Indexing Spatiotemporal Trajectories with Chebyshev Polynomials

by

Yuhan Cai

B.Sc. (Honours), The University of British Columbia, 2002

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

we accept this thesis as conforming
to the required standard

The University of British Columbia

April 2004

Library Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

YUHAN CAI — Name of Author *(please print)*

22/04/2004 — Date (dd/mm/yyyy)

Title of Thesis: Indexing Spatiotemporal Trajectories with Chebyshev Polynomials

Degree: Master of Science Year: 2004

Department of Computer Science

The University of British Columbia
Vancouver, BC Canada

Abstract

In this thesis, we investigate the subject of indexing large collections of spatiotemporal trajectories for similarity matching. Our proposed technique is to first mitigate the dimensionality curse problem by approximating each trajectory with a low order polynomial-like curve, and then incorporate a multidimensional index into the reduced space of polynomial coefficients. There are many possible ways to choose the polynomial, including Fourier transforms, splines, non-linear regressions, etc. Some of these possibilities have indeed been studied before. We hypothesize that one of the best approaches is the polynomial that minimizes the maximum deviation from the true value, which is called the *minimax* polynomial. Minimax approximation is particularly meaningful for indexing because in a branch-and-bound search (i.e., for finding nearest neighbours), the smaller the maximum deviation, the more pruning opportunities there exist. In general, among all the polynomials of the same degree, the optimal minimax polynomial is very hard to compute. However, it has been shown that the Chebyshev approximation is almost identical to the optimal minimax polynomial, and is easy to compute [32]. Thus, we shall explore how to use the Chebyshev polynomials as a basis for approximating and indexing d-dimensional ($d \geq 1$) trajectories.

The key analytic result of this thesis is the Lower Bounding Lemma. That is, we show that the Euclidean distance between two d-dimensional trajectories is lower bounded by the weighted Euclidean distance between the two vectors of Chebyshev coefficients. This lemma is not trivial to show, and it ensures that indexing with Chebyshev coefficients admits no false negatives. To complement the analytic result, we conduct comprehensive experimental evaluation with real and generated 1-dimensional to 4-dimensional data sets. We compare the proposed scheme with the Adaptive Piecewise Constant Approximation (APCA) scheme. Our preliminary results indicate that in all situations we test, Chebyshev indexing dominates APCA in pruning power, I/O and CPU costs.

Contents

List of Tables

List of Figures

Acknowledgments

First of all, I would like to express my gratitude to my supervisor, Dr. Raymond T. Ng, for his rewarding guidance and constant support throughout the course of my work. It was his supervision and inspiration that have not only enabled me to complete this thesis but also led me to the right direction of research.

Additionally, special thanks go to both Dr. Eamonn Keogh at the University of California, Riverside, for his APCA code and experimental data sets, and Dr. Christos Faloutsos at Carnegie Mellon University for his DR-tree package. I would also like to thank Dr. Jason Harrison and Dr. Michiel van de Panne for providing me with their motion capture data.

I did most of my research in the Database Systems Laboratory, where there was an enlightening and stimulating academic environment. My appreciation is extended to all my colleagues for their friendship and help. I owe many thanks to Dr. Edwin Knorr for his valuable comments and precious suggestions on my thesis.

Moreover, for two years, my studies were mainly funded by scholarships from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Institute of Robotics and Intelligent Systems (IRIS).

Finally, I am indebted to my parents for their love, education and encouragement during the eighteen years of my schooling.

YUHAN CAI

The University of British Columbia

April 2004

To my parents.

Chapter 1

Introduction

A **spatiotemporal trajectory** is a time-stamped sequence of vectors representing space and/or time information. More formally, a **d-dimensional trajectory** is an ordered collection S in the form

$$S = \langle (t_1, \vec{v}_1), (t_2, \vec{v}_2), \ldots, (t_N, \vec{v}_N) \rangle$$

where

- N is the length of the trajectory S.
- $t_1 < t_2 < \cdots < t_N$ are time stamps.
- Each vector $\vec{v}_i$ is of arity d for all $1 \leq i \leq N$.
- Each pair $(t_i, \vec{v}_i)$ records the values of a vector of d scalars at time t_i.

For example, if $d = 1$, then the trajectory is a time series. For a second example, $\vec{v}_i$ may capture the 2-dimensional or 3-dimensional coordinates of a moving object at time t_i, in which case we have a spatiotemporal trajectory. For yet another example, a trajectory may represent the change of the attributes or features of an entity over time.

1.1 Motivation

Time series are ubiquitous in temporal databases, which is a well-established area in database studies [16]. Massive time series or sequence data sets arise naturally in a variety of real world applications, such as medinformatics, meteorology, stock markets, and image/video databases. For example, doctors monitor the health conditions of their patients by keeping track of their body temperatures, geologists record monthly or annual rainfall data for weather forecasting, and financial analysts try to find patterns in their large pools of stock prices of different companies.

There are also many large collections of higher-dimensional spatiotemporal trajectories, thanks in part to the development of cost-effective mobile technologies, such as Geographic Information Systems, wireless communication electronics, and multimedia applications [14, 43, 56]. Examples include spatiotemporal trajectories of cars, airplanes, and other moving objects generated by motion tracking devices in surveillance applications and electronic games applications. Additionally, a video stream can also be regarded as a multidimensional trajectory, as it consists of a sequence of multiple frames, each of which is characterized by a set of feature attributes.

Specifically, as part of our collaboration with an electronic games company, we encounter large collections of 2-, 3- and 4-dimensional spatiotemporal trajectories. A 2-dimensional example is the coordinates of National Football League (NFL) players moving on a football field, or of National Hockey League (NHL) players skating on an ice rink. A 3-dimensional example is the positions of aircrafts during a flight simulation. Finally, a 4-dimensional example is the four angles of body joints of a person playing kung-fu or dancing. This type of data sets is useful for games developers and medical professionals. The point here is that beyond 1-dimensional

time series, applications of higher-dimensional spatiotemporal trajectories are very common.

Given those enormous databases of trajectories, what can we do with them, and how do we retrieve valuable information from them? One of the fundamental operations in mining trajectories is **similarity matching**, which refers to the process of finding trajectories that are similar to a given query trajectory. Similarity matching is useful in two aspects. First, it is a subroutine of many data mining tasks, such as classification, clustering, rule discovery, outlier detection, and query by contents. Second, it is important in its own right for exploratory data analysis. The following are typical similarity queries:

- *Identify companies who have similar sales patterns as Microsoft has.*
- *Find out if a given musical score is similar to any of the existing scores.*
- *Discover all images that contain regions similar to regions of a given image.*

1.2 Problem Statement

The problem of retrieving similar trajectories can be formatted as follows: given a reference trajectory database DB, a distance measure $Dist$, a query trajectory $\vec{q}$, and a positive number r, find the set R of trajectories that are within distance r of $\vec{q}$, or more precisely:

$$R = \{\vec{x} \in DB \mid Dist(\vec{x}, \vec{q}) \leq r\} \tag{1.1}$$

This is called a range query or radius search. Alternatively, one might be interested in finding the k nearest neighbours (kNN) of $\vec{q}$, which is equivalent to setting r so that $|R| = k$.

Similarity-based pattern querying has three major components: the similarity model that defines a distance measure between trajectories, the data representation that abstracts features from raw data sets, and the index structure that enables efficient searching for the closest matches.

1.2.1 Similarity Model

Many similarity distance functions have been studied in the literature, and which one is the "best" always depends on the specific user, data set and task. In general, they can be classified into two categories: metric functions and non-metric functions. A distance function D is a metric if it satisfies the following requirements:

- Symmetry: $D(a,b) = D(b,a)$.
- Non-negativity: $D(a,b) \geq 0$ if $a \neq b$, and $D(a,b) = 0$ if and only if $a = b$.
- Triangle Inequality: $D(a,b) \leq D(a,c) + D(c,b)$.

In most cases, a metric function is desired, because the triangle inequality can then be used to prune the index during search. The most popular distance metric is the $\mathcal{L}_p$-norm.

Definition 1.1 *Given two d-dimensional trajectories*

$$\overrightarrow{tu} = \langle (t_1, \vec{u_1}), (t_2, \vec{u_2}), \ldots, (t_N, \vec{u_N}) \rangle$$

and

$$\overrightarrow{tv} = \langle (t_1, \vec{v_1}), (t_2, \vec{v_2}), \ldots, (t_N, \vec{v_N}) \rangle,$$

the $\mathcal{L}_p$-norm distance between them is:

$$\mathcal{L}_p(\overrightarrow{tu}, \overrightarrow{tv}) = [\sum_{i=1}^{N} \sum_{j=1}^{d} |\overrightarrow{u_i^j} - \overrightarrow{v_i^j}|^p]^{\frac{1}{p}} \tag{1.2}$$

where each $\overrightarrow{u_i^j}$ *denotes the* j^{th} *component of the vector* $\overrightarrow{u_i}$.

It is called the Manhattan distance $Dist_{man}$ if $p = 1$, the Euclidean distance $Dist_{euc}$ if $p = 2$, and the Max distance $Dist_{\infty}$ if $p = \infty$. A simple variant of (1.2) is the weighted $\mathcal{L}_p$-norm defined by:

$$\mathcal{L}_p(\overrightarrow{tu}, \overrightarrow{tv}, W) = [\sum_{i=1}^{N} \sum_{j=1}^{d} W_{ij} |\overrightarrow{u_i^j} - \overrightarrow{v_i^j}|^p]^{\frac{1}{p}} \tag{1.3}$$

where W is a matrix of (nonnegative) weights for different points on different trajectories.

While Euclidean Distance $Dist_{euc}$ is used in most existing studies, it is nevertheless insufficient for all situations because:

- It works only for trajectories of the same length.
- It cannot handle outliers or noise.
- It is very sensitive to scale or amplitude.
- It does not work well with trajectories that are similar in shape, but out of phase.
- It does not allow stretching or compression of the time axis.

As a result, many attempts have been made to come up with distance functions that are invariant with respect to six transformations: shifting, uniform amplitude scaling, uniform time scaling, uniform bi-scaling, time warping and non-uniform amplitude scaling. Unfortunately, none of them is a metric. Some of the most famous distance notations are:

- Dynamic Time Warping (DTW) [4, 19]: The idea is to use dynamic programming to construct the warping path in the distance matrix that minimizes the

warping cost and then define the distance as the minimized cost. In effect, it allows shifting and stretching in order to align trajectories.

- Longest Common Subsequence (LCSS) [52, 53]: As a variant of edit distance, it describes how well two trajectories can match one another, by allowing them to stretch and to translate in space, without any rearrangements of the sequence of elements. One of the advantages is that it is robust to noise by giving more weight to the similar portions and paying less attention to regions of great dissimilarity.

- Landmark Model [25, 35]: Generally speaking, it identifies points of "great importance" as landmarks, based on which the similarity patterns are defined. For example, first-order landmarks are global or local extrema, second-order landmarks are inflection points, and so on. It is claimed to better match human intuition and episodic memory as it takes smoothing into account by letting certain landmarks be overshadowed by others.

In this thesis, we adopt the Euclidean distance function $Dist_{euc}$ for spatiotemporal trajectories. While this distance function is easy to compute, it is natural for many applications of spatiotemporal trajectories, including those for airplanes and other flying objects. Additionally, it allows scalable solutions to other problems such as clustering and classification. It is also the distance function adopted by most studies on indexing time series, including [21]. For more advanced distance functions such as time-warping [4] and longest common subsequence [53], we consider them future topics of investigation.

1.2.2 Data Representation

It is not necessary to separate data representation from the similarity model, but most previous works did, as an abstract representation permits more efficient computations than the raw data could, and may allow for an even more sophisticated indexing technique. While we shall discuss related works in greater detail in Chapter 2, it suffices to say that most existing frameworks are based on piecewise approximations, where each piece is either constant or linear. However, recall that, among the examples cited in Section 1.1, one thing in common is that they have smooth and continuous trajectories. This is because all those activities (e.g., human movement, flying objects) are governed by the laws of physics, giving rise to smooth motion trajectories. That is to say, a smooth and continuous trajectory is approximated with a piecewise discontinuous function. This mismatch may cause an unnecessary error or deviation, and may lead to a loss in pruning power in a branch-and-bound search.

In this thesis, we seek to approximate and index a d-dimensional spatiotemporal trajectory with a low order continuous polynomial-like curve. There are many possible ways to choose the polynomial, including (continuous) Fourier transforms, splines, non-linear regression, and so on. While all approximations are not exact by definition, the approximation that minimizes the maximum deviation from the true value is very desirable. This is called the *minimax* approximation. Minimax approximation is particularly meaningful for indexing because in a branch-and-bound search (i.e., for finding nearest neighbours), the smaller the maximum deviation, the more pruning opportunities there exist. However, in general, among all the polynomials of the same degree, the optimal minimax polynomial is very hard to compute. It has been shown that the Chebyshev approximation is almost identical

to the optimal minimax polynomial, and is easy to compute [32]. Thus, we shall explore how to use the Chebyshev polynomials as a basis for indexing d-dimensional trajectories.

1.2.3 Index Structure

Indexing is one of the many searching techniques available for similarity matching. If a searching mechanism retrieves a (proper) subset S of R, then the wrongly dismissed trajectories in $R - S$ are called *false dismissals* or *false negatives*. On the other hand, if S is a (proper) superset of R, then the wrongly retrieved trajectories in $S - R$ are called *false alarms* or *false positives*. As we can always remove false positives in a post-processing stage, they can be tolerated as long as there are not too many of them. Searching techniques that guarantee no false negatives are said to be *exact*; however, there are studies which consider providing faster *approximate* search at the expense of allowing both false positives and negatives [46, 25].

The most obvious brute-force solution for similarity matching would be a sequential scan of the whole database, in which we compute the distance between every trajectory $\vec{x} \in DB$ and $\vec{q}$, and return $\vec{x}$ if it qualifies. This approach requires that we access every single page in the database, which is clearly unrealistic for large data sets. Any mechanisms that avoid retrieving all the data pages could potentially increase the speed of the search, which automatically entails the idea of using an index. While we will discuss the existing indexing schemes in Chapter 2 and propose our own scheme in Chapter 4 and Chapter 5, here we shall give an outline of the desirable properties for any indexing technique [9]:

- It should be faster than sequential scanning.
- It should incur little space overhead.

- It should be able to handle queries of different lengths.
- It should be incremental, that is, it should allow insertions and deletions without rebuilding the index.
- It should guarantee no false negatives.

In addition, two other desirable crteria [21] are:

- It should be possible to build the index in a reasonable time.
- It should be able to handle more than one distance measure.

1.3 Contributions

As a preview, we make the following contributions in this thesis:

- Recall that a spatiotemporal trajectory is of the form $\langle(t_1, \vec{v}_1), \ldots, (t_N, \vec{v}_N)\rangle$. Thus, it is discrete in nature. We show how to approximate such a discrete "function" with Chebyshev polynomials. We first begin with the 1-dimensional case of time series. Our representation scheme allows us to prove a main result of this thesis – the *Lower Bounding Lemma.* That is, the true distance between two time series is lower-bounded by the distance in the index space (i.e., the space of Chebyshev coefficients in our case). As shown in Chapter 4, this is not a trivial result to prove.
- We generalize from the 1-dimensional case to the d-dimensional case ($d \geq 1$). Specifically, a d-dimensional trajectory is projected onto each dimension to create d 1-dimensional trajectories. We show that this projection preserves the Lower Bounding Lemma. We also give algorithms for building an index

of Chebyshev coefficients, and for supporting similarity searching of whole trajectories.

- To evaluate the effectiveness of the minimax property of Chebyshev polynomials on indexing, we conduct an extensive experimental evaluation. We use 1- to 4-dimensional real data sets, as well as generated (i.e. synthetic) data sets. For time series, the Adaptive Piecewise Constant Approximation (APCA) scheme has been shown to outperform all other schemes including Discrete Fourier Transform (DFT), Discrete Wavelet Transform (DWT) and Piecewise Aggregate Approximation (PAA) [21]. We obtain the APCA code from Keogh et al., and compare with Chebyshev approximation. We also extend APCA to d-dimensional situations as a "straw man" strategy.

  As a preview of our results, from 1- to 4-dimensional, real and generated data, Chebyshev dominates APCA in pruning power, I/O cost and CPU cost. Our empirical results indicate that Chebyshev approximation can deliver a 3- to 5-fold reduction on the dimensionality of the index space. For instance, it only takes 4 to 6 Chebyshev coefficients to deliver the same pruning power produced by 20 APCA coefficients. This is a very important advantage. As the dimensionality curse on the indexing structure is bound to set in sooner or later, Chebyshev coefficients are far more effective than APCA in delivering additional pruning power before that happens.

1.4 Thesis Outline

The thesis is organized as follows. In Chapter 2, we discuss related works. In Chapter 3, we review Chebyshev polynomials and their properties central to the

development of this thesis. In Chapter 4, we show how to approximate a time series with a Chebyshev polynomial, and give an example. We also propose a metric distance function between two vectors of Chebyshev coefficients. Finally, we prove the Lower Bounding Lemma. In Chapter 5, we generalize the earlier results for time series to deal with d-dimensional trajectories. In Chapter 6, we present our experimental setup and results. We compare Chebyshev and APCA with respect to pruning power, I/O costs and CPU costs.

Chapter 2

Related Works

2.1 Indexing 1-dimensional Time Series

Substantial efforts have been made on the problem of indexing one-dimensional time series. There are basically two ways to post a similarity query:

- **Whole Matching:** Given a collection DB of M time series, each of length N, and a query time series Q of the same length, we want to find those time series that are within distance r of Q. Mathematically, we seek the set

$$R = \{S \in DB | Dist(S, Q) \leq r\}$$

  Note that every time series, including Q, must have the same length N.

