(x)$ is the normal distribution function, $1 \leq i \leq \tau$, $h_{1,l} = min_{\mathcal{D}}$, and $h_{\tau,u} = max_{\mathcal{D}}$. Even though the normal distribution function is used to create equal area bin histogram, the idea can be easily extended to other data distributions.

With equal area bin segmentation, each histogram bin is assigned the same probability that data points of time series may fall in. For example, for the "cameramouse" data used in the experiments, the average filling ratio of 16 equal area bin histograms is about 98%. However, it is only 40% for the 16 equal size bin histograms. Figure 3.2 shows the values of equal area bin and equal size bin histograms of a randomly selected data from "cameramouse" data set. It can be observed that the values in the equal area bin histogram are well distributed and values in the equal size bin histogram concentrate in a few bins.

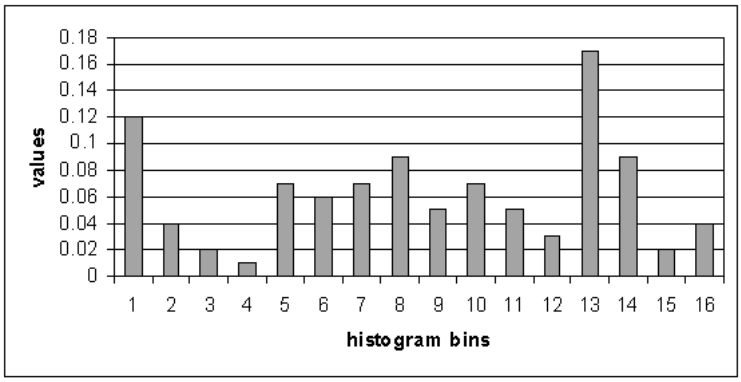
The relative effectiveness of equal size bin histograms and equal area bin histograms are compared experimentally in terms of classification accuracy.

## 3.3 Multi-scale Histograms

The time series histograms, as defined in the previous section, give a global view of the data distribution of time series. However, they do not consider the order of values in




(a) Values of a 16 equal size bin histogram




(b) Values of a 16 qual area bin histogram

Figure 3.2: A comparison of data distributions of an equal size bin histogram and an equal area bin histogram

the sequence. For example, in Figure 3.3, two time series, $R$ and $S$, are quite different in terms of their trend. However, they have the same histograms as shown in Figures 3.4(a) and 3.4(c). A multi-scale representation of a time series histogram is designed for better discrimination of time series based on their order details to facilitate shape match queries. Given a time series $R$ of length $N$, it can be equally divided into two segments, and each segment can be recursively divided into two, and so on. For each segment, its time series histogram can be computed and all these histograms form the multi-scale time series histograms of $R$. The number of levels (scales) is controlled by a single parameter, $\delta$,

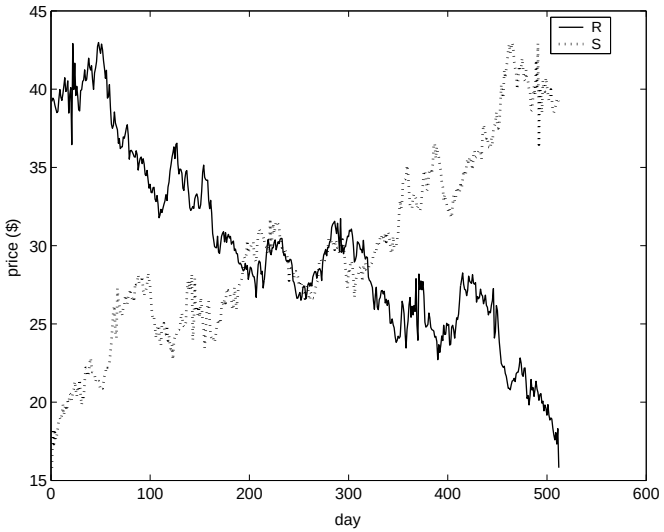
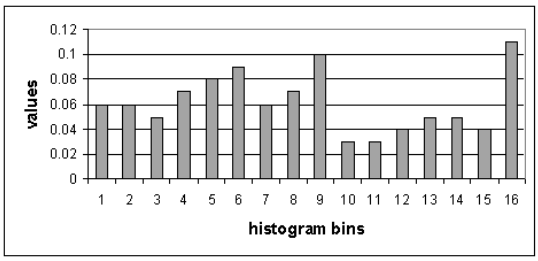



Figure 3.3: Two different time series with the same time series histogram

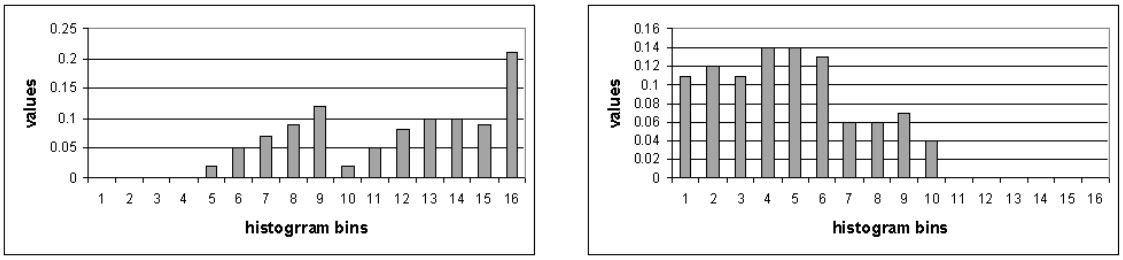
that is the *precision level* (i.e. *scale*) of the histogram. For $\delta = 1$, the histogram covers the entire time series, as defined in the previous section. If further precision is required, one can set $\delta > 1$, which would segment the time series data into $2^{(\delta-1)}$ equal length subsequences and histograms are computed for each segment. Figure 3.4 shows 2-scale histograms for time series $R$ and $S$ of Figure 3.3. In Figure 3.4, $R$ and $S$ can be easily distinguished at $\delta = 2$, although they cannot be differentiated at $\delta = 1$.

With multi-scale time series histograms, the shape match queries can be answered at several precision levels. It has to be guaranteed that similar time series at a higher scale will also be identified as similar at a lower scale. In order to guarantee this property, the cumulative histogram distance must be formulated in such a way that the distance at the lower scale is the lower bound of the distance at the higher scale. In other words, the cumulative histograms must follow Lower Bounding Lemma introduced in Chapter 2. The average of the cumulative histogram distances is used as the result of comparison at scale $\delta$, which, as proven below, satisfies this property.

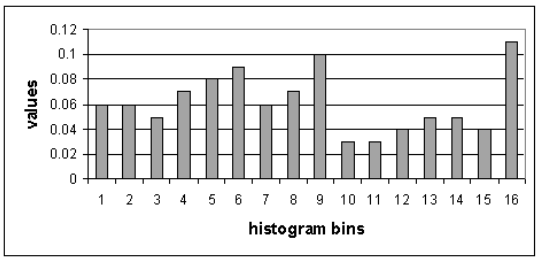
**Definition 3.4** *Given two time series $R$ and $S$, let their $\delta \geq 1$ scale histograms be $H^i_{R,\delta}$*



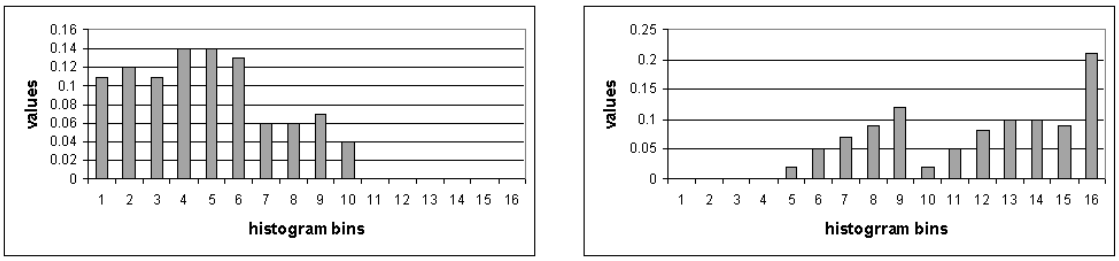

(a) a histogram of $R$ at scale 1




(b) two histograms of $R$ at scale 2




(c) a histogram of $S$ at scale 1




(d) two histograms of $S$ at scale 2

Figure 3.4: Two scale histograms of $R$ and $S$

*and* $H^i_{S,\delta}$*, respectively. where* $1 \leq i \leq 2^{\delta-1}$*. The CHD at scale* $\delta$ *is:*

$$CHD_\delta = \frac{\sum_{i=1}^{2^{\delta-1}} CHD(\hat{H}^i_{R,\delta}, \hat{H}^i_{S,\delta})}{2^{\delta-1}} \tag{3.6}$$

*where* $\hat{H}^i_{R,\delta}$ *(*$\hat{H}^i_{S,\delta}$*) denotes the cumulative histograms of* $i^{th}$ *segment of* $R$ *(*$S$*) at scale* $\delta$*.*

**Theorem 3.1** *In a* $\delta$*-level multi-scale time series histogram, if* $CHD_l$ *denotes the cumulative histogram distance between two time series histograms at scale* $l$*, then*

$$CHD_{l-1} \leq CHD_l \tag{3.7}$$

*where* $2 \leq l \leq \delta$*.*

This is an extension of a result on weighted Euclidean distance of color histograms [65].

**Proof.** By induction.

Base case: The claim holds for $l = 1$, $CHD_1 < CHD_2$.

Given two time series $R$ and $S$, their cumulative histograms at scale $i$ ($i \geq 1$) are denoted as: $\hat{H}_{R,i}$ and $\hat{H}_{S,i}$, respectively. Then:

$$CHD_1 = \sqrt{(\hat{H}_{R,1} - \hat{H}_{S,1})^T (\hat{H}_{R,1} - \hat{H}_{S,1})} \tag{3.8}$$

$$CHD_2 = \frac{1}{2} \sum_{i=1}^{2} \sqrt{(\hat{H}^i_{R,2} - \hat{H}^i_{S,2})^T (\hat{H}^i_{R,2} - \hat{H}^i_{S,2})} \tag{3.9}$$

Define $\|X\| = \sqrt{X^T X}$ and $\|X - Y\| = \sqrt{(X-Y)^T (X-Y)}$, where $X$ and $Y$ are $\tau$ dimensional vectors. Then,

$$CHD_1 = \|\hat{H}_{R,1} - \hat{H}_{S,1}\| \tag{3.10}$$

and

$$CHD_2 = \frac{1}{2} \sum_{i=1}^{2} \|\hat{H}^i_{S,2} - \hat{H}^i_{R,2}\| \tag{3.11}$$

$$\|X + Y\| = \sqrt{\sum_{i=1}^{n} (x_i + y_i)^2} \tag{3.12}$$

$$= \sqrt{\|X\|^2 + \|Y\|^2 + 2 \sum_{i=1}^{n} (x_i y_i)} \tag{3.13}$$

Using Cauchy's inequality, which is

$$(\sum_{i=1}^{n}(x_i y_i))^2 \leq \sum_{i=1}^{n}(x_i)^2 \sum_{i=1}^{n}(y_i)^2 \tag{3.14}$$

$$= \|X\|^2\|Y\|^2 \tag{3.15}$$

we get

$$\|X+Y\| \leq \|X\| + \|Y\| \tag{3.16}$$

It is known that

$$\hat{H}_{R,1} = \frac{1}{2}(\hat{H}^1_{R,2} + \hat{H}^2_{R,2}) \tag{3.17}$$

and

$$\hat{H}_{S,1} = \frac{1}{2}(\hat{H}^1_{S,2} + \hat{H}^2_{S,2}) \tag{3.18}$$

Thus:

$$\|\hat{H}_{R,1} - \hat{H}_{S,1}\| = \|\frac{1}{2}\sum_{i=1}^{2}(\hat{H}^i_{R,2}) - \frac{1}{2}\sum_{i=1}^{2}(\hat{H}^i_{S,2})\| \tag{3.19}$$

$$\leq \frac{1}{2}\sum_{i=1}^{2}\|\hat{H}^i_{R,2} - \hat{H}^i_{S,2}\|(\text{from 3.16}) \tag{3.20}$$

Therefore, $CHD_1 \leq CHD_2$.

Assume that the claim holds for $l \leq k$ $(k > 1)$. Histograms at scale $k$ and $k+1$ have the similar relationship as the ones shown in 3.17 and 3.18.

$$\hat{H}^i_{R,k} = \frac{1}{2}(\hat{H}^{2i-1}_{R,k+1} + \hat{H}^{2i}_{R,k+1}) \tag{3.21}$$

and

$$\hat{H}^i_{S,k} = \frac{1}{2}(\hat{H}^{2i-1}_{S,k+1} + \hat{H}^{2i}_{S,k+1}) \tag{3.22}$$

where $i \geq 1$.
Thus, for each element of $CHD_k$, $\|\hat{H}^i_{R,k} - \hat{H}^i_{S,k}\|$,

$$\|\hat{H}^i_{S,k} - \hat{H}^i_{R,k}\| \leq \frac{1}{2}(\|\hat{H}^{2i-1}_{S,k+1} - \hat{H}^{2i-1}_{R,k+1}\| + \|\hat{H}^{2i}_{S,k+1} - \hat{H}^{2i}_{R,k+1}\|) \tag{3.23}$$

Therefore, $CHD_k < CHD_{k+1}$. □

As stated earlier, time series histograms at scales have better discrimination power; however, the computation of CHD at higher scales is more expensive than those at lower scales. Fortunately, with the lower bound property of CHD at each scale (Theorem 3.1), when a shape match query needs answers at scale $l$, instead of directly computing the CHD at scale $l$, the CHD for each candidate time series can be computed starting from scale 1, and then scale 2 and so on [62]. This multi-step filtering strategy will not introduce false dismissals.

For both pattern existence queries and shape match queries, directly comparing $\tau$-dimensional time series histograms is computationally expensive, even with the help of spatial access methods such as the R-tree [38], Kd-tree [10], and X-tree [11]. Since $\tau$ is higher than 12-16 dimensions, the performance of using an indexing structure will be worse than that of sequential scan [112]. Therefore, the averages of time series cumulative histograms is used to avoid comparisons on full cumulative histograms.

**Definition 3.5** *Given a time series $R$ and its cumulative histogram at scale level 1, $\hat{H}_{R,1} = [\hat{h}_{R,1}, \hat{h}_{R,2}, \ldots, \hat{h}_{R,\tau}]$, where $\tau$ is the number of histogram bins, the average of cumulative histogram is:*

$$\hat{H}^{avg}_{R,1} = \frac{\sum_{i=1}^{\tau} \hat{h}_{R,i}}{\tau} \tag{3.24}$$

**Definition 3.6** *Given two time series $R$ and $S$, let $\hat{H}_{R,1}$ and $\hat{H}_{S,1}$ be their cumulative histograms at scale level 1. The distance between averages of cumulative histograms is:*

$$ACHD = \sqrt{\tau(\hat{H}^{avg}_{R,1} - \hat{H}^{avg}_{S,1})^2} \tag{3.25}$$

The averages of time series cumulative histograms are one dimensional data, which means that a simple $B^+$-tree can be used to improve the retrieval efficiency. Moreover, based on the definition of ACHD, the time series retrieved by comparing ACHD are guaranteed to include all the time series that should be retrieved by comparing cumulative histograms of time series at scale 1. This is stated in the following theorem.

**Theorem 3.2** *For any two $\tau$-dimensional time series cumulative histograms $\hat{H}_{R,1}$ and $\hat{H}_{S,1}$ at scale 1, $ACHD \leq CHD_1$.*

**Proof.** Given two histograms of scale 1 $\hat{H}_{R,1}$ and $\hat{H}_{S,1}$,

$$CHD_1^2 = \sum_{i=1}^{\tau} (\hat{h}_{R,i} - \hat{h}_{S,i})^2 \tag{3.26}$$

and

$$ACHD^2 = \frac{(\sum_{i=1}^{\tau} \hat{h}_{R,i} - \sum_{i=1}^{\tau} \hat{h}_{S,i})^2}{\tau} \tag{3.27}$$

Define $X = \hat{H}_{R,1} - \hat{H}_{S,1}$, where $x_i = \hat{h}_{R,i} - \hat{h}_{S,i}$ and $1 \leq i \leq \tau$. Therefore, the only thing that needs to be proven is:

$$\sum_{i=1}^{\tau} x_i^2 \geq \frac{(\sum_{i=1}^{\tau} x_i)^2}{\tau} \tag{3.28}$$

According to Arithmetic-Geometric Mean inequality:

$$\frac{(\sum_{i=1}^{\tau} x_i)^2}{\tau} \leq \frac{(\sum_{i=1}^{\tau} (x_i)^2 + \sum_{i=1}^{\tau-1} \sum_{i=1}^{\tau} (x_i)^2)}{\tau} \tag{3.29}$$

$$= \sum_{i=1}^{\tau} (x_i)^2 \tag{3.30}$$

□

Therefore, instead of directly computing the cumulative histogram distances, the distance between the averages of two time series cumulative histograms can be computed first to prune false candidates from the database. Averages can be considered as the first filter when multi-step filtering is used to answer a shape match query at a higher scale $l$. The algorithm for multi-step filtering is given in Algorithm 3.3, which shows one possible scan strategy to answer shape match queries that uses histograms of all scales up to $l$ to perform filtering. Another possible scan strategy is only using time series histograms at scale 1 and scale $l$. These two different scan strategies are compared in the next section.

## 3.4 Experimental Evaluation

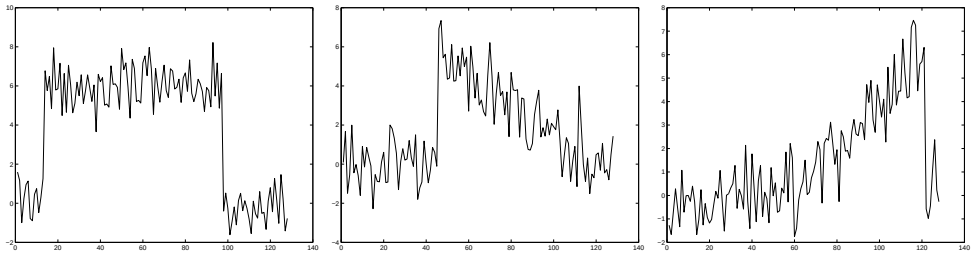
In this section, the results of experiments are reported. These experiments are designed to evaluate the efficacy and robustness of the histogram-based similarity measure and the matching algorithm. All programs are written in C and experiments are run on a Sun-Blade-1000 workstation under Solaris 2.8 with 1GB of memory.

**Algorithm 3.3** An algorithm for answering pattern queries with multi-scale filtering

**Input:** An example time series $Q$, precision level $\delta$, $\delta$ levels and average cumulative histograms of $Q$, a matching threshold $\epsilon$

**Output:** A list of time series which match $Q$

initialize result list $result_0$, ..., $result_\delta$
**for all** averages of time series cumulative histograms $\hat{H}_{R_i}^{avg}$ of $R_i$ in the database **do**
  compute $ACHD$ between $\hat{H}_Q^{avg}$ and $\hat{H}_{R_i}^{avg}$
  **if** $ACHD \leq \epsilon$ **then**
    insert the time series ID $ID_i$ into the result list $result_0$
  **end if**
**end for**
$j = 1$ {/* starting from scale 1 */}
**repeat**
  **for** each $ID_i$ in $result_{j-1}$ **do**
    **if** $CHD_j(\hat{H}_{R_i}, \hat{H}_Q) \leq \epsilon$ **then**
      insert the time series ID $ID_i$ into the result list $result_j$
    **end if**
  **end for**
  $j = j + 1$ {/*increase scale level by 1*/}
**until** $(j == \delta + 1)$ or ($result_{j-1}$ is empty)
**if** $result_{j-1}$ is empty **then**
  return NULL
**else**
  return $result_\delta$
**end if**



(a) Cylinder (b) Bell (c) Funnel

Figure 3.5: Cylinder-Bell-Funnel data set

### 3.4.1 Efficacy and Robustness of Similarity Measures

**Experiment 1.** This experiment is designed to test how well the cumulative histogram distances perform in finding patterns in time series. First the proposed approach is compared with Shatkay's algorithm [95] on a labelled time series sets: Cylinder-Bell-Funnel (CBF). In Shatkay's algorithm, first, a best fitting line algorithm is used to detect the movement slope of segments of time series. Then, the slope of each line segment is mapped to a symbol according to the predefined mapping tables (such as Table 1.1) and consecutive symbols are connected together to form a string. Finally, a string matching algorithm is used to find the matches between query regular expression and the converted strings[1]. The reason for using CBF data set is that it contains very simple distinct patterns, allowing a direct comparison of the techniques. The CBF data set has three distinct classes of time series: cylinder, bell and funnel (Figure 3.5). 1000 CBF data sets are generated with 100 examples for each class. Since the general shape of bell and funnel are treated as similar (because both of them contain a *peak*), only cylinder and bell data sets are used to test for existence queries.

In order to test robustness of the similarity measures, random Gaussian noise and local time shifting are added to both data sets using a modified version of the program in [109]. The modification includes the addition of non-interpolated noise and large time shifting

[1]Another related work [84] requires users to specify, in addition to the pattern, the "unit length" (the length of time series) in which the specified pattern may appear. It is quite difficult for users to know the "unit length" beforehand if they only want to check for the existence of a pattern. Therefore, this approach is not considered in this study.

$(20-30\%$ of the series length) since pattern existence queries only involve the existence of a given pattern.

For the query data set, another "pure" (without noise) CBF data set of size 150 is generated, where each class contains 50 examples. The time series histograms are also computed for the query data set. Precision and recall are used to measure the effectiveness of the retrieval results. Precision measures the proportion of correctly found time series, while recall measures the proportion of correct time series that are detected. In this experiment, Algorithm 3.1 is used to search the patterns in the time series. In order to set up a matching threshold for Algorithm 3.1, several values are tested. It is found that half the cumulative histogram distance between the query histogram and an empty histogram gives the best results. Besides using cumulative histogram distance, weighted Euclidean distance is also applied to compare their relative effectiveness.

The average of 50 executions of the query is depicted in Table 3.1 where histogram experiments are parameterized by the number of bins. In these experiments, results are reported as WED/CHD values. Even though Shatkay's approach achieves relatively high precision, its recall is very low; this is the effect of noise. The histogram-based approach (both WED and CHD) can achieve relatively high precision and recall, which confirms the proposed approach is suitable for answering pattern existence queries. Table 3.1 suggests that dividing the value space into too many sub-regions (histogram bins) does not improve the results significantly; reasonably good results are obtained around 16 bins. As expected, the results also show that CHD performs better than WED, this is because WED overestimates neighborhood similarity.

| | Cylinder | | Bell | |
|---|---|---|---|---|
| | precision(%) | recall(%) | precision(%) | recall(%) |
| Shatkay | 81 | 44 | 75 | 31 |
| his(8) | 72/80 | 88/91 | 63/ 70 | 71/ 74 |
| his(16) | 72/81 | 90/92 | 78/ 84 | 80/ 85 |
| his(32) | 75/81 | 88/90 | 79/ 85 | 85/ 87 |
| his(64) | 72/80 | 88/90 | 79/ 83 | 85/ 88 |

Table 3.1: Comparison on pattern existences queries

**Experiment 2.** This experiment is designed to check the effectiveness and robustness of multi-scale histograms in answering shape match queries. According to a recent survey [55], the efficacy of a similarity measure for shape match queries can be evaluated by classification and clustering results on labelled data sets. The classification error rate is defined as a ratio of the number of misclassified time series to the total number of the time series. The classification error rates in results produced by CHD are compared to those of DTW and LCSS by evaluating them on the Control-Chart (CC) data set. DTW and LCSS are selected for comparison, because DTW can also handle local time shifting and LCSS can handle local time shifting and noise disturbances. There are six different classes of control charts: "normal", "cyclic", "increasing trend", "decreasing trend", "upward shift", and "downward shift". Each class contains 100 examples. Non-interpolated noise and time shifting ($10-20\%$ of the series length) are added to test the robustness of cumulative histogram distance in answering shape match queries. For simplicity, a simple classification using 1-Nearest Neighbor with three similarity measures is used. The classification results are checked using the "leave one out" verification mechanism[2].

The average error rates using DTW and LCSS are 0.12 and 0.16, respectively. It has been claimed that LCSS is more accurate than DTW [108]. However, the experiments could not replicate this result. The possible reason for this discrepancy is the difference in the choice of the matching threshold value. Comparable results are reported using the cumulative histogram distances in Table 3.2. The experiment was run with different number of bins ($\tau$) and scales ($\delta$) using CHD on time series histograms with equal size bin.

The comparison of the error rates of DTW (0.12), LCSS (0.16) and those of Table 3.2 reveals that CHD performs better in answering shape match queries than DTW or LCSS. It is interesting that very high scales (i.e., high values of $\delta$) may lead to worse classification results. This is because the higher the scale, the more temporal details will be involved in computing CHD, and this causes time series histograms at that scale to be more sensitive to local time shifting and scaling. Another fact demonstrated in Table 3.2 is that higher number of bins may not lead to more accurate classification results. This is because, as

[2]The "leave one out" verification mechanism takes one sample data from a class and finds a nearest neighbor of the data in the entire data set by applying a predefined distance metric. If the found nearest neighbor belongs to the same class as the sample data, it is a hit, otherwise, it is a miss.

the number of bins increases, the more detailed information will be captured in time series histogram, including noise. These characteristics of multi-scale time series histograms suggest that it is sufficient to check only the first few scale histograms with small number of histogram bins. However, the improvements over DTW or LCSS as reported in Table 3.2 are modest, especially for the first few scale histograms.

| | number of bins $\tau$ | | | |
|---|---|---|---|---|
| scale $\delta$ | 8 | 16 | 32 | 64 |
| 1 | 0.62 | 0.59 | 0.56 | 0.53 |
| 2 | 0.22 | 0.16 | 0.12 | 0.13 |
| 3 | **0.14** | **0.10** | **0.11** | **0.11** |
| 4 | 0.20 | 0.13 | 0.14 | 0.15 |
| 5 | 0.24 | 0.19 | 0.20 | 0.20 |

Table 3.2: Error rates using CHD on time series histograms with equal size bin

To better understand these results, the filling ratio of equal size bin histograms were checked. The filling ratio is defined as the number of non-empty bins to the total number of bins. It was shown that nearly 50% of the bins are empty. Consequently, CHD between two time series histograms is not able to distinguish them properly, since most of the bins are identical. Therefore, the same experiment was repeated using time series histograms with equal area bins. Table 3.3 reports the results.

| | number of bins $\tau$ | | | |
|---|---|---|---|---|
| scale $\delta$ | 8 | 16 | 32 | 64 |
| 1 | 0.15 | 0.11 | 0.12 | 0.13 |
| 2 | 0.10 | 0.08 | 0.07 | 0.06 |
| 3 | **0.07** | **0.05** | **0.06** | **0.06** |
| 4 | 0.10 | 0.08 | 0.08 | 0.09 |
| 5 | 0.18 | 0.15 | 0.17 | 0.17 |

Table 3.3: Error rates using CHD on time series histograms with equal area bin

The results of Tables 3.2 and 3.3 demonstrate that the error rates of histograms with equal area bins are nearly half of histograms with equal size bins. The filling ratio of time series histogram with equal area bins is around 80%. Table 3.3 also shows that using only the first few scales (e.g. $\delta = 3$), provides reasonably high accuracy.

