Indexing Spatio-Temporal Trajectories with Chebyshev Polynomials

Yuhan Cai
University of British Columbia
2366 Main Mall
Vancouver, Canada
ycai@cs.ubc.ca

Raymond Ng
University of British Columbia
2366 Main Mall
Vancouver, Canada
rng@cs.ubc.ca

ABSTRACT

In this paper, we attempt to approximate and index a d-dimensional ($d \geq 1$) spatio-temporal trajectory with a low order continuous polynomial. There are many possible ways to choose the polynomial, including (continuous) Fourier transforms, splines, non-linear regression, etc. Some of these possibilities have indeed been studied before. We hypothesize that one of the best possibilities 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. However, 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 [16]. Thus, in this paper, we explore how to use the Chebyshev polynomials as a basis for approximating and indexing d-dimensional trajectories.

The key analytic result of this paper 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 conducted comprehensive experimental evaluation with real and generated 1-dimensional to 4-dimensional data sets. We compared the proposed scheme with the Adaptive Piecewise Constant Approximation (APCA) scheme. Our preliminary results indicate that in all situations we tested, Chebyshev indexing dominates APCA in pruning power, I/O and CPU costs.

1. INTRODUCTION

In this paper, our focus is on indexing large collections of spatio-temporal trajectories for similarity matching. A d-dimensional spatio-temporal trajectory is a sequence of the form $\langle (t_1, \vec{v}_1), \ldots, (t_N, \vec{v}_N) \rangle$, with $t_1 < \ldots < t_N$, and $\vec{v}_i$ of arity d for all $1 \leq i \leq N$. Each pair $(t_i, \vec{v}_i)$ records the values of a vector of scalars at time t_i. For example, if the vector is of arity 1, the trajectory is a time series. For a second example, $\vec{v}_i$ may capture the 2-dimensional or 3-dimensional coordinates of a flying object at time t_i.

Time series are ubiquitous in temporal databases, which is a well-established area in database studies. Stock prices, salary histories, etc. are typical examples of time series. There are also many large collections of higher-dimensional spatio-temporal trajectories, thanks in part to the development of cost-effective mobile technologies [22, 26]. Examples include spatio-temporal trajectories of cars, airplanes, and moving objects generated by motion tracking devices in surveillance applications and electronic games applications.

Specifically, as part of our collaboration with an electronic games company, we encounter large collections of 2-, 3- and 4-dimensional spatio-temporal trajectories, for which similarity matching of the whole trajectories is a fundamental operation. A 2-dimensional example is the coordinates of National Football League players moving on the football field, or National Hockey League (NHL) players skating on the ice rink. A 3-dimensional example is flight simulation data. 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 spatio-temporal trajectories are ubiquitous.

One thing in common among the examples cited above 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. While we will discuss related work in greater details below, it suffices to say that most existing indexing frameworks are based on piecewise approximations, where each piece is either constant or linear. That is to say, a smooth and continuous trajectory is approximated with a piecewise discontinuous function. This mismatch may cause unnecessary error or deviation, and may lead to a loss in pruning power in a branch-and-bound search.

In this paper, we seek to approximate and index a d-dimensional spatio-temporal trajectory with a low order continuous polynomial. There are many possible ways to choose the polynomial, including (continuous) Fourier transforms, splines, non-linear regression, etc.; and indeed, some of these possibilities have been studied before. 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 [16]. Thus, in this paper, we explore how to use the Chebyshev polynomials as a basis for indexing d-dimensional trajectories.

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

- Recall that a spatio-temporal 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 paper – 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 Section 3, 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 conducted an extensive experimental evaluation. We used 1- to 4-dimensional real data sets, as well as generated 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) [11]. We obtained the APCA code from Keogh et al., and compared with Chebyshev approximation. We also extended APCA to d-dimensional situations as a "straw man" strategy.

  From 1- to 4-dimensional, real to 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.

The paper is organized as follows. Below we discuss related work. In the next section, we review Chebyshev polynomials and their properties central to the development of this paper. In Section 3, 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 Section 4, we generalize the earlier results for time series to deal with d-dimensional trajectories. In Section 5, we present our experimental setup and results. We compare Chebyshev and APCA on pruning power, I/O and CPU costs.

1.1 Related Work

There is a vast body of literature on indexing 1-dimensional time series. Examples include earlier works by Faloutsos et al. [5], Agrawal et al. [1], Korn et al. [15], to more recent studies by Rafiei and Mendelzon [20], Chan and Fu [4], Wu et al. [27], Gunopulos and Das [7], Keogh et al. [12, 11], Popivanov and Miller [18]. These studies can be divided into the following categories based on the underlying approximation schemes:

- DFT: including [5, 1, 20]
- DWT: including [4, 27, 9, 18]
- PAA: including [12, 28]
- APCA: including [11]
- SVD: including [15, 10]

DWT, PAA and APCA fall into the category of approximation with a discontinuous piecewise function. DWT further requires the length of a time series be a power of two. The study presented in [11] shows that APCA dominates DWT and PAA in pruning power by an order of magnitude. Thus, the latter two schemes are not included in our experimental comparison.

