Open access • Journal Article • DOI:10.1145/376284.375680

Locally adaptive dimensionality reduction for indexing large time series databases

— Source link

KeoghEamonn, ChakrabartiKaushik, PazzaniMichael, MehrotraSharad

Published on: 01 May 2001 - Sigmod Record (ACMPUB27New York, NY, USA)

Topics: Dimensionality reduction, Search engine indexing and Nearest neighbor search

Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases

Kaushik Chakrabarti*♣ Microsoft Research, Redmond, WA 98052, kaushik@microsoft.com
Eamonn Keogh♦ Univ. of California, Riverside, CA 92521, eamonn@cs.ucr.edu
Sharad Mehrotra Univ. of California, Irvine, CA 92697, sharad@ics.uci.edu
Michael Pazzani Univ. of California, Irvine, CA 92697, pazzani@ics.uci.edu

Abstract

Similarity search in large time series databases has attracted much research interest recently. It is a difficult problem because of the typically high dimensionality of the data.. The most promising solutions involve performing dimensionality reduction on the data, then indexing the reduced data with a multidimensional index structure. Many dimensionality reduction techniques have been proposed, including Singular Value Decomposition (SVD), the Discrete Fourier transform (DFT), and the Discrete Wavelet Transform (DWT). In this work we introduce a new dimensionality reduction technique which we call Adaptive Piecewise Constant Approximation (APCA). While previous techniques (e.g., SVD, DFT and DWT) choose a common representation for all the items in the database that minimizes the global reconstruction error, APCA approximates each time series by a set of constant value segments of varying lengths such that their individual reconstruction errors are minimal. We show how APCA can be indexed using a multidimensional index structure. We propose two distance measures in the indexed space that exploit the high fidelity of APCA for fast searching: a lower bounding Euclidean distance approximation, and a non-lower bounding, but very tight Euclidean distance approximation and show how they can support fast exact searching, and even faster approximate searching on the same index structure. We theoretically and empirically compare APCA to all the other techniques and demonstrate its superiority.

Categories and Subject Descriptors: H.3.3 [**Information Search and Retrieval**] Search Process H.2.4 [**Systems**] Multimedia databases.
Keywords: Indexing, dimensionality reduction, time-series similarity retrieval.

1. Introduction

Time series account for a large proportion of the data stored in financial, medical and scientific databases. Recently there has been much interest in the problem of similarity search (query-by-content) in time series databases. Similarity search is useful in its own right as a tool for exploratory data analysis, and it is also an important element of many data mining applications such as clustering [13], classification [26, 33] and mining of association rules [12].

The similarity between two time series is typically measured with Euclidean distance, which can be calculated very efficiently. However the volume of data typically encountered exasperates the problem. Multi-gigabyte datasets are very common. As typical example, consider the MACHCO project. This database contains more than a terabyte of data and is updated at the rate of several gigabytes a day [48].

* **CONTACT AUTHOR**. [**Address**: One Microsoft Way, Redmond, WA 98052-8300, USA. **Phone**: (425)703-5137 **Fax**: (425) 936-7329]

♣ Work done while author was Ph.D. student at University of Illinois at Urbana Champaign

♦ Work done while author was Ph.D. student at University of California at Irvine

The most promising similarity search methods are techniques that perform dimensionality reduction on the data, then use a multidimensional index structure to index the data in the transformed space. The technique was introduced in [1] and extended in [39, 31,11]. The original work by Agrawal et. al. utilizes the Discrete Fourier Transform (DFT) to perform the dimensionality reduction, but other techniques have been suggested, including Singular Value Decomposition (SVD) [28, 24, 23], the Discrete Wavelet Transform (DWT) [9, 49, 22] and Piecewise Aggregate Approximation (PAA) [24, 52].

For a given index structure, the efficiency of indexing depends only on the fidelity of the approximation in the reduced dimensionality space. However, in choosing a dimensionality reduction technique, we cannot simply choose an arbitrary compression algorithm. What is required is a technique that produces an *indexable* representation. For example, many time series can be efficiently compressed by delta encoding, but this representation does not lend itself to indexing. In contrast SVD, DFT, DWT and PAA all lend themselves naturally to indexing, with each eigenwave, fourier coefficient, wavelet coefficient or aggregate segment mapping onto one dimension of an index tree.

The main contribution of this paper is to propose a simple, but highly effective compression technique, Adaptive Piecewise Constant Approximation (APCA), and show that it can be indexed using a multidimensional index structure. This representation was considered by other researchers, but they suggested it "*does not allow for indexing due to its irregularity*" [52]. We will show that indexing APCA is possible, and, using APCA is up to one to two orders of magnitude more efficient than alternative techniques on real world datasets. We will define the APCA representation in detail in Section 3, however an intuitive understanding can be gleaned from Figure 1.

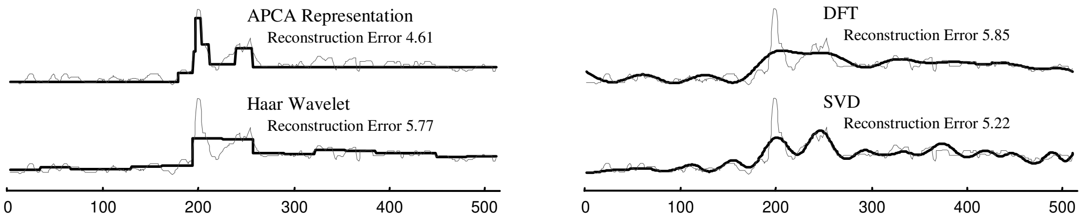


Figure 1: A visual comparison of the time series representation proposed in this work (APCA), and the 3 other representations advocated in the literature. For fair comparison, all representations have the same compression ratio. The reconstruction error is the Euclidean distance between the original time series and its approximation.

There are many situations in which a user would be willing to sacrifice some accuracy for significant speedup [5]. With this in mind we introduce two distance measures defined on the APCA representation. The first tightly lower bounds the Euclidean distance metric and is used to produce exact nearest neighbors. The second is not lower bounding, but produces a very close approximation of Euclidean distance and can be used to quickly find approximate nearest neighbors. Both methods can be supported by the same index structure so that a user can switch between fast exact search and even faster approximate search. Additionally we will show that the APCA representation can support queries where the distance measure is an arbitrary L_p norm (i.e. $p = 1,2,\ldots, \infty$).

The rest of the paper is organized as follows. In Section 2 we provide background on and review related work in time series similarity search. In Section 3 we introduce the APCA representation, an algorithm to compute it efficiently and two distance measures defined on it. In Section 4 we demonstrate how to index the APCA representation. Section 5 contains a comprehensive experimental comparison of APCA with *all* the competing techniques. In section 6 we discuss several advantages APCA has over the competing techniques, in addition to being faster. Section 7 offers conclusions and directions for future work.

2. Background and Related Work

Given two time series Q = {$q_1,\ldots,q_n$} and C = {$c_1,\ldots,c_n$} their Euclidean distance is defined as:

$$D(Q,C) \equiv \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2} \tag{1}$$

Figure 2 shows the intuition behind the Euclidean distance.

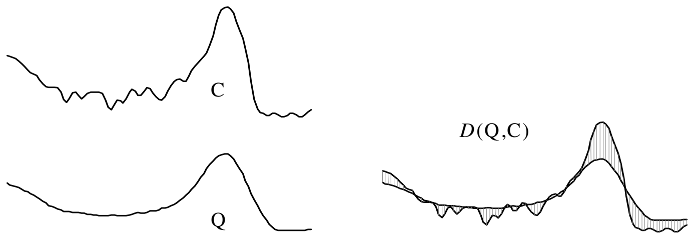


Figure 2: The intuition behind the Euclidean distance. The Euclidean distance can be visualized as the square root of the sum of the squared lengths of the gray lines.

There are essentially two ways the data might be organized [16]:

- **Whole Matching.**. Here it assumed that all sequences to be compared are the same length *n*.
- **Subsequence Matching.** Here we have a query sequence Q (of length *n*), and a longer sequence C (of length *m*). The task is to find the subsequence in C of length *n*, beginning at c_i, which best matches Q, and report its offset within C.

Whole matching requires comparing the query sequence to each candidate sequence by evaluating the distance function and keeping track of the sequence with the lowest distance. Subsequence matching requires that the query Q be placed at every possible offset within the longer sequence C. Note it is possible to convert subsequence matching to whole matching by sliding a "window" of length *n* across C, and making copies of the (*m-n*) windows. Figure 3 illustrates the idea. Although this causes storage redundancy it simplifies the notation and algorithms so we will adopt this policy for the rest of this paper.

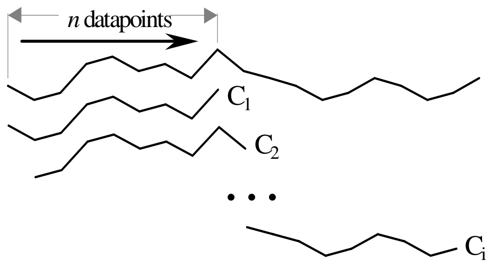


Figure 3: The subsequence matching problem can be converted into the whole matching problem by sliding a "window" of length *n* across the long sequence and making copies of the data falling within the windows.

There are two important kinds of queries that we would like to support in time series database, *range queries* (e.g., return all sequences within an epsilon of the query sequence) and *nearest neighbor* (e.g., return the K closest sequences to the query sequence). The brute force approach to answering these queries, sequential scanning, requires comparing every time series c_i to Q. Clearly this approach is unrealistic for large datasets.

Any indexing scheme that does not examine the entire dataset could potentially suffer from two problems, false alarms and false dismissals. False alarms occur when objects that appear to be close in the index are actually distant. Because false alarms can be removed in a post-processing stage (by confirming distance estimates on the original data), they can be tolerated so long as they are relatively infrequent. A false dismissal is when qualifying objects are missed because they appear distant in index space.