With the derived optimal parameter ($\delta = 3$) for computing CHD in Table 3.3, the "complete linkage" hierarchy clustering algorithm [47] is run using CHD on equal area time series histograms with different number of bins. This algorithm produces the best clustering results [108]. For each class in CC data set, 20 pieces of time series are picked as the representative data of that class. The set of all possible pairs of classes are partitioned into two clusters. The dendrogram of each clustered result is drawn to verify whether it correctly partitions the classes. The same clustering program is also run using DTW and LCSS. The clustering results using DTW, LCSS, and CHDs with 4 different sizes of bins are reported in Table 3.4. DTW and LCSS correctly clustered 10 and 9 clusters (out of a total 15 correct ones), respectively. CHD, on the other hand, matched these results (for $\tau = 8$) and performed better with 13 corrections for $\tau = 16$. This is because CHD compares the data distributions of two time series, which can reduce the effect of noise.

| Correct clustering results | DTW | LCSS | CHD (8) | CHD (16) | CHD (32) | CHD (64) |
|---|---|---|---|---|---|---|
| CC (total 15 correct) | 10 | 9 | 10 | 13 | 12 | 12 |

Table 3.4: Clustering results of three similarity measures

**Experiment 3.** In this experiment, the matching accuracy of multi-scale time histograms are tested versus DTW and LCSS on classifying time series with more complicated shapes. Classifying these types of time series requires matching on temporal details of the data. CHD, DTW and LCSS are evaluated by two labelled trajectory data sets that are generated from the Australian Sign Language (ASL)[3] data and the "cameramouse" [34]

[3]http://kdd.ics.uci.edu

data. Only $x$ positions are used for both data sets. A "seed" time series is first selected from each of the two data sets and then two additional data sets are created by adding interpolated Gaussian noise and small time shifting (5-10% of the time series length) to these seeds [109]. The ASL data set from UCI data consists of samples of Australian Sign Language signs. Various parameters were recorded as a signer was signing one of 95 words in ASL. One recording from each of following 10 words is extracted as "seed" time series of ASL data: "Norway", "cold", "crazy", "eat", "forget", "happy", "innocent", "later", "lose" and 'spend". The "cameramouse" data set contains 15 trajectories of 5 words (3 for each word) obtained by tracking the finger tip movement as people write various words. All the trajectories of each word are used as the seeds. The data set that is generated from seeds of "cameramouse" contains 5 classes and 30 examples of each class, while the one from ASL seeds contains 10 classes and each class has 10 examples. The classification and verification algorithms used in the second experiment are applied in this experiment. Based on the observation of the second experiment, time series histograms with equal area bins are used. Tables 3.5 and 3.6 report the results. The results of Tables 3.5 and 3.6

| | ASL data | Cameramouse data |
|---|---|---|
| DTW | 0.11 | 0.08 |
| LCSS | 0.29 | 0.25 |

Table 3.5: Error rates of LCSS and DTW

show that CHD again achieves better classification result. For both data sets, CHD can get 0 error rate. A surprising observation is that even at very low scales (e.g. $\delta = 2$), using CHD can already achieve better results than DTW and LCSS. Through the investigation of the filling ratios of both histograms, it was shown that the number of empty bins only accounts for less than 5% of the total number of bins. The standard deviations of both data sets are also very high with respect to their mean values, which indicates that data points are widely spread in their value space. Together with the results of the second experiment, these results indicate that CHD on time series whose data points are widely distributed in their value space (higher histogram filling ratio) can achieve better classification accuracy even at lower scales.

| | ASL DATA | | | | Cameramouse DATA | | | |
|---|---|---|---|---|---|---|---|---|
| | number of bins $\tau$ | | | | number of bins $\tau$ | | | |
| scale | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 |
| 1 | 0.12 | 0.11 | 0.12 | 0.12 | 0.13 | 0.13 | 0.07 | 0.07 |
| 2 | **0.06** | **0.05** | **0.04** | **0.04** | **0.04** | **0.03** | **0.02** | **0.02** |
| 3 | 0.06 | 0.06 | 0.04 | 0.04 | 0.03 | 0.02 | 0.02 | 0.02 |
| 4 | 0.03 | 0.04 | 0.04 | 0 | 0.04 | 0.01 | 0 | 0.01 |
| 5 | 0.06 | 0.06 | 0.04 | 0.02 | 0.04 | 0.04 | 0.02 | 0.04 |

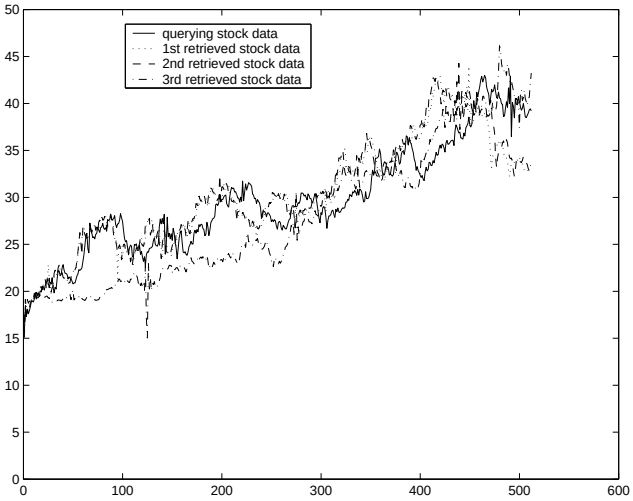
Table 3.6: Error rates using CHD on time series histograms with equal area bin

The hierarchy clustering test was also performed using different similarity measures for each data set. All possible pairs of words are selected and the same clustering program as in experiment 2 is used to partition them into two clusters. When CHD is used to cluster data, the scale is set to 4, and four different number of bins (8, 16, 32, 64) are used. The best results of each measure are reported in Table 3.7, which reveals that CHD achieves better clustering compared to DTW and LCSS. Regardless of the similarity measure, the number of correctly found clusters in ASL data set is less than half of the total clusterings (45). This is because the data length of ASL data is relatively small (average 65) and some trajectories from different words are quite similar to each other.

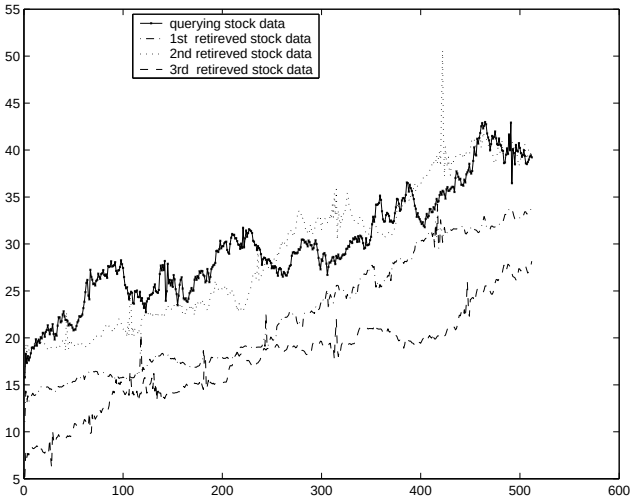
| Correct clustering results | DTW | LCSS | CHD (8) | CHD (16) | CHD (32) | CHD (64) |
|---|---|---|---|---|---|---|
| Cameramouse (total 10 correct) | 8 | 8 | 10 | 10 | 10 | 10 |
| ASL (total 45 correct) | 6 | 9 | 19 | 20 | 22 | 22 |

Table 3.7: Clustering results of three similarity measures

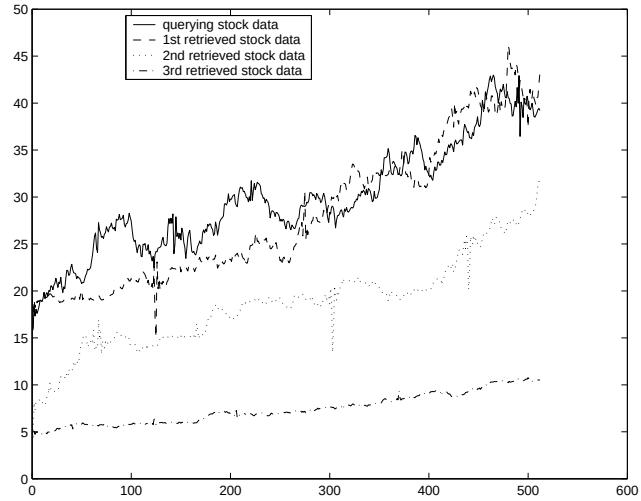
**Experiment 4.** This experiment tests the matching quality of CHD on multi-scale time histograms in answering $k$-nearest neighbor ($k$-NN) shape match search and compared




(a) 3-NN results on stock data using CHD




(b) 3-NN results on stock data using DTW




(c) 3-NN results on stock data using LCSS

Figure 3.6: A comparison of 3-NN queries using DTW, LCSS and CHD

it to DTW and LCSS. Given a time series $S$, $k$-NN shape match search will return the top $k$ time series in the database that are most similar to $S$. A real stock data set is used, and it contains 193 company stocks' daily closing price from late 1993 to early 1996, each consisting of 513 values [110]. Each time series is selected as a "seed" and a new data

set is created by adding interpolated Gaussian noise and small time shifting (2-5% of the time series length) to each seed. The new data set contains 1930 time series (each seed is used to create 10 new time series). Query data consist of randomly selected 10% of the sequences. The results are that CHD always performs better DTW and LCSS. Figure 3.6 shows one of the results of 3-NN search using CHD ($\delta = 4$, $\tau = 32$, which are set based on the previous experimental results), DTW, and LCSS on stock data.

### 3.4.2 Efficiency of Multi-Step Filtering

**Experiment 5.** Theorem 3.1 in Section 3.3 established that multi-step filtering using multi-scale time series histograms will not introduce false candidates. However, the question remains as to how many histogram comparisons are saved using multi-step filtering? This experiment tests that on the real stock data set that is used in the fourth experiment. A time series is randomly picked up and used to conduct a range query at precision level 4. The number of comparisons that are needed for three types of searching are computed: using only level 4, using level 1 and then jumping to 4, and using all 4 levels on different range thresholds. The experiments are run 100 times and the average results are reported in Table 3.8.

Step-by-step filtering is clearly the best strategy to reduce the total number of comparisons for small thresholds. For large thresholds, directly computing the distance at a higher level is a better choice in terms of the number of comparisons. However, large thresholds are not very useful for finding desirable results, since a larger portion of the data in the database will be returned.

| | level 4 only | level 1 and level 4 | all 4 levels |
|---|---|---|---|
| $r = 0.5$ | 11580 | 13498 | 17190 |
| $r = 0.2$ | 11580 | 12760 | 7590 |
| $r = 0.1$ | 11580 | 6838 | 3668 |
| $r = 0.05$ | 11580 | 2356 | 2072 |

Table 3.8: Number of comparisons of different filtering approaches

**Experiment 6.** The number of comparisons needed for computing the histogram distance only relates to the CPU computation cost. However, the I/O cost for sequentially reading in the data files becomes a bottleneck as the database size grows. If false candidates can be filtered out without reading in the data files, a significant speed-up will be achieved. For time series histograms, their averages are stored separately from the histograms. The distances between the average cumulative histogram of query data and those of stored data are first computed to remove possible false candidates. Therefore, the pruning power of average cumulative histograms is quite important. The pruning power is defined as the proportion of the time series in a database for which the true distance (e.g. CHD) does not need to be computed without introducing false dismissals.

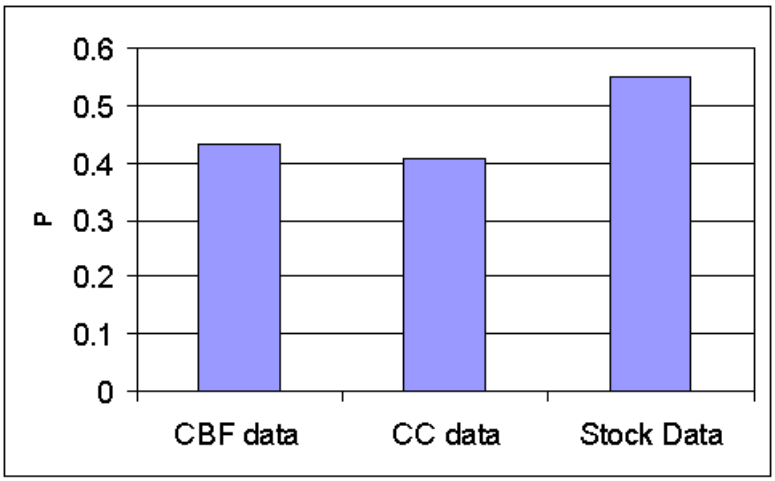



Figure 3.7: The punning power of averages of histograms

Figure 3.7 shows the pruning power of CHD with 16 bins on answering 1-NN queries (based on the previous experimental results, CHD can already achieve reasonable good results when the histogram bin size is 16). Because "cameramouse" and ASL data sets are small, they are not included in this experiment. Figure 3.7 demonstrates that using the distance of average time series histograms, it is possible to remove nearly 40% of the false candidates, which is helpful when the database size becomes large.

## 3.5 Comparison to Wavelets

In this section, time series histogram is compared with another multi-scale representation, wavelets. Wavelets have been widely used as a multi-resolution representation for image retrieval [74]; they have also been used as a dimensionality reduction technique in shape match queries in which Euclidean distance is used as the distance function [59]. Different from Fourier transform, wavelets can transform time series data into a multi-scale representation, where lower frequency bands are represented in lower scales and higher frequency bands are represented in higher scales. In this experiment, Euclidean distance is used to measure the similarity between two Haar wavelet coefficients (Haar wavelets are used because they are the most popular wavelet transformation). The same CBF data set used in the first experiment is used to measure the efficacy using different number of wavelet coefficients to answer pattern existence queries. Table 3.9 reports the results.

| | Cylinder | | Bell | |
|---|---|---|---|---|
| | precision | recall | precision | recall |
| 8 | 49 | 97 | 48 | 96 |
| 16 | 45 | 61 | 46 | 67 |
| 32 | 40 | 44 | 37 | 40 |
| 64 | 31 | 35 | 29 | 33 |

Table 3.9: Using wavelet for pattern existences queries

Compared to results that obtained by time series histograms (81/90 and 85/87), wavelets perform significantly worse in terms of accuracy. In fact, the wavelet transform transfers time series data into time-frequency domain, therefore, it is difficult, using wavelets, to answer pattern existence queries, since frequency appearance order is encoded in the wavelet coefficients. Even using only the first few coefficients (8), which have very lower time resolution (scale), high recall but low precision are obtained as shown in Table 3.9. The high recall is achieved by returning nearly all the data in the database (return all 200 time series of the CBF data set) as results and the lower precision is due to most of the important frequency bands are not used to answer queries. However, if more (32) coefficients are used, both precision and recall drop, because the high frequency bands have higher time

resolution, making them sensitive to time shifting.

For shape match queries, the efficacy of wavelets and multi-scale histograms are compared on CC data set. The ASL and Cameramouse data sets are not used because Euclidean distance requires the two compared wavelet coefficients to have the same length, thus the original time series must have the same length as well. Furthermore, since the wavelet transform requires the length of the sequences be a power of 2, the sequences are padded with 0 to make the length 64. The same classification test is carried out using different number of wavelet coefficients and the results are shown in Table 3.10.

| | 8 | 16 | 32 | 64 |
|---|---|---|---|---|
| error rate | 0.43 | 0.39 | 0.32 | 0.21 |

Table 3.10: Error rates using wavelet coefficients

Again the results are worse than the results achieved by using multi-scale histograms (0.06). The lower scale wavelets cannot capture the information about higher frequency band, thus, the error rate with lower scale is higher than that of higher scale. The higher scale wavelets contain too much detail about the appearance time of frequency bands, which causes the wavelets to be sensitive to time shifting. In fact, with Euclidean distance, using all wavelet coefficients is the same as using raw representation of time series [59]. Thus, wavelet representation is not robust to time shifting, making it unsuitable to answer pattern existence queries and introducing difficulties in answering shape match queries when data contain time shifting.

## 3.6 Conclusions

In this chapter, a novel representation of time series using multi-scale time series histograms is proposed to answer both pattern existence and shape match queries. This representation is based on the intuition that the distribution of time series can be an important cue for comparing similarity between time series. Multi-scale time series histograms are invariant to time shifting and scaling, and amplitude shifting and scaling. A robust similarity measure, cumulative histogram distance, is used to measure the similarity between two time

series histograms. Two different types of histograms: equal size bin and equal area bin are compared in terms of their efficacy for time series histograms.

The experimental results indicate that multi-scale time series histograms outperform string representations in finding patterns and outperform DTW and LCSS in answering shape match queries when the time series contain noise or time shifting and scaling. The experiments also show that equal area bin histograms are more suitable for time series comparison, and that distances of averages of histograms (one dimensional data) can effectively prune the false candidates from the database before computing the distance between two full histograms. The comparison study between time series histograms and wavelets confirms that, compared to wavelets, multi-scale histograms are more suitable for answering both type of queries when data contain time shifting and noise.

The multi-scale time histogram-based similarity retrieval has the following advantages:

- Multi-scale time series histograms can be used to answer shape match and pattern existence queries.
- The cumulative histogram distance reduces the boundary effects introduced by value space quantization and overcomes the shortcomings of overestimating (such as $L_1$-norm and $L_2$-norm) or underestimating (such as weighted Euclidean distance) the distance.
- Multi-scale time histograms are invariant to time shifting and scaling, amplitude shifting and scaling. Moreover, they can remove the disturbance introduced by noise.
- Multi-scale time histograms offer users flexibility to query the time series on different accuracy level. Furthermore, lower scale histograms can be used to prune the false candidates from the database before querying at higher scale histograms.

# Chapter 4

# Similarity-based Search Using Edit Distance with Real Penalty

## 4.1 Introduction

In the previous chapter, multi-scale time series histograms are developed for applications that require answers to pattern existence queries and shape match queries. The approach is more suitable for interactive applications, where users can refine their queries step-by-step by specifying different scales. Furthermore, it is also a good choice for answering pattern existence queries. For pattern existence queries, only the existence of a pattern is of interest. Time series histograms show the whole picture of data distribution, which fits the requirements of pattern existence queries. However, if users need fast answers to shape match queries, it may be hard for them to set up a proper scale value ($\delta$), unless they have some knowledge of the data set. Moreover, usually in this case, step-by-step refinement may not satisfy the requirement of fast response. Thus, in this and next chapters, solutions are developed for answering shape match queries.

As discussed in Chapter 2, the first issue in similarity search is the definition of a distance function. Among the distance functions that have been considered, $L_p$-norms [3, 31] are easy to compute, but they cannot handle local time shifting, which, as argued earlier, is essential for time series and trajectory data similarity matching. DTW [118, 57, 52] can deal with local time shifting, but it is a non-metric distance function, as is LCSS [13, 108],

which can handle local time shifting as well.

The efficiency of similarity search using distance functions, such as DTW and LCSS, is a particular problem since these non-metric distance functions violate triangle inequality that renders most indexing structures inapplicable. To this end, studies on this topic propose various lower bounds on the actual distance to guarantee no false dismissals [118, 57, 52, 119]. However, those lower bounds can admit a high percentage of false positives.

In this chapter, with the consideration of above two issues, the following questions are explored:

- *Is there a way to combine $L_p$-norms and the other distance functions so that one can get the best of both worlds* – namely being able to support local time shifting and being a metric distance function?
- With such a metric distance function, the triangle inequality can be applied for pruning, but *is it possible to develop a lower bound for the distance function?* If so, is lower bounding more efficient than applying the triangle inequality? Or, is it possible to do both?

In this chapter, a new distance function, called *Edit distance with Real Penalty* (ERP) is introduced. ERP borrows from both $L_1$-norm and DTW, and can handle local time shifting as well as being a metric distance function. The results of efficacy tests on benchmark data show that this distance function is natural for time series data. Furthermore, a new lower bound for ERP is proposed, which can be efficiently indexed with a standard $B^+$-tree; and a k-nearest neighbor ($k$-NN) algorithm that applies both lowering bounding and triangle inequality is developed as well. For simplicity, all the definitions, theorems and techniques are defined on one dimensional time series data; they can be easily extended to handle trajectory data where the dimensionality is more than two.

The rest of the chapter is arranged as follows. In Section 4.2, ERP is introduced along with benchmark tests on the efficacy of the proposed distance function. Section 4.3 introduces adapted lower bounds which were originally proposed for DTW and a new lower bound for ERP. A $k$-NN algorithm is also developed that applies both lowering bounding and triangle inequality. In Section 4.4, benchmark results are reported that

show the relative efficiency of lower bounding, triangle inequality, and the combination of both lowering bounding and triangle inequality.

## 4.2 A New Distance Function to Handle Local Time Shifting

### 4.2.1 Edit Distance with Real Penalty

To cope with local time shifting, one can borrow ideas from the domain of strings. A string is a sequence of elements, each of which is a symbol in an alphabet. Two strings, possibly of different lengths, are aligned so that they become identical with the smallest number of added, deleted or changed symbols. Among these three operations, deletion can be treated as adding a symbol in the other string. Hereafter, an added symbol is referred to as a *gap* element. This distance is called the *string edit distance* (ED) [63] which is defined as follows.

**Definition 4.1** *Given two strings $R$ and $S$ of lengths $M$ and $N$, respectively, the Edit Distance (ED) between $R$ and $S$ is the number of insert, delete, or replace operations that are needed to change $R$ into $S$. $ED(R, S)$ is defined as follows:*

$$ED(R,S) = \begin{cases} M & \text{if } N = 0 \\ N & \text{if } M = 0 \\ ED(Rest(R), Rest(S)) & \text{if } first(R) = first(S) \\ min\{ED(Rest(R), Rest(S)) + 1, & \\ ED(Rest(R), S) + 1, & \text{otherwise} \\ ED(R, Rest(S)) + 1\} & \end{cases} \tag{4.1}$$

where the cost/distance of introducing a gap element is set to 1, which can be generalized to the following formula.

$$dist(r_i, s_i) = \begin{cases} 0 & \text{if } r_i = s_i \\ \mathbf{1} & \textbf{if } r_i \textbf{ or } s_i \textbf{ is a gap} \\ 1 & \text{otherwise} \end{cases} \tag{4.2}$$

In the above formula, the second case is separated from the third and highlighted to indicate that if a gap is introduced in the alignment, the cost is 1. String edit distance has been proven to be a metric distance function [111], therefore, it satisfies triangle inequality. However, LCSS and DTW, which are closely related to ED [60], on time series do not follow triangle inequality. While the proof was given in Chapter 2, a detailed analysis of the reasons why LCSS and DTW break triangle inequality are given below, which provides hints for defining a metric distance function to handle local time shifting.

When LCSS applies to time series, the complication is that the elements $r_i$ and $s_i$ are not symbols, but real values. To take the real values into account, it relaxes equality to be within a certain tolerance (matching threshold) $\epsilon$. By borrowing the gap concept of ED on strings, the LCSS definition in Formula 2.10 can be rewritten as follows.

$$LCSS(R, S) = \begin{cases} 0 & \text{if } M = 0 \text{ or } N = 0 \\ max\{LCSS(Rest(R), Rest(S)) + score_{lcss}(r_1, s_1), & \\ \qquad LCSS(Rest(R), S) + score_{lcss}(r_1, gap), & \\ \qquad LCSS(R, Rest(S)) + score_{lcss}(gap, s_1)\} & \text{otherwise} \end{cases} \tag{4.3}$$

where

$$score_{lcss}(r_i, s_i) = \begin{cases} 1 & \text{if } |r_i - s_i| \leq \epsilon \\ 0 & \text{if } r_i \text{ or } s_i \text{ is a gap} \\ \mathbf{0} & \textbf{otherwise} \end{cases} \tag{4.4}$$

LCSS no longer satisfies triangle inequality. The problem arises precisely from relaxing equality, i.e., $|r_i - s_i| \leq \epsilon$. More specifically, for three elements $q_i, r_i, s_i$, there exist the case that $|q_i - r_i| \leq \epsilon$ and $|r_i - s_i| \leq \epsilon$ , but $|q_i - s_i| > \epsilon$. The third case of Formula 4.4 is separated from the others and highlighted to show where the relaxation may cause the violation of triangle inequality.

By borrowing the gap concept, DTW that was defined in Formula 2.8 can also be rewritten as follows:

$$DTW(R,S) = \begin{cases} 0 & \text{if } M = N = 0 \\ \infty & \text{if } M = 0 \text{ or } N = 0 \\ dist_{dtw}(r_1, s_1) + min\{DTW(Rest(R), Rest(S)), & \text{otherwise} \\ DTW(Rest(R), S), DTW(R, Rest(S))\} & \end{cases} \quad (4.5)$$

where

$$dist_{dtw}(r_i, s_i) = \begin{cases} |r_i - s_i| & \text{if } r_i, s_i \text{ not gaps} \\ |\mathbf{r_i} - \mathbf{s_{i-1}}| & \textbf{if } s_i \textbf{ is a gap} \\ |\mathbf{s_i} - \mathbf{r_{i-1}}| & \textbf{if } r_i \textbf{ is a gap} \end{cases} \quad (4.6)$$

DTW differs from LCSS in two key ways. First, unlike LCSS, DTW does not use a $\epsilon$ threshold to relax equality, the actual $L_1$-norm is used. Second, unlike LCSS, there is no explicit gap concept in its original definition in Formula 2.8. The replicated elements during the process of aligning two sequences are treated as gaps of DTW as highlighted in Formula 4.6. Therefore, the cost of a gap is not set to 1 as in LCSS; it amounts to replicating the previous element, based on which the $L_1$-norm is computed. The last two cases in Formula 4.6 deal with the possibility of replicating either $s_{i-1}$ or $r_{i-1}$.

To summarize, the key reason why DTW does not satisfy triangle inequality is that, when a gap needs to be added, it replicates the previous element. Thus, as shown in the second and third cases of Formula 4.6, the difference between an element and a gap varies according to $r_{i-1}$ or $s_{i-1}$. Contrast this situation with LCSS, which sets every difference to a constant 0 (second case in Formula 4.4). On the other hand, the problem for LCSS lies in its use of a $\epsilon$ tolerance. DTW does not have this problem, because it uses the $L_1$-norm between two non-gap elements.

Thus, in the definition of new distance function, *Edit distance with Real Penalty* (ERP), a *real penalty* ($L_1$-norm) is applied between two non-gap elements, but a constant value for computing the distance for gaps as shown by the last two cases of Formula 4.8.

**Definition 4.2** *Given two time series $R$ and $S$ of lengths $M$ and $N$, respectively, the Edit*

*Distance with Real Penalty (ERP) between R and S, ERP(R, S), is defined as follows:*

$$ERP(R,S) = \begin{cases} \sum_1^M |r_i - g| & \text{if } N = 0 \\ \sum_1^N |s_i - g| & \text{if } M = 0 \\ min\{ERP(Rest(R), Rest(S)) + dist_{erp}(r_1, s_1), & \text{otherwise} \\ \quad ERP(Rest(R), S) + dist_{erp}(r_1, g), & \\ \quad ERP(R, Rest(S)) + dist_{erp}(s_1, g)\} & \end{cases} \tag{4.7}$$

where

$$dist_{erp}(r_i, s_i) \quad = \quad \begin{cases} |r_i - s_i| & \text{if } r_i, s_i \text{ not gaps} \\ |r_i - g| & \text{if } s_i \text{ is a gap} \\ |s_i - g| & \text{if } r_i \text{ is a gap} \end{cases} \tag{4.8}$$

and $g$ denotes the gap which is assigned a constant value.

A careful comparison of $L_1$-norm and Edit Distance reveals that ERP can be seen as a combination of $L_1$-norm and Edit Distance. ERP differs from LCSS in avoiding the $\epsilon$ tolerance. On the other hand, ERP differs from DTW in not replicating the previous elements. The following lemma shows that for any fixed constant $g$, triangle inequality is satisfied.