- **Subsequence Matching:** Given a collection DB of M time series $S_1, \ldots, S_M$, where $|S_i| = N_i$ for $1 \leq i \leq M$, and a query time series Q of length $|Q| = N_Q \leq \min_{1 \leq i \leq M}\{N_i\}$, we want to find every time series S_i and every possible offset l such that

$$Dist(Q, S_i[l : (l + N_Q - 1)]) \leq r$$

where $S_i[l : (l + N_Q - 1)]$ denotes the subsequence of S_i starting at position l and ending at position $l + N_Q - 1$. Intuitively, we want to identify those sequences that contain matching subsequences.

It is possible to convert the subsequence matching problem into whole matching, by placing a sliding window of size N_Q at every offset of each sequence and taking each subsequence within the window as one "whole" sequence.

2.1.1 Whole Matching

A time series of length N is by definition a sequence of real numbers, and therefore can be considered as a point in N-dimensional space. This immediately suggests that we can use existing multidimensional index structures (a.k.a. Spatial Access Methods (SAMs)) to store and search such data. SAM examples include the R-tree family, quadtrees, k-d/B tree family and gridfiles. One of the problems common to all multidimensional indices is that their query performance degrades dramatically as dimensionality increases, and eventually reduces to sequential scanning or even worse. Experiments have shown that the R-tree family seems to be the most robust with respect to dimensionality, and that the R^*-trees work well up to 20 dimensions [10]. Since a time series may contain thousands of points, it is impossible to code the entire sequence directly into any multidimensional index. This phenomenon is known as the *dimensionality curse* problem, and in order to utilize the powers of SAMs we need to first perform *dimensionality reduction* (a.k.a. *Feature Extraction*) on the raw data [2].

A framework called GEneric Multimedia INdexIng (GEMINI) is introduced to accommodate any dimensionality reduction methods to allow efficient indexing [9]. The framework consists of three steps:

1. Establish a distance measure $Dist_{true}$ for the raw data series. In this thesis, we focus on Euclidean distance $Dist_{euc}$.

2. Produce a feature extraction function F that reduces the dimensionality of the data from the original length N to n that can be handled by an appropriate index structure.

3. Establish a distance measure $Dist_{feature}$ in the feature space (of n dimensions).

In fact, there is a specific requirement that $Dist_{feature}$ has to satisfy for the GEMINI framework to work out. A crucial result in [9] is that, for any search techniques that use feature extraction, in order to guarantee completeness, the distance measure in the feature space must match or underestimate the true distance.

Theorem 2.1 (Lower Bounding Lemma) *To guarantee no false negatives, the feature extraction function F must satisfy:*

$$Dist_{feature}(F(O_1), F(O_2)) \leq Dist_{true}(O_1, O_2)$$

where O_1 and O_2 are any two raw data series.

Proof: Let Q be the query object, O be a qualifying object and r be the radius. We want to prove that

$$Dist_{true}(Q, O) \leq r \Rightarrow Dist_{feature}(F(Q), F(O)) \leq r.$$

This is clear since

$$Dist_{feature}(F(Q), F(O)) \leq Dist_{true}(Q, O) \leq r.$$

□

Intuitively, this Theorem means that *if two objects are far apart in the feature space, then they must be far apart in the original space.* The performance of GEMINI methods depends solely on the tightness of the lower bound. The closer $Dist_{feature}$ is to $Dist_{true}$, the fewer false positives there are, and the more efficient the algorithm will be.

In the following sections, we shall review the existing dimensionality reduction techniques. Note that they are also applicable to the subsequence matching algorithm to be presented in Section 2.1.2. In general, these studies can be divided into the following categories based on the underlying approximation schemes:

- DFT: [9, 3, 40, 8]
- DWT: [7, 58, 17, 37]
- PAA: [24, 59]
- APCA: [21]
- SVD: [29, 18, 24]

Discrete Fourier Transform (DFT)

The first dimensionality reduction technique proposed for indexing time series in the literature is to use the Discrete Fourier Transform. The basic idea is that any realistic signal can be characterized by the superposition of a finite number of sine/cosine waves, each of which is represented by a single complex number known as a Fourier coefficient. The key observation is that, a signal of length N can be decomposed into N sine/cosine waves that can be recombined into the original signal, and many Fourier coefficients have a very low amplitude and therefore can be discarded without much loss of information in the reconstruction process.

The N-point Discrete Fourier Transform of a signal $\vec{x} = \langle x_0, \ldots, x_{N-1} \rangle$ is a sequence $\vec{X} = \langle X_0, \ldots, X_{N-1} \rangle$ of complex numbers, where

$$X_F = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} x_i \, e^{-\frac{2\pi i j F}{N}} \qquad F = 0, \ldots, N-1 \tag{2.1}$$

where $j = \sqrt{-1}$.

The signal $\vec{x}$ can be recovered by the inverse transform:

$$x_i = \frac{1}{\sqrt{N}} \sum_{F=0}^{N-1} X_F \, e^{\frac{2\pi i j F}{N}} \qquad i = 0, \ldots, N-1 \tag{2.2}$$

The energy $E(\vec{x})$ of a signal $\vec{x}$ is given by:

$$E(\vec{x}) \equiv \|\vec{x}\|^2 = \sum_{i=0}^{N-1} |x_i|^2 \tag{2.3}$$

One of the fundamental properties of DFT is the Parseval's Theorem, which states that the energy is preserved from time domain to frequency domain:

Theorem 2.2 (Parseval's Theorem)

$$\sum_{i=0}^{N-1} |x_i|^2 = \sum_{F=0}^{N-1} |X_F|^2 \tag{2.4}$$

For our dimensionality reduction purposes, we only keep the first n DFT coefficients as features, which will result in an underestimation of the distance between sequences. According to Theorem 2.1, no false negatives are guaranteed.

Discrete Wavelet Transform (DWT)

There are many different types of wavelet transforms and the one that is proposed for dimensionality reduction is the Discrete Haar Wavelet Transform [7]. It is similar to DFT in that it represents data in terms of the sum of a prototype function. However, it is different from DFT in that it is a multi-resolution representation and has the

time-frequency localization property. In most cases, DWT bears more information than DFT, in which only the frequency domain is considered. On the other hand, one of the drawbacks of DWT is that it is defined only for signals whose length is an integral power of 2.

Haar transform is a sequence of averaging and differencing operations in which we compute the average and difference between every two adjacent values of a discrete function (or a time series). For example, let $\vec{x} = \langle 9\ 7\ 3\ 5 \rangle$, then the DWT is computed as follows:

Resolution	**Averages**	**Coefficients**
4	(9 7 3 5)	
2	(8 4)	(1 −1)
1	(6)	(2)

The final Harr Transform $H(\vec{x}) = \langle c\ d_0^0\ d_0^1\ d_1^1 \rangle = \langle 6\ 2\ 1\ -1 \rangle$ is obtained by combining the last average value 6 and the coefficients on the third column, 2, 1 and -1. It is easy to see that we can get different resolutions by adding difference values back to or subtract differences from averages, and the original signal $\vec{x}$ can be recovered from $H(\vec{x})$ by essentially reversing the whole DWT process.

A crucial Theorem established in [7] is that the Euclidean distance between two time series can be expressed in terms of their Haar Transforms.

Theorem 2.3 *Let $\vec{x}$ and $\vec{y}$ be two sequences of length N, which is a power of 2. Suppose their Haar Transforms are $\vec{r}$ and $\vec{s}$, respectively. Let $\vec{r} - \vec{s} = (C\ D_1\ \ldots\ D_{N-1})$. Then the Euclidean distance $Dist_{euc}(\vec{x}, \vec{y}) = S_{\log_2 N}$ can be computed recursively by:*

$$S_0 = C \tag{2.5}$$

$$S_{i+1} = \sqrt{2(S_i^2 + D_{2^i}^2 + D_{2^i+1}^2 + \cdots + D_{2^{i+1}-1}^2)} \tag{2.6}$$

for $0 \leq i \leq \log_2 N - 1$.

One important Corollary of this Theorem is the Lower Bounding Lemma for DWT:

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

Piecewise Aggregate Approximation (PAA)

The idea is introduced independently by Yi and Faloutsos [59] and Keogh et al. [24, 23]. PAA divides each time series of length N into n segments of equal length, and uses the average value of each segment as a coordinate in the n-dimensional feature space. Mathematically, a time series $X = \langle x_1, \ldots, x_N \rangle$ of length N is represented in the n-dimensional feature space by $\bar{X} = \bar{x}_1, \ldots, \bar{x}_n$, where

$$\bar{x}_i = \frac{n}{N} \sum_{j=\frac{N}{n}(i-1)+1}^{\frac{N}{n}i} x_j \tag{2.7}$$

assuming that N is divisible by n.

Then, the distance metric $Dist_{feature}$ in feature space is defined as:

$$Dist_{feature}(\bar{X}, \bar{Y}) = DR(\bar{X}, \bar{Y}) = \sqrt{\frac{N}{n}} \sqrt{\sum_{i=1}^{n} (\bar{x}_i - \bar{y}_i)^2} \tag{2.8}$$

The proof of the Lower Bounding Lemma is long but straightforward. Readers are referred to [24] for a complete proof.

Another important result in [24] is:

Theorem 2.4 *If a raw time series is transformed to a feature space of dimensionality that is a power of 2, then the DWT representation and PAA representation are equivalent in the following ways:*

1. *The best possible representations using both techniques are identical.*

2. *The estimated distances between two objects in the feature space using both techniques are identical.*

This seemingly simple dimensionality reduction scheme has many advantages [24, 59]: it is easy to understand and implement; it is faster than most other transforms; it can handle more distance measures such as the general $\mathcal{L}_p$-norms and weighted distance functions.

Adaptive Piecewise Constant Approximation (APCA)

APCA is a generalization of PAA by relaxing the requirement that each segment must be of the same length. Intuitively, regions with great fluctuations are represented with several short segments, while relatively flat regions are represented with fewer long segments. As a result, APCA requires two numbers per segment, the first number recording the mean value of all the points in the segment and the second number recording the segment length.

Given a time series $S = \langle v_1, \ldots, v_N \rangle$, its APCA representation is defined to be:

$$C = \{\langle cv_1, cr_1 \rangle, \ldots, \langle cv_R, cr_R \rangle\}, \quad cr_0 = 0 \tag{2.9}$$

where R is the number of segments, cv_i is the mean value of the data points in segment i and cr_i is the right endpoint of segment i. For indexing reasons, the right endpoints are used instead of the lengths of segments.

While finding the optimal APCA representation (with the minimum reconstruction error) takes $O(MN^2)$ time, Keogh et al. [21] propose a sub-optimal algorithm using the Discrete Wavelet Transform, as shown in Figure 2.1.

```
Algorithm ComputeAPCA(S, R) {
  /* input: a time series S */
  /* input: the number of segments R to be used */
  /* output: the APCA representation of S */
  (1)   If length(S) is not a power of two, pad it with zeros.
  (2)   Perform the Haar Wavelet Transform on S
  (3)   Sort coefficients in order of decreasing normalized magnitude and
        keep only the first R coefficients.
  (4)   Reconstruct approximation of S from retained coefficients.
  (5)   If S was padded with zeros, truncate it to the original length.
  (6)   Replace approximate segment mean values with exact ones.
  (7)   WHILE (the number of segments is bigger than R)
  (8)     Merge the pair that can be merged with least rise of error
        end WHILE
} /* end algorithm */
```

Figure 2.1: Algorithm to Compute the APCA Representation of a Time Series

To define a distance metric that lower bounds the Euclidean distance $Dist_{euc}$, we must first introduce a special version of the APCA representation. Given a query Q, a time series S together with its APCA representation C, we define another sequence Q' as follows:

$$Q' = \{\langle qv_1, qr_1\rangle, \ldots, \langle qv_R, qr_R\rangle\} \tag{2.10}$$

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

Then the lower-bounding distance metric is defined to be:

$$D_{LB}(Q', S) = \sqrt{\sum_{i=1}^{R} (cr_i - cr_{i-1})(qv_i - cv_i)^2} \tag{2.11}$$

The key contribution Keogh et al. [21] make is that they show APCA is an *indexable* compression scheme. Each time series S is mapped to a point $C = \{\langle cv_1, cr_1\rangle, \ldots, \langle cv_R, cr_R\rangle\}$ in n-dimensional space (where $n = 2R$), and such points are referred to as APCA points. The distance between an APCA point C and Q is defined by $D_{LB}(Q', C)$ (Equations (2.10) and (2.11)) while the distance between Q

and a node U in the index is defined by the minimum distance $MINDIST(Q, Rect)$ between Q and the Minimum Bounding Rectangle (MBR) *Rect* associated with U, as to be discussed below.

Let U be a leaf node in a multidimensional index and $Rect = (L, H)$ be the MBR associated with U. We define $L = \{l_1, \ldots, l_n\}$ and $H = \{h_1, \ldots, h_n\}$ as follows:

$$\begin{aligned} l_i &= MIN_{C\ in\ U}\{cmin_{(i+1)/2}\} && \text{if } i \text{ is odd} \\ &= MIN_{C\ in\ U}\{cr_{i/2}\} && \text{if } i \text{ is even} \\ h_i &= MAX_{C\ in\ U}\{cmax_{(i+1)/2}\} && \text{if } i \text{ is odd} \\ &= MAX_{C\ in\ U}\{cr_{i/2}\} && \text{if } i \text{ is even} \end{aligned} \tag{2.12}$$

where each $cmin_i$ and $cmax_i$ denote the minimum and maximum values of the corresponding time series S among the data points in segment i, that is,

$$\begin{aligned} cmin_i &= MIN_{t=cr_{i-1}+1}^{cr_i}(S_t) \\ cmax_i &= MAX_{t=cr_{i-1}+1}^{cr_i}(S_t) \end{aligned} \tag{2.13}$$

for $i = 1, \ldots, R$.

For an index node U with MBR $Rect = (L, H)$, we can view *Rect* as two APCA representations $L = \{\langle l_1, l_2\rangle, \ldots, \langle l_{n-1}, l_n\rangle\}$ and $H = \{\langle h_1, h_2\rangle, \ldots, \langle h_{n-1}, h_n\rangle\}$. It is clear that any time series S under node U is "contained" within the two sequences L and H. To formalize this notion of containment, we define a set of R regions associated with *Rect*: the i^{th} region $G_i^{Rect}(i = 1, \ldots, R)$ associated with *Rect* is the two-dimensional rectangular region that fully contains the i^{th} segment of all sequences stored under U. Formally speaking, the boundary of the i^{th} region

is:

$$\begin{aligned} G_i^{Rect}[1] &= l_{2i-1} \\ G_i^{Rect}[2] &= l_{2i-2}+1 \\ G_i^{Rect}[3] &= h_{2i-1} \\ G_i^{Rect}[4] &= h_{2i} \end{aligned} \tag{2.14}$$

where $G[1]$ and $G[2]$ are the low bounds, and $G[3]$ and $G[4]$ are the high bounds along the value and time axes. At time instance $t = 1, \ldots, N$, we say that a region G_i^{Rect} is active if and only if $G_i^{Rect}[2] \leq t \leq G_i^{Rect}[4]$.

Given a query time series Q, the minimum distance $MINDIST(Q, Rect, t)$ between Q and $Rect$ at time instant $t = 1, \ldots, N$ is given by:

$$MINDIST(Q, Rect, t) = MIN_{G \text{ is active at } t}\{MINDIST(Q, G, t)\}$$

where

$$\begin{aligned} MINDIST(Q, G, t) &= (G[1] - Q_t)^2 \quad \text{if } Q_t \leq G[1] \\ &= (Q_t - G[3])^2 \quad \text{if } G[3] \leq Q_t \\ &= 0 \qquad\qquad\quad \text{otherwise.} \end{aligned}$$

And finally, we define:

$$MINDIST(Q, Rect) = \sqrt{\sum_{t=1}^{N} MINDIST(Q, Rect, t)} \tag{2.15}$$

As the proof of the Lower Bounding Lemma (that both D_{LB} and $MINDIST$ lower bound $Dist_{euc}$) is not trivial to present, we shall omit the details here.

Experiments have shown that the APCA representation has a very high fidelity to the original signal and a relatively low reconstruction error [21]. As a result, for indexing time series data, APCA outperforms all the above schemes, including DFT, DWT and PAA.

Singular Value Decomposition (SVD)

SVD is also known as Karhunen-Loeve (K-L) transform [57] and Principle Component Analysis (PCA) in statistics.

Given a collection of N-dimensional vectors $\{\vec{x}^1, \ldots, \vec{x}^M\}$, we collect them into an M x N matrix A. The Singular Value Decomposition of A is given by:

$$A = U\Sigma V^T \tag{2.16}$$

where U is an M x N matrix and Σ and V are N x N matrices. We have:

$$UU^T = I_M \quad U^TU = I_N$$

and V is orthonormal:

$$VV^T = V^TV = I_N$$

Σ is a diagonal matrix with non-negative elements called singular values along its diagonal:

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_N \end{pmatrix}$$

Since V is orthonormal, we can multiply both sides of Equation (2.16) by V and we get:

$$AV = U\Sigma \tag{2.17}$$

The product $U\Sigma$ contains a set of N-dimensional vectors $\{\vec{X}^1, \ldots, \vec{X}^M\}$, which are rotated from the original vectors $\{\vec{x}^1, \ldots, \vec{x}^M\}$. As rotation preserves length, we have:

$$\|\vec{x}^i\| = \|\vec{X}^i\| \text{ for } i = 1, \ldots, M$$

From Equation (2.17), we have:

$$\begin{pmatrix} \overrightarrow{X^1}^T \\ \vdots \\ \overrightarrow{X^M}^T \end{pmatrix} = \begin{pmatrix} \overrightarrow{U^1}^T \\ \vdots \\ \overrightarrow{U^M}^T \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_N \end{pmatrix}$$

We can reduce the dimensionality by discarding the least significant singular values in Σ and the corresponding entries in A, U and V. We have:

$$\begin{pmatrix} \overrightarrow{X_n^1}^T \\ \vdots \\ \overrightarrow{X_n^M}^T \end{pmatrix} = \begin{pmatrix} \overrightarrow{U_n^1}^T \\ \vdots \\ \overrightarrow{U_n^M}^T \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n \end{pmatrix} = U_n \Sigma_n$$

The key property of SVD is that it provides the best least squares fit to any matrix of data points. In other words, the product $U_n \Sigma_n$ contains the truncated versions of the original vectors rotated to the directions of best least squares fit. Consequently, SVD is optimal in the sense that it minimizes the reconstruction error among all (linear) transformations. However, since the whole database has to be examined before the transformation, SVD is a global technique and therefore not incremental. A single insertion to the data set would result in a recomputation of the entire reduction matrix. Additionally, since SVD requires $O(MN^2)$ time and $O(MN)$ space, we shall omit it in our experimental comparison.