We will refer to similarity-searching techniques that guarantee no false dismissals as *exact*, and techniques that do not have this guarantee as *approximate*. Approximate techniques can still be very useful for exploring large databases, particularly if the probability of false dismissal is low. We will review approximate techniques in section 2.1 and exact techniques in section 2.2.

2.1 Approximate techniques for similarity searching.

Several researchers have suggested abandoning the insistence on exact search in favor of a much faster search that returns approximately the same results. Typically this involves transforming the data with a lossy compression scheme, then doing a sequential search on the compressed data. Typical examples include [42, 27, 30, 46], who all utilize a piecewise linear approximation. Others have suggested transforming the data into a discrete alphabet and using string-matching algorithms [2, 20, 34, 29, 21, 38]. All these approaches suffer from some limitations. They are all evaluated on small datasets residing in main memory, and it is unclear if they can be made to scale to large databases. Further, the systems are evaluated without considering precision and recall, thus we can say little or nothing about the quality of the returned answer set.

The work of [3, 36, 45, 25, 26] differs from the above in that they focus in providing a more flexible query language and not on performance issues.

2.2 Exact techniques for similarity searching.

A time series $C = \{c_1,\ldots,c_n\}$ with n datapoints can be considered as a point in n-dimensional space. This immediately suggests that time series could be indexed by multidimensional index structure such as the R-tree and its many variants [17]. Since realistic queries typically contain 20 to 1,000 datapoints (i.e. n varies from 20 to 1000) and most multidimensional index structures have poor performance at dimensionalities greater than 8-12 [6], we need to first perform dimensionality reduction in order to exploit multidimensional index structures to index time series data. In [16] the authors introduced GEneric Multimedia INdexIng method (GEMINI) which can exploit any dimensionality reduction method to allow efficient indexing. The technique was originally introduced for time series, but has been successfully extend to many other types of data [28].

An important result in [16] is that the authors proved that in order to guarantee no false dismissals, the distance measure in the index space must satisfy the following condition:

$$D_{\text{index space}}(A,B) \leq D_{\text{true}}(A,B)$$

This theorem is known as the lower bounding lemma or the contractive property. Given the lower bounding lemma, and the ready availability of off-the-shelf multidimensional index structures, GEMINI requires just the following three steps.

- Establish a distance metric from a domain expert (in this case Euclidean distance).
- Produce a dimensionality reduction technique that reduces the dimensionality of the data from n to N, where N can be efficiently handled by your favorite index structure.
- Produce a distance measure defined on the N dimensional representation of the data, and prove that it obeys $D_{\text{index space}}(A,B) \leq D_{\text{true}}(A,B)$.

Table 1 contains an outline of the GEMINI indexing algorithm. All sequences in the dataset **C** are transformed by some dimensionality reduction technique [1] and then indexed by the index structure of choice. The indexing tree represents the transformed sequences as points in N dimensional space. Each point contains a pointer to the corresponding original sequence on disk.

[1] We remove the mean (optionally) because, for many applications, we are only interested in the similarity based on the shape of the sequence and not its vertical offset from the x-axis. If the offset is not removed, it would dominate the Euclidean distance function leading to unintuituve notions of similarity [9]. We remove the mean for the experiments in this paper. For the illustrative examples we use in this paper, we do not remove the mean for simplicity.

```
Algorithm BuildIndex(C,n);      // C is the dataset, n is the size of the window
begin
1. for all objects in database
2.    $C_i$ ← $C_i$ - Mean($C_i$);          // Optional: remove the mean of $C_i$
3.    $\overline{C_i}$ ← SomeTransformation($C_i$);//$\overline{C_i}$ is any dimensionality reduced representation
4.    Insert $\overline{C_i}$ into the Spatial Access Method with a pointer to $C_i$ on disk;
5. endfor
end
```

Table 1: An outline of the GEMINI indexing building algorithm.

Note that each sequence has its mean subtracted before indexing. This has the effect of shifting the sequence in the y-axis such that its mean is zero, removing information about its offset. This step is included because for most applications the offset is irrelevant when computing similarity.

Table 2 below contains an outline of the GEMINI range query algorithm.

```
Algorithm RangeQuery(Q,ε)
begin
1. Project the query Q into the same feature space as the index.
2. Find all candidate objects in the index within ε of the query.
3. Retrieve from disk the actual sequences pointed to by the candidates.
4. Compute the actual distances, and discard false alarms.
end
```

Table 2: The GEMINI range query algorithm.

The range query algorithm is called as a subroutine in the K Nearest Neighbor algorithm outlined in Table 3. There are several optimizations to this basic K Nearest Neighbor algorithm that we utilize in this paper [41]. We will discuss them in more detail in Section 4.

```
Algorithm K_NearestNeighbor(Q,K)
begin
1. Project the query Q into the same feature space as the index.
2. Find the K nearest candidate objects in the index.
3. Retrieve from disk the actual sequences pointed to by the candidates.
4. Compute the actual distances and record the maximum, call it εmax.
5. Issue the range query, RangeQuery(Q,εmax);
6. Compute the actual distances, and choose the nearest K.
end
```

Table 3: The GEMINI nearest neighbor algorithm.

The efficiency of the GEMINI query algorithms depends only on the quality of the transformation used to build the index. The tighter the bound on $D_{\text{index space}}(A,B) \le D_{\text{true}}(A,B)$ the better, as tighter bounds imply fewer false alarms hence lower query cost [7]. Time series are usually good candidates for dimensionality reduction because they tend to contain highly correlated features. For brevity, we will not describe the three main dimensionality reduction techniques, SVD, DFT and DWT, in detail. Instead we refer the interested reader to the relevant papers or to [24] which contains a survey of all the techniques. We will briefly revisit related work in Section 6 when the reader has developed more intuition about our approach.

3. Adaptive Resolution Representation

In recent work Keogh et. al. [24] and Yi & Faloutsos [52] independently suggested approximating a time series by dividing it into equal-length segments and recording the mean value of the datapoints that fall within the segment. The authors use different names for this representation ([24] calls it "Piecewise Aggregate Approximation" while [52] calls it "Segmented-Means"), we will refer to it as Piecewise Aggregate Approximation (PAA) in this paper. This simple technique is surprisingly competitive with the more sophisticated transforms.

The fact that each segment in PAA is the same length facilitates indexing of this representation. Suppose however we relaxed this requirement and allowed the segments to have arbitrary lengths, does this improve the quality of the approximation? Before we consider this question, we must remember that the approach that allows arbitrary length segments, which we call Adaptive Piecewise Constant Approximation (APCA), requires two numbers per segment. The first number records the mean value of all the datapoints in segment, the second number records the length. So a fair comparison is N PAA segments to M APCA segments, were $M = \lfloor N/2 \rfloor$.

It is difficult to make any intuitive guess about the relative performance of the two techniques. On one hand PAA has the advantage of having twice as many approximating segments. On the other hand APCA has the advantage of being able to place a single segment in an area of low activity and many segments in areas of high activity. In addition one has to consider the structure of the data in question. It is possible to construct artificial datasets where one approach has an arbitrarily large reconstruction error, while the other approach has reconstruction error of zero.

Figure 4 illustrates a fair comparison between the two techniques on several real datasets. Note that for the task of indexing, subjective feelings about which technique "looks better" are irrelevant. All that matters is the quality of the approximation, which is given by the reconstruction error (because lower reconstruction errors result in tighter bounds on $D_{\text{index space}}(A,B) \leq D_{\text{true}}(A,B)$.).

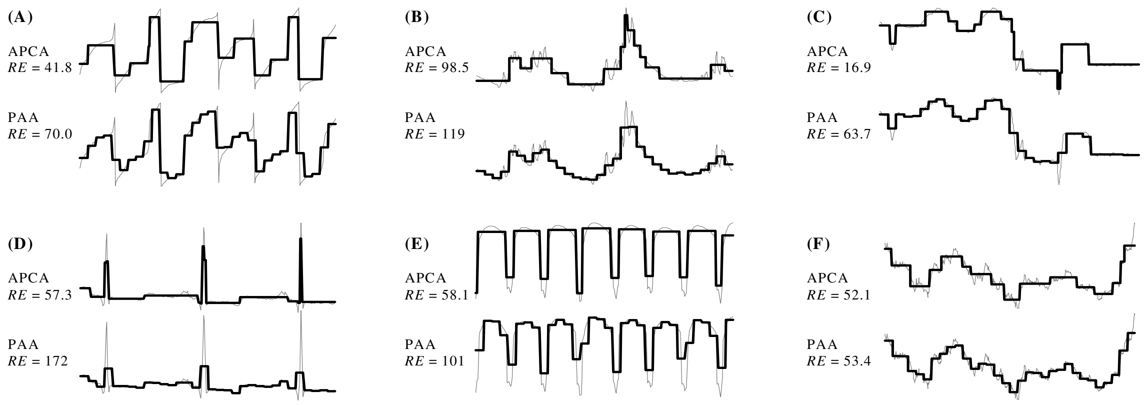


Figure 4: A comparison of the reconstruction errors of the equal-size segment approach (PAA) and the variable length segment approach (APCA), on a collection of miscellaneous datasets. A) INTERBALL Plasma processes. B) Darwin sea level pressures. C) Space Shuttle telemetry. D) Electrocardiogram. E) Manufacturing. F) Exchange rate.

On five of the six time series APCA outperforms PAA significantly. Only on the Exchange Rate data are they essentially the same. In fact, we repeated similar experiments for more than 40 different time series datasets, over a range of sequence lengths and compression ratios and we found that APCA is always at least as good as PAA, and usually much better. This comparison motivates our approach. If the APCA representation can be indexed, its high fidelity to the original signal should allow very efficient pruning of the index space (i.e. few false alarms, hence low query cost). We will show how APCA can be indexed in the next section (Section 4). In the rest of this section, we define the APCA representation formally, describe the algorithm to obtain the APCA representation of a time series and discuss the distance measures for APCA.