**Lemma 4.1** *For any three elements $q_i, r_i, s_i$, any of which can be a gap element, it is necessary that $dist(q_i, s_i) \leq dist(q_i, r_i) + dist(r_i, s_i)$ based on Formula 4.8.*

**Proof.** The proof of this lemma is rather straightforward. When none of $q_i, r_i, s_i$ is a gap element, the above inequality is satisfied because $L_1$-norm satisfies triangle inequality. If any of $q_i, r_i, s_i$ is a gap element, then that element is treated as having the value $g$. The above inequality is satisfied because $L_1$-norm satisfies triangle inequality. □

**Theorem 4.1** *Let $Q, R, S$ be three time series of arbitrary length. Then it is necessary that $ERP(Q, S) \leq ERP(Q, R) + ERP(R, S)$.*

The proof of this theorem is a consequence of Lemma 4.1 and the proof of the result by Waterman et al. [111] on string edit distance. The Waterman proof essentially shows that defining the distance between two strings based on their best alignment in a dynamic

programming style preserves triangle inequality, as long as the underlying distance function also satisfies triangle inequality. The latter requirement is guaranteed by Lemma 4.1.

Lemma 4.1 states that for any value of $g$, as long as it is fixed, $dist_{erp}(r_i, s_i)$ satisfies triangle inequality. In this study, $g = 0$ is picked for two reasons. First, $g = 0$ admits an intuitive geometric interpretation. Consider plotting the time series with the $x$-axis representing (equally-spaced) time points and the $y$-axis representing the values of the elements. In this case, the $x$-axis corresponds to $g = 0$. Thus, the distance between two time series $R, S$ corresponds to the difference between the area under $R$ and the area under $S$.

Second, to best match $R$, $S$ is aligned to form $\widetilde{S}$ with the addition of gap elements. However, since the gap elements have value $g = 0$, it is easy to see that $\sum \widetilde{s}_i = \sum s_j$, making the area under $S$ and that under $\widetilde{S}$ the same. The following lemma states this property. In the next section, the computational significance of this lemma is shown.

**Lemma 4.2** *Let $R, S$ be two time series. By setting $g = 0$ in Formula 4.8, $\sum \widetilde{s}_i = \sum s_j$, where $S$ is aligned to form $\widetilde{S}$ to match $R$.*

**Proof.** Since the inserted gap value $g = 0$, $\sum \widetilde{s}_i = \sum s_j + \sum g = \sum s_j$. □

Consider the example time series that were used in Chapter 2 to show LCSS and DTW do not follow triangle inequality are used here for testing ERP (repeated here for ease of reading): $Q = [0], R = [1, 2]$ and $S = [2, 3, 3]$. To best match $R$, $Q$ is aligned to be $\widetilde{Q} = [0, 0]$. Thus, $ERP(Q, R) = 1 + 2 = 3$. Similarly, to best match $S$, $R$ is aligned to be $\widetilde{R} = [1, 2, 0]$, giving rise to $ERP(R, S) = 5$. Finally, to best match $S$, $Q$ is aligned to be $\widetilde{Q} = [0, 0, 0]$, leading to $ERP(Q, S) = 8 \leq ERP(Q, R) + ERP(R, S) = 3 + 5 = 8$, satisfying triangle inequality.

To see how local time shifting works for ERP, change $Q = [3]$. Then $ERP(Q, R) = 1 + 1 = 2$, as $\widetilde{Q} = [0, 3]$. Similarly, $ERP(Q, S) = 2 + 3 = 5$, as $\widetilde{Q} = [0, 3, 0]$. The triangle inequality is satisfied as expected.

Notice that none of the results in this section are restricted to $L_1$-norm. That is, if another $L_p$-norm is used to replace $L_1$-norm in Formula 4.8, the lemma and the theorem remain valid. For the rest of the chapter, $L_1$-norm is used for simplicity.

Even though ERP is a metric distance function, it is a valid question to ask whether ERP is "natural" for time series. In general, whether a distance function is natural mainly

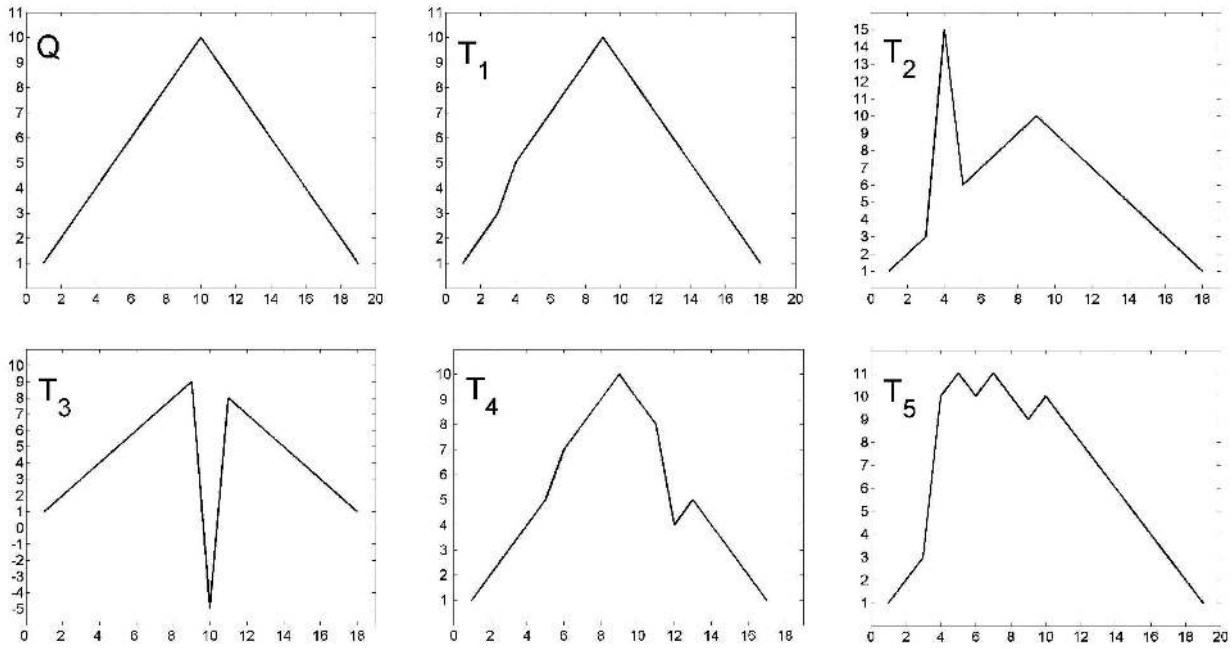



Figure 4.1: Subjective Evaluation of Distance Functions

depends on the application semantics. Nonetheless, the two experiments below suggest that ERP appears to be at least as natural as the existing distance functions.

The first experiment is a simple sanity check. A simple time series $Q$ was generated along with other five time series ($T_1$-$T_5$), by adding time shifting or noise data on one or two positions of $Q$ (Figure 4.1). For example, $T_1$ was generated by shifting the sequence values of $Q$ to the left starting from position 4, and $T_2$ was derived from $Q$ by introducing noise in position 4. Finally, $L_1$-norm, DTW, LCSS, and ERP were used to rank the five time series relative to $Q$. The rankings are as follows (where the leftmost is the most similar to $Q$):

**$L_1$-norm:** $T_1$, $T_4$, $T_5$, $T_3$, $T_2$
**DTW:** $T_1$, $T_4$, $T_3$, $T_5$, $T_2$
**LCSS:** $T_1$, $\{T_2, T_3, T_4\}$, $T_5$
**ERP:** $T_1$, $T_2$, $T_4$, $T_5$, $T_3$

As shown in these results, $L_1$-norm is sensitive to noise, as $T_2$ is considered the worst match. LCSS focuses only on the matched parts and ignores all the unmatched portions. As such, it gives $T_2, T_3, T_4$ the same rank, and considers $T_5$ the worst match. DTW gives $T_3$ a higher rank than $T_5$. Finally, ERP gives a ranked list different from all the others. Notice that the point here is *not* that ERP is the most natural. Rather, the point is that ERP appears to be no worse, if not better, than the existing distance functions.

| Avg. Error Rate | $L_1$ | DTW | LCSS | ERP |
|---|---|---|---|---|
| CBFtr | 0.03 | 0.01 | 0.01 | 0.01 |
| ASL | 0.16 | 0.10 | 0.11 | 0.09 |
| CM | 0.4 | 0.00 | 0.06 | 0.00 |

Table 4.1: Comparison of classification error rates of four distance functions

In the second experiment, a more objective evaluation is performed. The method used in Chapter 3 to evaluate the efficacy of CHD is applied in this experiment, specifically, the efficacy of distance functions are measured with classification error rate as in Table 4.1. The average classification error rate are reported using three benchmarks: the Cylinder-Bell-Funnel (CBFtr) data [33, 52], the ASL data [108] and the "cameramouse" data [108]. Compared to the standard CBF data [55] used in experiments of Chapter 3, temporal shifting is introduced in the CBFtr data set. The CBFtr data set is a 3-class problem. The other two data sets: the ASL and the "cameramouse" are same as the ones used in experimental study of Chapter 3. As shown in the Table 4.1, for three data sets, ERP performs (one of) the best, showing that it is not dominated by other well known alternatives.

## 4.3 Indexing for ERP

ERP can be considered a variant of LCSS and DTW. In particular, they share the same computational behavior. Thus, like LCSS and DTW, it takes $O(M * N)$ time to compute $ERP(Q, S)$ for time series $Q, S$ of length $M, N$ respectively. For large time series databases, for a given query $Q$, it is important to minimize the computation of the true distance

between $Q$ and $S$ for all series $S$ in the database. The topic explored here is indexing for $k$-NN queries.

Given that ERP is a metric distance function, one obvious way to prune is to apply triangle inequality, which is explained in Chapter 2. An algorithm is presented to perform triangle inequality pruning in Section 4.3.1. Another common way to prune is to apply the GEMINI framework [31] – that is, using lower bounds to guarantee no false negatives. In addition to adapting three lower bounds that were originally proposed for DTW [118, 57, 52] to ERP, Section 4.3.2 proposes a new lower bound for ERP, which has the nice property that it can be indexed by a simple $B^+$-tree.

### 4.3.1 Pruning by the Triangle Inequality

The procedure TrianglePruning shown in Algorithm 4.1 shows how the Triangle inequality is applied. The array *procArray* stores the true ERP distances computed so far. That is, if $\{R_1, \ldots, R_u\}$ is the set of time series for which $ERP(Q, R_i)$ has been computed, the distance $ERP(Q, R_i)$ is recorded in *procArray*. For time series $S$ currently being evaluated, triangle inequality ensures that $ERP(Q, S) \geq ERP(Q, R_i) - ERP(R_i, S)$, for all $1 \leq i \leq u$. Thus, it is necessary that $ERP(Q, S) \geq (max_{1 \leq i \leq u}\{ERP(Q, R_i) - ERP(R_i, S)\})$. This is implemented in lines 2 to 4. If the computed distance $maxPruneDist$ is already worse than the current $k$-NN distance stored in *result*, then $S$ can be skipped entirely. Otherwise, the true distance $ERP(Q, S)$ is computed, and *procArray* is updated to include $S$. Finally, the *result* array is updated, if necessary, to reflect the current $k$-NN neighbours and distances in sorted order.

Algorithm 4.2 shows how the *result* and *procArray* should be initialized when the procedure TrianglePruning is called repeatedly in line 4. Line 3 of the algorithm represents a simple sequential scan of all the time series in the database. This does *not* mean that a sequential scan should be used; it is included in the algorithm for two reasons. The first reason is to show how the procedure TrianglePruning can be invoked. The second reason is that a sequential scan algorithm can be used to illustrate the *pure* pruning power of triangle inequality, independent of the indexing structure. This issue will be discussed later when the inequality is used in conjunction with index structures. Moreover, triangle inequality needs not be applied only in the manner shown in Algorithm 4.1. It can be

**Algorithm 4.1** Procedure $TrianglePruning(S, procArray, pmatrix, result, Q, k)$

**Input:** $S \equiv$ the current time series; $procArray \equiv$ the array of time series with computed true distance to $Q$; $pmatrix \equiv$ precomputed pairwise distance matrix;

**Output:** $result \equiv$ the $k$-NN

```
1: maxPruneDist = 0
2: for each time series R in procArray do
3:    if (procArray[R].dist − pmatrix[R, S]) > maxPruneDist then
4:       maxPruneDist = procArray[R].dist − pmatrix[R, S]
5:    end if
6: end for
7: bestSoFar = result[k].dist {/* the k-NN distance so far */}
8: if maxPruneDist ≤ bestSoFar then {/* cannot be pruned */}
9:    realDist = ERP(Q, S) {/* compute true distance */}
10:   insert S and realDist into procArray
11:   if realDist < bestSoFar then {/* update result */}
12:      insert S and realDist into result
13:      sorted in ascending order of ERP distance
14:   end if{/* end-if, line 11 */}
15: end if
```

applied directly with existing metric space index structures such as VP-trees [105], M-trees [26], and OMNI-family of access methods [32]. Again the issue of comparison will be addressed in Section 4.4.

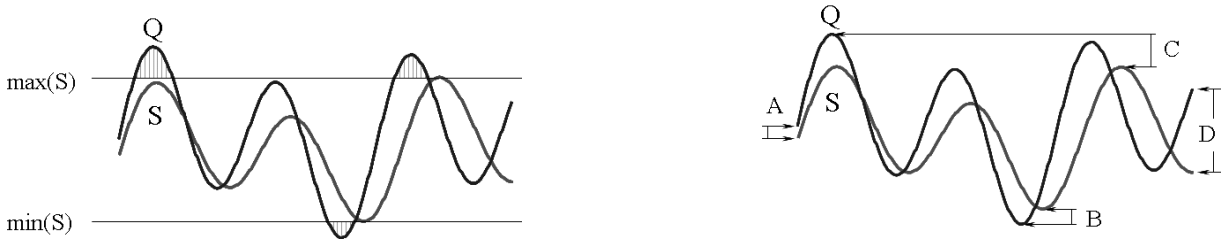
Finally, for large databases, the procedure TrianglePruning makes two assumptions. The first assumption is that the matrix $pmatrix$ is small enough to be contained in main memory. For every pair of time series $R, S$ in the database, $pmatrix[R, S]$ records the distance $ERP(R, S)$. For large databases, $pmatrix$ may be too large. The second assumption is that the size of $procArray$ is small enough. The size is left unspecified because procedure TrianglePruning works regardless of the actual size of the array. The larger the size of $procArray$, the more time series can be used for pruning. These two assumptions will be addressed in Section 4.3.3.

**Algorithm 4.2** Procedure SequentialScan($Q, k, pmatrix$)

**Input:** $Q \equiv$ the query time series; $pmatrix \equiv$ precomputed pairwise distance matrix

**Output:** $result \equiv$the $k$-NN

1: initialize $result$ so that $result[k].dist = \infty$
2: initialize $procArray$ to empty
3: **for** each time series $S$ in the database **do**
4: TrianglePruning($S, procArray, pmatrix, result, Q, k$)
5: **end for**
6: return result




(a) Yi's Lower Bound of DTW

(b) Kim's Lower Bound of DTW

Figure 4.2: Lower Bounds of Yi and Kim to DTW (figures from [52])

### 4.3.2 Pruning by Lower Bounding

For non-metric distance functions, like DTW, the main pruning strategy is lower bounding. However, lower bounding is not restricted to non-metric distance functions. In this section, a number of lower bounds are developed for ERP, which is a metric distance function.

#### Adapting Existing Lower Bounds

Given the similarities between ERP and DTW, existing lower bounds for DTW are adapted to become lower bounds for ERP. Specifically, the lower bounds proposed by Yi et al. [118], Kim et al. [57], and Keogh et al. [52] are considered.

The lower bound proposed by Yi et al., denoted as $DLB^{Yi}_{DTW}(Q, S)$, is based on the area covered by $Q$ that is greater than $max(S)$ and the area of $Q$ that is less than $min(S)$,

which is shown as the area connected by gray lines in Figure 4.2(a). $max(S)$ and $min(S)$ denote the maximum and minimum elements of $S$. $R_s$ is used to denote the value range $\langle min(S), max(S)\rangle$ of time series $S$. The formal definition of $DLB^{Yi}_{DTW}$ for $DTW(Q,S)$ is [118]:

$$DLB^{Yi}_{DTW}(Q,S) = \begin{cases} \sum_{q_i>max(S)} |q_i - max(S)| + \sum_{s_i<min(Q)} |s_i - min(Q)| \\ \qquad\qquad \text{if } R_Q \text{ overlaps } R_S \\ \sum_{q_i>max(S)} |q_i - max(S)| + \sum_{q_i<min(S)} |q_i - min(S)| \\ \qquad\qquad \text{if } R_Q \text{ encloses } R_S \\ max(\sum_{i=1}^{m} |q_i > max(S)|, \sum_{j=1}^{n} |s_j - min(Q)| \\ \qquad\qquad \text{if } R_Q \text{ disjoins } R_S \end{cases} \tag{4.9}$$

Adapting to ERP is rather straightforward. Because ERP uses a constant gap element $g$, comparing the original minimum and maximum elements with $g$ is needed, i.e., $min^*(S) = min\{min(S), g\}$ and $max^*(S) = max\{max(S), g\}$[1]. $R^*_S$ is used to denote the range $= \langle min^*(S), max^*(S)\rangle$. The lower bound for $ERP(Q,S)$ is defined as follows:

$$DLB^{Yi}_{ERP}(Q,S) = \begin{cases} \sum_{q_i>max^*(S)} |q_i - max^*(S)| + \sum_{s_i<min^*(Q)} |s_i - min^*(Q)| \\ \qquad\qquad \text{if } R^*_Q \text{ overlaps } R^*_S \\ \sum_{q_i>max^*(S)} |q_i - max^*(S)| + \sum_{q_i<min^*(S)} |q_i - min^*(S)| \\ \qquad\qquad \text{if } R^*_Q \text{ encloses } R^*_S \\ max(\sum_{i=1}^{m} |q_i > max(S)|, \sum_{j=1}^{n} |s_j - min(Q)| \\ \qquad\qquad \text{if } R^*_Q \text{ disjoins } R^*_S \end{cases} \tag{4.10}$$

**Theorem 4.2** *Let $Q$, $S$ be two time series data, then it is necessary that*

$$DLB^{Yi}_{ERP}(Q,S) \leq ERP(Q,S) \tag{4.11}$$

**Proof.** By changing the $max$ and $min$ accordingly, the proof is a simple extension of the original proof for $DLB^{Yi}_{DTW}(Q,S) \leq DTW(Q,S)$ [118]. □

Kim et. al [57] extract a 4-tuple feature vector $\langle first(S), last(S), max(S), min(S)\rangle$ for each time sequence $S$, where functions $first$, $last$, $max$, and $min$ are used to get the first, last, maximum, and minimum elements of $S$, respectively. The lower bound to $DTW(Q,S)$

[1]Recall that in this study $g = 0$

is the maximum of: $|first(Q) - first(S)|$, $|min(Q) - min(S)|$, $|max(Q) - max(S)|$, and $|last(Q) - last(S)|$, which are shown as $A$, $B$, $C$, and $D$ in Figure 4.2(b). $DLB^{Kim}_{DTW}(Q, S)$ is defined as follows:

$$DLB^{Kim}_{DTW}(Q,S) = max \begin{cases} |first(Q) - first(S)| \\ |last(Q) - last(S)| \\ |max(Q) - max(S)| \\ |min(Q) - min(S)| \end{cases} \tag{4.12}$$

To adapt this lower bound for ERP, $min(S)$ and $max(S)$ need to be changed to consider $g$, it is also necessary to consider the first or the last element may be a gap during the process of aligning two time series. Thus, $first^*(S) = g$ or $first(S)$ and $last^*(S) = g$ or $last(S)$. With this simple change, the lower bound for $ERP(Q, S)$ is defined as follows:

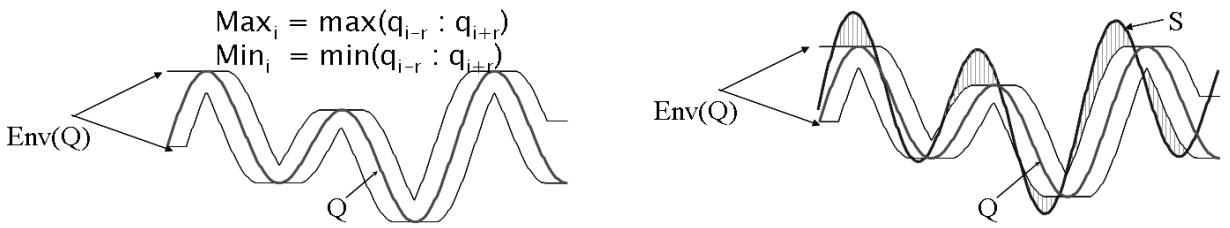
$$DLB^{Kim}_{ERP}(Q,S) = max \begin{cases} |first^*(Q) - first^*(S)| \\ |last^*(Q) - last^*(S)| \\ |max^*(Q) - max^*(S)| \\ |min^*(Q) - min^*(S)| \end{cases} \tag{4.13}$$

**Theorem 4.3** *Let $Q$, $S$ be two time series data, then it is necessary that*

$$DLB^{Kim}_{ERP}(Q,S) \leq ERP(Q,S) \tag{4.14}$$

**Proof.** By changing the $first$, $last$, $max$, and $min$ according to $g$, the proof is a simple extension of the original proof for $DLB^{Kim}_{DTW}(Q, S) \leq DTW(Q, S)$ [57]. □

Finally, Keogh et al. introduced the concept of a bounding envelop for a time series $Q$, which is of the form: $Env(Q) = [(Min_1, Max_1), \ldots, (Min_n, Max_n)]$. $Min_i$ and $Max_i$ represent the minimum and maximum values of the elements in the *warping range* ($wr$) of an element, as shown in Figure 4.3(a). The warping range of an element is the maximum number of times the element can be duplicated in local time shifting. Two constraints were made before deriving the lower bound distance: (1) all the sequences are of the same length, (2) the warping range for each element of a sequence is constrained. The lower bound distance is computed by summing the distances between values that are located outside of the envelop to the nearest boundary of the envelop, which is shown as the




(a) Keo's Envelop of Query Sequence Q (b) Keo's Lower Bound of DTW

Figure 4.3: Envelope and Lower Bounds of Keo to DTW (figures are from [52])

area connected by the gray lines in Figure 4.3(b). Given two time series data $Q$ and $S$, $DLB^{Keogh}_{DTW}(Q,S)$ is defined as follows:

$$DLB^{Keogh}_{DTW}(Q,S) = \sum_{i=1}^{N} \begin{cases} |q_i - Max_i| & \text{if } q_i > Max_i \\ |q_i - Min_i| & \text{if } q_i < Min_i \\ 0 & \text{otherwise} \end{cases} \quad (4.15)$$

To adapt the envelope to ERP, $Min_i$ and $Max_i$ are adjusted with the gap $g$, exactly as in $DLB^{Yi}_{ERP}(Q,S)$: $Max^*_i = max(Max_i, g)$ and $Min^*_i = min(Min_i, g)$. Given two time series $Q$ and $S$, the adapted lower bound for ERP $DLB^{Keogh}_{ERP}(Q,S)$ is defined as follows:

$$DLB^{Keogh}_{ERP}(Q,S) = \sum_{i=1}^{N} \begin{cases} |q_i - Max^*_i| & \text{if } q_i > Max^*_i \\ |q_i - Min^*_i| & \text{if } q_i < Min^*_i \\ 0 & \text{otherwise} \end{cases} \quad (4.16)$$

**Theorem 4.4** *Let $Q, S$ be two time series, then it is necessary that*

$$DLB^{Keogh}_{ERP}(Q,S) \leq ERP(Q,S) \quad (4.17)$$

**Proof.** After adjusting $Max_i$ and $Min_i$ to $Max^*_i$ and $Min^*_i$, respectively, the original proof for $DLB^{Keogh}_{DTW}(Q,S) \leq DTW(Q,S)$ [52] can be applied directly. □

It was shown that $DLB^{Keogh}$ is tighter than $DLB^{Kim}$ and $DLB^{Yi}$. However, compared to the other two lower bounds, $DLB^{Keogh}$ has two constraints and needs more information.

For a time sequence $S$ of length $N$, an envelope of $DLB^{Keogh}$ needs to store $2N$ values, while $DLB^{Kim}$ and $DLB^{Yi}$ only need 4 and 2 values, respectively. The effectiveness of the three adapted lower bounds will be empirically evaluated in the next section. The focus will be on the pruning power of these lower bounds. Notice that the lower bound $DLB^{Keogh}$ is not suitable for applying spatial access methods because of high dimensionality of bounding envelopes. Thus, Keogh [52] and Shasha et al. [119] proposed two different techniques based on PAA to reduce the dimensionality of query time series and bounding envelope, and defined new lower bounds for $DLB^{Keogh}$ [2].

Finally, as the base case for comparisons in the next section, a sequential scan strategy is considered for exploiting these lower bounds. Algorithm 4.3 shows how to use lower bounds to prune false candidates.

**Algorithm 4.3** Procedure $LowerboundsPruning(S, result, Q, k)$

**Input:** $S \equiv$ the current time series; $Q \equiv$ the query time sequence

**Output:** $result \equiv$ the $k$-NN

```
1: maxPruneDist = DLB(Q, S)
2: bestSoFar = result[k].dist {/* the k-NN distance so far */}
3: if maxPruneDist ≤ bestSoFar then {/* cannot be pruned */}
4:    realDist = ERP(Q, S) {/* compute true distance */}
5:    if realDist < bestSoFar then {/* update result */}
6:       insert S and realDist into result
7:       sorted in ascending order of ERP distance
8:    end if
9: end if
```

[2] These two technique are not included in the experimentation because comparison with $DLB^{Keogh}_{ERP}$ will be made directly. As a preview, the experimental results will show that the proposed new lower bound dominates $DLB^{Keogh}_{ERP}$, which translates to a superiority over the Keogh's and Shasha's lower bounds of $DLB^{Keogh}_{ERP}$.

### A New Lower Bound for ERP

In this section, a new lower bound specific to ERP is defined. Given a time series $S$ of length $N$, $sum(S) = \sum_{i=1}^{N} s_i$. The distance $DLB_{ERP}^{Chen}(Q, S)$ between $Q$ of length $M$ and $S$ of length $N$ is defined as:

$$DLB_{ERP}^{Chen}(Q, S) \quad = \quad |sum(Q) - sum(S)| \tag{4.18}$$

A key result here is the following theorem stating that this is indeed a lower bound for ERP.

**Theorem 4.5** *Let $Q$, $S$ be two time series data, then it is necessary that $DLB_{ERP}^{Chen}(Q, S) \leq ERP(Q, S)$.*

Before giving a proof of the theorem, the following results from the literature on a convex function are needed.

**Definition 4.3** *A function $f(x)$ is* convex *on an interval $[a, b]$ if for any two points $x_1$ and $x_2$ in $[a, b]$, $f[\frac{1}{2}(x_1 + x_2)] \leq \frac{1}{2}[f(x_1) + f(x_2)]$.*

**Lemma 4.3** *[83] Let $\lambda_1, \ldots, \lambda_K$ be non-negative real values such that $\sum_{i=1}^{K} \lambda_i = 1$. If $f$ is a convex function on reals, then $f(\lambda_1 x_1 + \ldots + \lambda_K x_K) \leq \lambda_1 f(x_1) + \ldots + \lambda_K f(x_K)$, where $x_1, \ldots, x_K$ are real values.*

The following corollary can be established for $L_p$-norms which are convex functions on reals.

**Corollary 4.1** *For any sequence $S = [s_1, \ldots, s_K]$, and $1 \leq p < \infty$, $K|mean(S)|^p \leq \sum_{i=1}^{K} |s_i|^p$ holds.*

**Proof:** Take $f(x) = |x|^p$, and $\lambda_i = 1/K$. Then $f(\sum_{i=1}^{K} \lambda_i s_i) = f((\sum_{i=1}^{K} s_i)/K) = |mean(S)|^p$. Thus, the inequality follows. □

With Corollary 4.1 and Lemma 4.3, Theorem 4.5 can be proven as follows.