2.1.2 Subsequence Matching

Subsequence matching is a more difficult problem, as different time series might have different lengths. There are not as many studies in this area as the ones for whole matching.

I-adaptive Method

This is the first work on indexing techniques for subsequence matching in time series databases, and it generalizes the GEMINI framework, which is originally designed for whole matching problems [9].

Without loss of generality, assume that the minimum query length is w. We use a sliding window of width w and place it at every possible offset l on every data sequence S_i, $i = 1, \ldots, M$. For each placement of the window, we extract n features from the subsequence inside the window, where n can be handled by an appropriate multidimensional index structure. Note that, the feature extraction can be done using any of the dimensionality techniques introduced in Section 2.1.1. Thus, a time series of length $|S_i| = N_i$ is transformed into a trail in feature space, consisting of $N_i - w + 1$ points, one for each possible offset of the sliding window. By taking advantage of the fact that adjacent points on the same trail will probably be very close to each other, we then divide the trail of a given data sequence into subtrails and represent each subtrail by its Minimum Bounding Rectangle (MBR). Finally, each time series is transformed into a set of MBRs in feature space, and we insert all the rectangles for every sequence into a multidimensional index.

Attempting to minimize the number of disk accesses, Faloutsos et al. use an adaptive heuristic *I-adaptive* based on a greedy algorithm to group trail points into subtrails. The algorithm defines a cost function that estimates the disk accesses:

Definition 2.1 *Given the sides $\vec{L} = (L_1, \ldots, L_n)$ of the n-dimensional MBR of a node in an index, the average number of disk accesses $DA(\vec{L})$ that this node will contribute for the average range query is:*

$$DA(\vec{L}) = \prod_{i=1}^{n} (L_i + 0.5) \tag{2.18}$$

assuming that the rectangles have been normalized to $[0,1)^n$.

Definition 2.2 *Given a subtrail of k points with an MBR of sizes* $\vec{L} = (L_1, \ldots, L_n)$ *the marginal cost mc of each point in this subtrail is:*

$$mc = \frac{DA(\vec{L})}{k} \tag{2.19}$$

The algorithm for dividing a trail into subtrails is then as follows: we assign the first point of the trail in a (trivial) subtrail. Then, for each successive point, if it increases the marginal cost of the current subtrail, start a new subtrail; otherwise, include it in the current subtrail.

For range searches, there are two cases:

- If the query sequence Q has a length equal to w, we map Q to a point q_f in feature space, and then the range query corresponds to a sphere centred at q_f with radius r. We retrieve subtrails whose MBRs intersect the query region and examine the real subsequences to discard false positives if any.

- If the length of Q is longer than w, we split the query into p pieces of length w each, process each subquery separately and merge the results. Moreover, the tolerance/radius for each subquery can in fact be reduced to $r/\sqrt{p}$ [9].

Other Approaches

Keogh et al. [22] introduce STB-indexing for subsequence matching. Their idea is to divide the data sequences into non-overlapping parts of a prespecified window size. If the points within a segment are mostly increasing, then this segment is represented by the number 1. If they are mostly decreasing, then the segment is represented by 0. The transformed sequences are then stored in bins, each of which

also contains a matrix that records the distance between all pairs of sequences. For query processing, the sequence is used for bin-pruning and the distance matrix is used for interbin-pruning.

Kahveci et al. [17] propose another method to handle range searches for queries of variable lengths. Their algorithm splits a given query into non-overlapping subqueries at different resolutions. For each subquery, a search in the index is performed corresponding to the resolution of the subquery. The results are then used to refine the radius of the next subquery. The search volume decreases exponentially as the query radius decreases, and consequently, this dramatically reduces redundant computations and disk reads.

2.2 Indexing Multidimensional Trajectories

There are not many studies in the literature on indexing multidimensional trajectories for similarity matching.

Theodoridis et al. [49] provide a formal specification for spatiotemporal index structures and multidimensional access methods in SpatioTemporal Database Management Systems (STDBMS). They describe a classification scheme for efficient indexing and query processing in spatiotemporal databases. They discuss three types of specification issues on:

- data types and data sets supported
- index construction
- query processing operations

They evaluate the existing proposals according to the above issues, and observe that

most of those methods do not follow the full list of specifications proposed, and thus they should be extended and revised.

Lee et al. [30] extend the traditional similarity search methods on time series data to support multidimensional trajectories. They first define the distance between two multidimensional sequences as a variant of Euclidean distance, and then introduce two lower bounding metrics, based on which they propose an algorithm to prune a database of irrelevant sequences and to find the solution interval of the selected sequences. Their methods have the following advantages:

- The search algorithm is based on Minimum Bounding Rectangles (MBRs), so it is fast and needs small storage overhead.
- The framework is designed to handle sophisticated similarity search, such as finding subsequences of a selected sequence.
- Query and data sequences can have arbitrary length.
- Any multidimensional access structure can be used.

Vlachos et al. [52, 53] investigate techniques for analysis and retrieval of trajectories in two-dimensional or three-dimensional space. In order to cope with the noisy nature of such data, they formalize non-metric similarity functions based on the Longest Common Subsequence (LCSS) framework, which is very robust to noise. As the exact computation of these measures is unavoidably inefficient, they present approximation algorithms with provable performance bounds. In addition, they prove a weaker version of the triangle inequality that can be used to prune the index for answering nearest neighbour queries based on hierarchical clustering. And finally, they compare their framework to the widely used Euclidean and Time Warping distance functions and show the superiority of their approach.

Kollios et al. [28], Papadopoulos et al. [34] and Saltenis et al. [44] consider indexing mobile objects and one-dimensional and two-dimensional space to answer range queries over the object locations into the future. One example of such queries is: "Report all objects that will be inside a specific region after a certain amount of time." They model the objects as points moving at a constant speed starting from a specific location, and approximate each trajectory by straight line segments in their indexing scheme. They also give an approximation algorithm with linear space and expected logarithmic query time in a dynamic external memory setting, as well as an algorithm with guaranteed logarithmic query time for a restricted version of the problem.

In addition to those similarity-based and coordinated-based queries discussed above, Pfoser et al. [36] introduce the concept of trajectory-based queries, which can be further classified into topological and navigational queries. Topological queries involve the whole or part of a trajectory, and they are important as well as expensive. Some of the basic spatial predicates are: *meet, overlap, contain, equal,* and *cross.* On the other hand, navigational queries seek dynamic information that is not explicitly stored, but has to be derived from the trajectories. For example, the average speed of a object is computed by dividing the distance travelled by the time taken, the direction of an object is determined by considering a vector between the starting and ending positions, and the area an object covers is derived by computing the convex hull of its trajectory. Having described the types of data and queries, the authors present two access methods for indexing such data, namely, the SpatioTemporal R-tree (STR-tree) and the Trajectory-Bundle tree (TB-tree).

Hadjieleftheriou et al. [12] and Kollios et al. [28] propose to index animated objects as a spatiotemporal evolution so as to efficiently answer queries about their

position in time and space. While most of the previous research has concentrated on the raw and feature levels and examines similarity-based queries, their work is focused on the semantic level and the queries are topological in nature. They classify evolutions as the degenerate case and the general case. In the degenerate case, objects are simply added or deleted from the movie. In the general case, objects are allowed to move and grow/shrink among frames during their life time. On the assumption that objects can only move or grow/shrink according to a polynomial function of time, they combine a spatial index with a partially persistent methodology, such that a new record is inserted into the index only when the parameters describing the movement (or extent) change.

Chapter 3

Background: Chebyshev Polynomials

A polynomial is a real function $P(x)$ that can be written in the form:

$$P(x) = a_0 + a_1 x + \cdots + a_m x^m$$

where $a_0, \cdots, a_m$ are real numbers and x is a real variable. If $a_m \neq 0$, we say that $P(x)$ has a degree of m.

Polynomials have many nice properties. For example, they can be differentiated as many times as we want for any values of x, and they can also be integrated over any intervals. Additionally, $P(x)$ is uniquely defined by the $m+1$ coefficients $a_0, \ldots, a_m$. As a result, polynomials have been the top choice for the approximation and interpolation of more complicated functions.

Chebyshev Polynomials are a special group of polynomials, whose properties and applications were discovered a century ago by the Russian Mathematician Pafnuty Lvovich Chebyshev. Their importance for practical computation, however, was rediscovered 60 years ago by Cornelius Lanczos, the father of Numerical

Mathematics. Thanks to their orthogonality and minimax properties, Chebyshev Polynomials have played a significant role in nearly every area of numerical analysis, including polynomial approximation, numerical integration, integral equations, and spectral methods for Partial Differential Equations.

In this chapter, we shall give an overview of the basic definitions and key formulae of Chebyshev Polynomials, and their applications in approximations and series expansions [32, 38, 41].

3.1 Trigonometric Definitions and Recurrences

Definition 3.1 *The Chebyshev Polynomial $T_m(t)$ of the first kind is a polynomial of degree m $(m = 0, 1, \ldots)$, defined by:*

$$T_m(t) = cos(m\ cos^{-1}(t)) \tag{3.1}$$

for $t \in [-1, 1]$.

There are actually four kinds of Chebyshev Polynomials, but $T_m(t)$ of the first kind is by far the most important and influential group and in this thesis we shall use the expression "Chebyshev Polynomials" to refer exclusively to the Chebyshev Polynomials $T_m(t)$ of the first kind.

While it is inconvenient and inefficient to compute each $T_m(t)$ directly from Equation (3.1), we can utilize the trigonometric identity

$$cos\, m\theta + cos\, (m-2)\theta = 2\ cos\, \theta\ cos\, (m-1)\theta$$

to derive the fundamental recurrence relation:

$$T_m(t) = 2tT_{m-1}(t) - T_{m-2}(t) \tag{3.2}$$

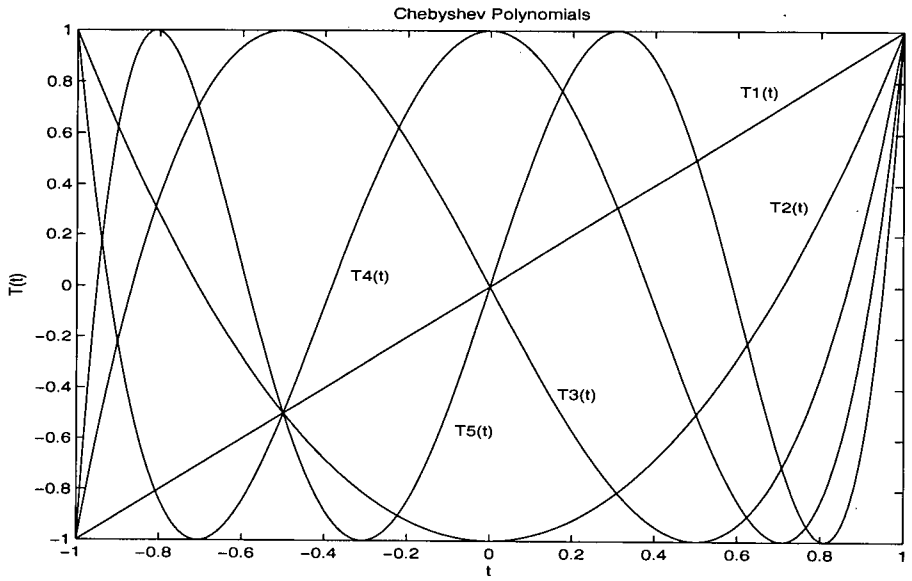


Figure 3.1: Chebyshev Polynomials

for all $m \geq 2$ with $T_0(t) = 1$ and $T_1(t) = t$.

From the above definition, the first six Chebyshev polynomials are:

$$\begin{aligned} T_0(t) &= 1 \\ T_1(t) &= t \\ T_2(t) &= 2t^2 - 1 \\ T_3(t) &= 4t^3 - 3t \\ T_4(t) &= 8t^4 - 8t^2 + 1 \\ T_5(t) &= 16t^5 - 20t^3 + 5t \end{aligned}$$

And Figure 3.1 shows the graphs of $T_1(t)$ to $T_5(t)$.

Even though in the above definitions, t is defined only over the interval $[-1, 1]$, we may define Chebyshev Polynomials appropriate to any given finite inter-

val $[a, b]$ $(-\infty < a < b < +\infty)$ by transforming this domain into $[-1, 1]$ of a new variable s under the linear transformation:

$$s = \frac{2x - (a + b)}{b - a}$$

Then, the Chebyshev Polynomials appropriate to $[a, b]$ are $T_m(s)$. Without loss of generality, hereafter we simply focus on the interval $[-1, 1]$.

Finally, we conclude this section with a key property of Chebyshev Polynomials:

Theorem 3.1 *The Chebyshev Polynomial $T_m(t)$ has m zeros and $m + 1$ local extrema in $[-1, 1]$.*

The zeros are:

$$t_j = cos\frac{(j - \frac{1}{2})\pi}{m}, \quad j = 1, 2, \ldots, m \tag{3.3}$$

The extrema are:

$$t_i = cos\frac{i\pi}{m}, \quad i = 0, 1, \ldots, m \tag{3.4}$$

Note that all the zeros are interior to $[-1, 1]$, while there are two extrema at the endpoints ± 1, and $m - 1$ "true" alternate maxima and minima where the first derivative is 0.

3.2 Approximation Theory

As described before, polynomials are among the simplest classes of functions in the sense that they can be easily specified, compactly represented, and efficiently evaluated. However, there exist many other functions that are complex in nature. This is where approximation comes in – it is useful and sometimes essential to

approximate any given function f by a much simpler function f^*, such that the approximating value $f^*(x)$ is very close to the corresponding real value $f(x)$. In this section, we shall review the fundamental concepts of approximation theory, with an emphasis on uniform (minimax) approximation, and then introduce the minimax property of Chebyshev Polynomials.

The approximation theory consists of three major components:

1. A function class $\mathcal{F}$ (containing all the functions to be approximated).

2. A form (for the approximation function f^*) that is parameterized by a few adjustable coefficients. This defines a set $\mathcal{A}$ of possible approximations to the given f.

3. A norm $\|\cdot\|$ (of the approximation error) that measures how good the approximation is. That is, $\|f - f^*\|$ defines the closeness of f^* to f.

There exists a vast body of literature on how to define different approximation problems by making appropriate selections of these components, however, in this thesis, our main focus is as follows:

1. Function class $\mathcal{F} = \mathcal{L}_p[a, b]$, that is, the set of $\mathcal{L}_p$-integrable functions on $[a, b]$, defined by:

$$\mathcal{F} = \mathcal{L}_p[a, b] = \{h(x) \text{ for which } \int_a^b w(x)|h(x)|^p \, dx \text{ exists}\} \tag{3.5}$$

   where $w(x)$ is a given nonnegative weight function, and $1 \leq p < \infty$.

2. Form

$$\mathcal{A} = \Pi_m = \{f^*(x) = P_m(x) = a_0 + a_1 x + \cdots + a_m x^m\} \tag{3.6}$$

   where the adjustable coefficients are $a_0, \ldots, a_m$.

3. Norm $\|\cdot\| = \|h\|_p = \mathcal{L}_p$-norm where

$$\|h\|_p = [\int_a^b w(x)|h(x)|^p \, dx]^{\frac{1}{p}} \quad (1 \leq p \leq \infty) \tag{3.7}$$

Specifically, the Chebyshev (minimax) norm is:

$$\|h\|_\infty = max_{a \leq x \leq b}|h(x)| \tag{3.8}$$

With respect to a specific function class, a form, and a norm, we are particularly concerned with the following questions:

1. What is a best approximation?
2. Does there always exist a best approximation?
3. Is the best approximation unique?
4. How do we construct a best approximation?

Our polynomial approximation problem, as defined by Equations (3.5) through (3.8), is one of the classical problems in the literature. In the remainder of this section, we shall cite a few theorems to characterize the properties of a best approximation, and delay the discussion of (4) to a later section. All the theorems are by themselves very important in numerical analysis, and they require an extensive amount of mathematical proof for which we shall refrain from going into details.

Definition 3.2 *An approximation f_B^* in $\mathcal{A}$ is a* **best approximation** *to f if, for any other approximation f^* in $\mathcal{A}$,*

$$\|f - f_B^*\| \leq \|f - f^*\|$$

In the case of $\mathcal{L}_\infty$-norm, we often use the terminology **minimax** *in place of* **best**.

Note that the above definition only shows what a best approximation looks like, but says nothing about whether one exists at all or whether there is more than one for the same function. The next Theorem establishes the existence and uniqueness of the best (polynomial) approximation.

Theorem 3.2 (Weierstrass's Theorem) *For any given continuous function f and for any given $\epsilon > 0$, there exists a polynomial P_m for some sufficiently large m such that $\|f - P_m\|_p < \epsilon$ for any $p \geq 1$. Furthermore, there exists a unique best polynomial approximation to any function $f \in \mathcal{L}_p[a,b]$ in the $\mathcal{L}_p$-norm, where $w(x) = 1$ in the case $p \to \infty$.*

Best approximations also exist in $\mathcal{L}_p$-norm on finite point sets for $1 \leq p \leq \infty$ and are unique if and only if $p > 1$. Such $\mathcal{L}_p$ norms are defined by:

$$\|f - f^*\|_p = [\sum_{i=1}^{N} w_i |f(x_i) - f^*(x_i)|^p]^{\frac{1}{p}}$$

where $\{w_i\}_{i=1}^N$ are positive scalar weights and $\{x_i\}_{i=1}^N$ is a discrete set of fitting points.