DFT falls into the category of approximating a time series with a continuous function. But while Fourier transformation is connected to Chebyshev approximation, the former does not have the minimax property that the latter enjoys. Because the study in [11] shows that APCA dominates DFT in pruning power by an order of magnitude, we omit DFT from our experimental comparison as well.

SVD approximation consists of space rotation and truncation. It is a global technique, and requires the computation of eigenvalues and eigenvectors of large matrices. Thus, it is far more expensive than the schemes mentioned above and the proposed scheme based on Chebyshev polynomials. And in terms of pruning power, it is not clear whether SVD is comparable to APCA. Thus, SVD is also omitted from our experimental comparison.

Studies considering indexing d-dimensional trajectories are not as many as the ones for time series. In [14], Kollios et al. considers indexing of moving objects to answer range queries of their future positions. Their framework answers queries of the form: "report the objects residing inside a given rectangle during a specified time period". As such, it is not designed for similarity retrieval of trajectories. Furthermore, it is based on the assumption that all the objects move with a constant velocity. The indexing scheme approximates each trajectory by straight line segments.

In [3], Chakrabarti and Mehrotra propose using SVD for indexing in high dimensional spaces. Specifically, clustering is first performed, and SVD is then applied to the local clusters, making it a semi-local approach. The proposed scheme based on Chebyshev polynomials is local and far less expensive in computational effort.

While most of the aforementioned studies are based on using the Euclidean distance, there are studies considering other possibilities. In [17], Perng et al. propose a new similarity model for time series called the Landmark model. The model is claimed to better match human intuition and episodic memory than the usual L_p-norm framework. In two recent studies, Kollios and Gunopulos et al. consider indexing multi-dimensional trajectories based on non-metric distance functions such as the longest common subsequence distance [24, 8]. A weaker version of the Triangle Inequality is used to answer k-nearest-neighbors (kNN) queries [21].

Even though approximations are used in the filtering step, the indexing proposed in this paper belongs to the class of exact schemes for similarity searching of the whole trajectories. That is to say, the schemes guarantee no false negatives. In the literature, there are studies which consider providing faster approximate similarity search, at the expense of allowing both false positives and negatives. Examples include the studies by Shatkay and Zdonik [23], and Keogh and Smyth [13]. A piecewise linear approach is used in all these studies.

2. BACKGROUND: CHEBYSHEV APPROXIMATION

In this section, we review Chebyshev polynomials and their key properties. Specifically, we discuss their orthogonality and the computation of the Chebyshev coefficients. These concepts are central to the following sections.

Definition 1. *(a) The Chebyshev Polynomial $P_m(t)$ is a polynomial in t of degree m, defined as:*

$$P_m(t) = cos(m\, cos^{-1}(t))$$

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

(b) Given the trigonometric identity $cosm\theta + cos(m-2)\theta = 2cos\theta cos(m-1)\theta$, the Chebyshev polynomials can be rewritten with the recurrence relation:

$$P_m(t) = 2tP_{m-1}(t) - P_{m-2}(t)$$

for all $m \geq 2$ with $P_0(t) = 1$ and $P_1(t) = t$.

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

$$\begin{aligned} P_0(t) &= 1 \\ P_1(t) &= t \\ P_2(t) &= 2t^2 - 1 \\ P_3(t) &= 4t^3 - 3t \\ P_4(t) &= 8t^4 - 8t^2 + 1 \end{aligned}$$

Even though in the above definition, t is defined over the interval [-1,1], the definition can be easily extended to any interval $[a, b]$. See [16] for more details. Without loss of generality, hereafter we simply focus on the interval [-1,1].

Concerning Chebyshev polynomials, a key property is that the system of Chebyshev polynomials $P_0(t), \ldots, P_m(t)$ is orthogonal. Two polynomials are orthogonal if their inner product is equal to 0. The inner product of two Chebyshev polynomials is defined as:

$$\langle P_i, P_j \rangle = \int_{-1}^{1} \frac{P_i(t)P_j(t)}{\sqrt{1-t^2}} dt$$

The following theorem holds for the inner product.

Theorem 1 ([16]). *The following is true for the Chebyshev polynomials $P_0(t), \ldots, P_m(t)$:*

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

As introduced above, hereafter we refer to:

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

as the *Chebyshev weight function*. The purpose of the weight function is to make the result of the integration exact (e.g., π) [16].