**Proof:** Given $Q$ of length $M$ and $S$ of length $N$, let the aligned sequences that give the best local time shifting outcome be $\widetilde{Q}$ and $\widetilde{S}$ respectively. Both $\widetilde{Q}$ and $\widetilde{S}$ are of the same length, say $K$.

$$\begin{aligned}
ERP(Q,S) &= \sum_{i=1}^{K} |\widetilde{s}_i - \widetilde{q}_i| && \text{(Formula 4.8)} \\
&\geq K|mean(\widetilde{S} - \widetilde{Q})| && \text{(Corollary 4.1, } p = 1) \\
&= K|\frac{\sum_{i=1}^{K} \widetilde{s}_i - \sum_{i=1}^{K} \widetilde{q}_i}{K}| \\
&= |\sum_{i=1}^{K} \widetilde{s}_i - \sum_{i=1}^{K} \widetilde{q}_i| \\
&= |\sum_{i=1}^{N} s_i - \sum_{j=1}^{M} q_j| && \text{(Lemma 4.2)} \\
&= DLB_{ERP}^{Chen}(Q,S)
\end{aligned}$$

□

There are two points worth mentioning about Theorem 4.5. The first point concerns the application of Lemma 4.2, i.e., $\sum_{i=1}^{K} \widetilde{s}_i = \sum_{i=1}^{n} s_i$. This is due to the fact that the gap element is $g = 0$. The beauty of this equality is that regardless of $\widetilde{S}$, which depends on $Q$, the quantity $sum(S)$ is unchanged and can be inserted into an index for lower bounding purposes for *any future* query $Q$. This leads to the second point – $sum(S)$ is a single value, thus, only a 1-dimensional B$^+$-tree is sufficient to index it.

Algorithm 4.4 shows the steps to answer a $k$-NN query using a B$^+$-tree on lower bounding, $sum(S)$. Procedure DLBerp$k$-NN first conducts a standard search for the value $sum(Q)$. This results in a leaf node $I$. The first $k$ time series pointed to by $I$ are used to initialize the *result* array. Then the leaf nodes of the tree are traversed using the pointer connecting a leaf node to its siblings. All the data values bigger than $sum(Q)$ are visited in ascending order. Similarly, all the data values smaller than $sum(Q)$ are visited in descending order. In both cases, if the lower bound distance $DLB_{ERP}^{Chen}(Q,S)$ is smaller than the best $k$-NN distance so far, the true distance $ERP(Q,S)$ is computed and updates are made if necessary. Otherwise, the remaining data values can be skipped entirely.

**Algorithm 4.4** Procedure DLBerp$k$-NN$(Q, k, Tree, result)$

**Input:** $Tree \equiv$ a B$^+$-tree storing $sum(S)$ for all $S$ in database

**Output:** $result \equiv$ the $k$-NN

1: $sumQ = sum(Q)$
2: conduct a standard B$^+$-tree search on $Tree$ using $sumQ$ and let $I$ be the leaf node the search ends up with
3: pick the first $k$ time series pointed to by $I$ and initialize $result$ with the $k$ true sorted $ERP$ distances
4: let $v_1, \ldots, v_h$ be the data values in all the leaf nodes larger than the data values visited in line 3, $v_1, \ldots, v_h$ are sorted in ascending order
5: **for** each $v_i$ **do**
6: $bestSoFar = result[k].dist$ {/* the $k$-NN distance so far */}
7: **if** $((v_i - sumQ) \leq bestSoFar)$ **then** {/*need to check */}
8: **for** each $S$ pointed to by the pointer with $v_i$ **do**
9: $realDist = ERP(Q, S)$ {/* compute true distance*/}
10: **if** $realDist < bestSoFar$ **then** {/* update $result$ */}
11: insert $S$ and $realDist$ into $result$, sorted in ascending order of ERP distance
12: $bestSoFar = result[k].dist$
13: **end if**
14: **end for**
15: **else** {/* else line 7 skip the rest */}
16: break
17: **end if**
18: **end for**
19: repeat line 4 to 18 this time to the values $w_1, \ldots, w_j$ in all the leaf nodes which are smaller than the data values visited in line 3. $w_1, \ldots, w_j$ are sorted in descending order. Line 7 is modified to: if $((sumQ - w_i) \leq bestSoFar)$
20: return result

**Algorithm 4.5** Procedure ERPCombine$k$-NN$(Q, k, Tree, result)$

**Input:** $Tree \equiv$ a B$^+$-tree storing $sum(S)$ for all $S$ in database

**Output:** $result \equiv$ the $k$-NN

1: $sumQ = sum(Q)$
2: conduct a standard B$^+$-tree search on $Tree$ using $sumQ$ and let $I$ be the leaf node the search ends up with
3: pick the first $k$ time series pointed to by $I$ and initialize $result$ with the $k$ true sorted $ERP$ distances
4: let $v_1, \ldots, v_h$ be the data values in all the leaf nodes larger than the data values visited in line 3, $v_1, \ldots, v_h$ are sorted in ascending order
5: **for** each $v_i$ **do**
6: $bestSoFar = result[k].dist$ {/* the $k$-NN distance so far */}
7: **if** $((v_i - sumQ) \leq bestSoFar)$ **then** {/*need to check */}
8: **for** each $S$ pointed to by the pointer with $v_i$ **do**
9: invoke Procedure TrianglePruning() in Algorithm 4.1
10: **end for**
11: **else** {/* else line 7 skip the rest */}
12: break
13: **end if**
14: **end for**
15: repeat line 4 to 14 this time to the values $w_1, \ldots, w_j$ in all the leaf nodes which are smaller than the data values visited in line 3. $w_1, \ldots, w_j$ are sorted in descending order. Line 7 is modified to: if $((sumQ - w_i) \leq bestSoFar)$
16: return result

### 4.3.3 Combining the Two Pruning Methods

For a non-metric distance function like DTW, lower bounding can be used, but pruning by triangle inequality cannot. ERP is, of course, different. It is possible to combine both methods – use triangle inequality to save the computation of the true distance $ERP(Q, S)$ *after lower bounding*. Algorithm 4.5 shows the steps of applying first $DLB_{ERP}^{Chen}$ followed by triangle inequality.

Recall from Algorithm 4.1 that in applying the TrianglePruning procedure, there are two unresolved issues: (i) the size of the pairwise distance matrix $pmatrix$; and (ii) the size of $procArray$, i.e., the maximum number of time series whose true ERP distances are kept for triangle inequality pruning. These issues are resolved below to make procedure ERPCombine$K$-NN in Algorithm 4.5 practical for large databases and for limited buffer space situations.

Let $maxTriangle$ denote the maximum number of time series whose true ERP distances are kept for triangle inequality pruning. This value should be determined at query time by the query engine. Hereafter these time series are called the *reference series*. There are two ways to pick these reference series:

- *static*, where reference series are randomly picked *a priori*. For that matter, they may not even be series contained in the database.
- *dynamic*, where reference series are picked as procedure ERPCombine$K$-NN runs. The choices are thus query dependent. In the implementation, the first $maxTriangle$ time series are simply picked to fill up the fixed-size $procArray$.

For the static reference series strategies, the entire $pmatrix$ is not needed. Only the pairwise ERP distance matrix for each pair of reference series and data series is needed. Thus, if $L$ is the number of time series in the database, then the size of this matrix is $L * maxTriangle$. For the dynamic reference series strategy, again the entire $pmatrix$ is not needed. As the reference series are picked and kept, the appropriate column of the matrix is read into the buffer space. The buffer space requirement is $maxTriangle$ columns, each of size $L$. Thus, the total buffer space required is again $L * maxTriangle$.

While the buffer space requirement is the same, there are tradeoffs between the two strategies. For the dynamic strategy, the advantage is that the reference series are chosen

based on previously compared time series. Thus, there is no extra cost involved. The disadvantage is that at the beginning of running ERPCombine$K$-NN, there may not be enough reference series for triangle inequality to be effective. It takes at least $maxTriangle$ iterations to fill up $procArray$. In contrast, for the static strategy, all $maxTriangle$ reference time series are available right at the beginning of running ERPCombine$K$-NN. The disadvantage is that the ERP distance must be computed between the query and each reference time series, which represents additional overhead. These two strategies are experimentally compared in the next section.

Finally, note that ERPCombine$K$-NN can be generalized to apply to any metric distance function – representing an extension to the GEMINI framework. That is, within the GEMINI framework, once lower bounding has been applied, an additional step can be added to apply triangle inequality to eliminate more false positives, as long as the underlying distance function is a metric distance function.

## 4.4 Experimental Evaluation

### 4.4.1 Experimental Setup

In this section, experimental results are presented based on the 24 benchmark data sets used in [119, 52][3], the stock data set used in [110], and the random walk data set tested in [54, 52, 119]. All experiments were run on a Sun-Blade-1000 workstation with 1G memory under Solaris 2.8. All the algorithms listed in previous sections were implemented in C. The various lower bound strategies for DTW [118, 57, 52], OMNI-sequential and the HF algorithm for selecting reference series [32], were also implemented. The M-tree code is obtained from *http://www-db.deis.unibo.it/Mtree/*.

The experiments measure either total time or pruning power. Total time includes both CPU and I/O, and is measured in seconds. The pruning power $P$, defined in Chapter 3, is the fraction of the time series $S$ in the data set for which $ERP(Q, S)$ does not need to be computed (without introducing false negatives). Following [52, 119], pruning power

[3]1.Sunspot; 2.Power; 3.Spot Exrates; 4.Shuttle; 5.Water; 6. Chaotic; 7.Streamgen; 8.Ocean; 9.Tide; 10.CSTR; 11.Winding; 12.Dryer2; 13.Ph Data; 14.Power Plant; 15.Balleam; 16.Standard &Poor; 17.Soil Temp; 18.Wool; 19.Infrasound; 20.EEG; 21.Koski ECG; 22.Buoy Sensor; 23.Burst; 24.Randow walk.

| $DLB_{ERP}^{Kim}$ | $DLB_{ERP}^{Yi}$ | $DLB_{ERP}^{Keogh}$ | $DLB_{ERP}^{Chen}$ | TR |
|---|---|---|---|---|
| 0.0 | 0.28 | 0.44 | 0.54 | 0.63 |

Table 4.2: Pruning power comparison of different lower bounds on the third data set

$P$ is measured because this is an indicator free of implementation bias (e.g., page size, thresholds).

### 4.4.2 Lower Bounding vs Triangle Inequality

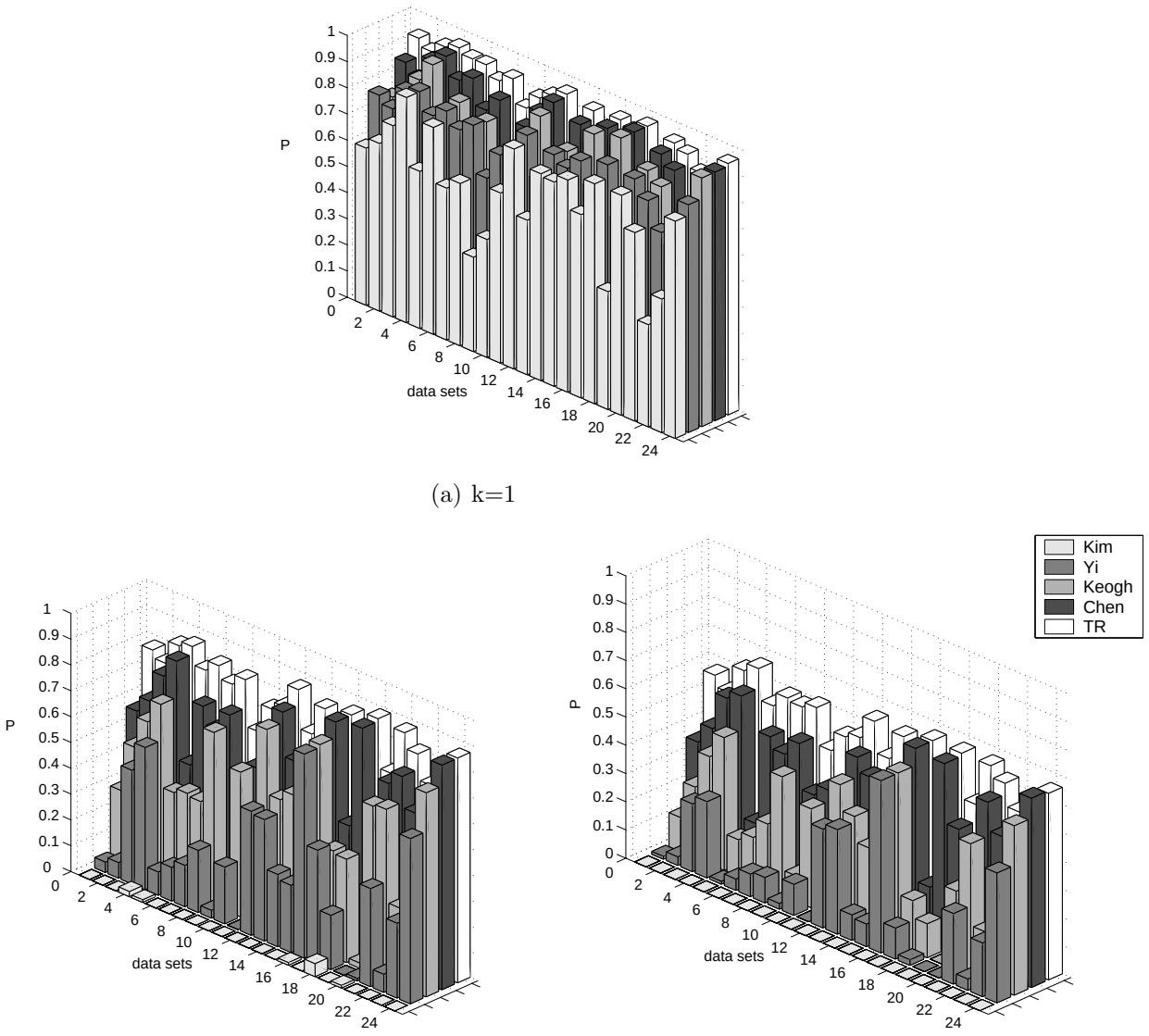
The first experiment addresses the issue of lower bounding versus triangle inequality. To make the comparisons fair, sequential scans were used. Figure 4.4 shows the pruning power of $DLB_{ERP}^{Yi}, DLB_{ERP}^{Kim}, DLB_{ERP}^{Keogh}, DLB_{ERP}^{Chen}$ and triangle inequality (denoted as TR) on the 24 benchmark data sets for $k = 1, 5, 20$. The pruning power shown is the average over 50 random queries. Each data set contains 200 time series, each with length 256. As a concrete example, an arbitrary data set is selected from the 24 and the detailed pruning power figures below for $k = 20$ are shown in Table 4.2. The third data set is picked and it contains the spot prices (foreign currency in dollars) over 10 years (10/9/86 to 8/9/96).

From the results shown in Table 4.2 and Figure 4.4, it is obvious that among the three previously proposed lower bounds, $DLB_{ERP}^{Keogh}$ performs uniformly the best. However, it is dominated by $DLB_{ERP}^{Chen}$ proposed in this chapter. Furthermore, triangle inequality (TR) consistently outperforms all the lower bounds. The larger the value of $k$, the larger is the difference. This shows the value of having a metric distance function.

### 4.4.3 Combining $DLB_{ERP}^{Chen}$ with Triangle Inequality

From Figure 4.4, triangle inequality (TR) performs the best in pruning power. In Figure 4.5, TR is compared further with:

- M-tree (page size 16K). Compared to other metric space indexing structures, such as MVP-tree and Sa-tree, M-tree is designed to minimize both I/O cost and distance computations.




(a) k=1

(b) k=5

(c) k=20

Figure 4.4: Lower Bounding vs the Triangle Inequality: Pruning Power

- OMNI-sequential (OMNI-seq). OMNI-seq is tested with different number of reference points that ranges from 5 to 100.
- $B^+$-tree (page size 1K). This implements $DLB_{ERP}^{Chen}$ with an index structure (i.e., Algorithm 4.4).
- $B^+$-tree incorporating both the triangle inequality and $DLB_{ERP}^{Chen}$ (i.e., Algorithm 4.5). Both the static and dynamic versions are included.

The experiment reveals that the pruning power of OMNI-seq with 100 reference points behaves very similar to that of TR. This is because they use the same strategies to remove false alarms; the only difference is the number of reference points that they use. Therefore, the results of OMNI-seq are not included in Figure 4.5.

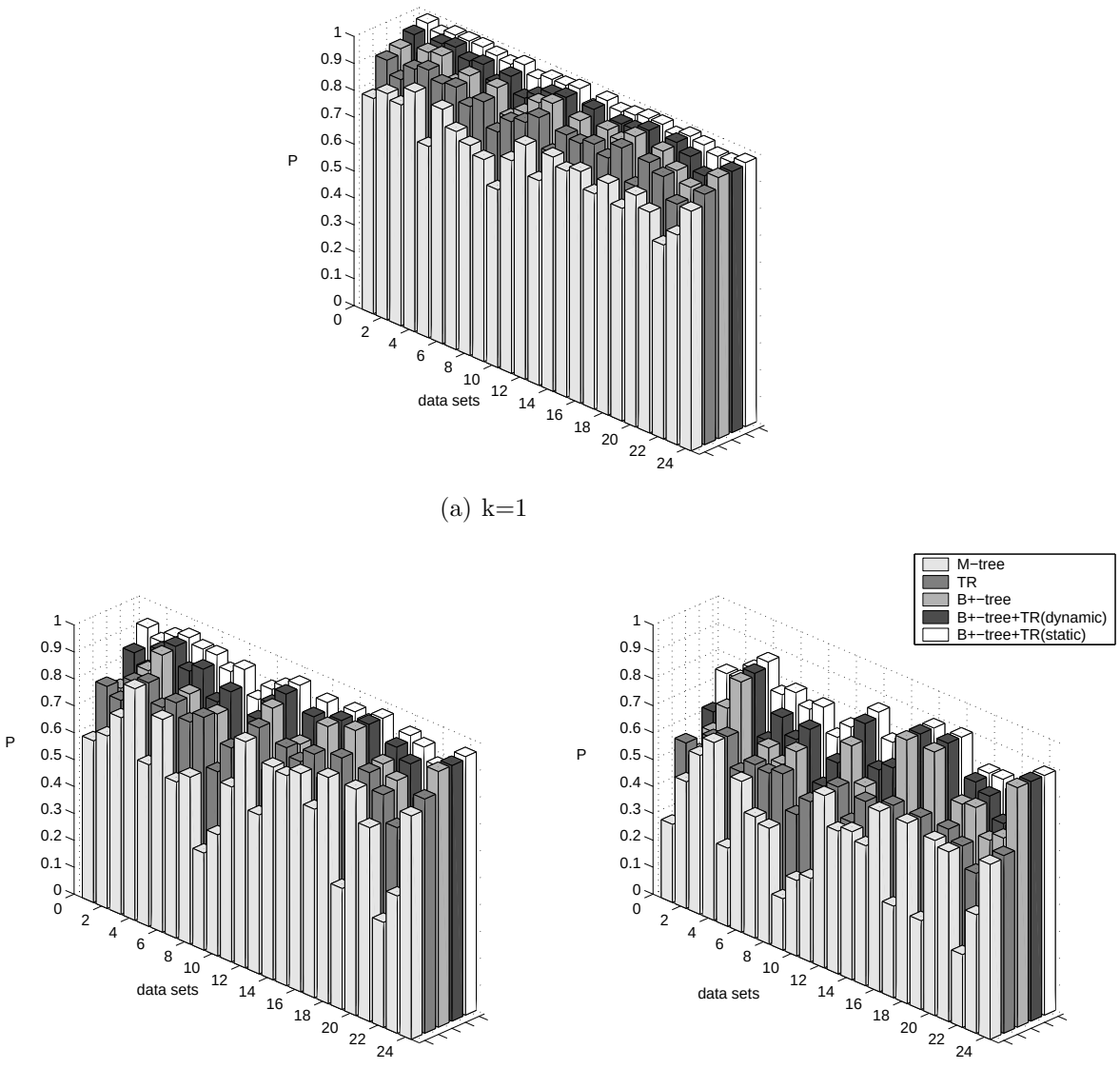
The first observation is that all the methods do well when $k = 1$. But with $k = 20$, the separation is clear. M-tree is by far the worst, even dominated by triangle inequality based on sequential scan. This is because, with M-tree, the algorithm uses the distance information below each scanned node, rather than on all the true distances previously computed. This restricts the effectiveness of pruning.

The second observation is that while TR (based on sequential scan) outperforms $DLB_{ERP}^{Chen}$ with sequential scan, the reverse is true when $DLB_{ERP}^{Chen}$ is used with a $B^+$-tree. This shows the effectiveness of the $B^+$-tree index in facilitating lower bound pruning.

The third observation is that the combination approaches $B^+$TR (static) and $B^+$TR (dynamic) are the best – better than $B^+$-tree alone or triangle inequality alone. This shows the power of combining both lower bounding and pruning by triangle inequality. For both the static and dynamic combination approaches, the number of reference series varies from 5 to 100. The graph only shows the result with 100 reference series. Table 4.3 is included to show the pruning power in greater details on the third data set with $k = 20$.

For both the static and dynamic combination approaches, increasing the number of reference series improves the pruning power as expected. The static approach appears to perform better than the dynamic approach. However, as will be shown in later experiments, this perspective on pruning power is not yet the complete picture.

Table 4.3 also includes the pruning power results using the HF algorithm to select the reference series and the OMNI-sequential with 100 reference points [32]. Both are




(a) k=1

(b) k=5

(c) k=20

Figure 4.5: Lower Bounding Together with the Triangle Inequality: Pruning Power

| M-tree | TR | B$^+$tree | B$^+$TR(static) | | |
|---|---|---|---|---|---|
| | | | 20 | 50 | 100 |
| 0.59 | 0.63 | 0.72 | 0.76 | 0.80 | 0.82 |

| B$^+$TR(dynamic) | | | B$^+$HF | OMNI-seq |
|---|---|---|---|---|
| 20 | 50 | 100 | (100) | (100) |
| 0.74 | 0.76 | 0.77 | 0.76 | 0.63 |

Table 4.3: Pruning power comparison of different indexing methods on the third data set

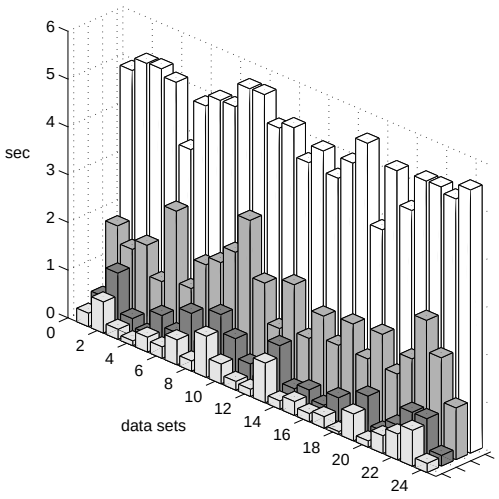
| M-tree | B$^+$tree | B$^+$TR (static) | B$^+$TR (dynamic) | OMNI-seq |
|---|---|---|---|---|
| 3.89 | 1.84 | 6.36 | 1.58 | 5.68 |

Table 4.4: Total time comparison of different indexing methods on the third data set

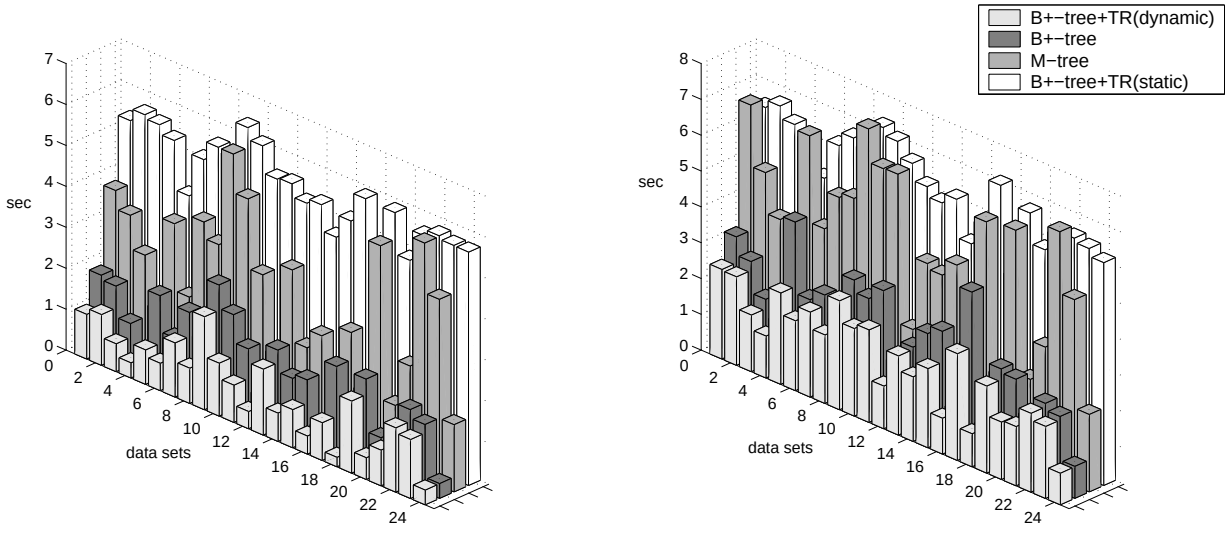
dominated by the static and dynamic approaches proposed in Section 4.3.3.

### 4.4.4 Total Time Comparisons

The pruning power results presented so far do not represent the complete picture because higher pruning power may require additional time spent on pre-processing. Figure 4.6 shows the total time spent on answering the $k$-NN queries. This time includes the pre-processing time and the query processing time (both CPU and I/O included). The time taken by TR (triangle inequality alone) is not shown in the figure because it takes much longer than the others. It was found that the time cost of OMNI-seq is very similar to that of B$^+$TR(static)), because both of them spend the same amount of time on computing the distance between query time series and the reference time series. Thus, the results of OMNI-seq are not included in Figure 4.6. Again, the graphs only show the results of combined methods with 100 reference sequences. The detailed run time (in seconds) on the third data set with $k = 20$ and 100 reference series is shown in Table 4.4. Among




(a) k=1




(b) k=5

(c) k=20

Figure 4.6: Total Time comparisons: 24 Benchmark Data Sets

the approaches shown in the Figure 4.6 and Table 4.4, the static combination approach (i.e., $B^+$TR(static)) takes the most time, followed by OMNI-seq, M-tree and the $B^+$-tree alone. The dynamic combination approach appears to be the best, in delivering very good pruning power but not at the expense of sacrificing pre-processing time.

However, it would be a mistake to conclude at this point that the dynamic combination approach is the clear winner. The main reason is that the 24 benchmarks are all small data sets, none of which contains more than 200 time series. They are used because they have been widely used as benchmarks, and represent a wide spectrum of applications and data characteristics. However, to address the size issue, the following set of results is based on much larger data sets.