3.1 The APCA representation

Given a time series C = $\{c_1,\ldots,c_n\}$, we need to be able to produce an APCA representation, which we will represent as

$$C = \{<cv_1,cr_1>,\ldots,<cv_M,cr_M>\}, \quad cr_0 = 0 \tag{2}$$

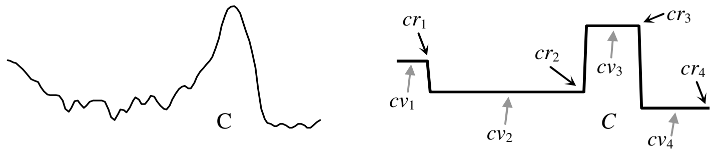


Figure 5: A time series C and its APCA representation C, with M = 4

where cv_i is the mean value of datapoints in the ith segment (i.e. $cv_i = mean(c_{cr_{i-1}+1,\ldots,}c_{cr_i})$) and cr_i the right endpoint of the i^th segment. We do not represent the length of the segments but record the locations of their right endpoints instead for indexing reasons as will be discussed in Section 4. The length of the i^th segment can be calculated as $cr_i - cr_{i-1}$. Figure 5 illustrates this notation.

Symbols	Definitions
S	Number of objects in the database.
n	Length of time series (a.k.a. query length, original dimensionality)
C = $\{c_1,\ldots,c_n\}$	A time series in a database, stored a vector of length n.
Q = $\{q_1,\ldots,q_n\}$	A query time series, represented as a vector of length n.
N	Dimensionality of index structure, with $N \ll n$.
M	Number of segments in APCA representation, with $M = \lfloor N/2 \rfloor$.
$C = \{<cv_1,cr_1>,\ldots,<cv_M,cr_M>\}$	An adaptive piecewise constant approximation of C, with c_i the mean value of i^th segment and cr_i the right endpoint of i^th segment.
$Q' = \{<qv_1,qr_1>,\ldots,<qv_M,qr_M>\}$	Also an adaptive piecewise constant approximation, but obtained using a special algorithm as describe in Equation 4
D(Q,C)	Euclidean distance
D_{AE}(Q,C)	An non-lower bounding approximation of Euclidean distance
D_{LB}(Q,C) or $D_{LB}(Q',C)$	A lower bounding approximation of the Euclidean distance
$cmax_i$, $cmin_i$	The max and min values of APCA representation C in i^th segment
R=(L, H)= $(\{l_1,\ldots,l_N\}, \{h_1,\ldots,h_N\})$	MBR associated with a node (say U) of the index built on N-dimensional APCA space; L=$\{l_1,\ldots,l_N\}$ and H=$\{h_1,\ldots,h_N\}$ denote the lower and higher endpoints of the major diagonal of R.
$\overline{C} = (\{cmin_1, cr_1, \ldots, cmin_M, cr_M\}, \{cmax_1, cr_1, \ldots, cmax_M, cr_M\})$	APCA rectangle corresponding to APCA point C
$G^R_i = \{G^R_i[1], G^R_i[2], G^R_i[3], G^R_i[4]\}$	i^th region associated with R; $G^R_i[1]$ and $G^R_i[3]$ are low and high bounds along the value axis; $G^R_i[2]$ and $G^R_i[4]$ are those along the time axis
MINDIST(Q, R)	Minimum distance of MBR R from query time series Q
MINDIST(Q, R, t)	Minimum distance of MBR R from Q at time instant t
MINDIST(Q, G, t)	Minimum distance of region G from Q at time instant t

Table 4: The notation used in this paper.

3.2 Obtaining the APCA representation

As mentioned before, the performance of the index structure built on the APCA representation defined in Equation 2 depends on how closely the APCA representation approximates the original signal. Closer the approximation, fewer the number of false alarms, better the performance of the index. We say that an M-segment APCA representation C of a time series C is *optimal* (in terms of the quality of approximation) iff C has the least reconstruction error among all possible M-segment APCA representations of C. Finding the optimal piecewise polynomial representation of a time series requires a O(Mn^2) dynamic programming algorithm [15, 35]. This is too slow for high dimensional data. In this paper, we propose a new algorithm to produce almost optimal APCA representations in O($n$log(n)) time. The algorithm works by first converting the problem into a wavelet compression problem, for which there are well known optimal solutions, then converting the solution back to the ACPA representation and (possibly) making minor modifications. The algorithm leverages off the fact that the Haar wavelet transformation of a time series signal can be calculated in O(n), and that an optimal reconstruction (i.e., having least reconstruction error) of the signal for any level of compression (i.e., #retained coefficients/n) can be obtained by sorting the coefficients in order of decreasing normalized magnitude, then truncating off the smaller coefficients [44]. The segments in the reconstructed signal may have approximate mean values (due to truncation); we replace them by the exact mean values to get a valid APCA representation as defined in Equation 2. There are, however, two issues we must address before utilizing this approach.

1) The DWT is defined only for time series with a length that is an integer power of two while n may not necessarily be a power of two. This problem can be solved easily by padding those time series with zeros, then truncating the corresponding segment after performing the DWT.
2) There is no exact mapping between the number of Haar coefficients retained and the number of segments in the APCA representation resulting from the reconstruction. For example a single coefficient Haar approximation could produce a 1, 2 or 3-segment APCA representation. Our solution is to keep the largest M coefficients; this will produce an APCA representation with the number of segments between M and 3M. If the number of segments is more than M, adjacent pairs of segments are merged until exactly M segments remain. The segment pairs targeted for merging are those that can be fused into a single segment with the minimum increase in reconstruction error.

Table 5 contains the outline of the algorithm.

```
Algorithm Compute_APCA(C,M)[2]
begin
1. if length(C) is not a power of two, pad it with zeros to make it so.
2. Perform the Haar Discrete Wavelet Transform on C.
3. Sort coefficients in order of decreasing normalized magnitude, truncate after M.
4. Reconstruct approximation (APCA representation) of C from retained coeffs.
5. If C was padded with zeros, truncate it to the original length.
6. Replace approximate segment mean values with exact mean values.
7. while the number of segments is greater than M
8.   Merge the pair of segments that can be merged with least rise in error
9. endwhile
end
```

Table 5: An algorithm to produce the APCA.

[2] The parameter M should be chosen judiciously. If M is too large, the dimensionality N of index structure (N=2M) will be high resulting in high query cost (due to dimensionality curse). If M is too small, the reconstruction error may become large leading to too many false positives and hence high query cost. So, M should be chosen such that the overall reconstruction error remains low without letting the dimensionality exceed the critical threshold of the index structure (above which it performs worse than sequential scan). The actual choice of M would depend on the dataset and the multidimensional index structure used.

We illustrate the working of the above algorithm using a numerical example.

Example 1 (Computing APCA representation)
Let us consider a time series C=[7, 5, 5, 3, 3, 3, 4, 6]. Table 6 shows the Haar wavelet decomposition of the series. We start by pairwise averaging the values to get a new "lower-resolution" representation of the data with the following average values [(7+5)/2, (5+3)/2, (3+3)/2, (4+6)/2] = [6, 4, 3, 5]. Obviously, some information is lost in this averaging process. To be able to reconstruct the original series, we need to also store the differences of the (second of the) averaged values from the computed pairwise average, i.e., [6-5, 4-3, 3-3, 5-6] = [1, 1, 0, -1]. These are called the detail coefficients. Applying the pairwise averaging and differencing process recursively on the lower-resolution array containing the averages, we get the full decomposition shown in Table 6.

Resolution	**Averages**	**Detail Coefficients**
3	[7, 5, 5, 3, 3, 3, 4, 6]	-
2	[6, 4, 3, 5]	[1, 1, 0, -1]
1	[5, 4]	[1, -1]
0	[4.5]	[0.5]

Table 6: Haar Wavelet Transform for APCA Computation

The wavelet transform W_C of C consists of the single coefficient representing the overall average of data values followed by the detailed coefficients in the order of increasing resolution. So W_C = [4.5, 0.5, 1, -1, 1, 1, 0, -1]. To take into account the importance of a coefficient with regard to the reconstruction of the original series (i.e., the number of elements in the series it contributes to the reconstruction of), we normalize the transform by dividing each coefficient by $2^{(l/2)}$ where l is the level of resolution of the coefficient. So the normalized wavelet transform of C is $W_C = [4.5, 0.5, \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, \frac{1}{2}, \frac{1}{2}, 0, \frac{-1}{2}]$.

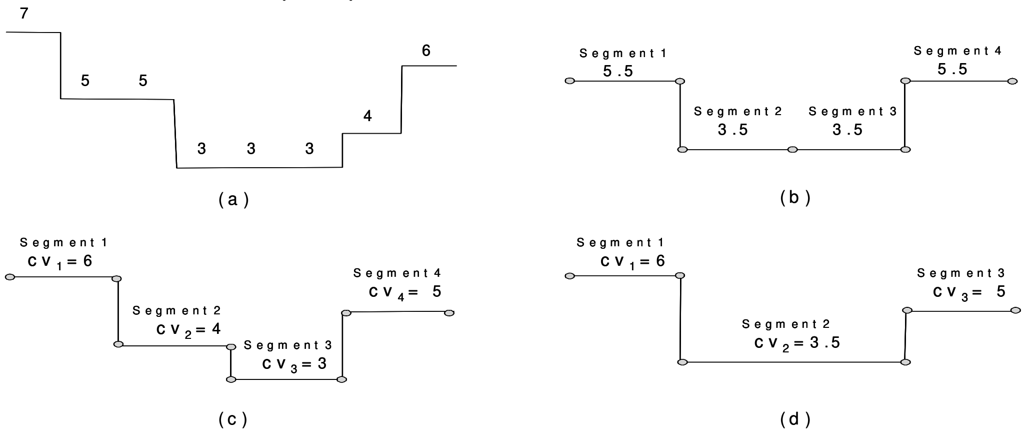


Figure 6: Step-by-step working of Compute_APCA algorithm. (a) Original time series C = [7,5,5,3,3,3,4,6] (b) Time series reconstructed from the M (3 in this case) best wavelet coefficients of C. The reconstructed series has 4 segments (segment boundaries indicated by dots). The mean value of each segment is shown just above the segment. (c) Reconstructed time series with approximate means replaced by exact means (cv_i's). (d) Final APCA representation obtained by merging segments 2 and 3 (to reduce the number of segments to M=3).