Given the orthogonality of the Chebyshev polynomials, they can be used as a base for approximating any function. That is, given a function $f(t)$, it can be approximated as:

$$f(t) \simeq c_0P_0 + \ldots + c_mP_m$$

The approximation is exact if $f(t)$ is a polynomial and its degree is less than or equal to m.

As a framework for indexing, the key question is how easy to compute the coefficients $c_0, \ldots, c_m$. The following theorem is called the Gauss-Chebyshev formula.

Theorem 2 ([16]). *Let $f(t)$ be a function to be approximated. $P_m(t)$ has m roots, namely $t_j = cos\frac{(j-0.5)\pi}{m}$ for $1 \leq j \leq m$. Then:*

(a) The coefficient c_0 is given by:

$$c_0 = \frac{1}{m}\sum_{j=1}^{m} f(t_j)P_0(t_j) = \frac{1}{m}\sum_{j=1}^{m} f(t_j)$$

(b) For all $1 \leq i \leq m$, the coefficient c_i is given by:

$$c_i = \frac{2}{m}\sum_{j=1}^{m} f(t_j)P_i(t_j)$$

The above formula gives an explicit way to compute each coefficient c_i. Note that in the above formulas for c_0 and c_i's, the constants differ by a factor of 2. This is a direct consequence of the second and third cases shown in Theorem 1 (i.e., π versus $\pi/2$). Figure 1 summarizes all the symbols used in this paper.

3. INDEXING WITH NO FALSE NEGATIVES

In this section, we focus on 1-dimensional spatio-temporal trajectories, i.e., time series. In Section 4, we generalize to higher dimensional trajectories.

Given a collection of time series of length N, we intend to represent each time series by its Chebyshev approximation

Notations	Meanings
$P_m(t)$	Chebyshev Polynomial of degree m
m	the degree of the polynomial $P_m(t)$ also the degree of the approximated polynomial
n	the number of Chebyshev coefficients, i.e., $n = m + 1$
c_i	the coefficient of $P_i(t)$ in an approximation
M	the number of trajectories
N	the (padded) length of each trajectory
d	dimensionality of the trajectories
$\vec{v}$	a vector of arity d
S	a spatio-temporal 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

Figure 1: Summary of Notations

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 [5]. We first establish a distance metric for the Chebyshev coefficients. In this paper, we use Euclidean distance (denoted as $Dist_{euc}$) to measure the distance between two time series S_1, S_2. We propose in Section 3.4 a natural Euclidean variant (denoted as $Dist_{cby}$) for measuring the distance between the two corresponding vectors of Chebyshev coefficients $\vec{C}_1, \vec{C}_2$. We then establish the important *Lower Bounding Lemma* in Section 3.5:

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

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.

3.1 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 occur at 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., $(t_i - t_{i-1}) = (t_{i+1} - t_i)$.

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 [4, 27, 9, 18], 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 paper do not require the same-set-uniformly-spaced assumption.

3.2 Chebyshev Approximation of a Time Series

Given a time series, we begin with the computation of the Chebyshev coefficients. However, Theorem 2 is not immediately applicable because the given formula is 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 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:

$$S(t) \quad = \quad \begin{cases} v_i & \text{if } t = t_i \\ undefined & \text{otherwise} \end{cases} \tag{1}$$

To apply Theorem 2, 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:

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

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

$$g(t) = v_i \quad \text{if } t \in I_i, \quad (\text{for } 1 \leq i \leq N) \tag{3}$$

To create an interval function based on the original time series in Equation (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 paper is that the lemma is satisfied with the inclusion of the Chebyshev weight function (defined in Section 2) and the length of each subinterval:

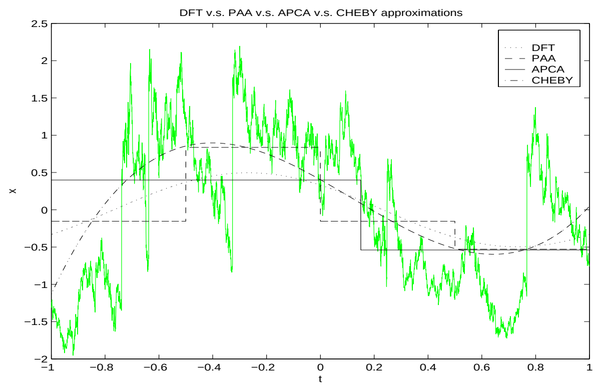
$$f(t) = \frac{g(t)}{\sqrt{w(t)|I_i|}} \quad \text{if } t \in I_i \quad (\text{for } 1 \leq i \leq N) \tag{4}$$

where $g(t)$ is as defined in Equation (3) and $|I_i|$ is the length of subinterval I_i.