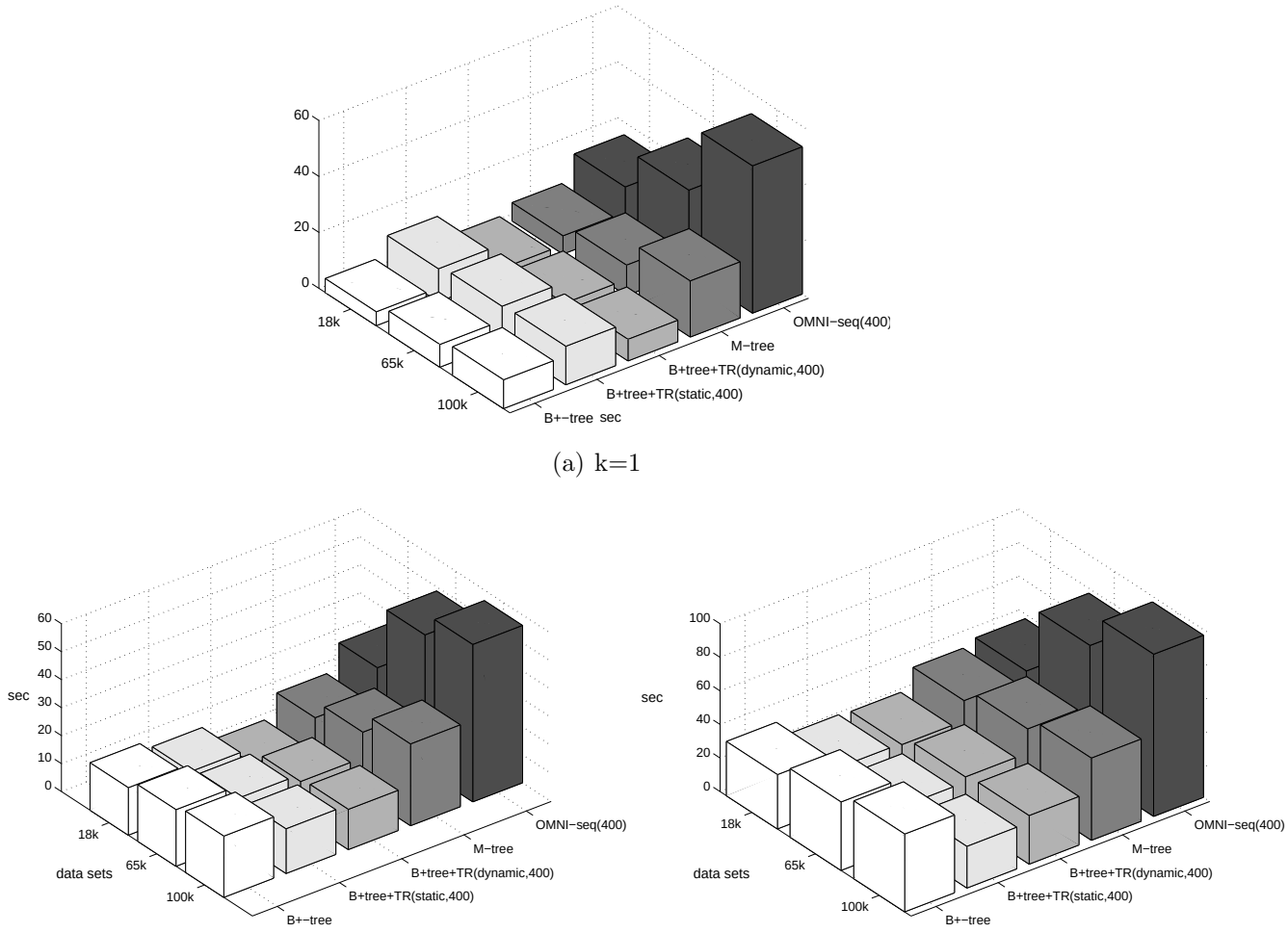
### 4.4.5 Scalability Tests

Three random walk data sets [54, 52, 119] are used in this experiment. The size of three data sets are 18432, 65,536 and 100,000, respectively, each of length 256.

Figure 4.7 shows the total time taken to answer $k$-NN queries. The time taken for five methods are included: M-tree, $B^+$-tree alone, $B^+$TR(static,400), $B^+$TR(dynamic,400) and OMNI-sequential(400), where 400 refers to the number of reference time series. It is clear that the static and dynamic approaches perform the best. In fact, the static approach manages to catch up with the dynamic approach, and in some cases, it even outperforms. The reason is that when the data set is large and selectivity is high ($k > 1$), the pre-processing represents a small overhead relative to the search. The improvement in pruning power can then compensate for the overhead.

## 4.5 Conclusions

In this chapter, a new distance function, ERP, is proposed. ERP can be viewed as a perfect marriage between $L_1$-norm and Edit Distance. It resembles $L_1$-norm since it is a metric distance function, and it resembles Edit Distance since it can handle local time shifting. The results of efficacy test on benchmark data test show that ERP is as natural for time series as existing distance functions such as DTW and LCSS.




(a) k=1

(b) k=5

(c) k=20

Figure 4.7: Total Time Comparisons: Large Data Sets

Because ERP is a metric distance function, triangle inequality can be used for pruning in answering $k$-NN queries. Besides that, developing a lower bound to ERP and applying GEMINI frame work is another solution to improve the retrieval efficiency. Therefore, besides adapting the lower bounds proposed for DTW and using them for ERP, a new lower bound is also developed. Unlike adapted lower bounds which require accessing with a multi-dimensional indexing structure, the proposed lower bound is 1-dimensional and can simply be implemented in a $B^+$-tree, which reduces the storage requirement and save

the possible I/O cost as a consequence. Because pruning by triangle inequality and by lower bounds are orthogonal, the two strategies are combined in the $k$-NN algorithm. This combination can be considered as an extension of GEMINI framework for indexing time series data with a metric distance function.

Extensive experiments were conducted using 24 benchmarks and other large data sets. Consistently, the new lower bound of ERP dominates existing strategies. More importantly, the combination approaches which incorporate both lower bounding and pruning by triangle inequality deliver superb pruning power as well as wall clock time performance. Among the two combination approaches, both the static and the dynamic approach are viable alternatives depending on $k$ and the size of the database.

# Chapter 5

# Similarity-based Search Using Edit Distance on Real Sequences

## 5.1 Introduction

In the previous chapter, similarity-based search over time series or trajectory data that contain local time shifting is studied. This chapter focuses on similarity-based retrieval on data containing both local time shifting and noise. In this chapter, all the definitions, theorems, and techniques are defined over trajectory data. This is because compared to time series data, trajectories may have more outliers [108]. Unlike stock, weather, or commodity price data, trajectories of moving objects are captured by recording the positions of the objects from time to time (or tracing the moving object from frame-to-frame in video data). Therefore, due to sensor failures, disturbance signals or errors in detection techniques, many outliers may appear. In fact, all the techniques used here can be applied to time series as well when they contain local time shifting and noise.

As pointed out in chapter 2, $L_p$-norm and DTW are sensitive to noise. LCSS can handle noise but it does not consider various gap differences between similar subsequences, which leads to inaccuracies. Here the gap differences refer to sub-trajectories in between two identified similar components of two trajectories. Even ERP, developed in Chapter 4, is sensitive to noise although it is robust against local time shifting. This is because ERP takes the differences of real values as distance. Revisiting the example trajectory data that contain

noise (Chapter 2), $Q = [(t_1, 1), (t_2, 2), (t_3, 3), (t_4, 4)]$,$R = [(t_1, 10), (t_2, 9), (t_3, 8), (t_4, 7)]$, $S = [(t_1, 1), (t_2, 100)$,$(t_3, 2), (t_4, 3), (t_5, 4)]$, $P = [(t_1, 1), (t_2, 100), (t_3, 101), (t_4, 2), (t_5, 4)]$, ERP produces the same rank as Euclidean distance and DTW, which confirms that it is sensitive to noise.

Therefore, in this chapter, a new distance function is introduced to addresses the case that data contain local time shifting and noise. The retrieval efficiency issues using this distance function are also discussed.

The rest of the chapter is arranged as follows. Section 5.2 presents a new distance function, EDR, as well as comparative efficacy test results on benchmark data sets. In Section 5.3, three pruning techniques are introduced that can be used to improve the retrieval efficiency. An optimization is also proposed by combining the three pruning methods. Experimental studies on retrieval efficiency in terms of pruning power and speedup ratio for each pruning technique and the combination method are presented in Section 5.4.

## 5.2 Edit Distance on Real Sequences

In this section, a new distance function, called *Edit Distance on Real sequence* (EDR), is proposed to handle data contain local time shifting and noise. EDR is more robust and accurate than the existing distance functions in measuring the similarity between two trajectories. For simplicity and without loss of generality, objects are treated as points that move in a two-dimensional space ($x-y$ plane) and time is considered to be discrete. Thus, given a trajectory $R = [(t_1, r_1) \ldots, (t_N, r_N)]$, $r_i$ is a pair, $(r_{i,x}, r_{i,y})$ and $(t_i, r_i)$ is defined as an *element* of trajectory $R$. Given $R$, its $x$ and $y$ position values are normalized using the corresponding mean $(\mu_x)$, $(\mu_y)$ and standard deviation $(\sigma_x)$, $(\sigma_y)$, respectively [37]:

$$Norm(R) = [(t_1, (\frac{r_{1,x} - \mu_x}{\sigma_x}, \frac{r_{1,y} - \mu_y}{\sigma_y})), \ldots, (t_N, (\frac{r_{N,x} - \mu_x}{\sigma_x}, \frac{r_{N,y} - \mu_y}{\sigma_y}))] \tag{5.1}$$

Normalization is recommended so that the distance between two trajectories are invariant to spatial scaling and shifting. Throughout this chapter, $R$ is used to denote $Norm(R)$.

Similar to ERP, EDR is based on Edit Distance (ED) [63]. As defined in Chapter 4, given two strings $R$ and $S$, $ED(R, S)$ is the number of insert, delete, or replace operations that are needed to convert $R$ into $S$. Since trajectories are not strings but numerical value

pair sequences, for EDR, it is crucial to properly define what is meant by *matching* between element pairs of different trajectories.

**Definition 5.1** *A pair of trajectory element vectors $r_i$ and $s_j$ from two trajectories $R$ and $S$, respectively, are said to match ($match(r_i, s_j) = true$) if and only if $|r_{i,x} - s_{j,x}| \leq \epsilon$ and $|r_{i,y} - s_{j,y}| \leq \epsilon$, , where $\epsilon$ is the matching threshold.*

**Definition 5.2** *Given two trajectories $R$ and $S$ of lengths $M$ and $N$, respectively, the Edit Distance on Real sequence (EDR) between $R$ and $S$ is the number of insert, delete, or replace operations that are needed to change $R$ into $S$. $EDR(R, S)$ is defined as follows:*

$$EDR(R,S) = \begin{cases} M & \text{if } N = 0 \\ N & \text{if } M = 0 \\ min\{EDR(Rest(R), Rest(S)) + subcost, & \\ \quad EDR(Rest(R), S) + 1, EDR(R, Rest(S)) + 1\} & \text{otherwise} \end{cases} \tag{5.2}$$

where $subcost = 0$ if $match(r_1, s_1) = true$ and $subcost = 1$ otherwise.

In Definition 5.2, the cost of a replace, insert, or delete operation is only 1, which corresponds to the original definition of edit distance [63].

Compared to Euclidean distance, DTW, and LCSS, EDR has the following virtues:

- In EDR, the matching threshold reduces effects of noise by quantizing the distance between a pair of elements to two values, 0 and 1. Therefore, the effect of outliers on the measured distance is much less in EDR than that in Euclidean distance, DTW, and ERP.
- Like ERP, seeking the minimum number of edit operations required to change one trajectory to another offers EDR the ability to handle local time shifting.
- Contrary to LCSS, EDR assigns penalties to the gaps between two matched sub-trajectories according to the lengths of gaps, which makes it more accurate than LCSS.

Revisiting the previous example of trajectory data that contain noise, the similarity ranking relative to $Q$ with EDR ($\epsilon = 1$) is $S, P, R$, which is the expected result.

In order to compare the efficacy of different distance functions, the following objective evaluation is applied. The first test is to check the clustering performance with different distance functions. The hierarchy clustering methods, which had been used in Chapter 3 to test CHD, is again used here to compare the efficacy of four distance functions on two labelled data sets, the "cameramouse" [34] and the Australian Sign Language (ASL) data sets. Different values of $\epsilon$ are used to run the experiments and a quarter of the maximum standard deviation of trajectories is set as the matching threshold $\epsilon$, which leads to the best clustering results. In order to make a fair comparison, DTW is also tested with different warping lengths and the best results are reported. Since Euclidean distance requires sequences with the same length, the strategy that was used in [108] can be applied, where the shorter of the two trajectories slides along the longer one and the minimum distance is recorded. The best result of each distance function is reported in Table 5.1.

| Correct results | Euclidean distance | DTW | ERP | LCSS | EDR |
|---|---|---|---|---|---|
| CM (total 10 correct) | 2 | 10 | 10 | 10 | 10 |
| ASL (total 45 correct) | 4 | 20 | 21 | 21 | 21 |

Table 5.1: Clustering results of five distance functions

As shown in Table 5.1, EDR performs as well as DTW, ERP and LCSS. The poor clustering results of Euclidean distance confirm that it is very sensitive to local time shifting. In this test, DTW and ERP perform similar to LCSS and EDR, because the two trajectory data sets contain local time shifting, but very little or no noise, which confirms earlier results [108]. Thus, this test shows that EDR is as effective as DTW, ERP, and LCSS in measuring similarities of trajectories when the trajectories contain little or no noise.

The second test uses classification of labelled data to evaluate the efficacy of a distance function, using the same method as in Chapters 3 and 4. The same two data sets of the first test and the Gun data set [89] are used. The gun data set is a 2 class data set with 100 trajectories per class.

In order to test the ability of distance functions to handle local time shifting and noise, Interpolated Gaussian noise (about 10-20% of the length of trajectories) and local time shifting are added to each data set using the program in [109]. To get average values over a number of data sets, each raw data set is used as a seed to generate 50 distinct data sets that include noise and time shifting. The results are shown in Table 5.2. For three data sets, EDR performs the best, showing that it is superior to the other distance functions in handing noise and local time shifting.

| Avg. Error Rate | Euclidean distance | DTW | ERP | LCSS | EDR |
|---|---|---|---|---|---|
| CM | 0.25 | 0.11 | 0.10 | 0.10 | 0.03 |
| ASL | 0.28 | 0.16 | 0.14 | 0.14 | 0.09 |
| Gun | 0.15 | 0.11 | 0.11 | 0.11 | 0.08 |

Table 5.2: Classification results of five distance functions

To conclude, the results of these two tests prove that EDR performs as well as DTW, ERP, and LCSS when trajectories contain little or no noise, and it is more robust than DTW and ERP, and more accurate than LCSS in noisy conditions.

## 5.3 Efficient Trajectory Retrieval Using EDR

The matching threshold $\epsilon$ in EDR is introduced to remove the effect of noise. However, the introduction of the threshold causes EDR to violate triangle inequality, making it non-metric and, thus, non-indexable by traditional distance-based indexing methods. However, this does not mean that EDR is not a "good" distance function. As pointed out by Jacobs et. al [46], it is not the poor selection of features or careless design that cause a distance function not to follow triangle inequality. Inherently, distance functions that are robust to noisy data will usually violate triangle inequality. Many robust distance functions have been proposed in the domain of image retrieval, such as Hausdorff distance [44] and Dynamic Partial Function (DPF) [35], that do not follow triangle inequality. Furthermore, much work in psychology also suggest that human similarity judgements do

not follow triangle inequality either [104]. Therefore, given a "good", robust but non-metric distance function, the issue is how to improve the retrieval efficiency for similarity search. The computation cost of EDR is $O(M * N)$, where $M$ and $N$ are the lengths of the two trajectories (the cost of DTW, ERP, LCSS are quadratic as well), which removes the possibility of using sequential scan when the database size is large. Therefore, three pruning methods are proposed to reduce the number of computations between the query trajectory and trajectories in the database.

### 5.3.1 Pruning by Mean Value Q-gram

Given a string $S$, Q-grams of $S$ can be obtained by "sliding" a window of length $q$ over the characters of $S$. Thus, each Q-gram is a substring of size $q$. For example, in Figure 5.1, a sliding window of size 3 slides from the beginning of "AGOODBOY", and six Q-grams are obtained {"AGO", "GOO", "OOD", "ODB" "DBO", "BOY"}.

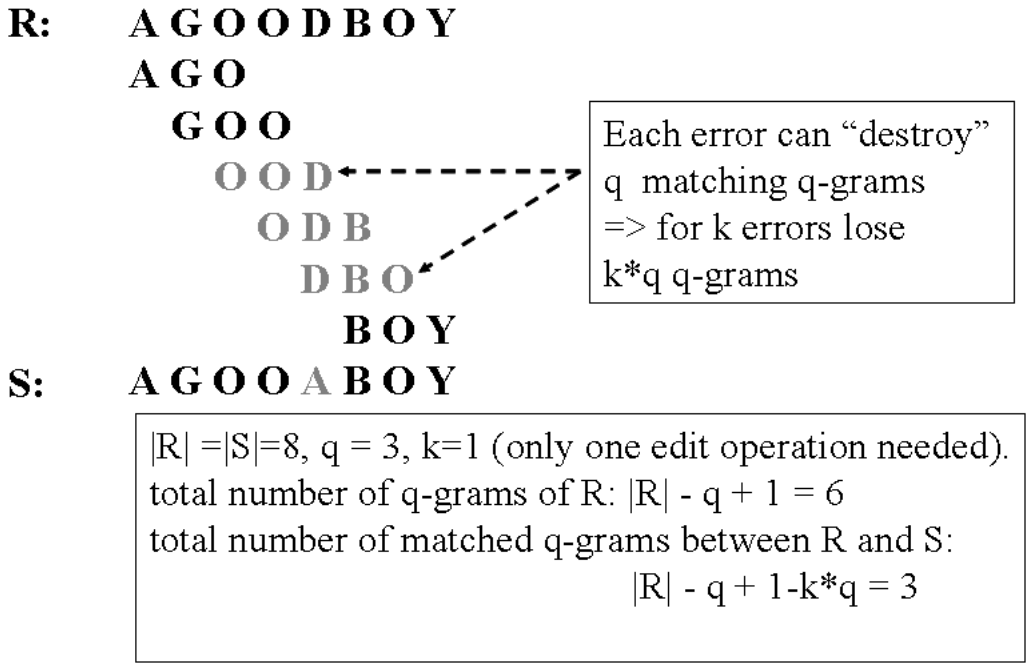



Figure 5.1: A Q-gram example

Q-grams have been well studied as a solution to the approximate string matching problem [48, 99], which is defined as follows: given a long text of length $N$ and a pattern of length $M$ ($M \leq N$), retrieve all the segments of the text whose *edit distance* to the pattern is at most $k$. If a pattern and a string are similar to each other, the number of substrings that are common to each other is high. This is the intuition of using Q-grams as a filter. As shown in Figure 5.1, only one edit operation is needed to transfer $R$ to $S$ and this single editing operation can affect at most 3 Q-grams from $R$ to be dissimilar to the corresponding Q-grams in $S$. The following theorem exactly reflects this intuition and is used to remove the segments that do not satisfy the requirement (at most $k$ editing operations) before using dynamic programming to compute the real edit distance between the pattern and the segment.

**Theorem 5.1** *Let $R$ and $S$ be strings of length $M$ and $N$. $R$ and $S$ within edit distance $k$ have at least $p = max(M, N) - q + 1 - kq$ common Q-grams [48].*

The value of $p$ consists of two parts: the first part, $max(M, N) - q + 1$, is the maximum number of Q-grams of size $q$ in $R$ or $S$, and the second part, $kq$, is the maximum number of Q-grams that can be affected between $R$ and $S$ by $k$ edit operations.

Theorem 5.1 can be used in the context of EDR to remove false candidates, but changes are required since the exact value match in counting common Q-grams between trajectories is not required. Thus, what it means to "match" has to be redefined as follows:

**Definition 5.3** *Given two Q-grams $r = [(r_{1,x}, r_{1,y}), \ldots, (r_{q,x}, r_{q,y})]$ and $s = [(s_{1,x}, s_{1,y}), \ldots, (s_{q,x}, s_{q,y})]$ of trajectories $R$ and $S$, respectively, $r$ matches $s$ if and only if each element of $r$ matches its corresponding element in $s$.*

However, the space requirement to store Q-grams is very high, since each Q-gram of a trajectory has to be stored. Furthermore, Theorem 5.1 only applies to one-dimensional strings, and naive implementation of Q-grams on multi-dimensional trajectories will not only increase the space cost but may also suffer the dimensionality curse problem [112]. Finally, Theorem 5.1 applies only to range queries (searching strings whose distances to the query string is at most $k$ edit operations). In most cases, users may not know the range a priori. In these situations, $k$-NN search is more meaningful.

Compared to strings, elements of trajectories are real values; thus, the properties of real values can be used to reduce the storage requirement of Q-grams. Based on Definitions 5.1 and 5.3, the following theorem holds:

**Theorem 5.2** *Given a matching threshold $\epsilon$, if two Q-grams $r = [(r_{1,x}, r_{1,y}), \ldots, (r_{q,x}, r_{q,y})]$ and $s = [(s_{1,x}, s_{1,y}), \ldots, (s_{q,x}, s_{q,y})]$ match, their mean value pairs $r_{mean} = (\frac{\sum r_{i,x}}{q}, \frac{\sum r_{i,y}}{q})$ and $s_{mean} = (\frac{\sum s_{i,x}}{q}, \frac{\sum s_{i,y}}{q})$ also match.*

**Proof.** Straightforward induction from inequalities of Definition 5.3. □

Based on Theorem 5.2, no more space is needed to store Q-grams than that is required to store a trajectory, regardless of the size of Q-gram. Most importantly, there is the possibility to index the Q-grams of trajectories with less dimensions. For example, given a trajectory $R = [(t_1, (1, 2)), (t_2, (3, 4)), (t_3, (5, 6)), (t_4, (7, 8)), (t_5, (9, 10))]$, Q-grams of size 3 for $R$ are: [(1,2), (3, 4), (5, 6)], [(3,4), (5,6), (7, 8)], [(5,6), (7,8), (9,10)]. A six-dimensional R-tree [38] is needed to index these Q-grams, and with increasing Q-gram sizes, the dimensionality will grow (e.g. for three dimensional trajectories, Q-grams of size 5 need a 15 dimensional R-tree) and may cause R-tree to perform worse than sequential scan [112]. However, the mean value Q-gram pairs of $R$ are: (3,4), (5,6), (7,8). Only a two dimensional R-tree is needed to index these pairs, even for Q-grams with larger size.

Q-grams were originally proposed as a filtering technique in answering range queries; in this section, two algorithms are proposed to extend this technique to answer $k$-NN queries. The first algorithm utilizes indexes on Q-grams to speed up the process of finding common Q-grams between two trajectories. Procedure Qgram$k$-NN-index (Algorithm 5.1) lists steps of the first algorithm, which conducts a standard search for each mean value pair $q_{mean}$ of the Q-grams in $Q$ and updates the corresponding Q-gram counter for each trajectory in the database. The Q-gram counters are then sorted in descending order and the first $k$ trajectories pointed by the first $k$ elements of the Q-gram counters are used to initialize the *result* array. Finally values in the rest of the Q-gram counters are visited in descending order. If the value satisfies the inequality stated in Theorem 1, the true distance $EDR(Q, S)$ is computed and updates to the result list are made if necessary. Otherwise, the remaining data values can be skipped entirely.

**Algorithm 5.1** Procedure Qgram$k$-NN-index$(Q, k, Tree, result)$

/*algorithm for applying Q-grams to answer $k$-NN query with indexes*/

**Input:** $Tree \equiv$ a R*-tree storing mean value pairs for all Q-grams in database

**Output:** $result \equiv$ the $k$-NN

1: **for** each Q-gram $q$ of trajectory Q **do**
2: $q_{mean} = mean(q)$
3: conduct a standard R*-tree search on $Tree$ using $q_{mean}$
4: increase the Q-gram counter for the trajectory that have a match mean value pair to $q_{mean}$
5: **end for**
6: sort the Q-gram counters of trajectories in descending order.
7: pick the first $k$ trajectories pointed by Q-gram counters and initialize $result$ with the $k$ true (sorted) $EDR$ distances
8: let $v_i, \ldots, v_n$ be the data values of Q-gram counters starts from $k+1$ and $|Q|$ be the length of query trajectory
9: **for** each $v_i$ **do**
10: $bestSoFar = result[k].dist$ {/* the $k$-NN distance so far */}
11: **if** $(v_i) \geq (max(|Q|, |S|) - (bestSoFar + 1) * size(Q - gram))$ **then** {/*need to check*/}
12: **for** each trajectory $S$ pointed by $v_i$ **do**
13: $realDist = EDR(Q, S)$ {/* compute true distance */}
14: **if** $(realDist < bestSoFar)$ **then** {/* update $result$ */}
15: insert $S$ and $realDist$ into $result$ sorted in ascending order of EDR distance
16: $bestSoFar = result[k].dist$
17: **end if**{/* end-if, line 14 */}
18: **end for**{/* end-for, line 12 */}
19: **else** {/* else, line 11, skip the rest */}
20: break
21: **end if**
22: **end for**{/* end-for, line 9 */}
23: return result

**Theorem 5.3** *Using procedure* Qgram$k$-NN-index *to answer a $k$-NN query does not introduce false dismissals.*

**Proof.** Prove by contradiction. Assume that procedure Qgram$k$-NN-index introduces false dismissals; then the following two statements are valid: (1) $v_i$ is the Q-gram counter value of trajectory $R$ (length $N$) and $v_i < max(M,N) + 1 - (bestSoFar + 1) * size(Qgram)$; (2) $EDR(Q,R) < bestSoFar$. According to Theorem 5.1, $v_i \geq max(M,N) + 1 - (EDR(Q,R) + 1) * size(Qgram)$. Based on statement (2), $v_i \geq max(M,N) + 1 - (bestSoFar + 1) * size(Qgram)$, which contradicts statement (1).□

Furthermore, in line 8 of the algorithm, the Q-gram counters are visited in descending order, which also guarantees that skipping the rest of the elements in line 19 will not introduce false dismissals. This is because, if $v_i \geq v_{i+1}$, and $v_i < max(M,N) + 1 - (bestSoFar+1)*size(Qgram)$, then $v_{i+1} < max(M,N)+1-(bestSoFar+1)*size(Qgram)$.

Procedure Qgram$k$-NN-index utilizes an index on mean values of Q-grams to find common Q-grams between the query trajectory and each data trajectory in the database. The computation cost of this pruning step (not including sorting) is $O(|Q| * \log(L * max(|R|)))$, where $L$ is the size of the database, $max(|R|)$ is the maximum length of trajectories in the database. However, when the database size $L$ increases, the index on Q-grams grows and the search operation on the index becomes expensive, which may increase the total execution time of each query as a consequence.

The second algorithm performs linear scan over sorted Q-grams. Procedure Qgram$k$-NN-seq (Algorithm 5.2) first conducts a merge join algorithm to find the common Q-grams between two trajectories without any indexes before computing the real EDR between them. The computation cost of this pruning step is only $O(|Q| + max(|R|))$ (not including sorting cost).

The above two approaches apply mean value Q-gram filters directly on trajectories to reduce the number of EDR calculations. Another possibility is to take the projection of the trajectory on each dimension, which produces a single dimensional data sequence, and apply Q-gram filters to them. Of course, it is necessary to ensure that no false dismissals will occur.