Suppose M=3. So we would retain the 3 coefficients with highest normalized magnitude, i.e., the first, third and fourth coefficients. Figure 6(a) and (b) show the original time series C and time series reconstructed from those 3 coefficients respectively. Figure 6(c) shows the reconstructed time series with approximate segment mean values replaced by the exact ones. Finally, we need to merge one pair of segments to reduce the number of segments to M=3; segment 2 and 3 is the best pair to merge as it results in the minimum increase in reconstruction error. Figure 6(d) shows the final 3-segment APCA representation of C produced by the Compute_APCA algorithm. ■

We experimentally compared this algorithm with several of the heuristic, merging algorithms [15, 35, 42] and found it is faster (at least 5 times faster for any length time series) and slightly superior in terms of reconstruction error.

3.3 Distance measures defined for the APCA representation

Suppose we have a time series C, which we convert to the APCA representation *C*, and a query time series Q. Clearly, no distance measure defined between Q and *C* can be exactly equivalent to the Euclidean distance *D*(Q,C) (defined in Equation 1) because *C* generally contains less information than C. However, we will define two distance measures between Q and *C* that approximate *D*(Q,C). The first, D_{AE}(Q,*C*) is designed to be a very tight approximation of the Euclidean distance, but may not always lower bound the Euclidean distance *D*(Q,C). The second, D_{LB}(Q,*C*) is generally a less tight approximation of the Euclidean distance, but is guaranteed to lower-bound, a property necessary to utilize the GEMINI framework. These distance measures are defined below, Figure 7 illustrates the intuition behind the formulas.

3.3.1 An approximate Euclidean measure D_{AE}

Given a query Q, in raw data format, and a time series *C* in the APCA representation, D_{AE}(Q,*C*) is defined as:

$$D_{AE}(Q,C) \equiv \sqrt{\sum_{i=1}^{M} \sum_{k=1}^{cr_i - cr_{i-1}} \left(cv_i - q_{k+cr_{i-1}} \right)^2} \qquad (3)$$

This measure can be efficiently calculated in O(*n*), and it tightly approximates the Euclidean distance, unfortunately it has a drawback which prevents its use for exact search.

Proposition 1 D_{AE}(Q,*C*) *does not satisfy the triangular inequality*

Proof: By counter example.

The triangular inequality states that for any objects α, β and χ

$$d(\alpha,\beta) \le d(\alpha,\chi) + d(\beta,\chi)$$

In other words, if D_{AE}(Q,*C*) obeys triangle inequality, there can exist no object A, B and C such that

$$D_{AE}(A,B) \ge D_{AE}(A,C) + D_{AE}(B,C)$$

We prove our proposition by finding three such objects.

Consider the time series A = {-1, -1, -2, 1, 2}, B = {1, 1, 0, -1, -1} and C = {0, 1, 0, 1, -2}. The 2-segment APCA representations of A, B and C as produced by the Compute_APCA algorithm are *A* = {<-1,2>, <$^1/_3$,5>}, *B* = {<$^2/_3$,3>,<-1,5>} and C = {<$^1/_2$,2>,< $^{-1}/_3$,5>} respectively.

According to Equation 3, $D_{AE}(A,B) \equiv \sqrt{(2/3-(-1))^2 + (2/3-(-1))^2 + (2/3-(-2))^2 + (-1-1)^2 + (-1-2)^2}$

$$\equiv \sqrt{2.7777 + 2.7777 + 7.1111 + 4 + 9}$$

$$\equiv 5.0662$$

Similarly, D_{AE}(A,*C*) = 3.8079 and D_{AE}(B,*C*) = 1.2247.

So, D_{AE}(A,*C*) + D_{AE}(B,*C*) = 5.0326.

⇒ $D_{AE}(A,B) \ge D_{AE}(A,C) + D_{AE}(B,C)$

This implies D_{AE}(Q,*C*) does not satisfy the triangular inequality. ■

The failure of D_{AE} to obey the triangular inequality means that it may not lower bound the Euclidean distance and thus cannot be used for exact indexing [51]. However, we will demonstrate later that it is very useful for approximate search.

3.3.2 An lower-bounding measure D_{LB}

To define D_{LB}(Q,C) we must first introduce a special version of the APCA. Normally the algorithm mentioned in Section 3.2 is used to obtain this representation. However we can also obtain this representation by "projecting" the endpoints of C onto Q, and finding the mean value of the sections of Q that fall within the projected intervals. A time series Q converted into the APCA representation this way is denoted as Q'. The idea can be visualized in Figure 7 III.

Q' is defined as:

$Q' = \{<qv_1,qr_1>,\ldots,<qv_M,qr_M>\}$, where $qr_i = cr_i$ and $qv_i = mean(q_{cr_{i-1}+1,\ldots,}q_{cr_i})$ (4)

$D_{LB}(Q',C)$ is defined as: $D_{LB}(Q',C) \equiv \sqrt{\sum_{i=1}^{M}(cr_i - cr_{i-1})(qv_i - cv_i)^2}$ (5)

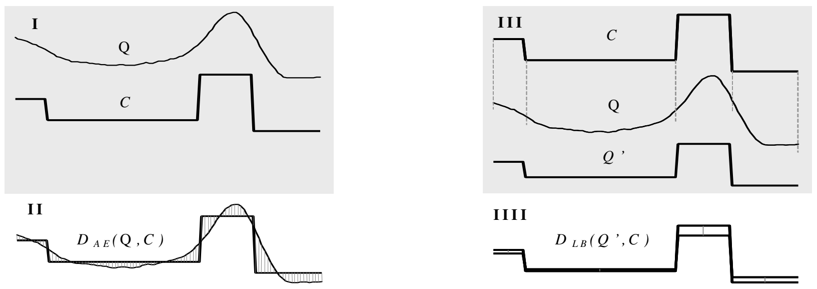


Figure 7: A visualization of the two distance measures define on the APCA representation. **I**) A query time series Q and a APCA object C. **II**) The D_{AE} measure can be visualized as the Euclidean distance between Q and the reconstruction of C. **III**) Q' is obtained by projecting the endpoints of C onto Q and calculating the mean values of the sections falling within the projected lines. **IIII**) The D_{LB} measure can be visualized as the square root of the sum of the product of squared length of the gray lines with the length of the segments they join.

We illustrate the computation of $D_{LB}(Q',C)$ using a numerical example below.

Example 2 (Computation of $D_{LB}(Q',C)$)

Let us consider a time series A={4, 6, 1, 0, 2}. The 2-segment APCA representation of A as produced by the Compute_APCA algorithm is A={<5,2>, <1,5>}. Let Q = {5, 3, 5, 6, 7} be a query time series. To compute $D_{LB}(Q',C)$, we first compute Q' ={<4,2>, <6,5>}. $D_{LB}(Q',C)$ $= \sqrt{(2-0)(4-5)^2 + (5-2)(6-1)^2}$ = 8.775. Note that $D_{LB}(Q',C)$ lower bounds D(Q,C) = $= \sqrt{(5-4)^2 + (3-6)^2 + (5-1)^2 + (6-0)^2 + (7-2)^2}$ = 9.327. The formal proof is shown below. ■

Lemma 1: $D_{LB}(Q',C)$ lower bounds the Euclidean Distance D(Q,C).

Proof:

We present a proof for the case where there is a single segment in the APCA representation. The more general proof for the M segment case can be obtained by applying the proof to each of the M segments.

Let W={$w_1, w_2, \ldots, w_p$} be a vector of p real numbers. Let $\overline{W}$ denote the arithmetic mean of W, i.e., $\overline{W} = (\Sigma w_i)/p$. We define a vector ΔW of real numbers where $\Delta w_i \equiv \overline{W} - w_i$. It is easy to see that $\Sigma\Delta w_i = 0$. The definition of Δw_i allows us to substitute w_i by $\overline{W} - \Delta w_i$, a fact which we will utilize in the proof below.

Let Q and C be the query and data time series respectively, with |Q| = |C| = n. Let Q' and C be the corresponding APCA vectors as defined in Equations 4 and 2 respectively.