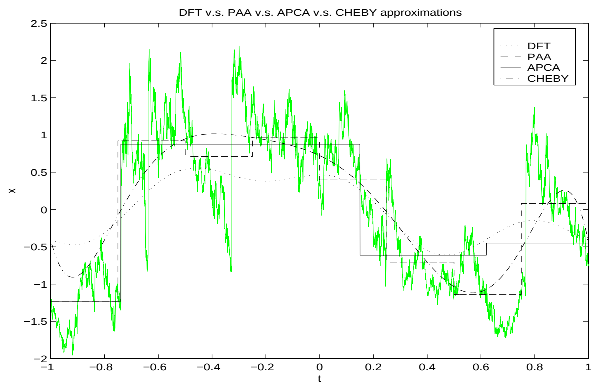
Because $f(t)$ is an interval function, we can use Theorem 2 to compute the coefficients of the Chebyshev approximation. Furthermore, for better approximation quality, we can use all the N data points and values of the time series. Of course, to reduce dimensionality, we only keep the first $n = m + 1$ Chebyshev coefficients for indexing. More specifically, we have the following formulas:

$$c_0 \quad = \quad \frac{1}{N} \sum_{j=1}^{N} f(t_j) P_0(t_j) \; = \; \frac{1}{N} \sum_{j=1}^{N} f(t_j) \tag{5}$$

$$c_i \quad = \quad \frac{2}{N} \sum_{j=1}^{N} f(t_j) P_i(t_j) \tag{6}$$



(a) $n = 4$



(b) $n = 8$

Figure 2: Comparing Chebyshev Approximation with Other Schemes

for all $1 \leq i \leq (n-1)$, where the t_j's are the roots of $P_N(t)$ as defined in Theorem 2.

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

3.3 An Example

Figure 2 shows the time series of the opening stock price of a Fortune500 company called ALCOA (ticker symbol: AA) for the period between February 28, 1978 to October 24, 2003 (for a total length of 6480 days). For $n = 4$, the left figure shows the original time series, the Chebyshev, the DFT, the PAA and the APCA approximations. The right figure 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.

Notice that for n equal to 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 Figure 2, 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. The following table shows the maximum deviation under the various schemes, normalized into the y-range of [-2, 2.5].

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

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 absolute guarantee that the maximum deviation decreases. Indeed, this phenomenon is shown in the above table. 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.

3.4 A Metric for Chebyshev Coefficients

Given two time series S_1, S_2, the previous subsection 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.

In this paper, we adopt the Euclidean distance function for spatio-temporal trajectories. While this distance function is simple, it is natural for many applications with spatio-temporal trajectories, including trajectories for airplanes and flying objects. It is also the distance function adopted by most studies on indexing time series, including [11]. For more advanced distance functions such as time-warping [2] and longest common subsequence [24], we consider them future topics of investigation.

Definition 2. *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:*

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

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 easy to establish.

Lemma 1. *$Dist_{cby}$ is a metric distance function.*

3.5 The Lower Bounding Lemma

We are now in a position to establish the Lower Bounding lemma: $Dist_{cby}(\vec{C}_1, \vec{C}_2) \leq Dist_{euc}(S_1, S_2)$. For space limitations, we omit a full proof of this result. Instead, we outline below some of the key steps in the proof.

Given $S_1 = \langle (t_1, v_1), \ldots, (t_N, v_N) \rangle$, and $S_2 = \langle (t_1, w_1), \ldots, (t_N, w_N) \rangle$, we consider the time series $Z = \langle (t_1, v_1 - w_1), \ldots, (t_N, v_N - w_N) \rangle$. Let $z_j = v_j - w_j$ for all $1 \leq j \leq N$. Then it is clear that the Euclidean distance between S_1, S_2 satisfies the following equality:

$$Dist^2_{euc}(S_1, S_2) = \sum_{j=1}^{N} (v_j - w_j)^2 = \sum_{j=1}^{N} z_j^2 \qquad (7)$$

Recall from Section 3.2 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 (3) and (4) in Section 3.2.

Lemma 2. *For all $t \in [-1, 1]$, $f_Z(t) = f_1(t) - f_2(t)$.*

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 (6). 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 3. *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 Sketch: In the following, we only focus on c_i for $1 \leq i \leq m$; the situation is almost identical for c_0.

$$\begin{aligned} c_i &= \tfrac{2}{N} \textstyle\sum_{j=1}^{N} f_Z(t_j) P_i(t_j) && \text{[Equation (6)]} \\ &= \tfrac{2}{N} \textstyle\sum_{j=1}^{N} [f_1(t_j) - f_2(t_j)] P_i(t_j) && \text{[previous lemma]} \\ &= a_i - b_i && \text{[Equation (6)]} \end{aligned}$$

□

Based on the above lemma and Definition (2), it is clear that:

$$Dist^2_{cby}(\vec{C}_1, \vec{C}_2) = \frac{\pi}{2} \sum_{i=0}^{m} c_i^2 \leq \frac{\pi}{2} \sum_{i=0}^{\infty} c_i^2 \qquad (8)$$

At this point, we need to use a known result for Chebyshev approximation. A function is *integrable* if it is bounded and has a finite number of discontinuities. Note that function $g(t)$ as defined in Equation (3) is integrable, as it is bounded and has $(N - 1)$ discontinuities. Then the function $f(t)$, as defined in Equation (4), is l_2-integrable with respect to the Chebyshev weight function. The following lemma thus applies.