**Theorem 5.4** *Let $R$ and $S$ be trajectories of length $M$ and $N$. If $EDR(R,S) \leq k$, the number of common Q-grams between two single dimensional data sequences $R^x$, $S^x$ (or $R^y$,*

**Algorithm 5.2** Procedure Qgram$k$-NN-seq($Q, k, result$)

/*algorithm for applying Q-grams to answer $k$-NN query with linear scan*/

**Input:** Mean value pairs of each trajectory are sorted in an ascending order of mean values of on $x$ axis

**Output:** $result \equiv$ the $k$-NN

1: sort mean value pairs of Q in ascending order on $x$ values
2: pick the first $k$ trajectories and initialize $result$ with the $k$ true (sorted) $EDR$ distances
3: starts from $i = k + 1$ and $|Q|$ be the length of query trajectory
4: **for** each trajectory $S_i$ in the database **do**
5: $bestSoFar = result[k].dist$ {/* the $k$-NN distance so far */}
6: $v_i$ =merge-join(mean-value-pairs($Q$), mean-value-pairs($S_i$))
{/* applying merge join to compute common mean value pairs */}
7: **if** $(v_i) \geq (max(|Q|, |S_i|) - (bestSoFar + 1) * size(Q - gram))$ **then** {/*need to check*/}
8: $realDist = EDR(Q, S)$ {/* compute true distance */}
9: **if** $(realDist < bestSoFar)$ **then** {/* update $result$ */}
10: insert $S$ and $realDist$ into $result$
11: sorted in ascending order of EDR distance
12: $bestSoFar = result[k].dist$
13: **end if**{/* end-if, line 9 */}
14: **end if**
15: **end for**
16: return result

*$S^y$) obtained by projecting $R$ and $S$ over dimension $x$ (or $y$) is at least $max(M, N) - q + 1 - kq$.*

**Proof.** Proving the theorem to hold on one of the projected dimensions is sufficient and $x$ dimension is selected. From Definition 5.1, if two Q-grams $r = [(r_{1,x}, r_{1,y}), \ldots, (r_{q,x}, r_{q,y})]$ and $s = [(s_{1,x}, s_{1,y}), \ldots, (s_{q,x}, s_{q,y})]$ of trajectories $R$ and $S$ match, then each element of $r_i$ matches its corresponding element in $s_i$. As a consequence, each $r_{i,x}$ matches $s_{i,x}$. Therefore, $p_x \geq p$, where $p_x$ and $p$ are the number of common Q-grams between $x$ dimensional data sequence and trajectories. From Theorem 5.1, $p \geq max(M, N) - q + 1 - kq$, thus $p_x \geq max(M, N) - q + 1 - kq$. □

Similar to applying Theorem 5.1 to remove false candidates, Theorem 5.4 can be used by only applying mean value Q-gram filters on projected single dimensional data sequences and removing the false candidates without introducing false dismissals.

Based on Theorems 5.2 and 5.4, only the mean value of each Q-gram in a one-dimensional data sequence needs to be stored. As a consequence, a simple B+-tree can be used to index mean values of Q-grams or the second merge join algorithm can be applied on projected single dimensional data sequences of trajectories. Compared to indexing Q-grams of trajectories using R-tree or merge join on two dimensional sequences, both the space and disk access times are reduced. However, because only information from one-dimensional data sequence is used, the pruning power is also reduced compared to that of using all the dimensions.

### 5.3.2 Pruning by Near Triangle Inequality

Even though EDR does not follow triangle inequality, the following property holds.

**Theorem 5.5** *(Near Triangle Inequality) Given three trajectories $Q$, $S$, and $R$, we have $EDR(Q, S) + EDR(S, R) + |S| \geq EDR(Q, R)$, where $|S|$ is the length of $S$.*

**Proof.** The first part of left hand side of the inequality, $EDR(Q, S) + EDR(S, R)$, can be viewed as the number of edit operations needed to convert trajectory $Q$ to $S$ and then to convert $S$ to $R$. $EDR(Q, S)+EDR(S, R)$ may be less than $EDR(Q, R)$, since an element $s_i$

of $S$ may match elements $q_i$ and $r_i$ of $Q$ and $R$, respectively, but $q_i$ and $r_i$ may not match. In this case, $EDR(Q, R)$ has one more edit operation than that of $EDR(Q, S) + EDR(S, R)$. The extreme case is that all the elements of $S$ match the elements of $Q$ and $R$, however those matched elements of $Q$ and $R$ do not match each other, thus $EDR(Q, R)$ is larger than $EDR(Q, S) + EDR(S, R)$ by at most $|S|$, which is the second part of the left hand side of the inequality. □

By rewriting the near triangle inequality (Theorem 5.5), $EDR(Q, S) \geq EDR(Q, R) - EDR(S, R) - |S|$ can be derived. If $EDR(Q, R)$ and $EDR(S, R)$ are known, $EDR(Q, R) - EDR(S, R) - |S|$ can be treated as a lower bound distance of $EDR(Q, S)$. Therefore, near triangle inequality can be used to prune out false candidates. Procedure NearTrianglePruning (Algorithm 5.3) gives the algorithm for the application of near triangle inequality to answer a $k$-NN query. The matrix $pmatrix$ holds the precomputed pairwise EDR distances of the trajectory database. The array $procArray$ stores the true EDR distances computed so far. That is, if $\{R_1, \ldots, R_u\}$ is the set of trajectories for which $EDR(Q, R_i)$ has been computed, the distance $EDR(Q, R_i)$ is recorded in $procArray$. Thus, for trajectory $S$ currently being evaluated, the near triangle inequality ensures that $EDR(Q, S) \geq EDR(Q, R_i) - EDR(R_i, S) - |S|$, for all $1 \leq i \leq u$. Thus, it is necessary that $EDR(Q, S) \geq (max_{1 \leq i \leq u}\{EDR(Q, R_i) - EDR(R_i, S) - |S|\})$ (lines 2 to 4). If distance $maxPruneDist$ is larger than the current $k$-NN distance stored in $result$, then $S$ can be skipped. Otherwise, the true distance $EDR(Q, S)$ is computed, and $procArray$ is updated to include $S$. Finally, the $result$ array is updated to reflect the current $k$-NN neighbours and distances in sorted order. The computation cost of the pruning step in NearTrianglePruning is constant, which is the size of $procArray$.

Application of the NearTrianglePruning procedure encounters two issues: (i) the size of the pairwise distance matrix $pmatrix$; and (ii) the size of $procArray$, i.e., the maximum number of trajectories whose true EDR distances are kept for near triangle inequality pruning. The dynamic strategies proposed in Chapter 4 are used to resolve these issues, which make the procedure practical for large databases and for limited buffer space situations. The computation cost of pruning step in NearTrianglePruning is constant, which is the size of $procArray$.

**Algorithm 5.3** Procedure NearTrianglePruning($S, procArray, pmatrix, result, Q, k$)

/*algorithm for near triangle inequality pruning*/

**Input:** $S \equiv$ the current trajectory; $procArray \equiv$ the array of trajectories with computed EDR to $Q$; $pmatrix \equiv$ precomputed pairwise distance matrix;

**Output:** $result \equiv$ the $k$-NN

1: $maxPruneDist = 0$
2: **for** each trajectory $R$ in $procArray$ **do**
3: **if** $procArray[R].dist - pmatrix[R, S] - |S|) > maxPruneDist$ **then**
4: $maxPruneDist = procArray[R].dist - pmatrix[R, S] - |S|$
5: **end if**
6: **end for**
7: $bestSoFar = result[k].dist$ {/* the $k$-NN distance so far */}
8: **if** $maxPruneDist \leq bestSoFar$ **then** {/* cannot be pruned */ }
9: $realDist = EDR(Q, S)$ {/* compute true distance */}
10: insert $S$ and $realDist$ into $procArray$
11: **if** $realDist < bestSoFar$ **then** {/* update $result$ */}
12: insert $S$ and $realDist$ into $result$
13: sorted in ascending order of EDR distance
14: **end if**
15: **end if**

Let $maxTriangle$ denote the maximum number of trajectories whose EDR distances to the rest of trajectories in the database are kept for near triangle inequality pruning. This value should be determined at query time by the query engine. Hereafter these trajectories are called the *reference trajectories*. Dynamic strategies pick these reference trajectories as procedure NearTrianglePruning runs, therefore, the entire $pmatrix$ is not needed. As the reference trajectories are picked and kept, the appropriate column of the distance matrix is read into the buffer space. The buffer space requirement is $maxTriangle$ columns, each of size $L$ ($L$ is the trajectory database size). Thus, the total buffer space required is $L * maxTriangle$. The choices are thus query dependent. In the implementation, the first $maxTriangle$ trajectories that fill up the $procArray$ are picked.

However, the near triangle inequality is a "weak" version of triangle inequality, as it filters only when trajectories have different lengths (both query and data trajectories). If all the trajectories have the same length, applying near triangle inequality will not remove any false candidates.

A general approach, called Constant Shift Embedding (CSE) [91], is also investigated to see whether it can be used to improve the retrieval efficiency. CSE is proposed to convert a distance function that does not follow triangle inequality to another one that follows it. The idea is briefly described as follows:

Consider a distance function $dist$ that is defined on data space $\mathcal{D}$ and $dist$ does not follow triangle inequality, then there exist three data elements $x, y, z \in \mathcal{D}$, $dist(x, y) + dist(y, z) < dist(x, z)$. $dist$ can be converted to another distance function, $dist'$, by adding a positive value $c$ to each distance value calculated by $dist$. For example, $dist'(x, y) = dist(x, y) + c$. If $c$ is large enough, $dist'(x, y) + dist'(y, z) \geq dist'(x, z)$ may hold (which equals $dist(x, y) + dist(y, z) + c \geq dist(x, z)$). Thus, triangle inequity holds on $dist'$.

However, CSE approach is not applied to EDR due to the following reasons:

1. All the pairwise distances in the data set have to be investigated to find $c$. In [91], the minimum eigenvalue of pairwise distance matrix is selected. This minimum eigenvalue is tested with some trajectory data sets, such as ASL, Kungfu, and Slip (the details of these data sets are explained in the experiment section). The embedded results of CSE show that very few distance computation can be saved. An analysis of the converted pairwise distance matrix showed that this minimum eigenvalue is

quite large and makes the pruning by triangle inequality meaningless, since the lower bounding of $dist(x, y)$, $(dist(x, z) - dist(y, z) - c)$ is too small to prune anything. Reducing the minimum eigenvalue may increase pruning ability, but it may cause some distances not to follow triangle inequality and introduce false dismissals.

2. Usually, for similarity search, query data are not inside the database. The constant value $c$ derived by only investigating the data in the database may not be large enough to make the distances between query data to any data in the database follow triangle inequality. Using the CSE approach to compute the distances between query data and all the data in the database does not make sense, since these distance computations are exactly what the pruning techniques try to save in answering a query.

### 5.3.3 Pruning by Histograms

Embedding methods have been used to improve the efficiency of $k$-NN queries on strings under edit distance. The basic idea is to embed strings into a vector space and define a distance function in the embedded vector space. To avoid false dismissals, the distance in the embedded space is required to be the lower bound of the edit distance on strings. A number of embedding methods have been proposed for strings [2, 6, 27, 49]; however, only two of these [49, 2] avoid introducing false dismissals. Both of these take a similar approach in that they transform strings into a multidimensional integer space by mapping strings to their *frequency vectors* (FV). A frequency vector of a string over an an alphabet records the frequency of occurrence of each character of the alphabet in that string. It is proven that the *frequency distance* (FD) between the FVs of two strings is the lower bound of the actual edit distance. FD of two points $u$ and $v$ in $s$-dimensional space, $FD(u, v)$, is defined as the minimum number of steps (insertion, deletion, replacement operations) that is required to go from $u$ to $v$ (or equivalently from $v$ to $u$).

In fact, frequency vectors are one-dimensional histograms over strings, where each bin is a character in the alphabet. Therefore, an embedding technique is proposed to transform trajectories into trajectory histograms. Then, these histograms are used to remove false candidates. Two dimensional histograms of trajectories are developed in the following way. Given the maximum ($max_x$) and minimum ($min_x$) $x$ dimension values of trajectories, the

range $[min_x, max_x]$ is divided into $\tau_x$ disjoint equal size subranges and the size of each subrange is $\epsilon$. The same procedure is carried on the $y$ dimension to get $\tau_y$ disjoint equal size subranges. Any distinct combination of these two subranges is called a *histogram bin*. Given a trajectory $S$, its histogram $H_S$ can be computed by counting the number of elements $h_i$ $(1 \leq i \leq \tau_x * \tau_y)$ that are located in each histogram bin $i$: $H = [h_1, \ldots, h_{\tau_x * \tau_y}]$.

Based on this embedding, a histogram distance $DH$ on histograms of trajectories is defined as follows:

**Definition 5.4** *Let $H_R$ and $H_S$ be histograms of two trajectories $R$ and $S$, respectively. The histogram distance, $HD(H_R, H_S)$, is defined as the minimum number of steps required to go from $H_R$ to $H_S$ (or equivalently from $H_S$ to $H_R$) by moving to a neighbour point at each step. $H_R$ and $H_S$ are neighbours if $R$ can be obtained from $S$ (or vice versa) using a single edit operation of EDR.*

Since EDR is defined based on a matching threshold $\epsilon$, neighbour points of histograms are different from those of FVs. For example, two FVs $v_1 = \langle 1, 0 \rangle$ and $v_2 = \langle 0, 1 \rangle$ are neighbours according to the definition of edit distance, and frequency distance between them is 1. Given two one-dimensional trajectories $R = [(t_1, 0.9)]$ and $S = [(t_1, 1.2)]$ and $\epsilon = 1$, the histograms of $R$ and $S$ are exactly the same vectors as $v_1$ and $v_2$. However, they are not neighbours according to Definition 5.4. This is because the transformation from $R$ to $S$ does not need any edit operations; 0.9 and 1.2 are treated as matched elements under EDR. As a consequence, the corresponding histogram distance is also 0. Therefore, to overcome the problem that elements located near the boundary of two different histogram bins may match each other under EDR, the elements from two different histogram bins are treated as if they were from the same bin if these two histogram bins *approximately match*.

**Definition 5.5** *Given two histograms $H_R$ and $H_S$, histogram bin $h_{R,i}$ of $H_R$ approximately matches histogram bin $h_{S,j}$ of $H_S$, if $h_{R,i}$ and $h_{S,j}$ are the same bin or they are adjacent bins.*

For example, given two histograms of $H_R = [h_{R,1}, h_{R,2}, h_{R,3}]$ and $H_S = [h_{S,1}, h_{S,2}, h_{S,3}]$ of two one-dimensional trajectory data, $h_{R,1}$ approximately matches $h_{S,1}$ as well as $h_{S,2}$,

and $h_{R,2}$ approximately matches $h_{S,1}$, $h_{S,2}$, and $h_{S,3}$. Procedure CompHisDist shown in Algorithm 5.4 presents the steps for computing HD between two histograms $H_R$ and $H_S$. In procedure CompHisDist, the first for loop is used to compute the difference between two histograms, the second loop (line 6-10) is used to find the elements in the histogram bins that approximately match each other, and the third loop is used to count the minimum number of steps that is required to transfer $H_R$ to $H_S$.

**Theorem 5.6** *Let $R$ and $S$ be two trajectories, $\epsilon$ be a matching threshold and $H_R$ and $H_S$ be the histograms of $R$ and $S$, respectively. Then, $HD(H_R, H_S) \leq EDR(R, S)$.*

**Proof.** Any single edit operation on $R$ to convert it to $S$ corresponds to a change in its histogram $H_R$: deletion of one element from $R$ corresponds to subtracting one from the value of some histogram bin; insertion of one element to $R$ corresponds to adding one to the value of some histogram bin; replacement of an element in $R$ corresponds to adding one in some bin and subtracting one in the other bin. Furthermore, each movement step that is used to transform $H_R$ to $H_S$ moves $H_R$ to its neighbour point, and the change of $H_R$ made by each movement step is same as that caused by a single edit operation. Thus, the number of steps used in the transformation of the histograms is the lower bound of the number of edit operations in EDR. □

With Theorem 5.6, to answer $k$-NN queries, HDs can be computed to prune out false candidates from the trajectory database. Most importantly, the computation cost of HD is almost linear. The nested for loops in CompHisDis may suggest that the computation time of HD is non-linear. However, as the number of bins that approximately match each other in the histogram space is limited to a small constant, the computation time of CompHisDist is still linear. The algorithm that uses HD as lower bound distance to prune false candidates can be achieved by modifying procedure NearTrianglePruning (Algorithm 5.3) as follows:

- Delete lines 2 to 6 and line 10, as it is no longer necessary to keep the array $procArray$.
- Change line 1 to: $maxPruneDist = HD(H_Q, H_S)$.

**Algorithm 5.4** Procedure CompHisDist($H_R$, $H_S$)

/*algorithm for Computing Histogram Distances*/

**Input:** $H_R$ and $H_S \equiv$ histograms of trajectories

**Output:** $result \equiv$ histogram distance

1: $posDist = 0, negDist = 0$
2: **for** each histogram bin of $H_R$ **do**
3: $H_{R,i} = H_{R,i} - H_{S,i}$
4: **end for**
5: **for** each histogram bin of $H_R$ **do**
6: **for** each approximately match bin $H_{R,j}$ of $H_{R,i}$ **do**
7: **if** $H_{R,j} * H_{R,i} < 0$ **then** {two values have opposite signs}
8: reduce the values of $H_{R,j}$ or $H_{R,i}$
{/* elements in the approximately match bins should treated as from the same bin */}
9: **end if**
10: **end for**
11: **end for**
12: **for** each histogram bin of $H_R$ **do**
13: **if** $H_{S,i} > 0$ **then**
14: $posDist+ = H_{R,i}$
15: **else**
16: $negDist+ = 0 - H_{R,i}$
17: **end if**
18: **end for**
19: return $result = max(posDist, negDist)$

This modified algorithm searches trajectory histograms one after another and it does not utilize previously computed histogram distances. This algorithms is named *Histogram SEquential scan (HSE)*. Another algorithm, *Histogram SoRted scan (HSR)*, is also proposed to answer $k$-NN queries using trajectory histograms. HSR first computes all the histogram distances between query and data trajectories. Then it sorts the histogram distances in ascending order. Finally, the trajectories are accessed according to the histogram distance order and EDR is computed if necessary. It is obvious that the pruning power of HSR is better than that of HSE since the trajectories are accessed in an ascending order of lower bound distances (histogram distances) of EDR. However, to achieve this improvement, HSR requires an additional sorting step. Their relative efficiency is compared in the following section.

When histograms are constructed, $\epsilon$ is used as the histogram bin size. If the matching threshold $\epsilon$ is small, trajectory histograms will have a lot of bins. The storage and computation cost will increase as a consequence. To address this issue, two solutions are proposed to reduce the number of bins:

1. Create histograms with a larger histogram bin size, which is $\delta$ ($\delta \geq 2$) times the matching threshold $\epsilon$.

2. Create individual histograms for each one-dimensional data sequence of trajectories using $\epsilon$ as the histogram bin size.

Assume that the number of bins for trajectory histograms with bin size $\epsilon$ is $\tau_x * \tau_y$, where $\tau_x$, $\tau_y$ are the number of histogram bins in each dimension. The two methods above reduce the number of bins by a factor of $\delta * \delta$ and $\frac{\tau_x * \tau_y}{\tau_x + \tau_y}$, respectively. Most importantly, they do not introduce false dismissals based on the following.

**Theorem 5.7** *Given two trajectories $R$, $S$, and a matching threshold $\epsilon$, $EDR_{\delta*\epsilon}(R, S) \leq EDR_{\epsilon}(R, S)$ holds where $\delta \geq 2$ and $EDR_{\delta*\epsilon}$ stands for EDR computed with $\delta * \epsilon$ as a matching threshold.*

**Proof.** It is clear that if an element of $S$ matches an element of $R$ within $\epsilon$, they must match each other within $\delta * \epsilon$. Thus, the number of matching elements will not be reduced if the matching threshold is increased from $\epsilon$ to $\delta * \epsilon$. As a consequence, the number of

edit operations needed to convert $R$ to $S$ within $\delta * \epsilon$ is not higher than that of conversion within $\epsilon$. $\square$.

**Theorem 5.8** *Given two trajectories $R$ and $S$, and a matching threshold $\epsilon$, $EDR_\epsilon^{x(y)}(R,S) \leq EDR_\epsilon(R,S)$ holds where $EDR_\epsilon^{x(y)}(R,S)$ is EDR on projected one-dimensional data sequence ($x$ or $y$) of trajectories.*

**Proof.** Similar to the proof of Theorem 5.7. If an element of $S$ matches an element of $R$ within $\epsilon$, the individual values of each dimension between two elements must match each other within threshold $\epsilon$. Thus, the number of matching elements is not reduced for each single dimensional data sequence compared to that of whole trajectories. $\square$.

**Corollary 5.1** *Let $R$ and $S$ be two trajectories, $H_{(R,\delta*\epsilon)}$ and $H_{(S,\delta*\epsilon)}$ be the histograms created with bin size $\delta * \epsilon$, and $H^x_{(R,\epsilon)}$ and $H^x_{(S,\epsilon)}$ be the histograms on $x$ dimensional data sequence. $HD(H_{(R,\delta*\epsilon)}, H_{(S,\delta*\epsilon)}) \leq EDR_\epsilon(R,S)$ and $HD(H^x_{(R,\epsilon)}, H^x_{(S,\epsilon)}) \leq EDR_\epsilon(R,S)$ hold.*

Using Corollary 5.1, either histograms with larger bin size or histograms on projected one-dimensional data sequence of trajectories can be used to prune false candidates.

### 5.3.4 Combination of three pruning methods

Because the three pruning techniques introduced earlier are orthogonal, it is possible to combine three methods – use one pruning method to save the computation of the true distance $EDR(Q,S)$ after another. An example of combination is shown in Algorithm 5.5. In the example procedure EDRCombine$K$-NN, histogram pruning is applied first, then, mean value Qgram filters are applied. Finally, the procedure NearTrianglePruning is invoked to remove more false candidates based on computed real EDR distances. In the experiments, other combinations are also tested, such as applying mean value Q-gram filtering before trajectory histograms and near triangle inequality pruning or applying near triangle inequality pruning before the other two. The results are discussed in the next section.

**Algorithm 5.5** Procedure EDRCombine$K$-NN($Q, procArray, pmatrix, result, Q, k$)

/*combination algorithm for applying histogram pruning followed by mean value Qgram and near triangle inequality pruning*/

**Input:** $S \equiv$ the current trajectory; $procArray \equiv$ the array of trajectories with computed EDR to $Q$; $pmatrix \equiv$ precomputed pairwise distance matrix;

**Output:** $result \equiv$ the $k$-NN

1: pick the first $k$ trajectories and initialize $result$ with the $k$ true (sorted) $EDR$ distances
2: $bestSoFar = result[k].dist$ and $i = k + 1$
3: **for** each trajectory $S_i$ in the database **do**
4: $maxPruneDist = HD(H_Q, H_{S_i})$ {/* trajectory histogram distance*/}
5: **if** $maxPruneDist \leq bestSoFar$ **then** {/* cannot be pruned by trajectory histogram*/}
6: $v_i$ =merge-join(mean-value-pairs($Q$), mean-value-pairs($S_i$))
{/* applying merge join to compute common mean value pairs */}
7: **if** $((v_i) \geq (max(|Q|, |S_i|) - (bestSoFar + 1) * size(Q - gram)))$ **then** {/* cannot be pruned by mean value Qgrams*/}
8: invoke procedure NearTrianglePruning() in Algorithm 5.3
9: **end if**
10: **end if**
11: **end for**

## 5.4 Experimental Evaluation

Experimental results on the efficiency of each pruning technique as well as the combination of methods are presented here, both speedup ratio and pruning power are measured. Speedup ratio is defined as follows:

$$speedup = \frac{\text{the total time required for a sequential scan to answer a query}}{\text{the total time needed with a pruning technique}} \tag{5.3}$$

where "total time" includes both CPU and I/O cost, which is used to answer a $k$-NN query.

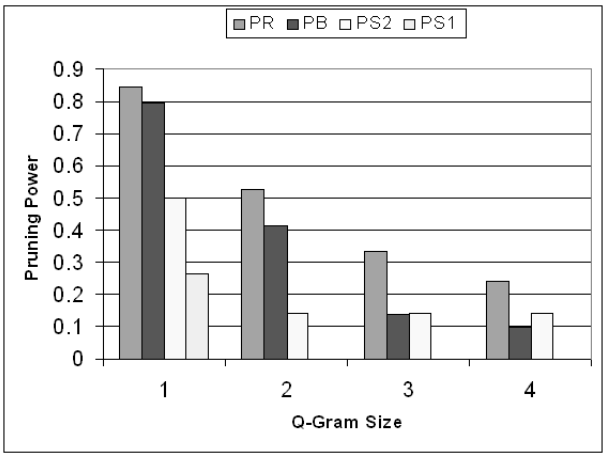
Similar to the tests in Chapter 4, various values of $k$ (1, 3, 5, 10, and 20) are tested. The results of $k = 20$ are reported because the performance differences of the pruning methods can be clearly observed. For all the $k$-NN queries, the matching threshold of a data set is set to $std_{max}/4$, where $std_{max}$ is the maximum standard deviation of trajectories in the data set. The matching threshold is set according to the previous efficacy test on EDR. All experiments were run on a Sun-Blade-1000 workstation with 1GB memory under Solaris 2.8.

### 5.4.1 Efficiency of Pruning by Q-gram Mean Values

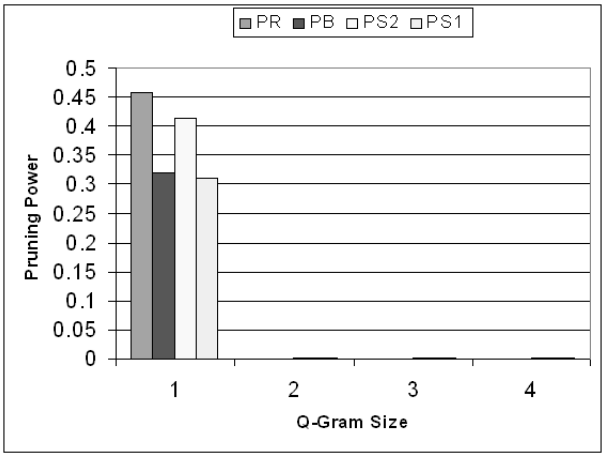
In this experiment, ASL data set from UCI KDD archive[1], Kungfu and Slip data sets from [19] are used. The ASL data set contains 710 trajectories with lengths varying from 60 to 140. The Kungfu data set contains 495 trajectories that record positions of body joints of a person playing Kungfu and the length of each trajectory is 640. The Slip data set also has 495 trajectories which record positions of body joints of a person slipping down and trying to stand up and the length of each trajectory is 400. This experiment is designed to compare the pruning efficiency of (1) Q-grams with different sizes, (2) indexed Q-grams versus merge join, and (3) two dimensional Q-grams versus one-dimensional Q-grams.

Figure 5.2 shows the pruning power comparison of 4 different implementations of pruning by mean values of Q-grams with various sizes (from 1 to 4): pruning with a R-tree on two dimensional Q-grams (PR), pruning with B+-tree on one-dimensional Q-grams (PB), pruning with merge join on sorted two dimensional Q-grams (PS2), and pruning with merge join on sorted one-dimensional Q-grams (PS1).

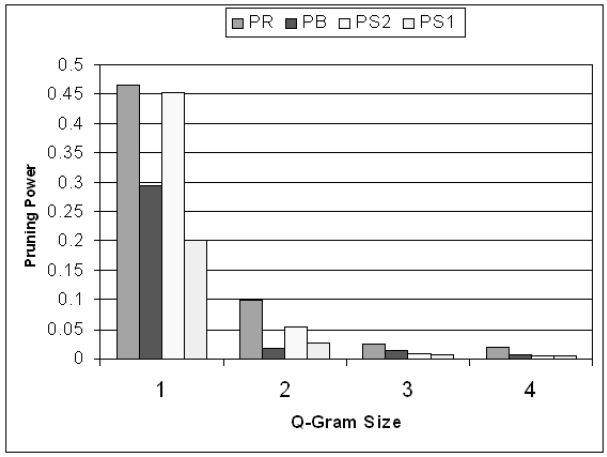
[1]This data set combines all the trajectories of ten word classes into one data set.