We want to prove $\sqrt{\sum_{i=1}^{n}(q_i - c_i)^2} \;\geq\; \sqrt{\sum_{i=1}^{M}(cr_i - cr_{i-1})(qv_i - cv_i)^2}$ Because we are considering just the single segment case, we can remove summation over *M* segments and rewrite the inequality as:

Assume
$$\sqrt{\sum_{i=1}^{n}(q_i - c_i)^2} \;\geq\; \sqrt{(cr_i - cr_{i-1})(qv_i - cv_i)^2}$$

Because $(cr_i - cr_{i-1}) = n$
$$\sqrt{\sum_{i=1}^{n}(q_i - c_i)^2} \;\geq\; \sqrt{n(qv_i - cv_i)^2}$$

Because the terms under the radicals must be nonnegative, we can square both sides
$$\sum_{i=1}^{n}(q_i - c_i)^2 \;\geq\; n(qv_i - cv_i)^2$$

qv_i is simply the mean of Q, so rewrite as $\overline{Q}$
$$\sum_{i=1}^{n}(q_i - c_i)^2 \;\geq\; n(\overline{Q} - \overline{C})^2$$

cv_i is simply the mean of C, so rewrite as $\overline{C}$

Substitute rearrangement of definition above
$$\sum_{i=1}^{n}\left((\overline{Q} - \Delta q_i) - (\overline{C} - \Delta c_i)\right)^2 \;\geq\; n(\overline{Q} - \overline{C})^2$$

Rearrange terms
$$\sum_{i=1}^{n}\left((\overline{Q} - \overline{C}) - (\Delta q_i - \Delta c_i)\right)^2 \;\geq\; n(\overline{Q} - \overline{C})^2$$

Binomial theorem
$$\sum_{i=1}^{n}(\overline{Q} - \overline{C})^2 - 2(\overline{Q} - \overline{C})(\Delta q_i - \Delta c_i) + (\Delta q_i - \Delta c_i)^2 \;\geq\; n(\overline{Q} - \overline{C})^2$$

Distributive law
$$\sum_{i=1}^{n}(\overline{Q} - \overline{C})^2 - \sum_{i=1}^{n}2(\overline{Q} - \overline{C})(\Delta q_i - \Delta c_i) + \sum_{i=1}^{n}(\Delta q_i - \Delta c_i)^2 \;\geq\; n(\overline{Q} - \overline{C})^2$$

Summation properties
$$n(\overline{Q} - \overline{C})^2 - 2(\overline{Q} - \overline{C})\sum_{i=1}^{n}(\Delta q_i - \Delta c_i) + \sum_{i=1}^{n}(\Delta q_i - \Delta c_i)^2 \;\geq\; n(\overline{Q} - \overline{C})^2$$

Associative law
$$n(\overline{Q} - \overline{C})^2 - 2(\overline{Q} - \overline{C})\left(\sum_{i=1}^{n}\Delta q_i - \sum_{i=1}^{n}\Delta c_i\right) + \sum_{i=1}^{n}(\Delta q_i - \Delta c_i)^2 \;\geq\; n(\overline{Q} - \overline{C})^2$$

$\Sigma\Delta w_i = 0$, proved above
$$n(\overline{Q} - \overline{C})^2 - 2(\overline{Q} - \overline{C})(0 - 0) + \sum_{i=1}^{n}(\Delta q_i - \Delta c_i)^2 \;\geq\; n(\overline{Q} - \overline{C})^2$$

Subtract like term from both sides
$$n(\overline{Q} - \overline{C})^2 + \sum_{i=1}^{n}(\Delta q_i - \Delta c_i)^2 \;\geq\; n(\overline{Q} - \overline{C})^2$$

$$\sum_{i=1}^{n}(\Delta q_i - \Delta c_i)^2 \;\geq\; 0$$

The sum of squares must be nonnegative, so our assumption was true.
Hence the proof. ■

3.4 Quality of proposed distance measures D_{LB} and D_{AE}

The quality of a lower bounding distance measure can be gauged by how tightly it lower bounds the true distance between all queries of interest and all objects in the database (because all queries of interest cannot be known in advance and the database may be very large, a large random sampling must suffice). For a non-lower bounding measure, the quality is a little more difficult to define. Intuitively we want the measure to tightly approximate the true distance but only rarely overestimate it. In addition, when the true distance is overestimated, it should not be by a large amount.

We devised a simple experiment to illustrate the quality of D_{LB} and D_{AE} compared to the DWT (Haar) and the DFT approaches. We randomly extracted two sequences A and B from a database and measured the true Euclidean distance D(A,B) between them. We also measured the distance between A and B using the various reduced dimensionality representations for a fixed value of N. The ratio of the estimated distance over the true distance for all combinations was used to plot a point in 2-space, as illustrated in Figure 8.

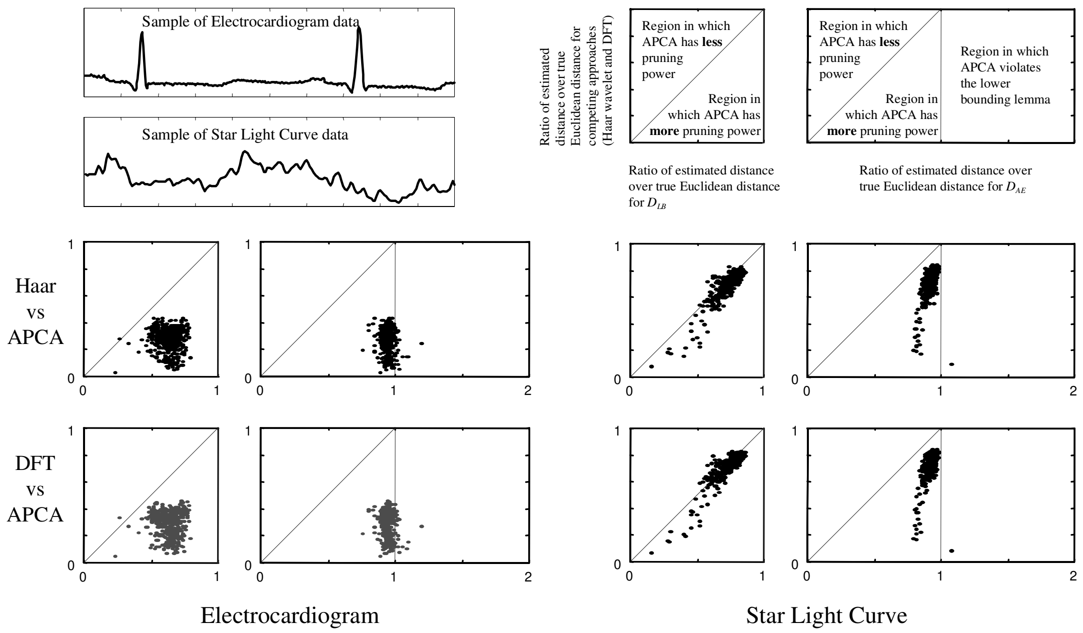


Figure 8: A visualization of the pruning power of the two distance measures defined on APCA as compared to the pruning power of the Haar wavelet and DFT approaches. Points on the diagonal indicate that the two approaches being compared have the same pruning power, points below the diagonal indicate that APCA is superior and points above the diagonal indicate that APCAs rival is superior.

We repeated this 1,000 times with randomly chosen sequences for each of two datasets. DWT and DFT behaved similarly, so for brevity we will only discuss the comparisons between DWT and the two measures defined on APCA.

For the Electrocardiogram dataset D_{LB} produces tighter lower bounds than the Haar wavelet approach 99.9% of the time, and the difference is usually quite significant. The D_{AE} measure very tightly approximated the true distance, and only violated lower bounding 2.9% of the time, generally by a very small amount.

For the Star Light Curve dataset D_{LB} produces tighter lower bounds than the Haar wavelet approach 81.3% of the time. The D_{AE} measure only violates lower bounding 0.1% of the time and generally is an extremely tight approximation of the true distance.

The quality of these results strongly suggests that APCA would be superior to existing approaches if indexable. We will address this issue in the next section.

4. Indexing the APCA representation

The APCA representation proposed in Section 3.1 defines a N-dimensional feature space ($N = 2M$). In other words, the proposed representation maps each time series C = $\{c_1,\ldots,c_n\}$ to a point $C = \{cv_1, cr_1, \ldots, cv_M, cr_M\}$ in a N-dimensional space. We refer to the N-dimensional space as the APCA space and the points in the APCA space as APCA points. In this section, we discuss how we can index the APCA points using a multidimensional index structure (e.g., R-tree) and use the index to answer range and K nearest neighbors (K-NN) queries efficiently. We will concentrate on K-NN queries in this section; range queries will be discussed briefly at the end of the section.

```
Algorithm ExactKNNSearch(Q,K)
Variable queue: MinPriorityQueue;
Variable list: temp;
begin
1.  queue.push(root_node_of_index, 0);
2.  while not queue.IsEmpty() do
3.     top = queue.Top();
4.     for each time series C in temp such that $D(Q,C) \le$ top.dist
5.        Remove C from temp;
6.        Add C to result;
7.        if |result| = K return result;
8.     endfor
9.     queue.Pop();
10.    if top is an APCA point $C$
11.       Retrieve full time series C from database;
12.       temp.insert(C, $D(Q,C)$);
13.    else if top is a leaf node
14.       for each data item C in top
15.          queue.push(C, $D_{LB}(Q',C)$);
16.       endfor
17.    else                                 // top is a non-leaf node
18.       for each child node U in top
19.          queue.push(U, MINDIST(Q,R)) // R is MBR associated with U
20.       endfor
21.    endif
22. enddo
end
```

Table 6: K-NN algorithm to compute the exact K nearest neighbors of a query time series Q using a multidimensional index structure

A K-NN query (Q, K) with query time series Q and desired number of neighbors K retrieves a set **C** of K time series such that for any two time series C ∈ **C,** E ∉ **C,** D(Q, C) ≤ D(Q, E). The algorithm for answering K-NN queries using a multidimensional index structure is shown in Table 6[3]. The above algorithm is an optimization on the GEMINI K-NN algorithm described in Table 3 and was proposed in [41]. Like the basic K-NN algorithm [19,40], the algorithm uses a priority queue *queue* to navigate nodes/objects in the index in the increasing order of their distances from Q in the indexed (i.e. APCA) space. The distance of an object (i.e. APCA point) C from Q is defined by $D_{LB}(Q',C)$ (cf. Section 3.3.2) while the distance of a node **U** from Q is defined by the minimum distance MINDIST(Q,R) of the minimum bounding rectangle (MBR) R associated with U from Q (definition of MINDIST will be discussed later). Initially, we push the root node of the index into the *queue* (Line 1). Subsequently, the algorithm navigates the index by

[3] In this paper, we restrict our discussion to only feature-based index structures i.e. multidimensional index structures that recursively cluster points using minimum bounding rectangles (MBRs). Examples of such index structures are R-tree, X-tree and Hybrid Tree . Note that the MBR-based clustering can be logical i.e. the index structure need not store the MBRs physically as long as they can be derived from the physically stored information. For example, space partitioning index structures like the hB-tree and the Hybrid Tree store the partitioning information inside the index nodes as kd-trees [14, 6]. Since the MBRs can be derived from the kd-trees, the techniques discussed here are applicable to such index structures [6].