Lemma 4 ([16]). *Let c_i be the Chebyshev coefficients for $f(t)$ which is l_2-integrable with respect to the Chebyshev weight function. Then it is the case that:*

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

With this lemma, we can finally put the various pieces together and conclude with the following theorem.

Theorem 3 (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 Sketch:

$$\begin{aligned} Dist^2_{cby}(\vec{C}_1, \vec{C}_2) &\leq \tfrac{\pi}{2} \textstyle\sum_{i=0}^{\infty} c_i^2 && \text{[Equation (8)]} \\ &= \int_{-1}^{1} \frac{f_Z^2(t)}{\sqrt{1-t^2}} dt && \text{[Lemma 4]} \\ &= \textstyle\sum_{j=1}^{N} |I_j| \frac{z_j^2}{|I_j|} && \text{[Equation (4)]} \\ &= \textstyle\sum_{j=1}^{N} z_j^2 && \\ &= Dist^2_{euc}(S_1, S_2) && \text{[Equation (7)]} \end{aligned}$$

□

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.

Furthermore, we can extend the strict Euclidean distance framework analyzed so far to a weighted Euclidean framework. We need to re-define $Dist_{euc}$ and $Dist_{cby}$ to include weights for the Lower Bounding Lemma to hold. For space limitations, we omit a proof of this extension.

4. INDEXING MULTI-DIMENSIONAL TRAJECTORIES

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

4.1 Lower Bounding for The Multi-Dimensional Case

Let S be a d-dimensional spatio-temporal 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 need 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 2 to give a metric distance function between two vectors of Chebyshev coefficients for two d-dimensional trajectories.

Definition 3. *Let S, R be d-dimensional spatio-temporal 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:*

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

```
Algorithm BuildIndex($DB$, $Index$, $n_1, \ldots, n_d$) {
  /* input: a database $DB$ of $M$ $d$-dimensional trajectories */
  /* input: $Index$, a multi-dimensional index which may already
            contain some entries */
  /* input: $n_i$ ($1 \leq i \leq d$) 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 $\{Dim_1, \ldots, Dim_d\}$
         creating $S_{Dim_1}, \ldots, S_{Dim_d}$
    (3)  for ($1 \leq i \leq d$) {
    (4)      apply Equations (3) to (6) 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 3: Algorithm for Building an Index of Chebyshev Coefficients

The following corollary is a simple extension of Theorem 3 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 1. *Let S, R be d-dimensional spatio-temporal trajectories, and $\vec{C}, \vec{D}$ be the corresponding vectors of Chebyshev coefficients. Then:*

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

4.2 Algorithms for Building and Searching the Index

Having established the Lower Bounding Lemma in Corollary 1 for the d-dimensional case, we can build an index of Chebyshev coefficients. Figure 3 shows a skeleton of an algorithm which takes M d-dimensional spatio-temporal trajectories, obtains the Chebyshev coefficients for each trajectory, and inserts the vectors of coefficients into a multi-dimensional index.

Recall from Section 3.2 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 3 that building the index takes $O(dMN)$ time.

Next we consider range and kNN searches. In both cases, the search is rather straightforward, following the GEMINI framework [5]. Figure 4 shows a skeleton of the range search algorithm, and Figure 5 shows a skeleton of the kNN search algorithm.

5. EXPERIMENTAL EVALUATION

5.1 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.