(a) ASL data




(b) Slip data

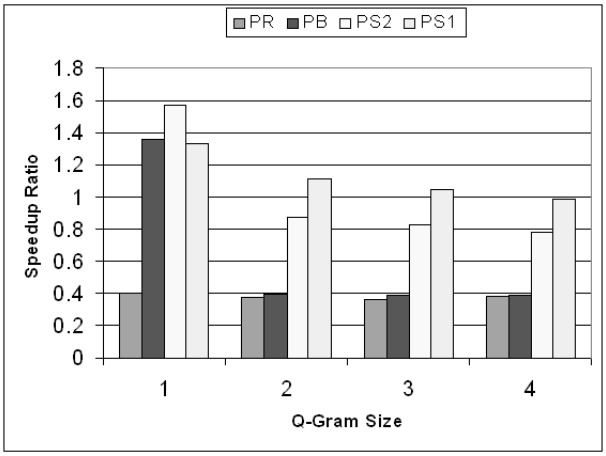



(c) Kungfu data

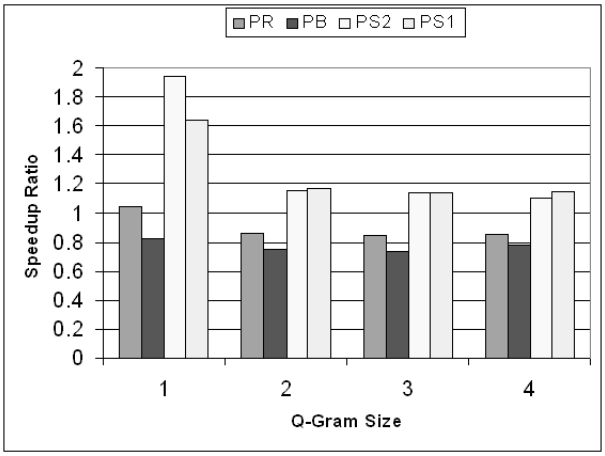
Figure 5.2: Pruning power comparisons of mean value Q-grams on three data sets

The results of three data sets show that, in terms of pruning power, PR is better than PB and PS2 is better than PS1, which confirms the previous claim that two dimensional Q-grams perform better than one-dimensional Q-grams. The results also show that with increasing Q-grams size, the pruning power drops; in particular for Slip data, the pruning power drops to 0 when the Q-gram size is greater than 1. Thus, Q-grams of size 1 are the most effective ones in removing false candidates. Furthermore, PR is always better than

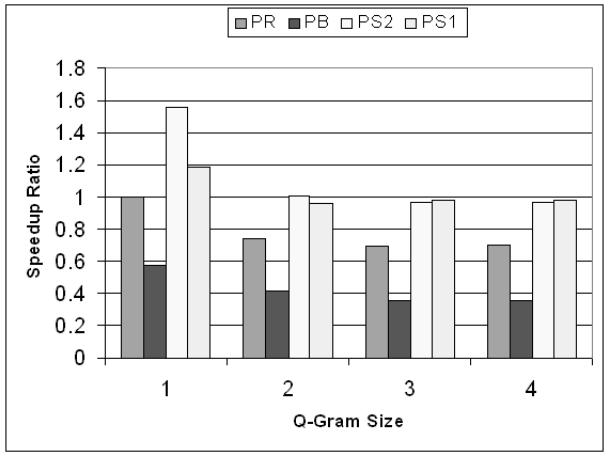
PS2 which indicates pruning with indexed Q-grams is better than pruning with merge join. However, the conclusion cannot be drawn that PR on Q-grams of size 1 is the best pruning method based on the pruning power test, since the higher pruning power may be the results of higher space and computation cost. Thus, the speedup ratios of four methods on three data sets are also compared. The results are shown in Figure 5.3. The results in Figure 5.3




(a) ASL data




(b) Slip data




(c) Kungfu data

Figure 5.3: Speedup ratio comparisons of mean value Q-grams on three data sets

seem to contradict the pruning power results in Figure 5.2, since the speedup ratios of PR

and PB are less than those of PS2 and PS1. This is due to the additional index search time and the time for counting the number of matching Q-grams. Even though PR and PB can remove more false candidates, this does not compensate the cost of index traversal. This also explains why PR or PB performs worse than a sequential scan (speedup ratio is less than 1) in some cases shown in Figure 5.3. The speedup ratio results also show that PR and PB perform worse on data sets with shorter length, such as ASL data (Figure 5.3(a)), than they do on trajectories with longer length, such as Kungfu data (Figure 5.3(c)). The reason is that the time required to compute EDR of short trajectories is less than that of longer trajectories; as a consequence, the total time saved from computing EDR of short trajectories is less than that of longer trajectories. The results also show that PS2 needs less time than PS1 with Q-grams of size 1, which reflects the fact that the total time spent on finding common Q-grams of two trajectories can be compensated by the time saved from removing more false candidates. However, as shown in Figure 5.3, when PS2 prunes little, such as Q-grams of size greater than 1, it performs worse than PS1 because the saved time cannot cover the cost of finding common Q-grams on trajectories. Two test results show that PS2 on Q-gram of size 1 is the best method to remove false candidates with mean value Q-grams.

### 5.4.2 Efficiency of Pruning by Near Triangle Inequality

If trajectories in a database have the same size, the near triangle inequality cannot remove false candidates. Kungfu and Slip data sets contain trajectories of the same length and are not used in this experiment. Instead, two random walk data sets are generated with different lengths (from 30 to 256), the lengths of one random walk data set follow uninform distribution (RandU) and the other one has normal distribution (RandN). There are 1,000 trajectories in each random walk data set. The pruning power and speedup ratio results for these and ASL data sets are shown in Table 5.3. The results show that both the pruning power and the speedup ratio of near triangle inequality is pretty low compared to the results of mean value Q-grams. This is because the factor $|S|$ introduced in near triangle inequality is too big, which reduces its pruning power. Finding a smaller suitable value is left as future work. The results also show that near triangle inequality works better on the data set whose lengths follow a uniform distribution (trajectory lengths of ASL data follow

| | ASL | RandN | RandU |
|---|---|---|---|
| Pruning Power | 0.09 | 0.07 | 0.26 |
| Speedup Ratio | 1.10 | 1.07 | 1.31 |

Table 5.3: Test results of near triangle inequality

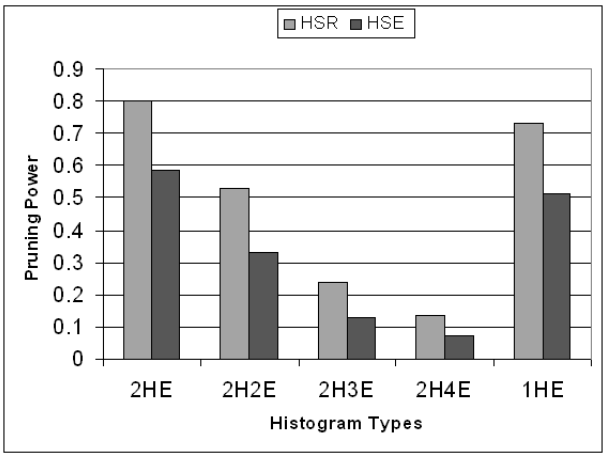
a near normal distribution), confirming the claim that it is more effective for trajectories of variable lengths.

### 5.4.3 Efficiency of Pruning by Histograms

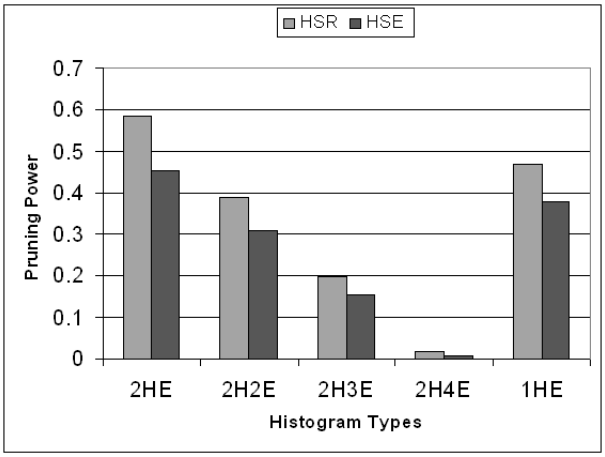
In this experiment, the efficiencies of different types of histograms, trajectory histograms of four different bin sizes ($\epsilon$, $2\epsilon$, $3\epsilon$, $4\epsilon$) and one-dimensional data sequence histograms of bin size $\epsilon$, together with two scan methods, histogram sorted scan (HSR) and histogram sequential scan (HSE) are tested. The same three data sets used in Section 5.4.1 are tested here. The test results of pruning power and speedup ratio are shown in Figures 5.4 and 5.5, respectively. In both figures, $1HE$ stands for one-dimensional data sequence histograms with bin sizes $\epsilon$. $2HE$, $2H2E$, $2H3E$, and $2H4E$ mean trajectory histograms with bin size $\epsilon$, $2\epsilon$, $3\epsilon$, and $4\epsilon$, respectively.

The pruning power results show that trajectory histograms with bin size $\epsilon$ have the highest pruning power on three data sets. A closer examination of three sub-figures of Figure 5.4 shows that pruning power of one-dimensional data sequence histograms is better than that of trajectory histograms with larger bin sizes. Thus, in terms of efficiency of two methods used to reduce the number of histogram bins, creating one-dimensional sequence histograms with the same bin size $\epsilon$ is better than enlarging the bin size $\epsilon$ of trajectory histograms.

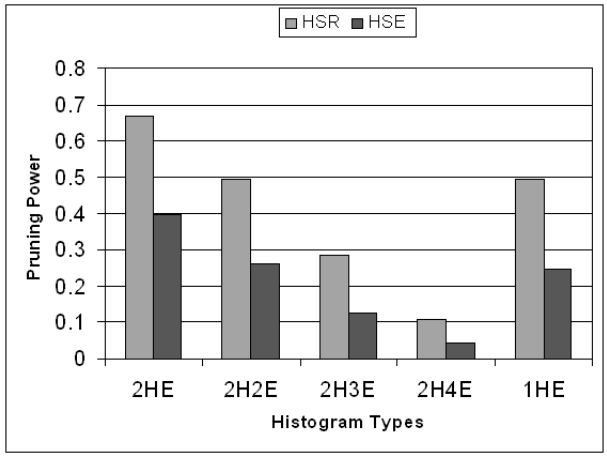
Even though HSR requires an additional sorting step, the results show that HSR beats HSE both in pruning power and speedup ratio tests, which indicates that it is worth sorting. Since the computation cost of histogram distance is linear, nearly all the speedup ratio results match the pruning power test results: that is, the method that has higher pruning power needs less time to answer $k$-NN queries. However, the speedup ratio of one-dimensional data sequence histograms is very close to or even more than that of trajectory




(a) ASL data



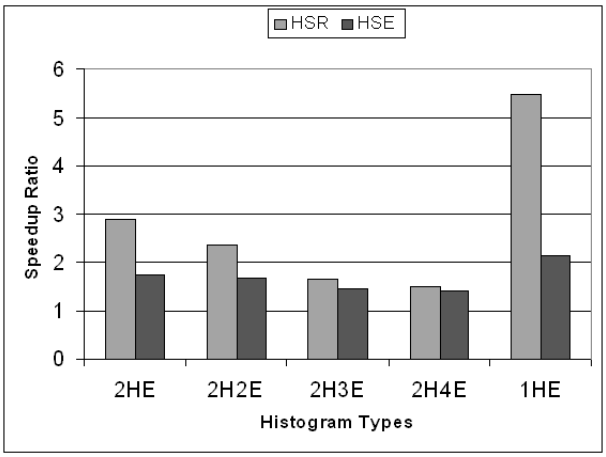

(b) Slip data



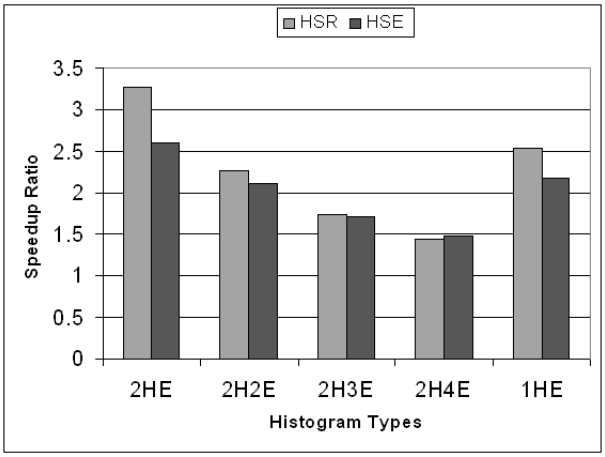

(c) Kungfu data

Figure 5.4: Pruning power comparisons of histograms on three data sets

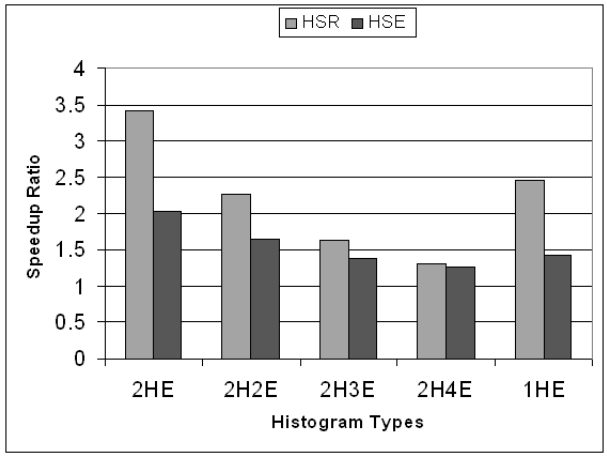
histograms with the same bin size (Figure 5.5(a)). This is because the pruning powers of two types of histograms are very similar and the time saved from computing distances using one-dimensional data sequence histograms is more than the time that is spent on computing the extra number of EDR. Comparing the pruning power and speedup ratio results of mean value Q-grams and histograms (Figures 5.2 vs. 5.4 and Figures 5.3 vs. 5.5) shows that histograms generally perform better than mean value Q-grams on removing




(a) ASL data




(b) Slip data



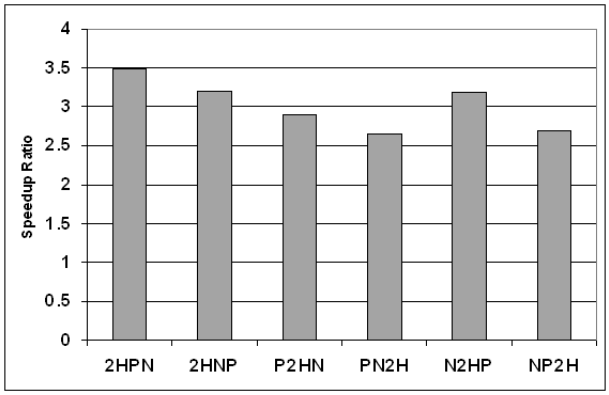

(c) Kungfu data

Figure 5.5: Speedup ratio comparisons of histograms on three data sets

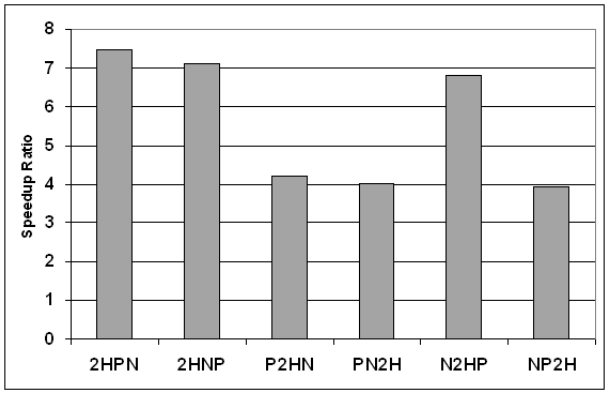
false candidates.

### 5.4.4 Efficiency of Combined Methods

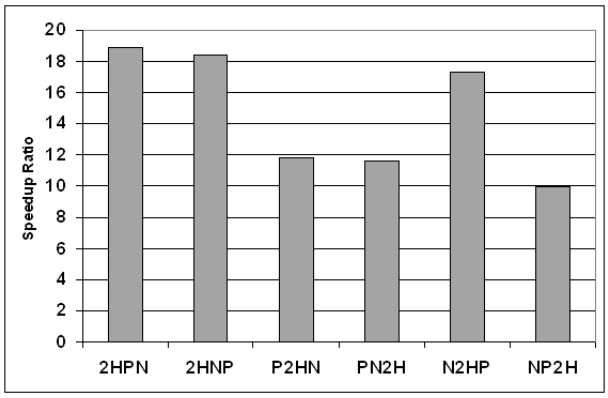
The combination of methods proposed in Section 5.3.4 are tested on NHL data [19], a mixed data set [107] and a randomwalk trajectory data set [23, 52]. The NHL data consists of




(a) NHL data



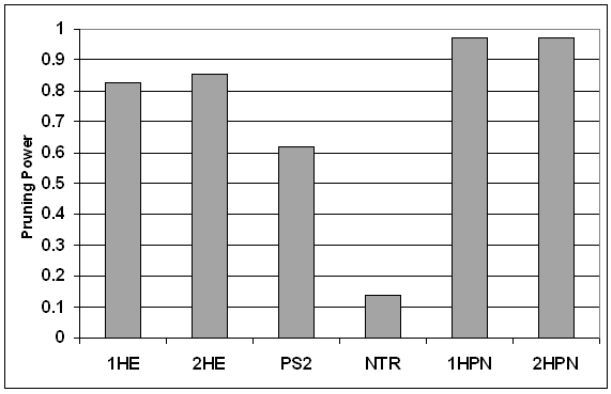

(b) Mixed data



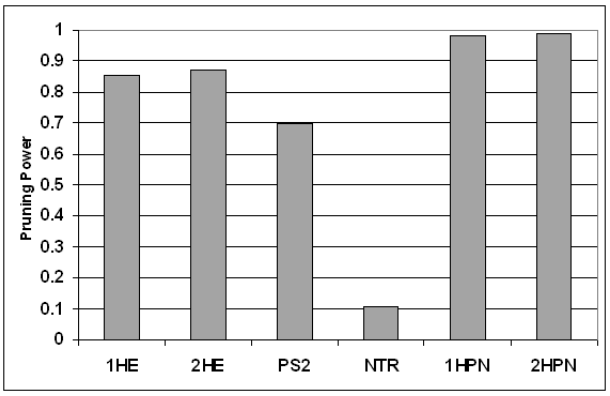

(c) Randomwalk data

Figure 5.6: Speedup ratio comparison of different applying orders of three pruning methods

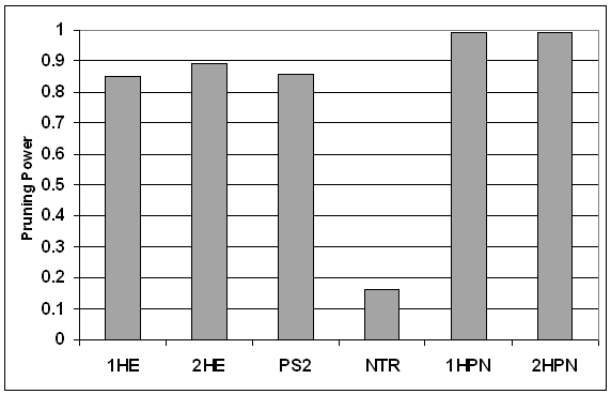
5000 two dimensional trajectories of National Hockey League players and their trajectory lengths vary from 30 to 256. The mixed data set contains 32768 trajectories whose lengths vary from 60 to 2000. The randomwalk data set contains 100,000 two dimensional trajectories and their lengths vary from 30 to 1024. Based on the results of above experiments, for trajectory histograms, HRE is selected rather than HSE, even though it requires an additional sorting step, since HRE outperforms HSE both in pruning power and speedup ratio. The PS2 method on mean value Q-grams is selected as the Q-grams filter. PR is not selected because it can only archive higher pruning power with very expensive search




(a) NHL data



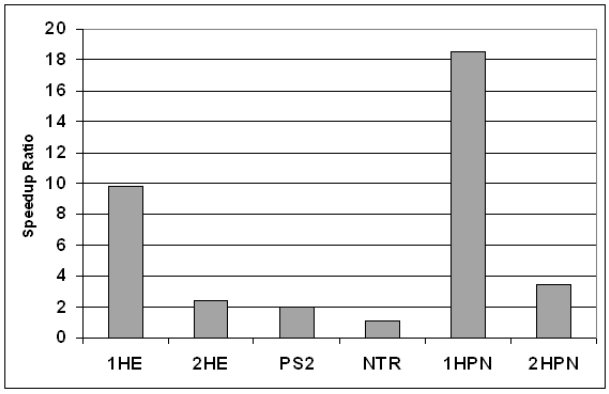

(b) Mixed data



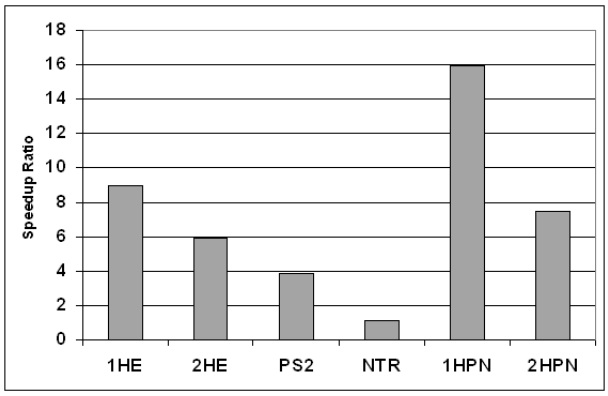

(c) Randomwalk data

Figure 5.7: Pruning power comparisons of combined methods on three large data sets

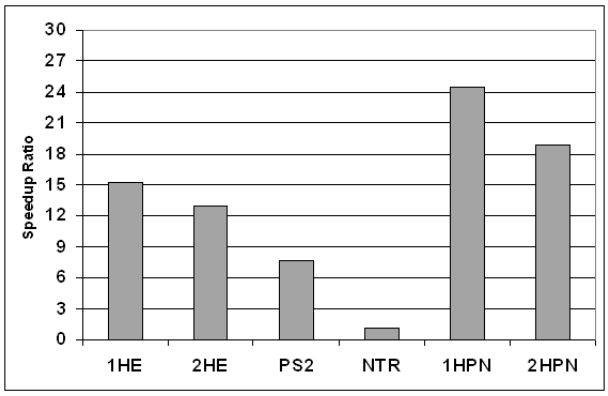
cost. In the experiment, all six possible ordering combinations are tested. The pruning power results show that the six combinations achieve the same pruning power on three data sets, which confirms the claim that the three pruning methods are independent of each other. With respect to speedup ratio, because of the differences in pruning power and computation cost of each pruning method, the application order affects the speedup ratio as shown in Figure 5.6. $2HPN$ means applying trajectory histograms with bin size $\epsilon$ pruning first, then Q-grams filtering, and at last, near triangle inequality pruning; the other symbols represent the rest of application orders. As shown in Figure 5.6, the example combination method listed in Algorithm 5.5 (applying histogram pruning first, then mean




(a) NHL data




(b) Mixed data




(c) Randomwalk data

Figure 5.8: Speedup ratio comparisons of combined methods on three large data sets

value Q-gram filtering, finally, near triangle inequality pruning) achieves the best performance in all three data sets. The reason is that applying a pruning method with more pruning power and less expensive computation cost first will cause fewer false candidates left for subsequent pruning, resulting in a decrease in the time cost of subsequent pruning.

Finally, the performance of different types of histograms in the combination method is tested. Two types of histograms are used: trajectory histograms and one-dimensional data sequence histograms with the same bin size $\epsilon$. Based on the performance results of above experiment, the combination method shown in Algorithm 5.5 is selected for comparison. The comparison results are shown in Figures 5.7 and 5.8. In both figures, $NTR$

stands for pruning by near triangle inequality, $1HPN$ stands for the combination with one-dimensional data sequence histograms with bin size $\epsilon$, merge join on two dimensional Q-grams, and near triangle inequality.

The results show that combined methods using one-dimensional data sequence histograms achieve the best performance. Both the pruning power and the speedup ratio are increased. The speedup ratio is nearly twice of using histogram pruning only, five times that of mean value Q-grams only, and twenty times that of near triangle inequality. The combined method with trajectory histograms also beats the method using trajectory histograms only. However, because of the large number of bins of trajectory histograms, its performance improvement is diminished by the time spent on computing the histogram distances, especially for large databases.

## 5.5 Conclusions

For similarity-based retrieval of time series and trajectory data contain local time shifting and noise, the existing distance functions are either sensitive to possible noise of trajectories, such as Euclidean distance, DTW, and ERP, or they can handle noise with less accuracy, such as LCSS. Therefore, in this chapter, a new distance function, *Edit distance on Real sequence* (EDR), is proposed to measure the similarity between data. The experimental results over benchmark data show that EDR performs as well as DTW, ERP, and LCSS when data contain little or no noise, and it is more robust than DTW and ERP, and more accurate than LCSS in noisy conditions.

In order to improve the retrieval efficiency of EDR, three pruning techniques are proposed: mean value Q-grams, near triangle inequality, and histograms. All three techniques are proven not to introduce false dismissals. Variant of mean value Q-grams and trajectory histograms are also developed, such as projected one-dimensional Q-grams and one dimensional sequence histograms. The efficiency of three pruning techniques and their variations are tested by extensive experimental studies. The results show that all three pruning techniques can greatly improve the retrieval performance compared to linear scan. Most importantly, the combination of three pruning methods can deliver superior retrieval efficiency.

# Chapter 6

# Conclusions and Future Work

## 6.1 Summary and Contributions

In this dissertation, techniques for similarity-based retrieval over time series and trajectory data are studied. Based on different applications and user requirements, various solutions are proposed.

The first issue that is studied is similarity-based retrieval for applications that need answers to both pattern existence and shape match queries. In these types of applications, users may be initially interested in retrieving all the time series data that have some specific patterns (pattern existence queries), followed by the retrieval of all the time series that are similar to an interesting time series that is found among the previous retrieval results (shape match queries). Previous approaches focus either on answering pattern existence queries or shape match queries, but not both. In order to address this issue, a novel representation of time series data, called *multi-scale time series histograms*, is proposed for iterative search over time series data for both types of queries. In addition to handling both query types, the novelties of multi-scale time series histograms are the following [24]:

- The cumulative histogram distance is used to measure the similarity between two time series data, which reduces the boundary effects introduced by value space quantization and overcomes the shortcomings of overestimating (such as $L_1$-norm and $L_2$-norm) or underestimating (such as weighted Euclidean) the distance.

- To answer pattern existence queries, multi-scale time histograms are invariant to time shifting and scaling, amplitude shifting and scaling. Moreover, they can reduce the effect introduced by noise.

- To answer shape match queries, multi-scale time histograms offer users flexibility to query the time series data at different accuracy levels. Averages and lower scale histograms can be used to prune the false candidates from the database before querying over higher scale histograms.

The thesis further studies effective and efficient handling of shape match queries focusing on distance function and retrieval techniques. This is a topic that has been extensively studied with many proposed approaches. However only two distance functions, Dynamic Time Warping (DTW) and Longest Common Subsequences (LCSS), can be used to handle local time shifting which is essential for time series or trajectory data similarity matching. DTW and LCSS do not follow triangle inequality, however, making it hard to improve the retrieval efficiency by traditional distance-based indexing structures.

The second contribution of this thesis is the introducing a new metric distance function, called *Edit Distance with Real Penalty* (ERP), which can be used to to handle local time shifting. ERP can be viewed as a variant of $L_1$-norm, except that it can support local time shifting. It can also be viewed as a variant of Edit Distance, except that it is a metric distance function. The benchmark results show that ERP is as natural as DTW and LCSS for time series data. A number of techniques are proposed to improve the retrieval efficiency of ERP [23].

- The previously proposed lower bound distances for DTW are modified and proved to be lower bound of ERP as well.

- A new lower bound distance for ERP is also proposed. Compared to the modified DTW lower bounds, the new one is more effective in removing false candidates, and most important of all, it can be indexed by a simple $B^+$-tree.

In terms of indexing time series or trajectory data, GEMINI framework is often used regardless of whether or not the underlying distance function is a metric distance function. The key is the use of a lower bound on the true distance to guarantee no false dismissals

when the index is used as a filter. However, when a distance function is a metric distance function, not only can the lower bound distance be used to remove false candidates, but also the triangle inequality property of a metric distance function can be used to reduce the search space. Thus, the third contribution of this thesis is a $k$-nearest neighbor ($k$-NN) algorithm that applies both lower bounding and the triangle inequality [23]. It is an extension of GEMINI framework on indexing metric distances of time series data. Using ERP as an example, the extensive experimental results show that this algorithm indeed gets the best of both pruning paradigms and delivers superb retrieval efficiency.