In particular, Theorem 3.2 guarantees the existence of a unique best approximation in the minimax norm; however, it does not tell us how to recognize such an approximation. Rather surprisingly, it is possible to do so explicitly, as the following powerful Theorem illustrates.

Theorem 3.3 (Alternation Theorem for Polynomials) *For any continuous function f a unique minimax polynomial approximation P_m exists and is uniquely characterized by the "alternating or equioscillation property" that there are at least $m+2$ points in $[a,b]$ at which $f(x) - P_m(x)$ attains its maximum absolute value with alternating signs.*

It is clear from Theorem 3.1 and Figure 3.1 in Section 3.1 that the Chebyshev Polynomial $T_m(t)$ has $m+1$ extrema with alternating signs on the interval $[-1,1]$. By invoking Theorem 3.3, we have:

Theorem 3.4 *The polynomial $2^{1-m}T_m(t)$ is the minimax approximation on $[-1,1]$ to the zero function by a monic polynomial of degree m.*

3.3 Orthogonality

In the last section, we showed that their minimax property have earned Chebyshev Polynomials a key position in the development of $\mathcal{L}_\infty$ approximations. On the other hand, Chebyshev Polynomials also possess another equally important property, namely, they are a family of orthogonal polynomials. The orthogonality of those polynomials, in addition to having a strong linkage with $\mathcal{L}_2$ (or least-squares) approximations, lends itself to the subject of series expansions.

Definition 3.3 *Two functions $f(x)$ and $g(x)$ in $\mathcal{L}_2[a,b]$ are orthogonal on the interval $[a,b]$ with respect to a given continuous and nonnegative weight function $w(x)$ if*

$$\int_a^b w(x)f(x)g(x)\,dx = 0 \tag{3.9}$$

If we use the "inner product" notation

$$\langle f,\ g\rangle = \int_a^b w(x)f(x)g(x)\,dx \tag{3.10}$$

then the orthogonality condition (3.9) is equivalent to saying that f is orthogonal to g if

$$\langle f,\ g\rangle = 0. \tag{3.11}$$

Furthermore, with this notation of inner product, the $\mathcal{L}_2$-norm is

$$\|h\| = \|h\|_2 := \sqrt{\langle h, h \rangle}. \tag{3.12}$$

Definition 3.4 *A family of polynomials $\{\phi_i(x) : i = 0, 1, \dots\}$ is orthogonal on the interval $[a, b]$ with respect to a given continuous and nonnegative weight function $w(x)$ if and only if for each $i = 0, 1, \dots$, and $j = 0, 1, \dots$*

1. *$\phi_i(x)$ is of degree i.*
2. *$\langle \phi_i(x), \phi_j(x) \rangle = 0$, if $i \neq j$.*
3. *$\langle \phi_i(x), \phi_i(x) \rangle = \|\phi_i\|^2 > 0$.*

The family is orthonormal if, for all i, $\langle \phi_i(x), \phi_i(x) \rangle = 1 = \|\phi_i\|^2$.

If we define the inner product (3.10) using the interval and weight function

$$[a, b] = [-1, 1], \qquad w(t) = (1 - t^2)^{-\frac{1}{2}}$$

then the following theorem holds for the inner product between Chebyshev Polynomials:

Theorem 3.5

$$\langle T_i, T_j \rangle = \int_{-1}^{1} \frac{T_i(t) T_j(t)}{\sqrt{1 - t^2}} dt = \begin{cases} 0 & \text{if } i \neq j \\ \frac{\pi}{2} & \text{if } i = j \neq 0 \\ \pi & \text{if } i = j = 0 \end{cases} \tag{3.13}$$

The system $\{T_i\}$ is therefore orthogonal with respect to $w(t)$ but not orthonormal. Hereafter we shall refer to

$$w(t) = (1 - t^2)^{-\frac{1}{2}}$$

as the **Chebyshev weight function.**

3.4 Series Expansions

One of the best ways to approximate a given function is to expand it in terms of infinite series of an orthogonal family of simpler functions. Indeed, many expansion techniques have been studied before, including the familiar Taylor series, Laurent series and Fourier series. We may write the Chebyshev series expansion of a given function $f(t)$ as

$$S_\infty^T(f)(t) = \lim_{m\to\infty} S_m^T(f)(t) = \sum_{i=0}^{\infty} c_i T_i(t) \tag{3.14}$$

where $S_m^T(f)$ denotes the partial sum of the infinite series $S_\infty^T(f)$.

The following Theorem asserts that $S_\infty^T(f)$ is in fact $\mathcal{L}_2$-convergent with respect to the Chebyshev weight function, provided that f is $\mathcal{L}_2$-integrable with respect to the same weight function.

Theorem 3.6 *If $f(t)$ is $\mathcal{L}_2$-integrable with respect to the inner product (3.10), then its Chebyshev series expansion (3.14) converges in $\mathcal{L}_2$, in other words,*

$$\int_{-1}^{1} (1-t^2)^{-\frac{1}{2}} [f(t) - S_m^T(f)(t)]^2 \, dx \to 0, \, as \; m \to \infty$$

Thus, we may write:

$$f(t) = S_\infty^T(f)(t) = \sum_{i=0}^{\infty} c_i T_i(t) \tag{3.15}$$

It follows, by taking inner products with T_j, that

$$\langle f, T_j \rangle = \sum_{i=0}^{\infty} c_i \langle T_i, T_j \rangle = c_j \langle T_j, T_j \rangle$$

since $\langle T_i, T_j \rangle = 0$ for $i \neq j$. Therefore,

$$c_j = \frac{\langle f, T_j \rangle}{\langle T_j, T_j \rangle} \tag{3.16}$$

Applying Equation (3.10) and Theorem 3.5, we have:

$$c_0 = \frac{\langle f, T_0 \rangle}{\langle T_0, T_0 \rangle} = \frac{1}{\pi} \int_{-1}^{1} \frac{f(t)}{\sqrt{1-t^2}} \, dt \tag{3.17}$$

$$c_j = \frac{\langle f, T_j \rangle}{\langle T_j, T_j \rangle} = \frac{2}{\pi} \int_{-1}^{1} \frac{f(t)T_j(t)}{\sqrt{1-t^2}}\, dt, \quad \text{for } j \geq 1. \tag{3.18}$$

Note that in the above formulae, for c_0 and c_j's, the constants differ by a factor of 2. This is a direct consequence of the second and third cases shown in Theorem 3.5 (i.e., π versus $\pi/2$).

For a particular degree m, the error function for the partial sum $S_m^T f$ is $e_m^T f = f - S_m^T f$ and it satisfies:

$$\begin{aligned}
\|e_m^T f\|^2 &= \langle f - S_m^T f, f - S_m^T f \rangle \\
&= \langle f, f \rangle - 2\langle f, S_m^T f \rangle + \langle S_m^T f, S_m^T f \rangle \\
&= \|f\|^2 - 2\textstyle\sum_{i=0}^{m} c_i \langle T_i, f \rangle + \sum_{i=0}^{m} c_i^2 \langle T_i, T_i \rangle \\
&= \|f\|^2 - 2\textstyle\sum_{i=0}^{m} c_i \frac{\pi}{2} c_i + \sum_{i=0}^{m} c_i^2 \frac{\pi}{2} \\
&= \|f\|^2 - \tfrac{\pi}{2} \textstyle\sum_{i=0}^{m} c_i^2
\end{aligned}$$

From Theorem 3.6, $e_m^T f \to 0$, as $m \to \infty$, therefore, we obtain an important convergence Theorem for Chebyshev Coefficients:

Theorem 3.7

$$\sum_{i=0}^{\infty} c_i^2 = \frac{2}{\pi}\|f\|^2 = \frac{2}{\pi} \int_{-1}^{1} (1-t^2)^{-\frac{1}{2}} f^2(t) dt. \tag{3.19}$$

Chapter 4

Indexing with No False Negatives

In this Chapter, we focus on 1-dimensional spatiotemporal trajectories, that is, time series. In Chapter 5, we shall generalize our framework to higher dimensional trajectories. Figure 4.1 summarizes all the symbols used in this thesis.

Given a collection of time series of length N, we intend to represent each time series by its Chebyshev approximation of degree m, with $m \ll N$. To facilitate fast searching, the $n = m + 1$ Chebyshev coefficients are to be stored in a multi-dimensional index structure. As such, n is typically small, say below 25.

To show that indexing the time series is reduced to indexing their Chebyshev coefficients, we follow the GEMINI framework [9]. We first establish a distance metric for the Chebyshev coefficients. In this thesis, we use the Euclidean distance (denoted as $Dist_{euc}$) to measure the distance between two time series S_1, S_2. We propose in Section 4.4 a natural Euclidean variant (denoted as $Dist_{cby}$) for measuring the distance between the two corresponding vectors of Chebyshev coefficients $\vec{C}_1, \vec{C}_2$.

Notations	Meanings
$T_m(t)$	Chebyshev Polynomial of degree m
m	the degree of the polynomial $T_m(t)$ also the degree of the approximated polynomial
n	the number of Chebyshev coefficients, i.e., $n = m + 1$
c_i	the coefficient of $T_i(t)$ in an approximation
M	the number of trajectories
N	the (padded) length of each trajectory
d	the dimensionality of the trajectories
$\vec{v}$	a vector of arity d
S	a spatiotemporal trajectory $\langle(t_1, \vec{v}_1), \ldots, (t_N, \vec{v}_N)\rangle$
$\vec{C}$	the vector of n Chebyshev coefficients for S
$f(t)$	an interval function defined for trajectory S
$Dist_{euc}$	Euclidean Distance
$Dist_{euc_w}$	Weighted Euclidean Distance

Figure 4.1: Summary of Notation

We then establish the important *Lower Bounding Lemma* in Section 4.5:

$$Dist_{cby}(\vec{C}_1, \vec{C}_2) \leq Dist_{euc}(S_1, S_2)$$

This lemma is critical in guaranteeing no false negatives in using the index as a filter. And the tighter the lower bound, the smaller is the number of false positives.

4.1 Assumptions

Concerning the studies of time series and trajectories in the literature, the following assumptions may be made:

- **Same-length**: That every time series has the same length, i.e., N time points.
- **Power-of-2**: That every time series has a length 2^k for some positive integer k.
- **Same-set**: That every time series has the same *set* of time points $\{t_1, \ldots, t_N\}$. Thus, this assumption automatically includes the same-length assumption. However, the width between a pair of successive time points t_i and t_{i+1} is not necessarily the same as the width between any other pair.

- **Same-set-uniformly-spaced**: This extends the same-set assumption to require that the width between a pair of successive time points be the same everywhere, i.e., $(t_i - t_{i-1}) = (t_{i+1} - t_i)$ for $i = 2, \ldots, N-1$.

In this study, like most frameworks on trajectory matching, we make the same-length assumption. If the time series are not of the same length, padding techniques may be applied (see Matlab for example). Unlike some other frameworks, like wavelet decompositions [7, 58, 17, 37], we do not require the power-of-2 assumption. Furthermore, we make the same-set assumption to make the ensuing analysis easier. If this assumption is not met, interpolation techniques may be applied. The results to be presented in this thesis do not require the same-set-uniformly-spaced assumption.

4.2 Chebyshev Approximation of a Time Series

Given a time series, we begin with the computation of the Chebyshev coefficients. However, Equations (3.17) and (3.18) are not immediately applicable because the given formulae are restricted to *interval functions*. By "interval functions", we mean functions whose domain is an interval (in our case, the interval [-1,1]). The function may or may not be continuous, but is defined everywhere over the interval.

In contrast, the time series is a *discrete* function, as the domain is a discrete set, rather than an interval. Specifically, let the time series be $S = \langle (t_1, v_1), \ldots, (t_N, v_N) \rangle$, where $-1 \leq t_1 < \ldots < t_N \leq 1$. (Recall that time t is normalized into the range [-1,1].) This can be rewritten in functional form below:

$$S(t) = \begin{cases} v_i & \text{if } t = t_i \\ undefined & \text{otherwise} \end{cases} \tag{4.1}$$

To apply Equations (3.17) and (3.18), we need to extend the above discrete function into an interval function. We first divide the interval $[-1, 1]$ into N disjoint subintervals as follows:

$$I_i = \begin{cases} [-1, \frac{t_1+t_2}{2}) & \text{if } i = 1 \\ [\frac{t_{i-1}+t_i}{2}, \frac{t_i+t_{i+1}}{2}) & \text{if } 2 \leq i \leq N-1 \\ [\frac{t_{N-1}+t_N}{2}, 1] & \text{if } i = N \end{cases} \tag{4.2}$$

An obvious choice for an interval function would be the following *step* function:

$$g(t) = v_i \quad \text{if } t \in I_i, \quad (\text{for } 1 \leq i \leq N) \tag{4.3}$$

To create an interval function based on the original time series in Equation (4.1), the above function defines the previously undefined parts as follows. Between a pair of successive time points t_i and t_{i+1}, the mid-point $\frac{t_i+t_{i+1}}{2}$ is used as a "divider" – the first half retains the value v_i, while the second half adopts the value v_{i+1}. Special attention is paid to the boundary conditions: t_1 with respect to the left end-point of the interval, and t_N with respect to the right end-point.

While the above function is simple, it does not immediately satisfy the Lower Bounding Lemma. A key result to be proven later in this thesis is that the lemma is satisfied with the inclusion of the Chebyshev weight function (defined in Section 3.3) and the length of each subinterval:

$$f(t) = \frac{g(t)}{\sqrt{w(t)|I_i|}} \quad \text{if } t \in I_i \quad (\text{for } 1 \leq i \leq N) \tag{4.4}$$

where $g(t)$ is as defined in Equation (4.3) and $|I_i|$ is the length of subinterval I_i.

At this point, we need to use an known result in integral calculus [51]:

Lemma 4.1 *A function is integrable over interval $[a, b]$ if it is bounded and has a finite number of discontinuities on $[a, b]$.*

Now, by applying Equation (4.4) and Lemma 4.1 we can claim that:

Lemma 4.2 *$f(t)$ is $\mathcal{L}_2$-integrable with respect to the Chebyshev weight function.*

Because $f(t)$ is also an interval function, we can use the formulae to compute the coefficients of the Chebyshev approximation. Furthermore, for better approximation quality, we can use all N data points and values of the time series. Of course, to reduce dimensionality, we only keep the first $n = m + 1$ Chebyshev coefficients for indexing. More specifically, we have the following formulae:

$$\begin{aligned} c_0 &= \frac{1}{\pi}\int_{-1}^{1} \frac{f(t)}{\sqrt{1-t^2}}\, dt \\ c_i &= \frac{2}{\pi}\int_{-1}^{1} \frac{f(t)T_i(t)}{\sqrt{1-t^2}}\, dt \end{aligned} \tag{4.5}$$

for all $1 \leq i \leq (n-1)$.

For a complex function $f(x)$, it might not be easy or efficient to evaluate the integrals exactly. We would rather evaluate them numerically using *Gauss-Chebyshev quadrature.*

Theorem 4.1 *For any $\mathcal{L}_2$-integrable function $F(t)$ with respect to the Chebyshev weight function $w(t)$,*

$$\int_{-1}^{1} w(t)F(t)dt \simeq \frac{\pi}{N}\sum_{j=1}^{N} F(t_j) \tag{4.6}$$

where $t_j = cos\frac{(j-\frac{1}{2})\pi}{N}$ $(j = 1, 2, \ldots, N)$ is the j^{th} root of $T_N(t)$, as in Equation (3.3).

Thus, the discretized version of Equation (4.5) is:

$$\begin{aligned} c_0 &= \frac{1}{N}\sum_{j=1}^{N} f(t_j)T_0(t_j) = \frac{1}{N}\sum_{j=1}^{N} f(t_j) \\ c_i &= \frac{2}{N}\sum_{j=1}^{N} f(t_j)T_i(t_j), \quad 1 \leq i \leq (n-1) \end{aligned} \tag{4.7}$$

It should be obvious that the complexity of computing each c_i is $O(N)$. Thus, the total complexity for approximating a time series is $O(nN)$ for computing

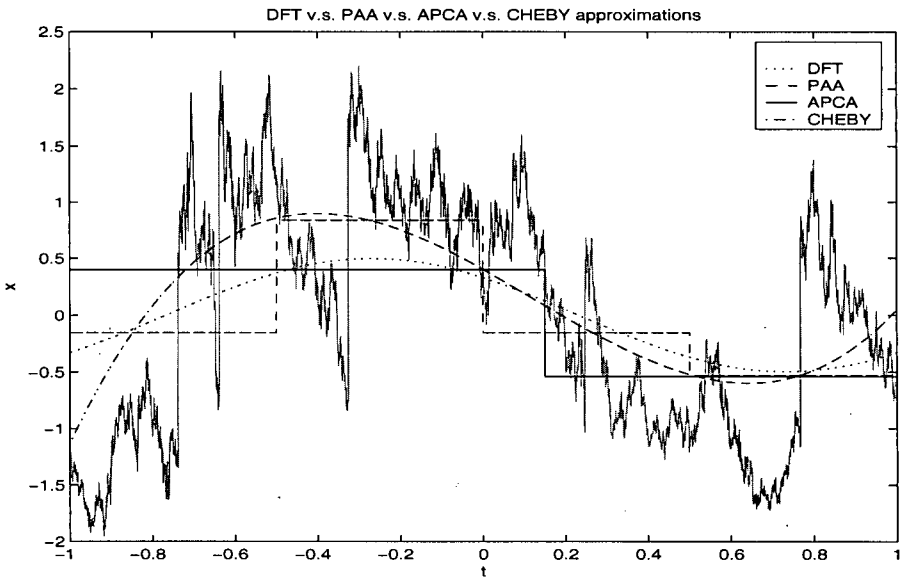


Figure 4.2: A Comparison of Approximation Schemes ($n = 4$)

all n coefficients. Because n is intended to be a small constant (e.g., ≤ 25), the complexity for Chebyshev approximation can thus be regarded as $O(N)$.