```
Algorithm RangeSearch($Q$, $Index$, $r$) {
  /* 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 a distance $r$ from $Q$
             with respect to $Dist_{euc}$ */
  (1)  apply Equations (3) to (6) to obtain the vector of
       coefficients for $Q$
  (2)  find all trajectories in $Index$ within $r$ 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 4: Algorithm for a Range Search

```
Algorithm kNNSearch($Q$, $Index$, $k$) {
  /* 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 (3) to (6) 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 RangeSearch($Q$, $Index$, $max$)
  (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: Algorithm for a kNN Search

Name	Dimensionality	Number	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

The Stocks data set consists of the daily opening prices of 500 companies traded in 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.

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. It is 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 h (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 h has h 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 (3) to (6). Recall from Figure 2 that there are various well-known schemes for time series indexing. As the study in [11] shows convincingly that APCA is almost always the best algorithm, we focus our empirical comparison only on 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 3 to 5.

Finally, many multi-dimensional indexing structures have been developed. See [6] for a comprehensive survey. For the results reported here, we used the DR-tree package developed by 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 3 with the APCA code. Similarly, we developed APCA versions of RangeSearch and kNNSearch procedures by basically replacing line (1) in Figures 4 and 5 with the APCA code.

5.2 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 4 and 5 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 be 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. [11]. 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.

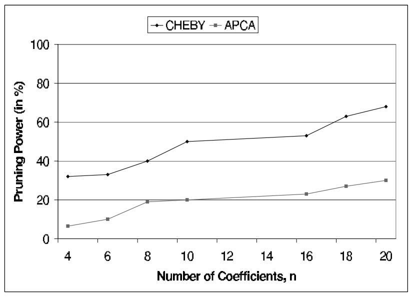
Apart from search times, we feel that it is essential to compare the pruning power for two reasons. First, in a search time comparison with indexing included, there are bias introduced by implementation details, including the choice of the indexing structure. A pruning power comparison is free of those implementation bias. 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.

5.3 Pruning Power Comparison: Real Data Sets

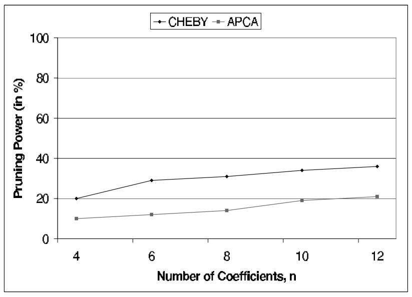
Figure 6 compares the pruning power of Chebyshev and APCA approximation. The value of k is 10 (i.e., 10-nearest neighbours). The figure consists of six graphs, one for each of the six real data sets. In all cases, the x-axis shows varying values of n (i.e., the number of coefficients allowed in the approximation). 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 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 even less than what 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.

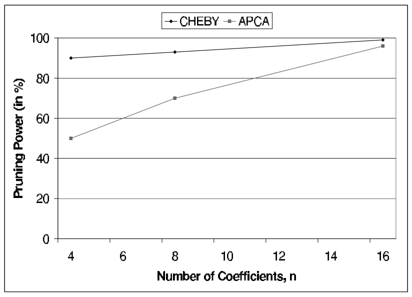
Let us turn our attention to higher-dimensional trajectories. Note that the value of n represents the number of coefficients for *each* dimension. For instance, for the graph in Figure 6(f), $n = 20$ corresponds to a total of 80 coefficients used for approximating the given 4-dimensional data. Note that we are not suggesting that in practice, we should build



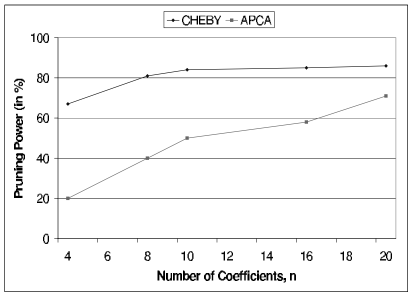
(a) 1-D Stocks data



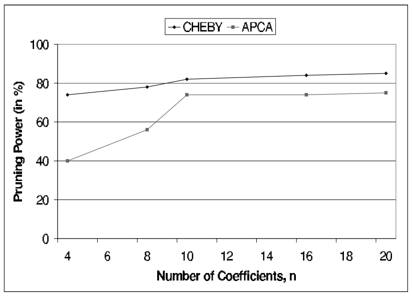
(b) 1-D ERP data



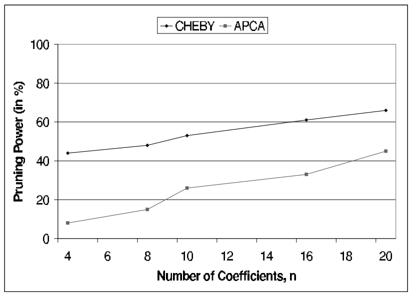
(c) 2-D NHL data



(d) 3-D Slips data



(e) 3-D Kungfu data



(f) 4-D Angle data

Figure 6: Pruning Power Comparisons: Real 1- to 4-Dimensional Data Sets

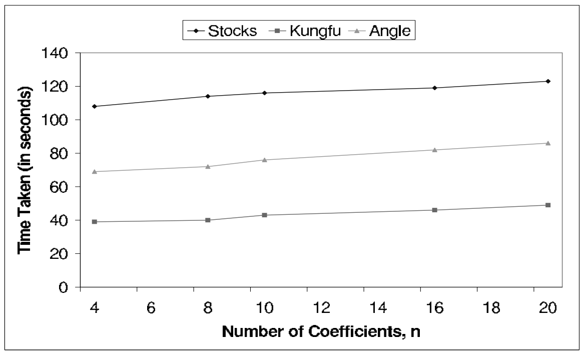


Figure 7: Computing Chebyshev Coefficients