The third part of this dissertation addresses similarity-based retrieval over time series or trajectory data contain local time shifting and noise. Current distance functions, such as Euclidean distance, DTW, and ERP, are not able to handle noise, or they handle noise by losing much accuracy (e.g. LCSS). Thus, a new distance function, called *Edit Distance on Real Sequence* (EDR), is proposed to measure the similarity of data that contain local time shifting and noise, which is the fourth contribution of this thesis. EDR removes the noise effects by quantizing the distance between a pair of elements to two values, 0 and 1. Seeking the minimum number of edit operations required to change one trajectory to another offers EDR the ability to handle local time shifting. Furthermore, assigning penalties to the unmatched parts solves the inaccuracy problem of LCSS. A set of objective tests on benchmark data show that EDR is more robust than Euclidean distance, DTW, and ERP, and more accurate than LCSS when it is used to measure the similarity between data that contain noise and local time shifting.

EDR does not follow triangle equality and the computation time is quadratic; thus, improving the retrieval efficiency becomes an important challenge. The following pruning techniques are proposed to address this challenge:

- two-dimensional and projected one-dimensional mean value Q-grams;
- near triangle inequality;
- two-dimensional and projected one-dimensional histograms.

All the above three techniques are proven not to introduce false dismissals. Furthermore, a combination mechanism is developed to combine the three pruning techniques and deliver superior pruning power.

## 6.2 Comparison with Related Work

A few approaches [4, 95, 84] have been proposed for pattern existence queries over time series data. The moving direction (the slope between two values) of an user specified interval [84], consecutive values [4], or a segment (obtained by a segmentation algorithm) [95] was represented as a distinct alphabet. Thus, these approaches converted time series data into strings and applied string-matching techniques to find the patterns. However, this conversion has two problems: (1) it is sensitive to noise, and (2) data points located near each other may be assigned to different symbols because of value quantization. Compared to these approaches, multi-scale time series histograms discussed in Chapter 3 are robust to noise and reduce the boundary effects introduced by value space quantization.

Many studies on shape match queries over time series were conducted in the past decade. These work focused on two main aspects: new dimensionality reduction techniques (assuming that the Euclidean distance is the similarity measure), and new approaches for measuring the similarity between two time series, which were reviewed in Chapter 2.

Compared to previous proposed distance functions, such as Euclidean distance [3, 31, 87], DTW [12, 118, 57, 52], and LCSS [14, 28, 108], ERP can handle time series data with different lengths and contain local time shifting. Most importantly, unlike DTW or LCSS, ERP is a metric distance function, which is indexable by distance-based indexing structures.

Most of the previous approaches on indexing time series follow the GEMINI framework. However, if the distance measure is a metric distance function, then existing indexing structures proposed for metric distance functions that use triangle inequality pruning may be applicable. Examples include the MVP-tree [15], the M-tree [26], the Sa-tree [75], and the OMNI-family of access methods [32]. The proposed indexing framework in this thesis combines the pruning power of lower bounding and triangle inequality, which delivers superb retrieval efficiency and dominates all existing strategies.

There are some work have been done on similarity-based search over trajectory data. Lee et al. [61] use the distance between minimum bounding rectangle to compute the distance between two multidimensional trajectories. Even though they can achieve very high recall, the distance function cannot avoid false dismissals. The methods presented in this thesis guarantees that there are no false dismissals. Cai and Ng [19] propose an

effective lower bounding technique for indexing trajectories. However, Euclidean distances are used as the similarity measure [61, 116, 19], and, as argued in Chapter 2, this measure is not robust to noise or time shifting which often appear in trajectory data. Little and Gu [66] use path and speed curves to represent the motion trajectories and measure the distance between two trajectories using DTW. Vlachos et al. [106] also use DTW on rotation invariant representation of trajectories, sequences of angle and arc-length pairs. However, DTW requires continuity along the warping path, which makes it also sensitive to noise. EDR distance function proposed in this thesis for trajectory retreival is robust to noise. Vlachos et al. [108] use LCSS to compare two trajectories with the consideration of spatial space shifting. Compared to LCSS, EDR is not only robust to noise, it also assigns penalties according to the sizes of the gaps in between similar shapes, which makes it more accurate. A cluster-based indexing tree is proposed in [108] to improve the retrieval efficiency using LCSS. The performance of this indexing method depends on the clustering results. However, since LCSS does not follow triangle inequity, it is hard to find good clusters and representing points in the data set [46]. The three pruning techniques proposed for trajectory retrieval do not have this limitation.

## 6.3 Future Work

There are a number of issues that require further investigation. The important ones are the following:

- For multi-scale time series histograms, the scale level needs to be set up by users when they issue a query. Ways to automatically set up the scale value for users is an interesting issue to study.
- The efficacy of ERP on data in other domains, such as speech recognition and musical retrieval, will be studied. Furthermore, the effectiveness of the combination method, which combines the pruning power of lower bounding and triangle inequality, will be studied on other metric distance functions.
- The property of the new lower bound of ERP will be investigated to check the possibility of proposing a new general lower bound for most of the distance functions,

such as Euclidean distance, DTW, ERP, LCSS, and EDR.

- Besides reducing the dimensionality of time series and trajectory data (proposing new lower bounds), designing an effective distance-based indexing structure will definitely help to improve the retrieval efficiency.

- For non-metric distance functions, such as EDR, the possibility of embedding them into a metric space without introducing false dismissals will be studied. Furthermore, directly designing an indexing method for non-metric distances is a very interesting long-term research direction.

The emergence and increasing deployment of sensor network technology has generated interesting problems and has attracted significant attention from both academic and industrial communities. Sensors are small battery-powered, wireless devices which are equipped with limited processing power and memory. The management of data generated by sensors is studied in the database community under the name of "stream data management" [8, 36]. The data collected by a sensor can be treated as a stream of time series data, which raises the possibility that some of the techniques proposed in this thesis may be applicable. However, compared to archived time series data, an important characteristics of stream time series data is that the data are frequently updated. Thus, previous methods that rely on indexing structures to improve the retrieval efficiency do not work in this scenario since the cost of updating indexes will diminish the saving of the query cost. A common approach to address this issue is batch updates, which is not a good solution in this context, since it leads to the loss of accuracy and this loss cannot be bounded. The following possible solutions will be investigated:

- Using previously proposed dimensionality reduction techniques (e.g. Discrete Fourier Transform) to extract a few features from stream time series data and developing an incremental approach to compute new features from previous features.

- Creating an index on extracted features and designing a novel mechanism to reduce the number of updates to the index with a bounded loss of accuracy.

- Offering two types of answers to similarity-based queries over stream time series data, approximate results that are derived from the index and exact results that are derived from the index as well as the stream data stored on disk.

# Chapter 7

# Summary of Notation

$d$ variable over the set of natural numbers

$i$ variable over the set of natural numbers

$j$ variable over the set of natural numbers

$g$ a gap element introduced for computing Edit Distance

$v$ staring shifting position

$m$ number of segments of a sequence

$n$ number of segments of a sequence

$r$ a constant real number for query range

$x$ a data item belongs to a defined data space $\mathcal{D}$

$y$ a data item belongs to a defined data space $\mathcal{D}$

$z$ a data item belongs to a defined data space $\mathcal{D}$

$\tau$ number of histogram bins

$\delta$ scale level of histograms

$\epsilon$ a real value for matching threshold

$t_i$ the $i^{th}$ timestamp of a sequence

$(t_i, q_i)$ the $i^{th}$ element of sequence $Q$

$(t_i, r_i)$ the $i^{th}$ element of sequence $S$

$(t_i, s_i)$ the $i^{th}$ element of sequence $R$

$q_i$ the vector of $i^{th}$ element of sequence $Q$

$r_i$ the vector of $i^{th}$ element of sequence $R$

$s_i$ the vector of $i^{th}$ element of sequence $S$

$q_{i,x}$ the $x$ coordinate of $i^{th}$ element vector of sequence $Q$

$q_{i,y}$ the $y$ coordinate of $i^{th}$ element vector of sequence $Q$

$r_{i,x}$ the $x$ coordinate of $i^{th}$ element vector of sequence $R$

$r_{i,y}$ the $y$ coordinate of $i^{th}$ element vector of sequence $R$

$s_{i,x}$ the $x$ coordinate of $i^{th}$ element vector of sequence $S$

$s_{i,y}$ the $y$ coordinate of $i^{th}$ element vector of sequence $S$

$w_i$ the $i^{th}$ weight of weight vector $W$, where $w_i \geq 0$

$L$ a constant natural number,the size of a database

$M$ a constant natural number,the length of a sequence

$N$ a constant natural number, the length of a sequence

$W$ a vector of weights for different elements of a sequence

$R$ a time series or trajectory $R = [(t_1, r_1), \ldots, (t_N, r_N)]$

$S$ a time series or trajectory $S = [(t_1, s_1), \ldots, (t_N, s_N)]$

$T$ a time series or trajectory $T = [(t_1, t_1), \ldots, (t_N, t_N)]$

$\mathcal{D}$ a set of time series of trajectory data

$|R|$ the length of $R$

$first(R)$ the first element vector of sequence $R$

$last(R)$ the last element vector of sequence $R$

$Rest(R)$ the subsequence of $R$ without the first element: $[(t_2, r_2), \ldots, (t_n, r_n)]$

$\widetilde{R}$ sequence $R$ after aligned with another data sequence

$DLB$ a lower bound of the distance

$H_R$ histograms of $R$

$p(x)$ the probability distribution function of $x$

$max$ a function to get the maximum value of the operand.

$min$ a function to get the minimum value of the operand.

$sum$ a function to get the sum of the operand.

$dist$ a distance function

$wr$ the warping range in dynamic time warping

$iap$ inverse angle probability

$ACHD$ The distance between two average cumulative histograms

$ED$ Edit Distance on strings

$WED$ Weight Euclidean Distance

$CHD$ Cumulative Histogram Distance

$DTW$ Dynamic Time Warping

$LCSS$ Longest Common Subsequence

$ERP$ Edit Distance with Real Penalty

$EDR$ Edit Distance on Real Sequence

# Bibliography


[1] J. I. Agbinya. Discrete wavelet transform techniques in speech processing. In *Proc. 1996 IEEE TENCON - Digital Signal Processing Applications*, pages 514–519, 1996.

[2] S. A. Aghili, D. Agrawal, and A. El Abbadi. Bft: Bit filtration technique for approximate string join in biological databases. In *Proc. Int. Conf. String Processing and Retrieval*, pages 326–340, 2003.

[3] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *Proc. 4th Int. Conf. of Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[4] R. Agrawal, G. Psaila, E. L. Wimmers, and M. Zaït. Querying shapes of histories. In *Proc. 21th Int. Conf. on Very Large Data Bases*, pages 502–514, 1995.

[5] A. N. Akansu and R. A. Haddad. *Multiresolution Signal Decomposition*. Academic Press, 1992.

[6] A. Andoni, M. Deza, A. Gupta, P. Indyk, and S. Raskhodnikova. Lower bounds for embedding edit distance into normed spaces. In *Proc. of the 14th annual ACM-SIAM symposium on Discrete algorithms*, pages 523–526, 2003.

[7] H. Andŕe-Jönsson and D. Z. Badal. Using signature files for querying time-series data. In *Proc. of the 1$^{st}$ European Symp.on Principles of Data Mining and Knowledge Discovery*, pages 211–220, 1997.

[8] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1–16, 2002.

[9] R. A. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proc. of the 5th Annual Symposium on Combinatorial Pattern Matching*, pages 198–212, 1994.

[10] J. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[11] S. Berchtold, D. A. Keim, and H-P. Kriegel. The x-tree: An index structure for high-dimensional data. In *Proc. 22th Int. Conf. on Very Large Data Bases*, pages 28–39, 1996.

[12] D. J. Berndt and J. Clifford. Finding patterns in time series: A dynamic programming approach. In *Advances in Knowledge Discovery and Data Mining*, pages 229–248, 1996.

[13] J. S. Boreczky and L. A. Rowe. Comparison of video shot boundary detection techniques. In *Proc. 8th Int. Symp. on Storage and Retrieval for Image and Video Databases*, pages 170–179, 1996.

[14] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. pages 357–368, 1997.

[15] T. Bozkaya and M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Trans. Database Sys.*, 24(3):361–404, 1999.

[16] S. Brin. Near neighbor search in large metric spaces. In *Proc. 21th Int. Conf. on Very Large Data Bases*, pages 574–584, 1995.

[17] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Commun. ACM*, 16(4):230–236, 1973.

[18] B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14):2357–2366, 2003.

[19] Y. Cai and R. Ng. Indexing spatio-temporal trajectoires with chebyshev polynomials. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 599–610, 2004.

[20] E. Ch'avez, J. Marroqu'in, and R. Baeza-Yates. Spaghettis: an array based algorithm for similarity queries in metric spaces. In *In Proc. Proc. 6th South American Symposium on String Processing and Information Retrieval*, 1999.

[21] E. Chaźez, J. Marroquín, and G. Navarro. Fixed queries array: A fast and economical data structure for proximity searching. *Multimedia Tools Appl.*, 14(2):113–135, 2001.

[22] E. Chaźez, G. Navarro, R. Baeza-Yates, and J. L. Marroquń. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.

[23] L. Chen and R. Ng. On the marriage of edit distance and Lp norms. In *Proc. 30th Int. Conf. on Very Large Data Bases*, pages 792–803, 2004.

[24] L. Chen and M. T. Özsu. Multi-scale histograms for answering queries over time series data. In *Proc. 20th Int. Conf. on Data Engineering*, page 838, 2004.

[25] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *CS Tech. Report. CS-2004-30, School of Computer Science, University of Waterloo.*

[26] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. 23th Int. Conf. on Very Large Data Bases*, pages 426–435, 1997.

[27] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proc. of the 13th annual ACM-SIAM symposium on Discrete algorithms*, pages 667–676, 2002.

[28] G. Das, D. Gunopulos, and H. Mannila. Finding similar time series. In *Proc. 1st European Symp. on Principles of Data Mining and Knowledge Discovery*, pages 88–100, 1997.

[29] I. Daubechies. *Ten Lectures on Wavelets*. SIAM, 1992.

[30] F. Dehne and H. Noltemeier. Voronoi trees and clustering problems. *Information Systems*, 12(2):171–175, 1987.

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

[32] R. F. S. Filho, A. J. M. Traina, C. Traina Jr., and C. Faloutsos. Similarity search without tears: The OMNI family of all-purpose access methods. In *Proc. 17th Int. Conf. on Data Engineering*, pages 623–630, 2001.

[33] P. Geurts. Pattern extraction for time series classification. In *Proc. 5th Int. Conf. on Principles and Practice of Knowledge Discovery in Databases*, pages 115–127, 2001.

[34] J. Gips, M. Betke, and P. Fleming. The camera mouse: Preliminary invertigation of automated visaul tracking for computer access. In *In Proc. Conf. on Rehabilitation Engineering and Assistive Technology Society of North America*, pages 98–100, 2000.

[35] K-S. Goh, B. T. Li, and Ed. Chang. Dyndex: a dynamic and non-metric space indexer. In *Proc. 10th ACM Int. Conf. on Multimedia*, pages 466–475, 2002.

[36] L. Golab and M. T. Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, 2003.

[37] D.Q. Goldin and P.C. Kanellakis. On similarity queries for time series data: Constraint specification and implementation. In *Proc. of the Int. Conf. on Principles and Pratice of Constraint Programming*, pages 23–35, 1995.

[38] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[39] J. Hafner, H.S. Sawhney, W. Equitz, M. Flickner, and W. Niblack. Efficient color histogram indexing for quadratic form distance functions. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17(7):729–739, 1995.

[40] A. Hanjalic, R.L. Lagendijk, and J.Biemond. Improving text retrieval for routing problem using laten semantic indexing. In *Proc. of the 17th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 282–291, 1994.

[41] Y. Hanjalic, J. Kärkkäinen, and H. Toivonen. Mining for similarities in aligned time series using wavelets. *Data Mining and Knowledge Discovery: Theory, Tools, and Technology*, 3695:150–169, 1999.

[42] A. Harr. Theorie der orthogonalen funktionen-systeme. *Mathematische Annalen*, pages 69–331, 1910.

[43] Y. Huang and P. S. Yu. Adaptive query processing for time-series data. In *Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 282–286, 1999.

[44] D. P. Huttenlocher, G. A. Klanderman, and W. A. Rucklidge. Comparing images using the hausdorff distance. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

[45] F. Itakura. Minimum predication residual principle applied to speech recognition. *IEEE Trans. Acoustics Speech Signal Process*, 23:52–72, 1975.

[46] D. W. Jacobs, D. Weinshall, and Y. Gdalyahu. Classification with nonmetric distances: Image retrieval and class representation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(6):583–600, 2000.

[47] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[48] P. Jokinen and E. Ukkonen. Two algorithms for approximate string matching in static texts. In *Proc. Int. Conf. of Mathematical Foudations of Computer Science*, pages 240–248, 1991.

[49] T. Kahveci and A. Singh. Variable length queries for time series data. In *Proc. 17th Int. Conf. on Data Engineering*, pages 273–282, 2001.

[50] K. Kalpakis, D. Gada, and V. Puttagunta. Distance measures for effective clustering of arima time-series. In *Proc. 2001 IEEE Int. Conf. on Data Mining*, pages 273–280, 2001.

[51] K. V. Ravi Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 166–176, 1998.

[52] E. Keogh. Exact indexing of dynamic time warping. In *Proc. 28th Int. Conf. on Very Large Data Bases*, pages 406–417, 2002.

[53] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3):263–286, 2000.

[54] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 151–162, 2001.

[55] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. In *Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 102–111, 2002.

[56] E. Keogh and P. Smyth. A probabilistic approach to fast pattern matching in time series databases. In *Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining*, pages 24–30, 1997.

[57] S Kim, S. Park, and W. Chu. An indexed-based approach for similarity search supporting time warping in large sequence databases. In *Proc. 17th Int. Conf. on Data Engineering*, pages 607–614, 2001.

[58] F. Korn, H. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 289–300, 1997.

[59] K.P.Chan and A.W-C Fu. Efficient time series matching by wavelets. In *Proc. 15th Int. Conf. on Data Engineering*, pages 126–133, 1999.

[60] JB. Kruskall. *Time warp, string edits and macromolecules.* Addsion Wisely Press, 1983.

[61] S.-L. Lee, S.-J. Chun, D.-H. Kim, J.-H. Lee, and C.-W. Chung. Similarity search for multidimensional data sequences. In *Proc. 16th Int. Conf. on Data Engineering*, pages 599–608, 2000.

[62] K. Leung and R. T. Ng. Multistage similarity matching for sub-image queries of arbitrary size. In *Proc. of 4th Working Conf. on Visual Database Systems*, pages 243–264, 1998.

[63] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8):707–710, 1966.

[64] J. Lin, E. Keogh, S. Lonardi, and P. Patel. Finding motifs in time series. In *Proc. 2nd Int. Workshop Temporal Data Mining*, pages 370–377, 2002.

[65] S. Lin, M. T. Özsu, V. Oria, and R. Ng. Multi-precision similarity querying of image databases. In *Proc. 27th Int. Conf. on Very Large Data Bases*, pages 221–230, 2001.

[66] J. L. Little and Z. Gu. Video retrieval by spatial and temporal sturcture of trajectories. In *Proc. 13th Int. Symp. on Storage and Retrieval for Image and Video Databases*, pages 544–553, 2001.

[67] L. Mic'o, J. Oncina, and R. C. Carrasco. A fast branch & bound nearest neighbour classifier in metric spaces. *Pattern Recogn. Lett.*, 17(7):731–739, 1996.

[68] M. L. Mico, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recogn. Lett.*, 15(1):9–17, 1994.

[69] Y-S. Moon, K-Y. Whang, and W-S. Han. General match: a subsequence matching method in time-series databases based on generalized windows. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 382–393, 2002.

[70] Y-S. Moon, K-Y. Whang, and W-K. Loh. Duality-based subsequence matching in time-series databases. In *Proc. 17th Int. Conf. on Data Engineering*, pages 263–272, 2001.

[71] Y. Morinaka, M. Yoshikawa, T. Amagasa, and S. Uemura. The lindex: An indexing structure for efficient subsequence matching in time sequence databases. In *Proc. 5th PacificAisa Conf. on Knowledge Discovery and Data Mining*, pages 51–60, 2001.

[72] C. S. Myers, L. R. Rabiner, and A. E. Rosenberg. Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. *IEEE Trans. Acoustics Speech Signal Process*, 28(6):623–635, 1980.

[73] M.A. Nascimento and J. R. O. Silva. Towards historical R-trees. In *Proc. 1998 ACM Symp. on Applied Computing*, pages 235–240, 1998.

[74] A. Natsev, R. Rastogi, and K. Shim. Walrus: a similarity retrieval algorithm for image databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 395–406, 1999.

[75] G. Navarro. Searching in metric spaces by spatial approximation. *The VLDB Journal*, 11:28–46, 2002.

[76] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin. The QBIC project: querying images by content using color, texture and shape. In *Proc. 5th Int. Symp. on Storage and Retrieval for Image and Video Databases*, pages 173–185, 1993.

[77] A. V. Oppenheim and R. W. Schafer. *Digital Signal Processing.* Prentice-Hall, 1975.

[78] S. Park and W. W. Chu. Similarity-based subsequence search in image sequence databases. *Int. Journal of of Images and Graphics*, (1):31–53, 2003.

[79] T. Pavlidis. Waveform segmentation through functional approximation. *IEEE Trans. Computers*, pages 689–697, 1973.

[80] C.-S. Perng, H. Wang, S.R. Zhang, and D.S. Parker. Landmarks: a new model for similarity-based pattern querying in time series databases. In *Proc. 16th Int. Conf. on Data Engineering*, pages 33–42, 2000.

[81] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches in query processing for moving object trajectories. In *Proceedings of 26th International Conference on Very Large Data Bases*, pages 395–406, 2000.

[82] I. Popivanov and R. J. Miller. Similarity search over time series data using wavelets. In *Proc. 17th Int. Conf. on Data Engineering*, pages 212–221, 2001.

[83] M. H. Protter and C. B. Morrey. *A First Course in Real Analysis*. Sprinter-Verlag, 1977.

[84] Y. Qu, C. Wang, , L. Gao, and X. S. Wang. Supporting movement pattern queries in user-specified scales. *IEEE Trans. Knowledge and Data Eng.*, 15(1):26–42, 2003.

[85] Y. Qu, C. Wang, and X. S. Wang. Supporting fast search in time series for movement patterns in multiple scales. In *Proc. 7th Int. Conf. on Information and Knowledge Management*, pages 251–258, 1998.

[86] L. Rabiner and B. Juang. *Fundamentals of Speech Recognition*. Prentice Hall, 1993.

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

[88] D. Rafiei and A. O. Mendelzon. Efficient retrieval of similar time sequences using DFT. In *Proc. 9th Int. Conf. of Foundations of Data Organization and Algorithms*, 1998.

[89] C. A. Ratanamahatana and E. Keogh. Making time-series classification more accurate using learned constraints. In *Proc. of SIAM International Conference on Data Mining*, pages 11–22, 2004.

[90] R.Bennett. *Spatial Time Series*. Pion Limited, 1979.

[91] V. Roth, J. Laub, J. Buhmann, and K.-R. Muller. Going metric: Denoising pairwise data. In *Proc. of the Int. Conf. on Neural Information Processing Systems*, pages 23–35, 2002.

[92] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. Acoustics Speech Signal Process*, 26:43–49, 1978.

[93] M. Shapiro. The choice of reference points in best-match file searching. *Commun. ACM*, 20(5):339–343, 1977.

[94] D. Shasha and T. L. Wang. New techniques for best-match retrieval. *ACM Trans. Inf. Syst.*, 8(2):140–158, 1990.

[95] H. Shatkay and S.B. Zdonik. Approximate queries and representations for large data sequences. In *Proc. 12th Int. Conf. on Data Engineering*, pages 536–545, 1996.

[96] E. J. Stollnitz, T. D. Derose, and D. H. Salesin. *Wavelets for computer graphics: theory and applications*. Morgan Kaufmann Publishers Inc., 1996.

[97] M. Stricker and M. Orengo. Similarity of color images. In *Proc. 7th Int. Symp. on Storage and Retrieval for Image and Video Databases*, pages 381–392, 1995.

[98] Z. Struzik and A. Siebes. The haar wavelet transform in the time series similarity paradigm. In *In Proc. of 3rd European Conf. on Principles of Data Mining and Knowledge Discovery*, pages 12–22, 1999.

[99] E. Sutinen and J. Tarhio. Filtration with q-samples in approximate string matching. In *Proc. of the 7th Annual Symposium on Combinatorial Pattern Matching*, pages 50–63, 1996.

[100] M. J. Swain and D. H. Ballard. Color indexing. *Int. Journal of Computer Vision*, 7(1):11–32, 1991.

[101] Y.F. Tao and D. Papadias. MV3R-tree: A spatio-temporal access method for timestamp and interval queries. In *Proc. 27th Int. Conf. on Very Large Data Bases*, pages 431–440, Rome, Italy, 2001.

[102] C. Tappert and S. Das. Memory and time improvements in a dynamic programming algorithm for matching specch patterns. *IEEE Trans. Acoustics Speech Signal Process*, 26:583–586, 1978.

[103] Y. Theodoridis, T. Sellis, A. N. Papadopoulos, and Y. Manolopoulos. Specifications for efficient indexing in spatiotemporal databases. In *Proc. of 1998 IEEE Int. Conf. on SSDBM*, pages 242–245, 1998.

[104] A. Tversky. Features of similarity. *Psychological Review*, 84:327–352, 1977.

[105] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letter*, 40(4):175–179, 1991.

[106] M. Vlachos, D. Gunopulos, and G. Das. Rotation invariant distance measures for trajectories. In *Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 707–712, 2004.

[107] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multidimensional time-series with support for multiple distance measures. In *Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 216–225, 2003.

[108] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Proc. 18th Int. Conf. on Data Engineering*, pages 673 – 684, 2002.

[109] M. Vlachos, J. Lin, E. Keogh, and D. Gunopulos. A wavelet-based anytime algorithm for k-means clustering of time series. In *Proc. Workshop on Clustering High Dimensionality Data and Its Applications*, pages 23–30, 2003.

[110] C.Z. Wang and X. Wang. Supporting content-based searches on time series via approximation. In *Proc. 12th Int. Conf. on Scientific and Statistical Database Management*, pages 69–81, 2000.

[111] M. S. Waterman, T. F. Smith, and W. A. Beyer. Some biological sequence metrics. advances in mathematics. *Advances in Mathematics*, (20):367–387, 1976.

[112] R. Weber, H.-J Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th Int. Conf. on Very Large Data Bases*, pages 194–205, 1998.

[113] W. Wei. *Time Series Analysis*. Addison-Wesley, 1994.

[114] Y-L Wu, D. Agrawal, and A. E. Abbadi. A comparison of dft and dwt based similarity search in time-series databases. In *Proc. 9th Int. Conf. on Information and Knowledge Management*, pages 488–495, 2000.

[115] X. Xu, J. Han, and W. Lu. RT-tree: An improved R-tree index structure for spatiotemporal databases. In *Proc. of 4th Int. Symp. on Spatial Data Handling*, pages 1040–1049, 1990.

[116] Y. Yanagisawa, J. Akahani, and T. Satoh. Shape-based similarity query for trajectory of mobile objects. In *Proc. 4th Int. Conf. on Mobile Data Management*, pages 63–77, 2003.

[117] B-K Yi and C. Faloutsos. Fast time sequence indexing for arbitrary Lp norms. In *Proc. 26th Int. Conf. on Very Large Data Bases*, pages 385–394, 2000.

[118] B-K Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proc. 14th Int. Conf. on Data Engineering*, pages 23–27, 1998.

[119] Y. Zhu and D. Shasha. Warping indexes with envelope transforms for query by humming. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 181–192, 2003.