popping out the item from the top of queue at each step (Line 9). If the popped item is an APCA point *C*, we retrieve the original time series C from the database, compute its exact distance D(Q,C) from the query and insert it into a temporary list *temp* (Lines 10-12). If the popped item is a node of the index structure, we compute the distance of each of its children from Q and push them into *queue* (Lines 13-20). We move a time series C from *temp* to *result* only when we are sure that it is among the K nearest neighbors of Q i.e. there exists no object E $\notin$ *result* such that D(Q,E) < D(Q,C) and $|result|$ < K. The second condition is ensured by the exit condition in Line 7. The first condition can be guaranteed as follows. Let **I** be the set of APCA points retrieved so far using the index (i.e. **I** = temp $\cup$ result). If we can guarantee that $\forall$ C $\in$ **I**, $\forall$ E $\notin$ **I**, $D_{LB}(Q',C) \leq D$(Q,E), then the condition "D(Q,C) $\leq$ top.dist" in Line 4 would ensure that there exists no unexplored time series E such that D(Q, E) < D(Q,C). By inserting the time series in *temp* (i.e. already explored objects) into *result* in increasing order of their distances D(Q,C) (by keeping temp sorted by D(Q,C)), we can ensure that there exists no explored object E such that D(Q, E) < D(Q,C). In other words, if $\forall$ C $\in$ **I**, $\forall$ E $\notin$ **I**, $D_{LB}(Q',C) \leq D$(Q,E), the above algorithm would return the correct answer.

Before we can use the above algorithm, we need to describe how to compute MINDIST(Q,R) such that the correctness requirement is satisfied i.e. $\forall$ C $\in$ **I**, $\forall$ E $\notin$ **I**, $D_{LB}(Q',C) \leq D$(Q,E). We now discuss how the MBRs are computed and how to compute MINDIST(Q,R) based on the MBRs. We start by revisiting the traditional definition of an MBR [17]. Let us assume we have built an index of the APCA points by simply inserting the APCA points $C = \{cv_1, cr_1, \ldots, cv_M, cr_M\}$ into a MBR-based multidimensional index structure (using the insert function of the index structure). Let **U** be a leaf node of the above index. Let R = (L, H) be the MBR associated with **U** where L = $\{l_1, l_2, \ldots, l_N\}$ and H = $\{h_1, h_2, \ldots, h_N\}$ are the lower and higher endpoints of the major diagonal of R. By definition, R is the smallest rectangle that spatially contains each APCA point $C = \{cv_1, cr_1, \ldots, cv_M, cr_M\}$ stored in **U**. Formally, R = (L, H) is defined as:

Definition 4.1 (Old definition of MBR)

$$\begin{aligned} l_i &= \min_{C\ in\ U}\ cv_{(i+1)/2} \text{ if i is odd} \qquad (6)\\ &= \min^{L}_{C\ in\ U}\ cr_{i/2} \quad \text{if i is even}\\ h_i &= \max^{L}_{C\ in\ U}\ cv_{(i+1)/2} \text{ if i is odd}\\ &= \max^{L}_{C\ in\ U}\ cr_{i/2} \quad \text{if i is even} \end{aligned}$$

The MBR associated with a non-leaf node would be the smallest rectangle that spatially contains the MBRs associated with its immediate children [17].

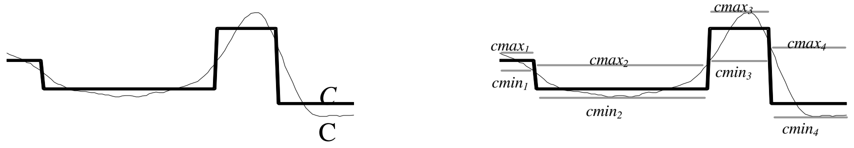


Figure 9: Definition of *cmin*i and *cmax*i for computing MBRs

However, if we build the index as above (i.e. the MBRs are computed as in Definition 4.1), it is not possible to define a MINDIST(Q,R) that satisfies the correctness criteria. To overcome the problem, we define the MBRs are follows. Let us consider the MBR R of a leaf node **U**. For any APCA point $C = \{cv_1, cr_1, \ldots, cv_M\ cr_M\}$ stored in node **U**, let *cmin*i and *cmax*i denote the minimum and maximum values of the corresponding time series C among the datapoints in the ith segment i.e.

$$cmin_i = \min_{t=cr_{i-1}+1}^{cr_i}(c_t) \quad \text{and} \quad (7)$$
$$cmax_i = \max_{t=cr_{i-1}+1}^{cr_i}(c_t)$$

The $cmin_i$ and $cmax_i$ of a simple time series with 4 segments is shown in Figure 9.
We define the MBR R = (L, H) associated with **U** as follows:

Definition 4.2 (New definition of MBR)

$$\begin{aligned} l_i &= \min_{C \text{ in } U} cmin_{(i+1)/2} && \text{if i is odd} \\ &= \min_{C \text{ in } U} cr_{i/2} && \text{if i is even} \\ h_i &= \max_{C \text{ in } U} cmax_{(i+1)/2} && \text{if i is odd} \\ &= \max_{C \text{ in } U} cr_{i/2} && \text{if i is even} \end{aligned} \quad (8)$$

As before, the MBR associated with a non-leaf node is defined as the smallest rectangle that spatially contains the MBRs associated with its immediate children.

How do we build the index such that the MBRs satisfy Definition 4.2. We insert rectangles instead of the APCA points. In order to insert an APCA point $C = \{cv_1, cr_1, .., cv_M, cr_M\}$, we insert a rectangle $\overline{C} = (\{cmin_1, cr_1, ..., cmin_M, cr_M\}, \{cmax_1, cr_1, ..., cmax_M, cr_M\})$ (i.e. $\{cmin_1, cr_1, ..., cmin_M, cr_M\}$ and $\{cmax_1, cr_1, ..., cmax_M, cr_M\}$) are the lower and higher endpoints of the major diagonal of $\overline{C}$) into the multidimensional index structure (using the insert function of the index structure). Since the insertion algorithm ensures that the MBR R of a leaf node **U** spatially contains all the $\overline{C}$'s stored in **U**, R satisfies definition 4.2 (as illustrated in example 3 below). The same is true for MBRs associated with non-leaf nodes.

Example 3 (Computation of MBRs)
Let us consider two time series A={4, 6, 1, 0, 2} and B={4, 3, 5, 1, 3}. The 2-segment APCA representations of A and B as produced by the Compute_APCA algorithm are $A = \{<av_1, ar_1>, <av_2, ar_2>\} = \{<5,2>, <1,5>\}$ and $B = \{<bv_1, br_1>, <bv_2, br_2>\} = \{<4,3>, <2,5>\}$ respectively. For A, $amin_1 = \min(4,6) = 4$, $amax_1 = \max(4,6) = 6$, $amin_2 = \min(1,0,2) = 0$, $amax_2 = \max(1,0,2) = 2$. For B, $bmin_1 = \min(4,3,5) = 3$, $bmax_1 = \max(4,3,5) = 5$, $bmin_2 = \min(1,3) = 1$, $bmax_2 = \max(1,3) = 3$. So, APCA rectangles $\overline{A} = (\{amin_1, ar_1, amin_2, ar_2\}, \{amax_1, ar_1, amax_2, ar_2\}) = (\{4, 2, 0, 5\}, \{6, 2, 2, 5\})$ and $\overline{B} = (\{bmin_1, br_1, bmin_2, br_2\}, \{bmax_1, br_1, bmax_2, br_2\}) = (\{3, 3, 1, 5\}, \{5, 3, 3, 5\})$. Since the MBR R of $\overline{A}$ and $\overline{B}$ is the smallest rectangle that spatially contains $\overline{A}$ and $\overline{B}$, R $= (\{\min(amin_1, bmin_1), \min(ar_1, br_1), \min(amin_2, bmin_2), \min(ar_2, br_2)\}, \{\max(amax_1, bmax_1), \max(ar_1, br_1), \max(amax_2, bmax_2), \max(ar_2, br_2)\})$ which satisfies definition 4.2. To complete the example, R $= (\{\min(4,3), \min(2,3), \min(0,1), \min(5,5)\}, \{\max(6,5), \max(2,3), \max(2,3), \max(5,5)\}) = (\{3, 1, 0, 5\}, \{6, 3, 3, 5\})$. ■

Since we use one of the existing multidimensional index structures to build the APCA index, the storage organization of the nodes follows that of the index structure (e.g., ⟨MBR, child_ptr⟩ array if R-tree is used, kd-tree if hybrid tree is used). For the leaf nodes, we need to store the cv_i's of each data point (in addition to the $cmax_i$'s, $cmin_i$'s and cr_i's) since they are needed to compute D_{LB} (Line 15 of the K-NN algorithm in Table 6). The index can be optimized (in terms of leaf node fanout) by *not* storing the $cmax_i$'s and $cmin_i$'s of the data points at the leaf nodes i.e. just storing the cv_i's and cr_i's (a total of 2M numbers) per data point in addition to the tuple identifier. The reason is that the $cmax_i$'s and $cmin_i$'s are not required for computing D_{LB}, and hence are not used by the K-NN algorithm. They are needed just to compute the MBRs properly (according to definition 4.2) at the time of insertion. The only time they are needed later (after the time of insertion) is during the recomputation of the MBR of the leaf node containing the data point after a node split. The insert function of the index structure can be easily modified to fetch the $cmax_i$'s and *cmin*$_i$'s of the necessary data points from the database (using the tuple identifiers) on such occasions. The small extra cost of such fetches during node splits is worth the improvement in search performance due to higher leaf node fanout. We have applied this optimization in the index structure for our experiments but we believe the APCA index would work well even without this optimization.