4.3 An Example

Figure 4.2 and Figure 4.3 show the time series of the opening stock price of a Fortune 500 company called ALCOA (ticker symbol: AA) for the period from February 28, 1978 to October 24, 2003 (for a total length of 6480 trading days). For $n = 4$, Figure 4.2 shows the original time series, and the Chebyshev, DFT, PAA and APCA approximations. Figure 4.3 shows the approximations for $n = 8$. The x-axis is normalized to the interval [-1, 1], and the y-axis is normalized according to the APCA framework.

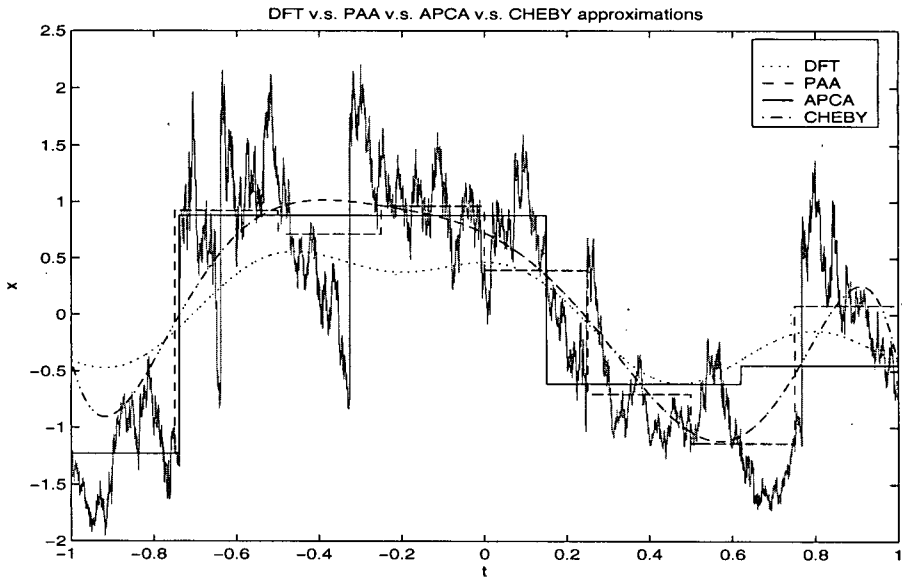


Figure 4.3: A Comparison of Approximation Schemes ($n = 8$)

Note that, if n is a power of 2, the PAA approximation is exactly the same as the wavelet transform. Also note that under APCA, because each piece is not of equal length, each piece requires two values for storage. Thus, for $n = 8$, there are only 4 pieces under APCA, as opposed to 8 pieces under PAA (and 8 coefficients for the Chebyshev approximation and DFT).

From the two Figures, it is easy to see that the Chebyshev approximation is different from the others. However, just based on the naked eye, it is hard to observe the minimax property of the Chebyshev approximation. Table 4.1 shows the maximum deviation under the various schemes, normalized into the y-range of [-2, 2.5].

The second and third column of the table show the situation for $n = 4$ and $n = 8$ respectively. Notice that for DFT as n increases, there is no guarantee that

Approximation Scheme	Maximum Deviation ($n = 4$)	Maximum Deviation ($n = 8$)
Chebyshev	1.88	1.84
DFT	2.00	2.09
APCA	2.35	2.23
PAA	2.31	2.28

Table 4.1: Maximum Deviations for Different Approximation Schemes

the maximum deviation decreases. Having said that, the general trend is that the maximum deviation decreases as n increases, as exhibited by the other schemes. In any event, for both values of n, the maximum deviation of the Chebyshev approximation is by far the smallest among the ones shown.

4.4 A Metric for Chebyshev Coefficients

Given two time series S_1, S_2, the previous section shows how to compute their corresponding vectors of Chebyshev coefficients, denoted by $\vec{C}_1, \vec{C}_2$ respectively. The next task is to define a distance function between the two vectors. Such a definition depends on the distance function used for the original time series S_1, S_2.

Definition 4.1 Let S_1, S_2 be two time series of length N, and let $\vec{C}_1, \vec{C}_2$ be the corresponding vectors of Chebyshev coefficients. Specifically, let $\vec{C}_1^T = [a_0, \ldots, a_m]$ and $\vec{C}_2^T = [b_0, \ldots, b_m]$. ($T$ denotes the transpose of the vector.) Define:

$$Dist_{cby}(\vec{C}_1, \vec{C}_2) = \sqrt{\frac{\pi}{2} \sum_{i=0}^{m} (a_i - b_i)^2} \tag{4.8}$$

The distance function $Dist_{cby}$ is basically a Euclidean distance function on the coefficients. It is weighted by the constant $\frac{\pi}{2}$ for the eventual Lower Bounding Lemma to work out. The following lemma is obvious.

Lemma 4.3 $Dist_{cby}$ is a metric distance function.

4.5 The Lower Bounding Lemma

We are now in a position to establish the Lower Bounding Lemma: $Dist_{cby}(\vec{C}_1, \vec{C}_2) \leq Dist_{euc}(S_1, S_2)$.

Given $S_1 = \langle (t_1, u_1), \ldots, (t_N, u_N) \rangle$, and $S_2 = \langle (t_1, v_1), \ldots, (t_N, v_N) \rangle$, we consider the time series $Z = \langle (t_1, u_1 - v_1), \ldots, (t_N, u_N - v_N) \rangle$. Let $z_j = u_j - v_j$ for all $1 \leq j \leq N$. Then it is clear that the Euclidean distance between S_1, S_2 satisfies the following equality:

$$Dist^2_{euc}(S_1, S_2) = \sum_{j=1}^{N} (u_j - v_j)^2 = \sum_{j=1}^{N} z_j^2 \tag{4.9}$$

Recall from Section 4.2 how an interval function is defined for a time series. Let the interval functions corresponding to S_1, S_2 and Z be f_1, f_2 and f_Z respectively. The lemma below is easy to establish by following Equations (4.3) and (4.4) in Section 4.2.

Lemma 4.4 For all $t \in [-1, 1]$, $f_Z(t) = f_1(t) - f_2(t)$.

Proof: $\forall i = 1, \ldots, N$, let $x \in I_i$ be arbitrarily given.

$$\begin{aligned} f_Z(x) &= \frac{g(x)}{\sqrt{w(x)|I_i|}} && \text{[Equation (4.4)]} \\ &= \frac{z_i}{\sqrt{w(x)|I_i|}} && \text{[Equation (4.3)]} \\ &= \frac{u_i - v_i}{\sqrt{w(x)|I_i|}} && \text{[Definition of } Z\text{]} \\ &= f_1(x) - f_2(x) && \text{[Equation (4.4)]} \end{aligned}$$

□

The above lemma can then be used to establish a useful result for Chebyshev approximation. Let us consider the Chebyshev approximation of Z based on Equation (4.7). Let the corresponding vector of Chebyshev coefficients be denoted as $\vec{C}_Z$. Given that Z is the "difference" between S_1, S_2, the following lemma says that the vector of Chebyshev coefficients preserves the difference.

Lemma 4.5 Let $\vec{C}_1, \vec{C}_2$ and $\vec{C}_Z$ be the vectors of Chebyshev coefficients for S_1, S_2 and Z respectively. Specifically, let $\vec{C}_1^T = [a_0, \ldots, a_m]$, $\vec{C}_2^T = [b_0, \ldots, b_m]$ and $\vec{C}_Z^T = [c_0, \ldots, c_m]$. Then for all $0 \leq i \leq m$, it is the case that $c_i = a_i - b_i$.

Proof: In the following, we only focus on c_i for $1 \leq i \leq m$; the situation is almost identical for c_0.

$$\begin{aligned} c_i &= \tfrac{2}{N} \textstyle\sum_{j=1}^{N} f_Z(t_j) T_i(t_j) && \text{[Equation (4.7)]} \\ &= \tfrac{2}{N} \textstyle\sum_{j=1}^{N} [f_1(t_j) - f_2(t_j)] T_i(t_j) && \text{[Lemma 4.4]} \\ &= a_i - b_i && \text{[Equation (4.7)]} \end{aligned}$$

□

Based on the above lemma and Definition 4.1, it is clear that:

$$Dist^2_{cby}(\vec{C}_1, \vec{C}_2) = \frac{\pi}{2} \sum_{i=0}^{m} c_i^2 \leq \frac{\pi}{2} \sum_{i=0}^{\infty} c_i^2 \tag{4.10}$$

Lemma 4.2 tells us that, the function $f(t)$, as defined in Equation (4.4), is $\mathcal{L}_2$-integrable with respect to the Chebyshev weight function. By invoking Theorem 3.7 in Chapter 3, we can finally put the various pieces together and conclude with the following theorem.

Theorem 4.2 (The Lower Bounding Lemma) Let S_1, S_2 be two time series, and $\vec{C}_1, \vec{C}_2$ be the corresponding vectors of Chebyshev coefficients. Then:

$$Dist_{cby}(\vec{C}_1, \vec{C}_2) \leq Dist_{euc}(S_1, S_2)$$

Proof:

$$\begin{aligned} Dist^2_{cby}(\vec{C}_1, \vec{C}_2) &\leq \tfrac{\pi}{2} \textstyle\sum_{i=0}^{\infty} c_i^2 && \text{[Equation (4.10)]} \\ &= \textstyle\int_{-1}^{1} \frac{f_Z^2(t)}{\sqrt{1-t^2}} dt && \text{[Theorem 3.7]} \\ &= \textstyle\sum_{j=1}^{N} |I_j| \frac{z_j^2}{|I_j|} && \text{[Equation (4.4)]} \\ &= \textstyle\sum_{j=1}^{N} z_j^2 \\ &= Dist^2_{euc}(S_1, S_2) && \text{[Equation (4.9)]} \end{aligned}$$

□

Notice that a key step in the above proof is $\int_{-1}^{1} \frac{f_Z^2(t)}{\sqrt{1-t^2}} dt = \sum_{j=1}^{N} |I_j| \frac{z_j^2}{|I_j|}$. This is due to the fact that the integrand is a step function, and hence stepwise integrable. The result of the integration at each step is the area under the curve, which is the width $|I_j|$ multiplied with the height $\frac{z_j^2}{|I_j|}$, that is, z_j^2.

4.6 Extension to the Weighted Euclidean Framework

In this Section, we shall extend the strict Euclidean framework into a generalized weighted version.

Definition 4.2 *For any two time series* $S_1 = \langle (t_1, u_1), \ldots, (t_N, u_N) \rangle$ *and* $S_2 = \langle (t_1, v_1), \ldots, (t_N, v_N) \rangle$, *the* weighted *Euclidean Distance between them is:*

$$Dist_{euc_w}(S_1, S_2) = \sqrt{\sum_{i=1}^{N} W_i (u_i - v_i)^2} \tag{4.11}$$

where $\{W_i\}_{i=1}^{N}$ *are nonnegative scalar weights.*

We redefine the function $g(t)$ (as in Equation (4.3)) as follows:

$$g(t) = \sqrt{W_i} v_i \quad \text{if } t \in I_i, \quad (\text{for } 1 \leq i \leq N) \tag{4.12}$$

where subintervals $\{I_i\}_{i=1}^{N}$ are defined in Equation (4.2).

We then define another function $f(t)$ as:

$$f(t) = \frac{g(t)}{\sqrt{w(t)|I_i|}} \quad \text{if } t \in I_i \quad (\text{for } 1 \leq i \leq N) \tag{4.13}$$

It is clear that $f(t)$ is $\mathcal{L}_2$-integrable with respect to the Chebyshev weight function. Therefore, we can apply Gauss-Chebyshev quadrature (Equation (4.6)) to obtain:

$$\begin{aligned} c_0 &= \tfrac{1}{N}\textstyle\sum_{j=1}^{N} f(t_j)T_0(t_j) = \tfrac{1}{N}\textstyle\sum_{j=1}^{N} f(t_j) \\ c_i &= \tfrac{2}{N}\textstyle\sum_{j=1}^{N} f(t_j)T_i(t_j), \quad 1 \leq i \leq (n-1) \end{aligned} \tag{4.14}$$

where $t_j = cos\frac{(j-\frac{1}{2})\pi}{N}$ $(j = 1, 2, \ldots, N)$ is the j^{th} root of $T_N(t)$.

Definition 4.3 Let S_1, S_2 be two time series of length N, and let $\vec{C}_1, \vec{C}_2$ be the corresponding vectors of Chebyshev coefficients obtained by applying Equation (4.12) through Equation (4.14). Specifically, suppose $\vec{C}_1^T = [a_0, \ldots, a_m]$ and $\vec{C}_2^T = [b_0, \ldots, b_m]$. ($T$ denotes the transpose of the vector.) Define:

$$Dist_{cby_w}(\vec{C}_1, \vec{C}_2) = \sqrt{\frac{\pi}{2}\sum_{i=0}^{m}(a_i - b_i)^2} \tag{4.15}$$

Thus, the Lower Bounding Lemma for the weighted Euclidean framework is:

Theorem 4.3 Let S_1, S_2 be two time series, and $\vec{C}_1, \vec{C}_2$ be the corresponding vectors of Chebyshev coefficients. Then:

$$Dist_{cby_w}(\vec{C}_1, \vec{C}_2) \leq Dist_{euc_w}(S_1, S_2)$$

Proof: Let $Z = \langle (t_1, z_1), \ldots, (t_N, z_N)$, where $z_j = u_j - v_j$ for $j = 1, \ldots, N$. Note that, if the interval functions corresponding to S_1, S_2 and Z are f_1, f_2 and f_Z respectively (Equations (4.12) and (4.13)), then both Lemma 4.4 and Lemma 4.5

are also valid for our weighted Euclidean framework. Therefore,

$$\begin{aligned}
Dist^2_{cby_w}(\vec{C}_1, \vec{C}_2) &\leq \tfrac{\pi}{2} \textstyle\sum_{i=0}^{\infty} c_i^2 && \text{[Equation (4.10)]} \\
&= \textstyle\int_{-1}^{1} \frac{f_Z^2(t)}{\sqrt{1-t^2}} dt && \text{[Theorem 3.7]} \\
&= \textstyle\sum_{j=1}^{N} |I_j| \frac{(\sqrt{W_j} z_j)^2}{|I_j|} && \text{[Equation (4.13)]} \\
&= \textstyle\sum_{j=1}^{N} W_j z_j^2 \\
&= Dist^2_{euc}(S_1, S_2) && \text{[Equation (4.11)]}
\end{aligned}$$

□

Chapter 5

Indexing Multidimensional Trajectories

So far, we have established indexing based on Chebyshev approximation for 1-dimensional time series. In this Chapter, we extend the framework to d-dimensional ($d \geq 1$) spatiotemporal trajectories. Then we present algorithms for indexing and kNN searches.

5.1 Lower Bounding for the Multidimensional Case

Let S be a d-dimensional spatiotemporal trajectory of the form $\langle (t_1, \vec{v}_1), \ldots, (t_N, \vec{v}_N) \rangle$, where $\vec{v}_i$ is of arity d. Let the d dimensions be $\{Dim_1, \ldots, Dim_d\}$. Then S is decomposed into d 1-dimensional series: $S_{Dim_1}, \ldots, S_{Dim_d}$. Let each of these series S_{Dim_i} be approximated and represented with the vector $\vec{C}_i$ of Chebyshev coefficients. The vector $\vec{C}_i$ is of arity n_i, and needs not be of the same arity as $\vec{C}_j$ for $j \neq i$. Finally, let $\vec{C}$ be the vector of Chebyshev coefficients for S, i.e., $\vec{C}^T = [\vec{C}_1^T, \ldots, \vec{C}_d^T]$.

We generalize Definition 4.1 to give a metric distance function between two

vectors of Chebyshev coefficients for two d-dimensional trajectories.

Definition 5.1 Let S, R be d-dimensional spatiotemporal trajectories. Let their vectors of Chebyshev coefficients be $\vec{C}^T = [\vec{C}_1^T, \ldots, \vec{C}_d^T]$ and $\vec{D}^T = [\vec{D}_1^T, \ldots, \vec{D}_d^T]$ respectively. Define:

$$Dist_{cby}(\vec{C}, \vec{D}) = \sqrt{\sum_{i=1}^{d} Dist_{cby}^2(\vec{C}_i, \vec{D}_i)}$$

The following corollary is a simple extension of Theorem 4.2 generalizing the the Lower Bounding Lemma from 1-dimensional to d-dimensional trajectories. This is because the d-dimensional distance is based on the sum-of-squares distances along each dimension.

Corollary 5.1 Let S, R be d-dimensional spatiotemporal trajectories, and $\vec{C}, \vec{D}$ be the corresponding vectors of Chebyshev coefficients. Then:

$$Dist_{cby}(\vec{C}, \vec{D}) \leq Dist_{euc}(S, R)$$

5.2 Algorithms for Building and Searching A Single Index

Having established the Lower Bounding Lemma in Corollary 5.1 for the d-dimensional case, we can build an index of Chebyshev coefficients. Figure 5.1 shows a skeleton of an algorithm which takes M d-dimensional spatiotemporal trajectories, obtains the Chebyshev coefficients for each trajectory, and inserts the vectors of coefficients into a multidimensional index.