an 80-dimensional index. Rather, we focus here on examining pure pruning effectiveness, independent of the index structure. To continue with Figure 6(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.

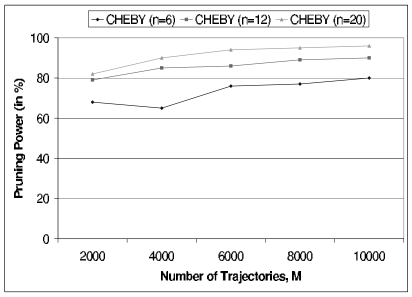
5.4 Building Time and the Choice of n

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, the larger the value of n, the higher is the pruning power. The obvious question to ask then is how large could n 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 later in Section 5.6.

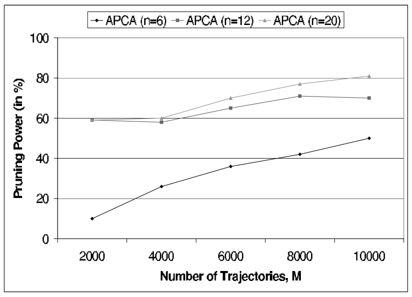
The second factor is the computation time of Chebyshev approximation. The key question here is how fast the time taken to compute the Chebyshev coefficients grows with respect to n. Figure 7 answers this question for the 1-dimensional Stocks data, 3-dimensional Kungfu data, and 4-dimensional Angle data. 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 6, 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 5.6 when indexing is included in our measurement, we shall return to this issue.

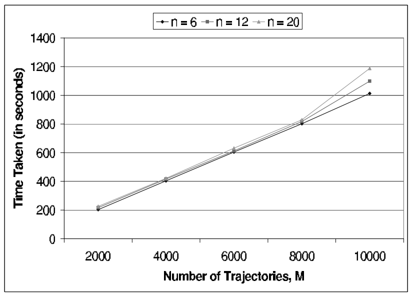
We do not include the building time for APCA here, as it takes at least an order of magnitude longer. But this is



(a) Chebyshev pruning power



(b) APCA pruning power



(c) Chebyshev building time

Figure 8: Scalability: Pruning Power and Building Time

not a fair comparison as APCA is implemented in Matlab, whereas our Chebyshev code is implemented in C++.

5.5 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 5.1. Figure 8 shows a representative situation – based on a 3-dimensional generated data set with an underlying polynomial of degree 10 and trajectory length of 720. Figure 8(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 3- to 5-fold reduction in the dimensionality of the approximation. Let us examine the first two graphs in Figure 8 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 8(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 7. 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.

5.6 Comparisons with Indexing Included

So far, our discussions have not yet taken into 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.

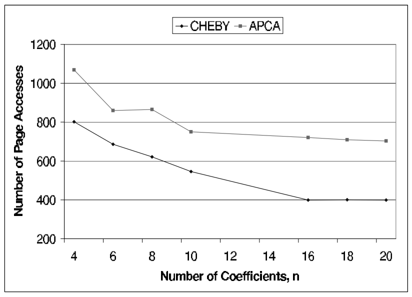
CPU time includes the time taken to naviagate the index nodes, the time taken to compute the lower bounded distances $Dist_{cby}(\vec{C}_Q, \vec{C}_S)$, and the time taken to compute the real Euclidean distances $Dist_{euc}(Q, S)$, whenever needed. As discussed before, the time taken to perform the initial 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.

5.6.1 I/O Cost Comparison

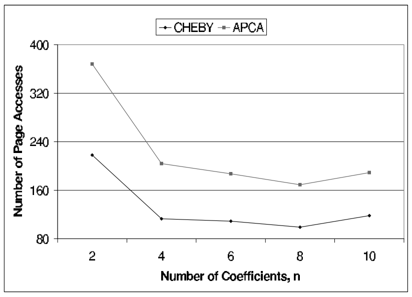
Figure 9 shows the I/O and CPU costs for the Stocks data, the Kungfu data and the 3-dimensional generated data with 10,000 trajectories each of length 720. (The results for the others are not shown for space limitations.) The x-axis of the graphs shows the number of coefficients used, n, for *each* dimension. 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 for $n = 8$, 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.

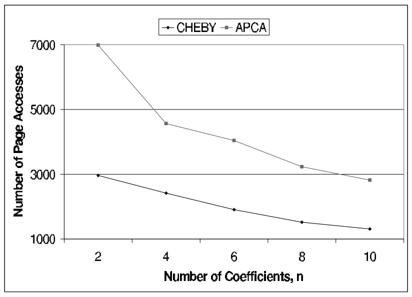
Dimensionality curse aside, the number of page accesses required by Chebyshev approximation in all cases is about 50% to 60% that of APCA. This improvement is highly consistent with the pruning power results shown earlier in Fig-



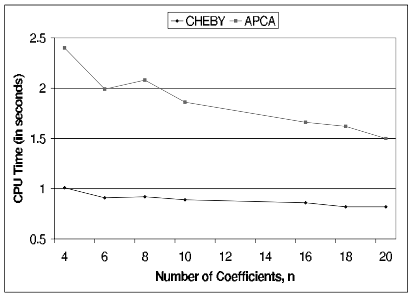
(a) 1-D Stocks data: I/O cost



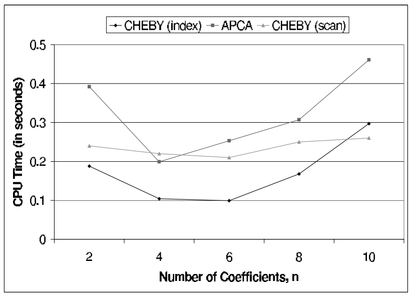
(b) 3-D Kungfu data: I/O cost



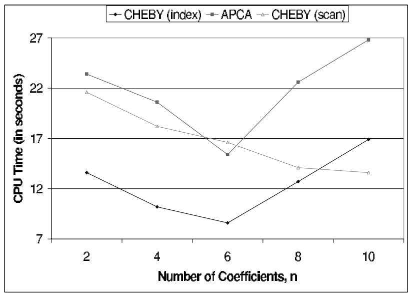
(c) 3-D Generated data: I/O cost



(d) 1-D Stocks data: CPU time



(e) 3-D Kungfu data: CPU time



(f) 3-D Generated data: CPU time

Figure 9: Search Time Comparison: Indexing Included

ure 6 and Figure 8.

5.6.2 CPU Cost Comparison