Once we have built the index as above (i.e. the MBRs satisfy Definition 4.2), we define the minimum distance MINDIST(Q,R) of the MBR R associated with a node **U** of the index structure from the query time series Q. For correctness, $\forall$ C $\in$ **I**, $\forall$ E $\notin$ **I**, $D_{LB}(Q',C) \leq D(Q,E)$ (where **I** denotes the set of APCA points retrieved using the index at any stage of the algorithm). We show that the above correctness criteria is satisfied if MINDIST(Q,R) lower bounds the Euclidean distance *D*(Q,C) of Q from any time series C placed under **U** in the index[2].

Lemma 2:

If MINDIST(Q,R) $\leq$ *D*(Q,C) for any time series C placed under **U**, the algorithm in Table 6 is correct i.e. $\forall$ C $\in$ **I**, $\forall$ E $\notin$ **I**, $D_{LB}(Q',C) \leq D(Q,E)$ where **I** denotes the set of APCA points retrieved using the index at any stage of the algorithm.

Proof:

According to the K-NN algorithm, any item *E* $\notin$ **I** *must* satisfy one of the following conditions:

1) E has been inserted into the queue but has not been popped yet i.e. $\forall$ C $\in$ **I**, $D_{LB}(Q', C) \leq D_{LB}(Q',E)$
2) E has not yet been inserted into the queue i.e. there exists a parent node **U** of E whose MBR R satisfies the following condition: $\forall$ C $\in$ **I**, $D_{LB}(Q',C) \leq$ MINDIST(Q,R).

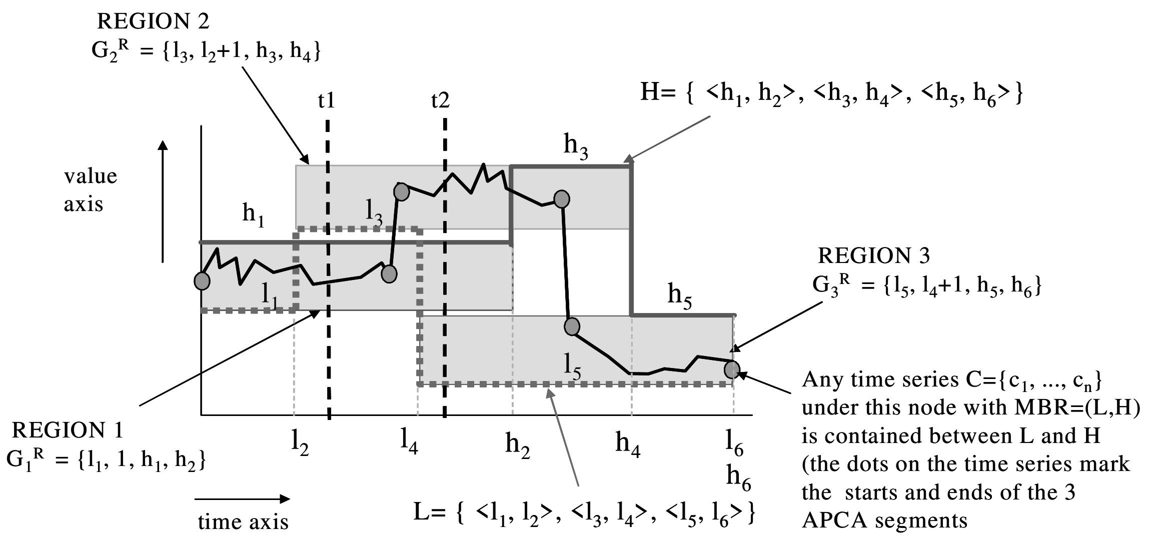
Since $D_{LB}(Q',E) \leq D(Q,E)$ (Lemma 1), (1) implies $\forall$ C $\in$ **I**, $D_{LB}(Q',C) \leq D(Q,E)$. If MINDIST(Q,R) $\leq$ *D*(Q,E) for any time series E under **U**, (2) implies that $\forall$ C $\in$ **I**, $D_{LB}(Q', C) \leq D(Q,E)$. Since either (1) or (2) must be true for any item E $\notin$ **I**, $\forall$ C $\in$ **I**, $\forall$ E $\notin$ **I**, $D_{LB}(Q',C) \leq D(Q,E)$.■

A trivial definition MINDIST(Q,R) that lower bounds *D*(Q,C) for any time series C under **U** is MINDIST(Q,R) = 0 for all Q and R. However, this definition is too conservative and would cause the K-NN algorithm to visit all nodes of the index structure before returning any answer (thus defeating the purpose of indexing). The larger the MINDIST, the more the number of nodes the K-NN algorithm can prune, the better the performance. We provide such a definition of MINDIST below[3].

Let us consider a node **U** with MBR R = (L,H). We can view the MBR as two APCA representations L={$<l_1, l_2>, \ldots, <l_{N-1}, l_N>$} and H = {$<h_1, h_2>, \ldots, <h_{N-1}, h_N>$}. The view of a 6-dimensional MBR ({$l_1,l_2,..,l_6$}, {$h_1,h_2,\ldots,h_6$}) as two APCA representations {$<l_1, l_2>, \ldots <l_5, l_6>$} and {$<h_1, h_2>, \ldots <h_5, h_6>$} is shown in Figure 10. Any time series C = {$c_1, c_2,\ldots, c_n$} under the node **U** is "contained" within the two bounding time series L and H (as shown in Figure 10). In order to formalize this notion of containment, we define a set of M regions associated with R. The i th region G^R_i (i = 1,…, *M*) associated with R is defined as the 2-dimensional rectangular region in the value-time space that fully contains the i th segment of all time series stored under **U**. The boundary of a region G, being

[2] Note that MINDIST (Q,R) does not have to lower bound $D_{LB}(Q,C)$ for any C under U; it just has to lower bound *D*(Q,C) for any C under U.

[3] Index structures can allow external applications to plug in domain-specific MINDIST functions and point-to-point distance functions and retrieve nearest neighbors based on those functions (e.g., Consistent function in GiST).



REGION 2
$G_2^R = \{l_3, l_2+1, h_3, h_4\}$
t1
t2
$H = \{ \langle h_1, h_2\rangle, \langle h_3, h_4\rangle, \langle h_5, h_6\rangle \}$
value axis
h_1
l_3
h_3
l_1
REGION 3
$G_3^R = \{l_5, l_4+1, h_5, h_6\}$
h_5
l_5
Any time series $C=\{c_1, ..., c_n\}$ under this node with MBR=(L,H) is contained between L and H (the dots on the time series mark the starts and ends of the 3 APCA segments
REGION 1
$G_1^R = \{l_1, 1, h_1, h_2\}$
l_2
l_4
h_2
h_4
l_6
h_6
time axis
$L = \{ \langle l_1, l_2\rangle, \langle l_3, l_4\rangle, \langle l_5, l_6\rangle \}$

Figure 10: The M Regions associated with a 2M-dimensional MBR. The boundary of a region G is denoted by G = {G[1], G[2], G[3], G[4]}

a 2-d rectangle, is defined by 4 numbers: the low bounds G[1] and G[2] and the high bounds G[3] and G[4] along the value and time axes respectively.

By definition,

$$G_i^R[1] = \min_{C\ under\ U}\ (cmin_i) \qquad (9)$$
$$G_i^R[2] = \min_{C\ under\ U}\ (cr_{i-1}+1)$$
$$G_i^R[3] = \max_{C\ under\ U}\ (cmax_i)$$
$$G_i^R[4] = \max_{C\ under\ U}\ (cr_i)$$

Based the definition of MBR in Definition 4.2, G_i^R can be defined in terms of the MBR R as follows:

Definition 4.3 (Definition of regions associated with MBR)

$$G_i^R[1] = l_{\{2i-1\}} \qquad (10)$$
$$G_i^R[2] = l_{\{2i-2\}}+1$$
$$G_i^R[3] = h_{\{2i-1\}}$$
$$G_i^R[4] = h_{\{2i\}}$$

Figure 10 shows the 3 regions associated with the 6-dimensional MBR ($\{l_1,l_2,..,l_6\}$, $\{h_1,h_2,\ldots,h_6\}$). We illustrate the region computation using a numeric example in Example 4.

Example 4 (Region Computation)
Let us consider the MBR R in Example 3. Recall R=($\{l_1, l_2, l_3, l_4\}$, $\{h_1, h_2, h_3, h_4\}$) = ({3, 1, 0, 5),{6, 3, 3, 5}). The two regions associated with R are:

G_1^R = {min($amin_1$, $bmin_1$), min(ar_0+1, br_0+1), max($amax_1$, $bmax_1$), max(ar_1, br_1)} (by Equation 9)
= $\{l_1, l_0+1, h_1, h_2\}$ (by Equation 10)
= {3, 1, 6, 3}

G_2^R = {min($amin_2$, $bmin_2$), min(ar_1+1, br_1+1), max($amax_2$, $bmax_2$), max(ar_2, br_2)}
= $\{l_3, l_2+1, h_3, h_4\}$
= {0, 2, 3, 5} ■

At time instance t (t = 1,…,*n*), we say a region G_i^R is active iff $G_i^R[2] \le t \le G_i^R[4]$. For example, in Figure 10, only regions 1 and 2 are active at time instant t_1 while regions 1, 2 and 3 are active at time instant t_2. The value c_t of a time series C under **U** at time instant t must lie within one of the regions active at t i.e. $\bigvee_{G_i^R \text{ is active}} G_i^R[1] \le c_t \le G_i^R[3]$.

Lemma 3:The value c_t of a time series C under U at time instant t must lie within one of the regions active at t.