Recall from Section 4.2 that the complexity of step (4) of the algorithm is $O(N)$, where N is the length of each trajectory. Thus, it is clear from Figure 5.1 that building the index takes $O(dMN)$ time.

```
Algorithm BuildOneIndex(DB, Index, n_1, ..., n_d) {
  /* input: a database DB of M d-dimensional trajectories */
  /* input: Index, a multidimensional index which may already
            contain some entries */
  /* input: n_i (1 ≤ i ≤ d) denotes the number of Chebyshev
            coefficients to be used for the i^th dimension */
  /* output: the trajectories approximated and added to Index */
  for each trajectory S {
      (1)   initialize C to be empty
      (2)   project S to its d dimensions {Dim_1, ..., Dim_d}
            creating S_Dim_1, ..., S_Dim_d
      (3)   for (1 ≤ i ≤ d) {
      (4)         apply Equations (4.2) to (4.7) to S_Dim_i
      (5)         add all the computed n_i coefficients to C
            } /* end for-loop */
      (6)   insert the coefficients in C as a single
            multi-dimensional point into Index
  } /* end for-loop */
} /* end algorithm */
```

Figure 5.1: Algorithm for Building a Single Index of Chebyshev Coefficients

Next we consider range and kNN searches. In both cases, the search is rather straightforward, following the GEMINI framework [9]. Figure 5.2 shows a skeleton of the range search algorithm, and Figure 5.3 shows a skeleton of the kNN search algorithm.

5.3 Algorithms for Building and Searching Multiple Indices

A keen reader would notice that the algorithm presented in Figure 5.1 does not scale very well with the dimensionality of the trajectories. For example, if the trajectories have a dimensionality of 10, then even as few as four coefficients per dimension would result in an index of 40 dimensions, which is clearly unacceptable, due to the dimensionality curse problem of multidimensional indices. In this Section, we

```
Algorithm RangeSearchOneIndex(Q, Index, r) {
  /* input: a d-dimensional query trajectory Q */
  /* input: the index of Chebyshev coefficients Index */
  /* input: a radius r for range search */
  /* output: all trajectories within distance r of Q
             with respect to Dist_euc */
  (1)  apply Equations (4.2) to (4.7) to obtain the vector of
       coefficients for Q
  (2)  find all trajectories in Index within distance r of Q using Dist_cby
  (3)  retrieve from disk the corresponding (full) trajectories
  (4)  compute the true distances using Dist_euc and
       discard all the false positives
} /* end algorithm */
```

Figure 5.2: Algorithm for a Range Search in a Single Index

shall propose an alternative approach, in which we "distribute" the Chebyshev coefficients of trajectories to multiple indices instead of using a single index. Figure 5.4 illustrates how we build multiple indices based on the Chebyshev coefficients of each trajectory in a collection of M d-dimensional spatiotemporal trajectories. Figure 5.5 gives an algorithm for range search, and Figure 5.6 is for kNN search.

Note that, in Step (5) of Figure 5.4, there are many ways to choose l. In fact, there are exponentially many ways to distribute the sets of Chebyshev coefficients to indices, and we shall leave the optimization of distributions as a topic of future research.

5.4 Properties of Chebyshev Indexing

Recall from Section 1.2.3 that we have outlined a list of desirable properties for indexing techniques. In this Section, we shall argue that Chebyshev indexing satisfies all those criteria:

- Indexing is intrinsically much faster than sequential scanning in terms of query

```
Algorithm kNNSearchOneIndex(Q, Index, k) {
  /* input: a d-dimensional query trajectory Q */
  /* input: the index of Chebyshev coefficients Index */
  /* input: k a positive integer */
  /* output: the k most similar trajectories to Q
             with respect to Dist_euc */
  (1)  apply Equations (4.2) to (4.7) to obtain the vector of
       coefficients for Q
  (2)  find the k-nearest neighbours to Q in Index using Dist_cby
  (3)  retrieve from disk the corresponding (full) trajectories
  (4)  compute the true distances using Dist_euc and
       record the maximum max
  (5)  invoke the range search RangeSearchOneIndex(Q, Index, max)
  (6)  retrieve from disk the corresponding (full) trajectories
  (7)  compute the true distances using Dist_euc and
       retain the nearest k trajectories
} /* end algorithm */
```

Figure 5.3: Algorithm for a kNN Search in a Single Index

performance, since it partitions the search space into hierarchical components and does not require access to every data page. As long as the dimensionality is not too high, indexing always dominates sequential scanning. This is also confirmed by our experimental results in Chapter 6.

- As n is quite small, our indexing structure does not take much space and is supposed to be memory resident.

- Chebyshev indexing allows both whole matching and subsequence matching.

- For each trajectory in the database, we compute its Chebyshev coefficients based only on the trajectory itself. Unlike SVD, Chebyshev approximation is incremental.

- No false negatives are guaranteed (Theorem 4.2 and Corollary 5.1).

- It is clear to see from Figure 5.1 that the index building process takes $O(dMN)$

```
Algorithm BuildMultipleIndices(DB, Index_1, ..., Index_g, n_1, ..., n_d) {
  /* input: a database DB of M d-dimensional trajectories */
  /* input: Index_1, ..., Index_g, g indices which may already
            contain some entries */
  /* input: n_i (1 ≤ i ≤ d) denotes the number of Chebyshev
            coefficients to be used for the i^th dimension */
  /* output: the trajectories approximated and indexed */
  for each trajectory S {
      (1)   initialize C_1, ..., C_g to be empty where C_j is
            the collection of coefficients to be inserted into Index_j
      (2)   project S to its d dimensions {Dim_1, ..., Dim_d}
            creating S_Dim_1, ..., S_Dim_d
      (3)   for (1 ≤ i ≤ d) {
      (4)         apply Equations (4.2) to (4.7) to S_Dim_i
                  to get the set of Chebyshev coefficients V_i for S_Dim_i
      (5)         add all the n_i coefficients in V_i to C_l for some l
            } /* end for-loop */
      (6)   for (1 ≤ j ≤ g) {
      (7)         insert the coefficients in C_j as a single
                  multidimensional point into Index_j
            } /* end for-loop */
  } /* end for-loop */
} /* end algorithm */
```

Figure 5.4: Algorithm for Building Multiple Indices of Chebyshev Coefficients

time. Thus, Chebyshev indexing is scalable with respect to all of d, M and N.

- As shown in Chapter 4, our framework can handle both the Euclidean distance and the weighted Euclidean distance.

```
Algorithm RangeSearchMultipleIndices(Q, Index_1, ..., Index_g, r) {
  /* input: a d-dimensional query trajectory Q */
  /* input: the indices of Chebyshev coefficients */
  /* input: a radius r for range search */
  /* output: all trajectories within distance r of Q
             with respect to Dist_euc */
  (1)   apply Equations (4.2) to (4.7) to obtain the vector C of
        coefficients for Q
  (2)   for (1 ≤ j ≤ g) {
  (3)       project C onto the dimensions on which Index_j is based
  (4)       find all trajectories in Index_j within r of Q using Dist_cby
  (5)       retrieve from disk the corresponding (full) trajectories
  (6)       compute the true distances using Dist_euc and
            discard all the false positives
        } /* end for-loop */
} /* end algorithm */
```

Figure 5.5: Algorithm for a Range Search in Multiple Indices

```
Algorithm kNNSearchMultipleIndices(Q, Index_1, ..., Index_g, k) {
  /* input: a d-dimensional query trajectory Q */
  /* input: the indices of Chebyshev coefficients */
  /* input: k a positive integer */
  /* output: the k most similar trajectories to Q
             with respect to Dist_euc */
  (1)   apply Equations (4.2) to (4.7) to obtain the vector C of
        coefficients for Q
  (2)   initialize set U to be empty
  (3)   for (1 ≤ j ≤ g) {
  (4)       project C onto the dimensions on which Index_j is based
  (5)       find the k-nearest neighbours to Q in Index_j using Dist_cby
  (6)       add those neighbours into set U
        } /* end for-loop */
  (7)   retrieve from disk the corresponding (full) trajectories
  (8)   compute the true distances using Dist_euc and
        record the k^th smallest distance ks
  (9)   invoke the range search
        RangeSearchMultipleIndices(Q, Index_1, ..., Index_g, ks)
  (10)  retrieve from disk the corresponding (full) trajectories
  (11)  compute the true distances using Dist_euc and
        retain the nearest k trajectories
} /* end algorithm */
```

Figure 5.6: Algorithm for a kNN Search in Multiple Indices

Chapter 6

Experimental Evaluation

6.1 Data Sets and Programs Used

We conducted an experimental evaluation on many real data sets. The following table provides a summary of those reported here.

The Stocks data set consists of the daily opening prices of 500 companies traded on the New York Stock Exchange for the past 25 years. The data set was obtained from http://finance.yahoo.com. The ERP data set was provided to us by Eammon Keogh. Both of these data sets consist of long 1-dimensional time series.

Name	Dimensionality	Number of Trajectories	Trajectory Length
Stocks	1	500	6480
ERP	1	496	6396
NHL	2	5000	256
Slips	3	495	400
Kungfu	3	495	640
Angle	4	657	640

Table 6.1: Data Sets Used

The NHL data set consists of 5000 National Hockey League players' 2-dimensional trajectories, each of length 256 time points. The trajectories were obtained by digitizing the Philadelphia Flyers' hockey games during the NHL 2001-2002 season. The data were provided to us by an electronic games company.

The Slips, Kungfu and Angle data sets were obtained from http://www.e-motek.com/entertainment/index.htm. The site belongs to a company which operates a motion capture facility for use by electronic game developers and medical professionals. The Slips data are 3-dimensional positions of body joints of a person slipping down and trying to stand up. The Kungfu data are 3-dimensional positions of body joints of a person playing kung fu. Finally, the 4-dimensional Angle data record the four angles of the body joints of a person playing kung fu.

The aforementioned data sets vary in dimensionality and length. But they are rather small in number (not necessarily in total size). To complement the situation so that scalability can be tested more thoroughly, we implemented a trajectory generator. Specifically, it uses a simple mixture contamination model, i.e., $Z(t) = (1-w)P(t) + w\mathcal{N}$. $Z(t)$ is the generated 1-dimensional time series. With a probability of $(1-w)$ (e.g., $w = 0.1$), the generated values follow the values of a polynomial $P(t)$ of a specified degree m (e.g., from 4 to 20). But with a probability w, Gaussian noise $\mathcal{N}(0,1)$ is introduced. The polynomial $P(t)$ of degree m has m roots, which are picked randomly within the range [-1,1]. This polynomial is then expanded and scaled. For a d-dimensional trajectory, the above generation procedure is invoked d times to generate the data on each dimension separately.

We implemented Chebyshev approximation in C++, corresponding to Equations (4.3) to (4.7). Recall from Figure 4.2 that there are various well-known schemes for time series indexing. As the study in [21] shows convincingly that APCA is al-

most always the best algorithm, we focus our empirical comparison only with APCA. We obtained the APCA code from Eammon Keogh, for which we are thankful. The APCA code was implemented in Matlab. We implemented the BuildIndex, RangeSearch and kNNSearch procedures shown in Figures 5.1 to 5.3.

Finally, many multidimensional indexing structures have been developed. See [10] for a comprehensive survey. For the results reported here, we used the DR-tree package developed by Christos Faloutsos and his group.

To come up with a "straw man" algorithm for a comparative analysis for d-dimensional trajectories, we developed another version of the BuildIndex procedure by replacing line (4) in Figure 5.1 with the APCA code. Similarly, we developed APCA versions of RangeSearch and kNNSearch procedures by basically replacing line (1) in Figures 5.2 and 5.3 with the APCA code.

6.2 Comparison Criteria: Pruning Power and Search Time

Note that because the APCA code is implemented in Matlab, and line (4) in BuildIndex is looped many times, it is unfair to compare the execution times of the two BuildIndex procedures directly. However, the situation is different for the RangeSearch and kNNSearch procedures. Because line (1) in Figures 5.2 and 5.3 is called only once per query, we did not measure the execution time of this line, but measured and compared the execution times of the rest of the procedures. We feel that this is a fair comparison between Chebyshev and APCA on their search performance with indexing taken into account – modulo the time taken to approximate the initial query.

In addition to the execution times, we also compared the pruning power of the two schemes. Our definition of pruning power is slightly simpler than the one used by Keogh et al. [21]. Adopting a branch-and-bound strategy, we used a sequential scan to conduct a kNN search. Specifically, let $S_1, \ldots, S_k$ be the *current* k-nearest trajectories based on their real Euclidean distances to query Q. Let max_{euc} be the maximum distance according to these k current best. For the next trajectory R to be evaluated, we compare max_{euc} with $Dist_{cby}(\vec{C}_Q, \vec{C}_R)$, where $\vec{C}_Q, \vec{C}_R$ denote the vectors of coefficients of query Q and trajectory R respectively. If max_{euc} is smaller, then by the Lower Bounding Lemma, R cannot possibly be nearer, thus saving one calculation of the real Euclidean distance $Dist_{euc}(Q, R)$. Otherwise, the real distance $Dist_{euc}(Q, R)$ is computed and the current k-nearest trajectories and max_{euc} may need to be updated. Thus, the pruning power essentially measures the percentage of *saved* real Euclidean distance calculations, as a result of the approximation. Note that this percentage depends on the initial k trajectories. To overcome this bias, we define the pruning power to be the average percentage of saved calculations over 10 randomly picked queries.

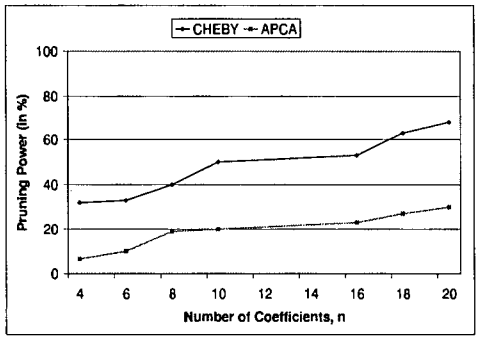
Apart from search times, we feel that it is essential to compare the pruning power for two reasons. First, in a search time comparison with indexing included, there are biases introduced by implementation details, including the choice of the indexing structure. A pruning power comparison is free of those implementation biases. Second, as indexing is included in a search time comparison, the dimensionality curse of the indexing structure may dominate at some point, and mask the true pruning effectiveness of the approximation schemes. The latter is best measured directly by a pruning power comparison.

6.3 Pruning Power Comparison: Real Data Sets

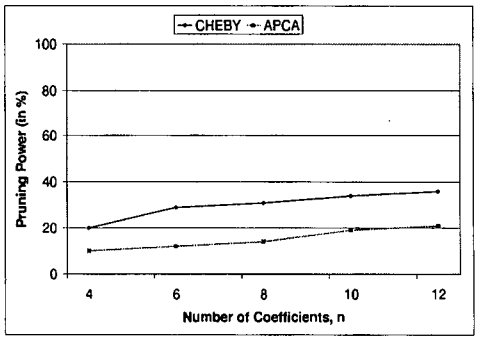
Figure 6.1 compares the pruning power of Chebyshev and APCA approximations. The value of k is 10 (i.e., 10-nearest neighbours). The figure consists of six graphs, one for each of the six real data sets. In all cases, the x-axis shows varying values of n (i.e., the number of coefficients allowed in the approximation). Notice that because APCA approximates a trajectory with variable-length pieces, each piece requires two coefficients. Thus, $n = 2$ for APCA corresponds to a single piece, which has little pruning power, and hence is omitted. Furthermore, the APCA code requires that the length of a trajectory be a multiple of n. Thus, the values plotted on the x-axes for the six graphs vary from data set to data set. For example, the NHL trajectories are each of length 256; the values of n that can be used must be powers of 2. The y-axis shows the percentage of saved Euclidean distance calculations.

Let us first take a closer look at the two 1-dimensional data sets: Stocks and ERP. As expected, as n increases, the pruning power increases. For the Stocks data, as n varies from 4 to 20, the pruning power of Chebyshev approximation increases from around 35% to about 70%. In contrast, the pruning power of APCA only increases from 8% to 30%. In other words, even if 20 coefficients are used for the APCA to approximate each trajectory, the pruning power it delivers is less than what the Chebyshev approximation can deliver with 4 coefficients. Thus, there is at least a 5-fold improvement in the dimensionality of the approximation. For the ERP data, as n varies from 4 to 12, the pruning power of Chebyshev approximation changes from 20% to 35%, whereas that of APCA changes from 10% to 20%. Thus, it takes APCA 12 coefficients to deliver the same pruning power as 4 Chebyshev coefficients can do.