For graphs (d) to (f), the y-axis shows the CPU time taken (in seconds) of 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 5.2) 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 sets 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 [25], 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 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 spatio-temporal 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 2, the computation of the distance between two vectors of Chebyshev coefficients is $O(n)$. However, based on the distance measure given in [11], the corresponding computation between two vectors of APCA coefficients is in fact $O(N)$, which requires extra CPU time.

Recommendations: In closing, we make the following suggestions regarding indexing for d-dimensional trajectories. They are based on the DR-tree we used and should be adjusted depending on the choice of the index structure. For 1-dimenionsional, using $n = 20$ Chebyshev coefficients appears to be the best. For 2-dimenisonal, 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, sequential scan using a higher number of Chebyshev coefficients is recommended.

6. CONCLUSIONS

In this paper, we explore how to apply Chebyshev polynomials for approximating and indexing d-dimensional spatio-temporal trajectories. Chebyshev polynomials enjoys the property that they are almost identical to the minimax polynomials; yet they are easy to compute. Computing Chebyshev coefficients is linear with respect to the data set size M, as well as the trajectory length N. Our experimental results further shows 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 paper 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 APCA in pruning power, I/O 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 [2] and the longest common subsequence distance [24]. We would also like to expand our framework to conduct sub-trajectory matching. The fixed-window strategy proposed in [5] 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.

7. REFERENCES

[1] 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.

[2] 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*, pp. 359–370, 1994.

[3] K. Chakrabarti and S. Mehrotra. Local Dimensionality Reduction: a new approach to indexing high dimensional spaces. *Proc. 2000 VLDB.*

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

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

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

[7] D. Gunopulos and G. Das. A Tutorial on Time Series Similarity Measures and Time Series Indexing. *Proc. 2001 SIGMOD.*

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

[9] T. Kahveci and A. Singh. Variable length queries for time series data. *Proc. 2001 ICDE.*

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

[11] 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.

[12] 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.

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

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

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

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

[17] 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.*

[18] I. Popivanov and R. Miller. Similarity Search Over Time Series Data Using Wavelets. *Proc. 2002 ICDE.*

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

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

[21] N. Roussopoulos, S. Kelley and F. Vincent. Nearest Neighbor Queries. *Proc. 1995 SIGMOD.*

[22] S. Saltenis and C. Jensen. Indexing of Moving Objects for Location-Based Services. *Proc. 2002 ICDE.*

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

[24] M. Vlachos, G. Kollios and D. Gunopulos. Discovering similar multidimensional trajectories. *Proc. 2002 ICDE.*

[25] 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.

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

[27] 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.

[28] B. Yi and C. Faloutsos. Fast Time Sequence Indexing for Arbitrary Lp Norms. *Proc. 2000 VLDB.*