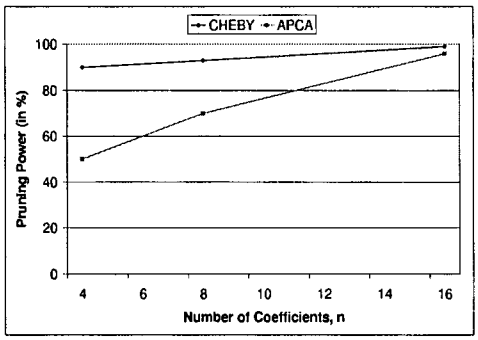
Let us turn our attention to higher-dimensional trajectories. Note that the



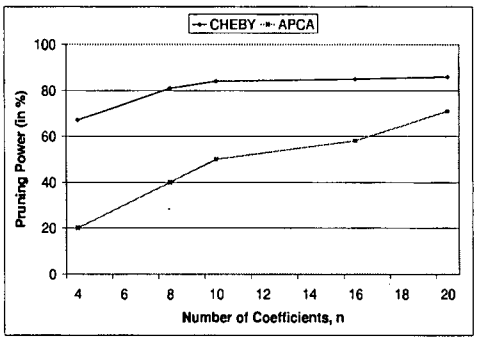
(a) 1-D Stocks data



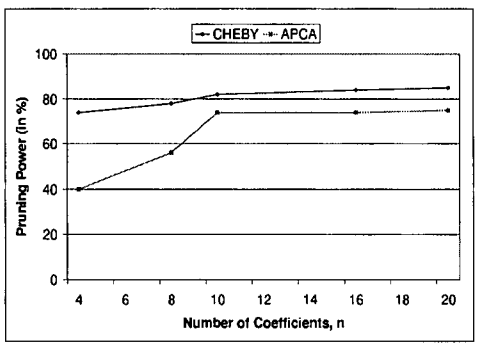
(b) 1-D ERP data



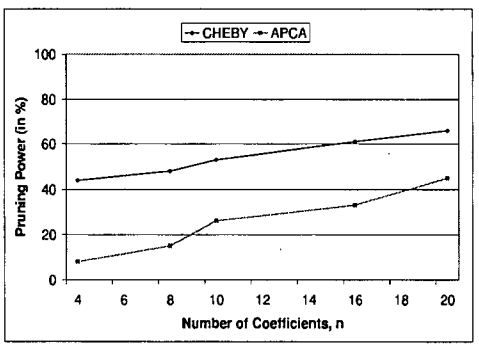
(c) 2-D NHL data



(d) 3-D Slips data



(e) 3-D Kungfu data



(f) 4-D Angle data

Figure 6.1: Pruning Power Comparisons: Real 1- to 4-Dimensional Data Sets

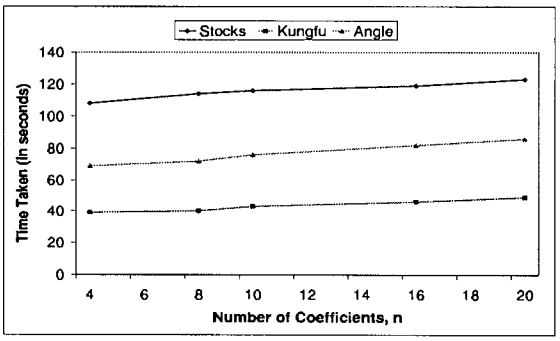


Figure 6.2: Computing Chebyshev Coefficients

value of n represents the number of coefficients for *each* dimension. For instance, for the graph in Figure 6.1(f), $n = 20$ corresponds to a total of 80 coefficients used for approximating the given 4-dimensional data. Note that we are not suggesting that in practice, we should build an 80-dimensional index. Rather, we focus here on examining pure pruning effectiveness, independent of the index structure. To continue with Figure 6.1(f), we observe that it takes APCA 20 coefficients to deliver what 4 Chebyshev coefficients can deliver, representing a 5-fold difference in dimensionality of approximation. Similar observation applies to the 2-dimensional and 3-dimensional data sets.

6.4 Building Time and the Choice of n

The above discussion focuses on comparing the dimensionality of Chebyshev approximation and APCA. Here we focus solely on Chebyshev approximation. In all the graphs shown in Figure 6.1, the larger the value of n, the higher the pruning power is. The obvious question to ask then is how large n could be. There are two key factors. The first factor is the dimensionality of the index, as the dimensionality

curse on the index structure may put a limit on the value of n. This issue will be addressed in Section 6.6.

The second factor is the computation time of Chebyshev approximation. The key question here is how fast the CPU time taken to compute the Chebyshev coefficients grows with respect to n. Figure 6.2 answers this question for the 1-dimensional Stocks data, 3-dimensional Kungfu data, and 4-dimensional Angle data. We omit the others to save space, as the same conclusion can be drawn. The x-axis of the graph shows varying values of n, and the y-axis shows the number of seconds in CPU time to compute the Chebyshev coefficients for all the trajectories. The machine used was an Intel PC with a single 1.8 GHz processor and 256 Mbytes of RAM. The timing figures represent averages of 10 randomly picked queries.

Across the three curves in the graph, the absolute time taken is not that important, as the time depends on the size and length of each data set. What is important, however, is that for each curve, the time taken is shown to be linear with respect to n, as predicted from the earlier equations. What is noteworthy is how small the rate of growth turns out to be, i.e., the slope of the "straight" line. The reason is that, as shown in Equation (4.7), the bottleneck of the computation of the coefficients is for computing $f(t_j)$ for all $1 \leq j \leq N$. This computation is done only once for all the n coefficients. The significance of this observation is that as long as increasing n delivers additional pruning power, the incremental building cost is not an obstacle at all. Of course, this does not represent the final verdict on the choice of n; later in Section 6.6 when indexing is included in our measurement, we shall return to this issue.

We do not include the building time for APCA here, as it takes at least an order of magnitude longer. But this is not a fair comparison as APCA is implemented

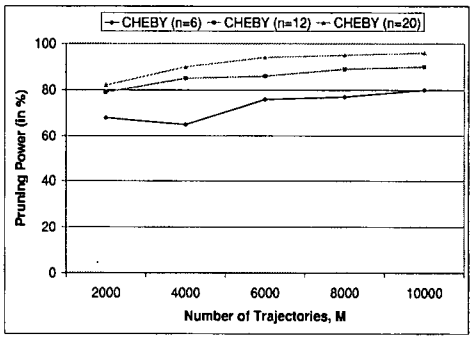
in Matlab, whereas our Chebyshev code is implemented in C++.

6.5 On Scalability: Generated Data

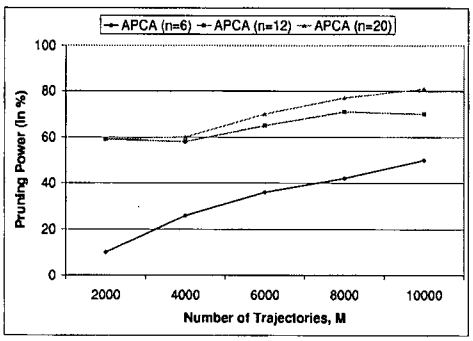
So far, all the empirical evaluations are based on the real data sets, all of which are small in M, the number of trajectories. Here we used the generated data sets, as described in Section 6.1. Figure 6.3 shows a representative situation – based on a 3-dimensional generated data set with an underlying polynomial of degree 10 and trajectory length of 720. Figures 6.3(a) and (b) compare the pruning power of Chebyshev approximation and APCA. The x-axis shows varying values of data set size M, and the y-axis shows the percentage of saved Euclidean calculations. To avoid crowding the graph, we only show the situation when $n = 6, 12$ and 20.

Recall from the earlier pruning power discussion that Chebyshev approximation can deliver a 3- to 5-fold reduction in the dimensionality of the approximation. Let us examine the first two graphs in Figure 6.3 to see if the same conclusion can be drawn for larger data sets. Take $M = 2000$ as the first example. The pruning power of Chebyshev approximation using $n = 6$ coefficients is roughly the same as the pruning power of APCA using $n = 20$ coefficients. Similar observations can be made for all other values of M shown in the graphs. Thus, this confirms the superiority of Chebyshev approximation for both real and generated data sets.

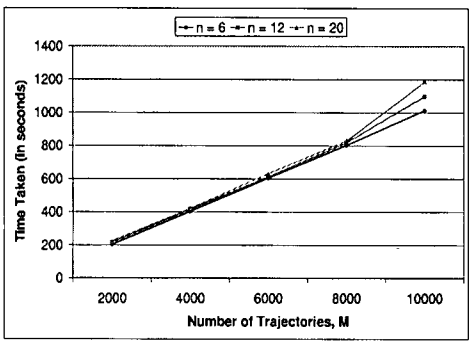
Figure 6.3(c) shows that the time taken to compute Chebyshev coefficients is linear with respect to M. This shows the scalability of Chebyshev approximation. Furthermore, the graph shows that there is little difference in time whether 6 or 20 coefficients are being computed, confirming an earlier observation surrounding Figure 6.2. This shows that the computation of Chebyshev coefficients is far more affected by the data set size M than by the number of coefficients n.



(a) Chebyshev pruning power

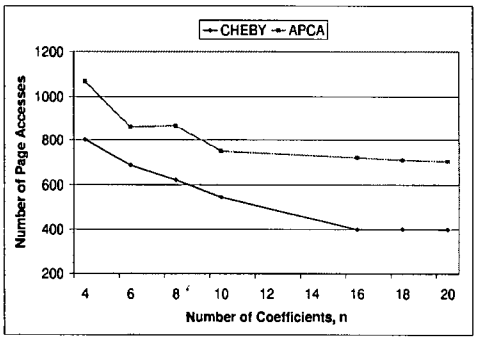


(b) APCA pruning power

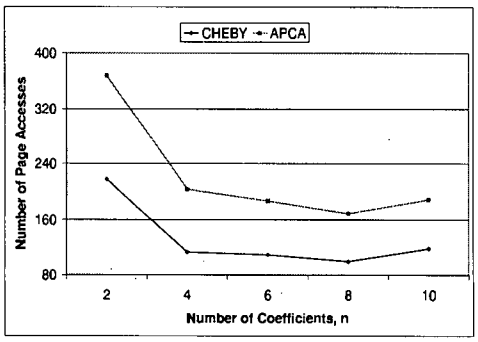


(c) Chebyshev building time

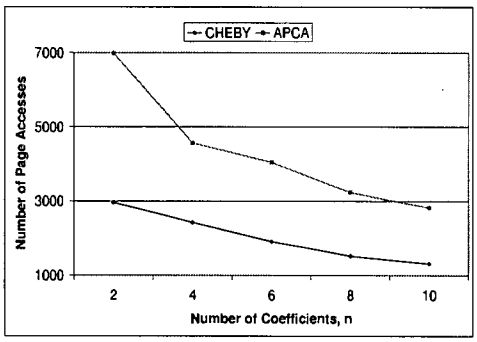
Figure 6.3: Scalability: Pruning Power and Building Time



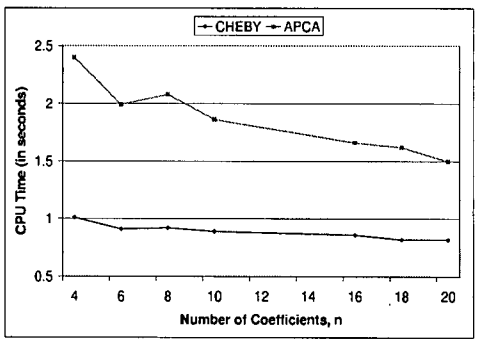
(a) 1-D Stocks data: I/O cost



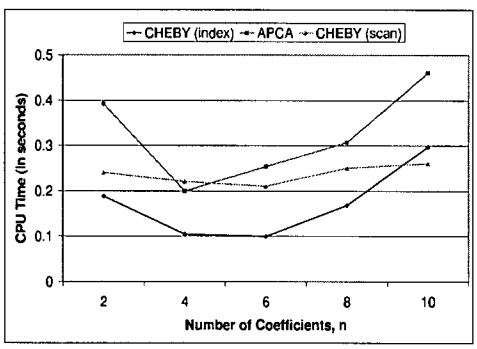
(b) 3-D Kungfu data: I/O cost



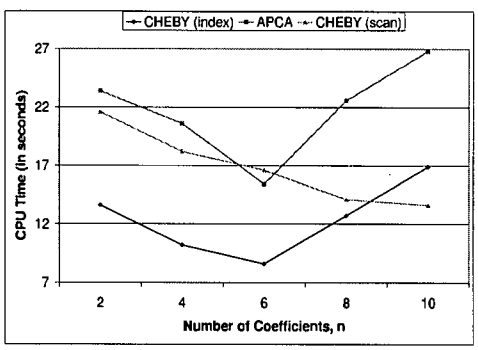
(c) 3-D Generated data: I/O cost



(d) 1-D Stocks data: CPU time



(e) 3-D Kungfu data: CPU time



(f) 3-D Generated data: CPU time

Figure 6.4: Search Time Comparison: Indexing Included

6.6 Comparisons with Indexing Included

So far, our discussions have not taken account of the indexing structure. The comparison between Chebyshev and APCA is based on pruning power and sequential scans. In the remainder of this section, we compare these two schemes with indexing included – in terms of both I/O cost and CPU cost. I/O cost, if reported in seconds, may depend heavily on implementation and experimentation details, such as buffer space, speed of a random page read, etc. To eliminate these details, we report I/O cost as the sum of the number of index nodes/pages accessed and the number of page reads required to retrieve the specific trajectories needed by the kNNSearch procedure. We used a page size of 10Kbytes.

CPU time includes the time taken to navigate the index nodes, the time taken to compute the lower bounded distances $Dist_{cby}(\vec{C}_Q, \vec{C}_S)$, and the time taken to compute the real Euclidean distances $Dist_{euc}(Q, S)$, whenever needed. As discussed before, the time taken to perform the initial approximation of the query is not included, due to the fact that the APCA code is written in Matlab. Even though the exact CPU time is highly dependent on the size of the data set and the length of the trajectories, the CPU time can at least be used as a relative measurement between Chebyshev and APCA. Like the figures reported on pruning power, the timing figures reported here on I/O and CPU costs represent the averages over 10 randomly picked queries.

6.6.1 I/O Cost Comparison

Figure 6.4 shows the I/O and CPU costs for the Stocks data, the Kungfu data and the 3-dimensional generated data with 10,000 trajectories each of length 720. The x-axis of the graphs shows the number of coefficients used, n, for *each* dimension.

For graphs (a) to (c), the y-axis shows the I/O cost in page accesses. Given the differences in length and number of trajectories in each data set, the absolute values in graphs (a) to (c) are relatively unimportant; what is important are the curves within each graph.

For the 1-dimensional Stocks data in graph (a), the reduction in page accesses as n increases flattens off for n beyond 16. For the 3-dimensional Kungfu data in graph (b), the number of page accesses reaches a minimum when $n = 8$, corresponding to a 24-dimensional index. Beyond that, the dimensionality curse on the index structure sets in, and the number of total page accesses starts to rise. For the 3-dimensional generated data in graph (c), there is not yet any observed increase in total page accesses beyond $n = 8$. However, recall that total page accesses come from two sources: the number of data pages and the number of index nodes/pages. As n increases, the former decreases due to the increase in pruning power. In contrast, the latter goes up due to increasing dimensionality, and accounts for a larger and larger percentage of total page accesses. Eventually, the latter dominates the former.

Dimensionality curse aside, the number of page accesses required by Chebyshev approximation in all cases is about 50% to 60% that of APCA. This improvement is highly consistent with the pruning power results shown earlier in Figure 6.1 and Figure 6.3.

6.6.2 CPU Cost Comparison

For graphs (d) to (f), the y-axis shows the CPU time taken (in seconds) for the entire kNNSearch. Within each graph, we show the times taken by Chebyshev and APCA, with indexing included. Furthermore, whenever the sequential scan strategy

(as described in Section 6.2) becomes competitive, the timing figures for scans are included as well. The key difference between indexing and sequential scans is that with the former, the dimensionality curse on the indexing structure will set in sooner or later. In graph (d), for the 1-dimensional Stocks data, the minimum CPU time occurs when $n = 20$. But for the 3-dimensional Kungfu and generated data in graphs (e) and (f), the minimum CPU time occurs when $n = 4$ or $n = 6$ (corresponding to a 12-dimensional or 18-dimensional index). And if the total time is considered by summing up the CPU and I/O costs, the best situation is when $n = 6$.

As expected and consistent with the literature [55], our sequential scan strategy starts to dominate indexing. For graphs (e) and (f), this occurs when $n = 10$. As our sequential scan strategy is not optimized, it is conceivable that a more optimized sequential scan procedure may dominate even earlier. Recall from Figure 6.1 that the pruning power continues to grow beyond $n = 8$. Thus, it is important to include sequential scans as a viable alternative to indexing for spatiotemporal trajectories.

For the comparison between Chebyshev and APCA, again the former dominates in CPU time taken. This is consistent with all the previous comparisons on pruning power and I/O cost. But besides pruning power, there is an additional reason why the CPU time for Chebyshev is lower than that for APCA. As defined in Definition 4.1, the computation of the distance between two vectors of Chebyshev coefficients is $O(n)$. However, based on the distance measure given in [21], the corresponding computation between two vectors of APCA coefficients is in fact $O(N)$ (where N is the length of each trajectory), which requires extra CPU time.

6.6.3 Recommendations

In closing, we make the following suggestions regarding indexing for d-dimensional spatiotemporal trajectories. They are based on the DR-tree we used and should be adjusted depending on the choice of the index structure. For the 1-dimensional case, using $n = 20$ Chebyshev coefficients appears to be the best. For the 2-dimensional case, the suggested value of n is 8-12 for each dimension. The corresponding suggestion is 4-6 for 3-dimensional trajectories. And finally, 4 coefficients for each dimension are recommended for 4-dimensional trajectories. For higher dimensionality, or for additional pruning power, a sequential scan using a higher number of Chebyshev coefficients is recommended.

6.7 A Single Index vs. Multiple Indices

Until now, the highest dimensionality of trajectories in all the experimental data sets used is 4, but there are many examples of higher dimensionality in real life. In this Section, we shall explore the performance of our algorithms with respect to the dimensionality of trajectories, focusing only on Chebyshev approximation.

We use our trajectory generator to generate five databases, each having $M = 5000$ trajectories, $N = 500$ in length, but with different dimensionalities ($d = 2, 4, 6, 8, 10$). We compare the performance under three situations (where $n = 3$ and 5):

1. SingleIndex corresponds to the scenario where n coefficients are computed for each attribute and a single index of $3d$ dimensions is used.

2. MultipleIndices corresponds to the scenario where n coefficients are computed for each attribute and the coefficients for two attributes are grouped into one

index. In total, $\frac{d}{2}$ indices of 6 dimensions are used.

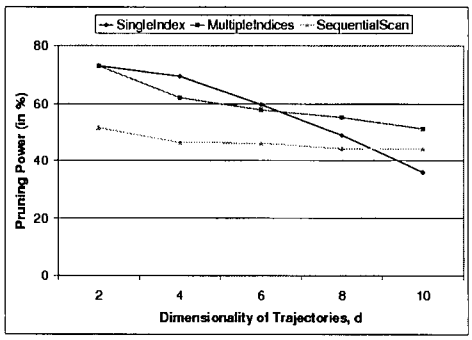
3. SequentialScan corresponds to the scenario where n coefficients are computed for each attribute and kNN search is conducted using sequential scan.

Figure 6.5 and Figure 6.6 show the comparisons in terms of pruning power, I/O cost and CPU cost, respectively. They both display a similar trend as dimensionality increases from 2 to 10. When d is small, putting all the coefficients together in one single index appears to be the best, but as d gets larger, the dimensionality curse sets in, and sequential scanning starts to dominate. It is interesting to note that, when $d \geq 6$, distributing Chebyshev coefficients to multiple indices turns out to be better than the single index approach, and the difference becomes more obvious as d increases. On the other hand, however, sequential scan is definitely the winner over the two indexing schemes at very high dimensionalities.

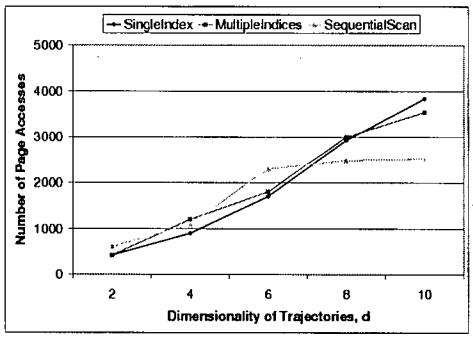
6.8 Subsequence Matching

All the previous Sections, whether indexing is included or not, whether a single index or multiple indices are used, are devoted to the problem of whole matching. In this Section, our focus is subsequence matching, as discussed in Section 2.1.2. We use our trajectory generator to create a database of $M = 500$ time series, each of which has a length of $N = 500$. Queries of length $w = 180$ are randomly generated and radius searches with a radius $r = 10$ are performed. In total, there are $(N - w + 1) * M = 160500$ subsequences of length w.

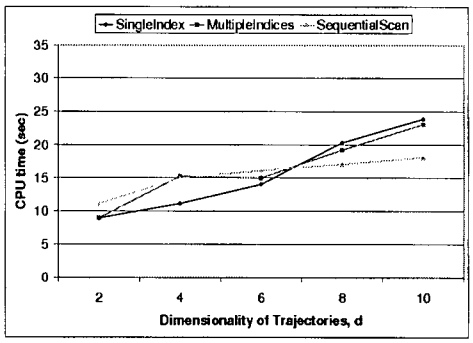
Note that, in the subsequence matching algorithm described in Section 2.1.2, both Chebyshev and APCA can be used for the dimensionality reduction step. In Figure 6.7, we compare Chebyshev approximation and APCA approximation in



(a) Pruning Power Comparison

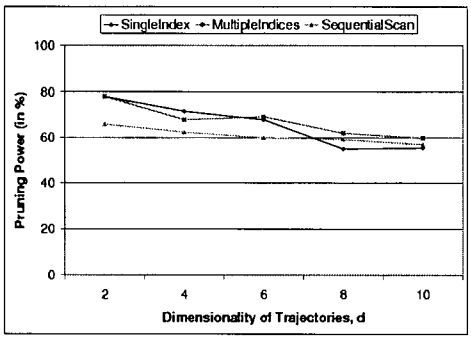


(b) I/O Cost Comparison

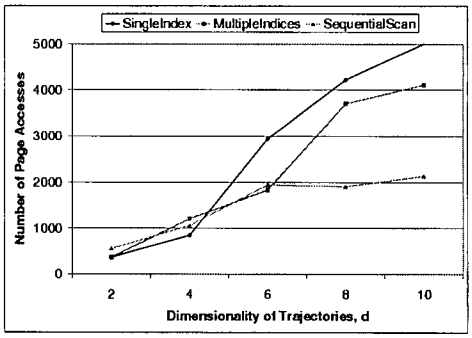


(c) CPU Time Comparison

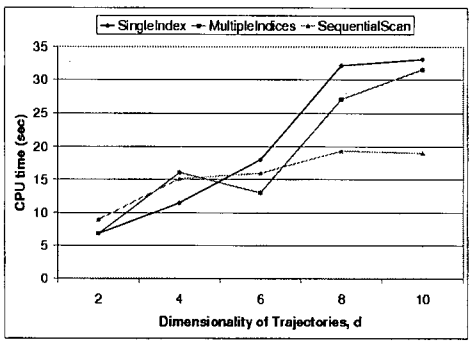
Figure 6.5: A Single Index vs. Multiple Indices ($n = 3$)



(a) Pruning Power Comparison

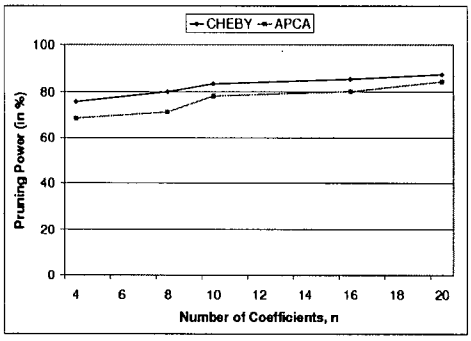


(b) I/O Cost Comparison

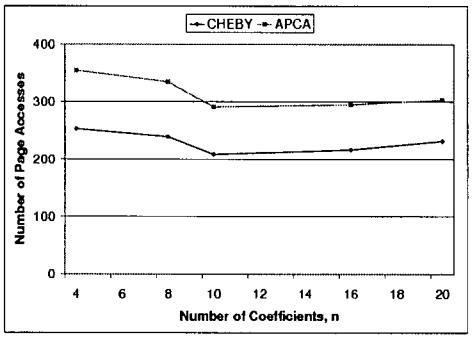


(c) CPU Time Comparison

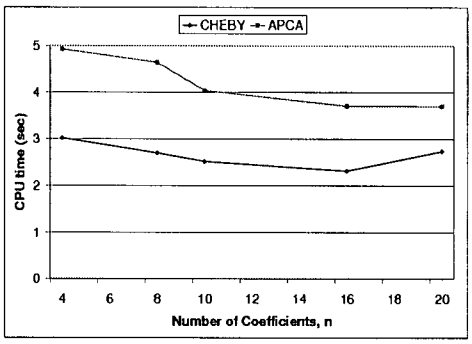
Figure 6.6: A Single Index vs. Multiple Indices ($n = 5$)



(a) Pruning Power Comparison



(b) I/O Cost Comparison



(c) CPU Time Comparison

Figure 6.7: Subsequence Matching: Pruning Power, I/O costs and CPU cost

terms of pruning power, I/O cost and CPU cost, respectively. As expected, the results are highly consistent with those for whole matching in Section 6.6.

Chapter 7

Conclusions

In this thesis, we explore how to apply Chebyshev polynomials for approximating and indexing d-dimensional spatiotemporal trajectories. Chebyshev polynomials enjoy the property that they are almost identical to the minimax polynomials; yet they are easier to compute. Computing Chebyshev coefficients is linear with respect to the data set size M, as well as to the trajectory length N. Our experimental results further show that computing extra Chebyshev coefficients takes negligible time (i.e., increasing n incurs little extra cost).

In order for Chebyshev approximation to be used for indexing, a key analytic result of this thesis is the Lower Bounding Lemma. To achieve this result, we need to extend a discrete trajectory into an interval function, so that Chebyshev approximation becomes applicable. We also need to define a distance function between two vectors of Chebyshev coefficients.

To evaluate the effectiveness of the minimax property of Chebyshev polynomials on indexing, we conducted an extensive experimental evaluation. From 1- to 4-dimensional, real to generated data, Chebyshev dominates the widely-used APCA

in pruning power, I/O costs and CPU costs. Our empirical results indicate that Chebyshev approximation can deliver a 3- to 5-fold reduction on the dimensionality of the index space. That is, it only takes 4 to 6 Chebyshev coefficients to deliver the same pruning power produced by 20 APCA coefficients. This is a very important advantage. As the dimensionality curse on the indexing structure is bound to set in sooner or later, Chebyshev coefficients are far more effective than APCA in delivering additional pruning power before that happens.

In ongoing work, we would like to extend the Lower Bounding Lemma to other distance functions, such as the dynamic time-warping distance [4] and the longest common subsequence distance [53]. We would also like to expand our framework to conduct sub-trajectory matching. The fixed-window strategy proposed in [9] is applicable; yet we seek to exploit properties of Chebyshev approximation for further optimization. The experimental results reported here are based on using the same number of coefficients for each dimension. In ongoing work, we would explore how to allocate a fixed number of Chebyshev coefficients to the d dimensions according to the "need" of each dimension. Finally, we would explore how to develop an optimized sequential scan algorithm to use in conjunction with Chebyshev coefficients.

Bibliography

[1] P. Agarwal, L. Arge and J. Erickson. Indexing Moving Points. *Proc. 2000 ACM PODS*, pp. 175–186.

[2] R. Agrawal, C. Faloutsos and A. Swami. Efficient similarity search in sequence databases. *Proc. of the 4th Conference on Foundations of Data Organization and Algorithms 1993*, pp. 69–84.

[3] R. Agrawal, K. Lin, H. Sawhney and K. Shim. Fast Similarity Search in the Presence of Noise, Scaling and Translation in Time-series databases. *Proc. 1995 VLDB*, pp. 490–501.

[4] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. *Working Notes of the Knowledge Discovery in Databases Workshop*, 1994, pp. 359–370.

[5] Y. Cai and R. T. Ng. Indexing Spatio-Temporal Trajectories with Chebyshev Polynomials. *Proc. 2004 SIGMOD*, to appear.

[6] K. Chakrabarti and S. Mehrotra. Local Dimensionality Reduction: a new approach to indexing high dimensional spaces. *Proc. 2000 VLDB*, pp. 89–100.

[7] K. Chan and A. Fu. Efficient Time Series Matching by Wavelets. *Proc. 1999 ICDE*, pp. 126–133.

[8] K. Chu and M. Wong. Fast Time-Series Searching with Scaling and Shifting. *Proc. 1999 PODS*, pp. 237–248.

[9] C. Faloutsos, M. Ranganathan and Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. *Proc. 1994 SIGMOD*, pp. 419–429.

[10] V. Gaede and O. Gunther. Multidimensional Access Methods. *ACM Computing Surveys*, 30, pp. 170–231, 1998.

[11] D. Gunopulos and G. Das. A Tutorial on Time Series Similarity Measures and Time Series Indexing. *Proc. 2001 SIGMOD*, pp. 243–307.

[12] M. Hadjieleftheriou, G. Kollios, V. Tsotras and D. Gunopulos. Efficient Indexing of Spatiotemporal Objects. *Proc. 2002 EDBT*, pp. 251–268.

[13] M. L. Hetland. A Survey of Recent Methods for Efficient Retrieval of Similar Time Sequences. *Data Mining in Time Series Databases.* Kandel, and H. Bunke, Eds. Singapore: World Scientific, to be published.

[14] T. Imielinski and B. Badrinath. Querying in Highly Mobile Distributed Environments. *Proc. 1992 VLDB*, pp. 41–52.

[15] H.V. Jagadish. On Indexing Line Segments. *Proc. 1990 VLDB*, pp. 614–625.

[16] C. Jensen and R. Snodgrass. Temporal Data Management. *TKDE 11(1)*, 1999, pp. 36–44.

[17] T. Kahveci and A. Singh. Variable length queries for time series data. *Proc. 2001 ICDE*, pp. 273–282.

[18] K. V. R. Kanth, D. Agrawal and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. *Proc. 1998 SIGMOD*, pp. 166–176.

[19] E. Keogh. Exact indexing of dynamic time warping. *Proc. VLDB 2002*, pp. 406–417.

[20] E. J. Keogh. Efficiently Finding Arbitrarily Scaled Patterns in Massive Time Series Databases. *Proc. 2003 PKDD*, pp. 253–265.

[21] E. Keogh, K. Chakrabarti, M. Pazzani and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. *Proc. 2001 SIGMOD*, pp. 151–162.

[22] E. Keogh and M. Pazzani. An Indexing Scheme for Fast Similarity Search in Large Time Series Databases. *Proc. 1999 SSDBM*, pp. 56–67.

[23] E. J. Keogh and M. J. Pazzani. A simple dimensionality reduction technique for fast similarity search in large time series databases. *Pacific-Asia Conference on Knowledge Discovery and Data Mining 2000*, pp. 122–133.

[24] E. Keogh, K. Chakrabarti, M. Pazzani and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Journal of Knowledge and Information Systems, 2000*, pp. 263–286.

[25] E. Keogh and P. Smyth. A probabilistic approach to fast pattern matching in time series databases. *Proc. 1997 KDD*, pp. 20–24.

[26] G. Kollios, D. Gunopulos and V. Tsotras. Nearest Neighbor Queries in a Mobile Environment. *Spatio-Temporal Database Management Workshop 1999*, pp. 119–134.

[27] G. Kollios, D. Gunopulos and V. Tsotras. On Indexing Mobile Objects. *Proc. 1999 PODS*, pp. 261–272.

[28] G. Kollios, D. Gunopulos, V. Tsotras, A. Delis and M. Hadjieleftheriou. Indexing Animated Objects Using Spatio-Temporal Access Methods. *IEEE Trans. Knowledge and Data Engineering 2001*, pp. 742–777.

[29] F. Korn, H. Jagadish and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. *Proc. 1997 SIGMOD*, pp. 289–300.

[30] S. L. Lee, S. J. Chun, D. H. Kim, J. H. Lee and C. W. Chung. Similarity Search for Multidimensional Data Sequences. *Proc. 2000 ICDE*, pp. 599–608.

[31] J. Kuan, P. Lewis. Fast k nearest neighbor search for R-tree family. *Proc. of First Int. Conf. on Information, Communication and Signal Processing 1997*, pp. 924–928.

[32] J. C. Mason and D. Handscomb. Chebyshev Polynomials. Chapman & Hall, 2003.

[33] Geographic Data Mining and Knowledge Discovery. Research Monographs in Geographic Information Systems, edited by H. Miller and J. Han, Taylor and Francis, 2000.

[34] Dimitris Papadopoulos, George Kollios, Dimitrios Gunopulos and Vassilis Tsotras. Indexing Mobile Objects on the Plane. *DEXA Workshops 2002*, pp. 693–697.

[35] 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. *Proc. 2000 ICDE*, pp. 33–42.

[36] D. Pfoser, C. J. Jensen and Y. Theodoridis. Novel approaches to the indexing of moving object trajectories. *Proc. 2000 VLDB*, pp. 395–406.

[37] I. Popivanov and R. Miller. Similarity Search Over Time Series Data Using Wavelets. *Proc. 2002 ICDE*, pp. 212–221.

[38] W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling. Numerical Recipes: The Art of Scientific Computing. Cambridge University Press, 1986.

[39] D. Rafiei. On similarity-based queries for time series data. *Proc. 1999 ICDE*, pp. 410–417.

[40] D. Rafiei and A. Mendelzon. Efficient Retrieval of Similar Time Sequences Using DFT. *Proc. 1998 FODO.*

[41] T. J. Rivlin. Chebyshev Polynomials: From Approximation Theory to Algebra and Number Theory (2nd Edition). John Wiley & Sons, 1990.

[42] N. Roussopoulos, S. Kelley and F. Vincent. Nearest Neighbor Queries. *Proc. 1995 SIGMOD*, pp. 71–79.

[43] S. Saltenis and C. Jensen. Indexing of Moving Objects for Location-Based Services. *Proc. 2002 ICDE*, pp. 463–472.

[44] S. Saltenis, C. Jensen, S. Leutenegger and M. Lopez. Indexing the Positions of Continuously Moving Objects. *Proc. 2000 SIGMOD*, pp. 331–342.

[45] T. K. Sellis, N. Roussopoulos and C. Faloutsos. The R^+-Tree: A Dynamic Index for Multi-dimensional Objects. *Proc. 1987 VLDB*, pp. 507–518.

[46] H. Shatkay and S. Zdonik. Approximate queries and representations for large data sequences. *Proc. 1996 ICDE*, pp. 546–553.

[47] P. Sistla, O. Wolfson, S. Chamberlain and S. Dao. Modeling and querying moving objects. *Proc. 1997 ICDE*, pp. 422–432.

[48] R. Snodgrass. The Temporal Query Language TQuel. *TODS 12(2)*, pp. 247–298, 1987.

[49] Y. Theodoridis, T. Sellis, A. Papadopoulos and Y. Manolopoulos. Specifications for Efficient Indexing in Spatiotemporal Databases. *Proc. 1998 SSDBM*, pp. 123–132.

[50] E. Tsoukatos and D. Gunopulos. Efficient Mining of SpatioTemporal Patterns. *Seventh International Symposium on Spatial and Temporal Databases 2001*, pp. 425–442.

[51] James Stewart. Single Variable Calculus: Early Transcendentals. Brooks/Cole Publishing Company, 1995.

[52] M. Vlachos, D. Gunopulos and G. Kollios. Robust Similarity Measures for Mobile Object Trajectories. *Proc. of DEXA Workshops 2002*, pp. 721–728.

[53] M. Vlachos, G. Kollios and D. Gunopulos. Discovering similar multidimensional trajectories. *Proc. 2002 ICDE*, pp. 673–684.

[54] C. Wang and S. Wang. Supporting content-based searches on time series via approximation. *Proc. 2000 SSDBM*, pp. 69–81.

[55] R. Weber, H. Schek and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. *Proc. 1998 VLDB*, pp. 194–205.

[56] O. Wolfson, B. Xu, S. Chamberlain and L. Jiang. Moving objects databases: Issues and solutions. *Proc. 1998 SSDBM*, pp. 111–122.

[57] D. Wu, D. Agrawal, A. El Abbadi, A. Singh and T. R. Smith. Efficient retrieval for browsing large image databases. *Proc. 1996 CIKM*, pp. 11–18.

[58] Y. Wu, D. Agrawal and A. Abbadi. A Comparison of DFT and DWT based Similarity Search in Time-Series Databases. *Proc. 2000 CIKM*, pp. 488–495.

[59] B. Yi and C. Faloutsos. Fast Time Sequence Indexing for Arbitrary $\mathcal{L}_p$ Norms. *Proc. 2000 VLDB*, pp. 395–406.