Proof:

Let us consider a region G_i^R that is not active at time instant t i.e. either $G_i^R[2] > t$ or $G_i^R[4] < t$. First, let us consider the case $G_i^R[2] > t$. By definition, $G_i^R[2] \le cr_{i-1} + 1$ for any C under **U** . Since $G_i^R[2] > t$, $t < cr_{i-1} + 1$ i.e. c_t is not in segment i.

Now let us consider the case $G_i^R[4] < t$. By definition, $G_i^R[4] \ge cr_i$ for any C under **U**. Since $G_i^R[4] < t$, $t > cr_i$ i.e. c_t is not in segment i.

Hence, if region G_i^R is not active at t, c_t cannot lie in segment i i.e. c_t can lie in segment i only if G_i^R is active. By definition of regions, c_t must lie within one of the regions active at t i.e. $\bigvee_{G_i^R \text{ is active}} G_i^R[1] \le c_t \le G_i^R[3]$.■

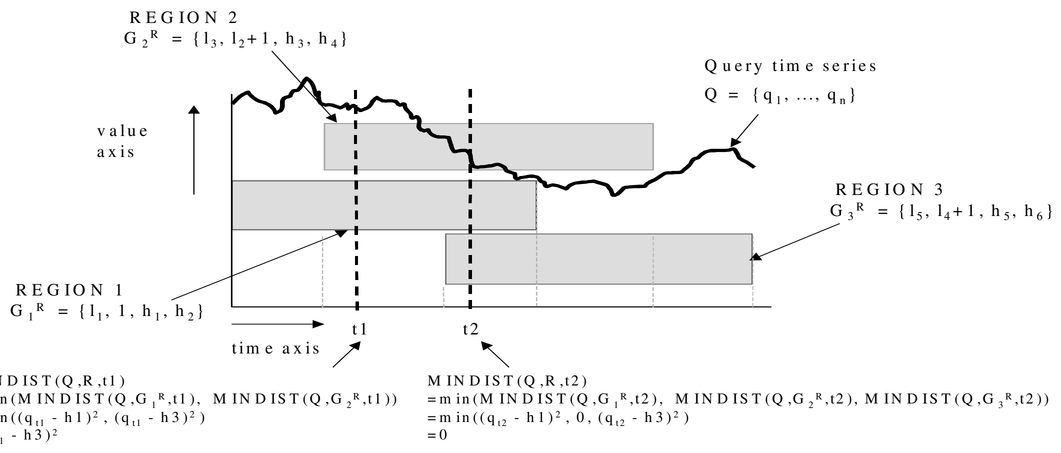

REGION 2
$G_2^R = \{l_3, l_2+1, h_3, h_4\}$
Query time series
$Q = \{q_1, \dots, q_n\}$
value axis
REGION 3
$G_3^R = \{l_5, l_4+1, h_5, h_6\}$
REGION 1
$G_1^R = \{l_1, 1, h_1, h_2\}$
time axis
t1
t2
MINDIST(Q,R,t1)
$=\min(\text{MINDIST}(Q,G_1^R,t1), \text{MINDIST}(Q,G_2^R,t1))$
$=\min((q_{t1} - h1)^2, (q_{t1} - h3)^2)$
$=(q_{t1} - h3)^2$
MINDIST(Q,R,t2)
$=\min(\text{MINDIST}(Q,G_1^R,t2), \text{MINDIST}(Q,G_2^R,t2), \text{MINDIST}(Q,G_3^R,t2))$
$=\min((q_{t2} - h1)^2, 0, (q_{t2} - h3)^2)$
$=0$

Figure 11: Computation of MINDIST

Given a query time series Q = {q_1, q_2, …, q_n}, the minimum distance MINDIST(Q,R,t) of Q from R at time instant t (cf. Figure 11) is given by $\min_{\text{region } G \text{ is active at } t}$ MINDIST(Q,G,t) where

$$\begin{aligned}\text{MINDIST}(Q,G,t) &= (G[1]-q^t)^2 \text{ if } q_t < G[1] \\ &= (q_t-G[3])^2 \text{ if } G[3] < q_t \\ &= 0 \text{ otherwise.}\end{aligned} \tag{11}$$

MINDIST(Q,R) is defined as follows:

$$\text{MINDIST(Q,R)} = \sqrt{\sum_{t=1}^{n} MINDIST(Q,R,t)} \tag{12}$$

We illustrate the MINDIST computation using a numeric example in Example 5.

Example 5 (MINDIST Computation)

Let us consider the MBR R in Examples 3 and 4 and its associated regions G_1^R and G_2^R. Let us consider the query time series Q = {5, 3, 5, 6, 7} in Example 2.

MINDIST(Q,R,1) = MINDIST(Q, G_1^R, 1) = 0

MINDIST(Q,R,2) = min(MINDIST(Q, G_1^R, 2), MINDIST(Q, G_2^R, 2)) = 0

MINDIST(Q,R,3) = min(MINDIST(Q, G_1^R, 3), MINDIST(Q, G_2^R, 3)) = 0

MINDIST(Q,R,4) = MINDIST(Q, G_2^R, 4) = $(6\text{-}3)^2$ = 9

MINDIST(Q,R,5) = MINDIST(Q, G_2^R, 5) = $(7\text{-}3)^2$ = 16

MINDIST(Q,R) = $\sqrt{9+16}$ = 5.

Note that MINDIST(Q,R) lower bounds *D*(Q,A)= $\sqrt{(5-4)^2+(3-6)^2+(5-1)^2+(6-0)^2+(7-2)^2}$ = 9.327 and *D*(Q,B)= $=\sqrt{(5-4)^2+(3-3)^2+(5-5)^2+(6-1)^2+(7-3)^2}$ = 6.481 (formal proof below) ■

Lemma4: MINDIST(Q,R) lower bounds *D*(Q,C) for any time series C under U.

Proof:

We will first show MINDIST(Q,R,t) lower bounds D(Q,C,t) = $(q_t\text{-}c_t)^2$ for any time series C under **U**. We know that c_t must lie in one of the active regions (Lemma 3). Without loss of generality, let us assume that c_t lies in an active region G i.e. G[1] ≤ c_t ≤ G[3]. Hence MINDIST(Q,G,t) ≤ D(Q,C,t). Also, MINDIST(Q,R,t) <= MINDIST(Q,G,t) (by definition of MINDIST(Q,R,t)). Hence MINDIST(Q,R,t) lower bounds D(Q,C,t). Since MINDIST(Q,R) = $\sqrt{\sum_{t=1}^{n} MINDIST(Q,R,t)}$ and *D*(Q,C) = $\sqrt{\sum_{t=1}^{n} MINDIST(Q,C,t)}$, MINDIST(Q,R,t) ≤ *D*(Q,C,t) implies MINDIST(Q,R) ≤ *D*(Q,C).■

Note that, in general, lower the number of active regions at any instant of time, higher the MINDIST, better the performance of the K-NN algorithm. Also, narrower the regions along the value dimension, higher the MINDIST. The above two principles justify our choice of the dimensions of the APCA space. The odd dimensions help clustering APCA points with similar *cv*ᵢ's, thus keeping the regions narrow along the value dimension. The even dimensions help clustering APCA points that are approximately aligned at the segment end points, thus ensuring only one region (minimum possible) is active for most instants of time.

```
Algorithm ExactRangeSearch(Q, ε, T)
begin
1.  if T is a non-leaf node
2.     for each child U of T
3.        if MINDIST(Q,R)≤ε  ExactRangeSearch(Q, ε, U); // R is MBR of U
4.     endfor
5.  else                                         // T is a leaf node
6.     for each APCA point C in T
7.         if $D_{LB}$(Q',C)≤ε
8.              Retrieve full time series C from database;
9.              if D(Q,C) ≤ε  Add C to result;
10.        endif
11.    endfor
12. endif
end
```

Table 7:Range search algorithm to retrieve all the time series within a range of ε from query time series Q. The function is invoked as `ExactRangeSearch(Q, ε, root_node_of_index)`.

Although we have focussed on K-NN search in this section, the definitions of D_{LB} and MINDIST proposed in this paper are also needed for answering range queries using a multidimensional index structure. The range search algorithm is shown in Table 7. It is a straightforward R-tree-style recursive search algorithm combined with the GEMINI range query algorithm shown in Table 2. Since both MINDIST(Q,R) and $D_{LB}(Q',C)$ lower bound D(Q,C), the above algorithm is correct [16].

In this section, we described how to find the exact nearest neighbors of a query time series using a multidimensional index structure. In Section 3.3.1, we proposed an approximate Euclidean distance measure $D_{AE}(Q,C)$ for fast approximate search. If we want to use the same index structure to answer both exact queries and approximate queries, we can simply replace the distance function $D_{LB}(Q,C)$ in Line 15 of the K-NN algorithm (Table 6) by $D_{AE}(Q,C)$ to switch from exact to approximate queries and vice-versa. Since $D_{AE}(Q,C)$ is a tighter approximation of D(Q,C) than $D_{LB}(Q',C)$, the K-NN algorithm would need to retrieve fewer APCA points from the index before the algorithm stops. This would result in fewer disk accesses to retrieve the full time series corresponding to the retrieved APCA points (Line 11 of Table 6), leading to lower query cost. Since the approximate distance $D_{AE}(Q,C)$ between a time series query Q = $\{q_1, q_2,\ldots q_n\}$ and an APCA point $C = \{cv_1, cr_1, \ldots, cv_M, cr_M\}$ *almost always* lower bounds the Euclidean distance $D(Q,C)$ between Q and the original time series C = $\{c_1, c_2,\ldots, c_n\}$ (see Figure 8), the approximate function can be used to get reasonably accurate results more efficiently using the same index structure.

If an index is used exclusively for approximate search based on D_{AE}, further optimizations are possible. For such an index, we can construct the MBRs as defined in Definition 4.1 i.e. by inserting the APCA point $C = \{cv_1, cr_1, \ldots, cv_M, cr_M\}$ itself instead of the corresponding rectangle $(\{cmin_1, cr_1,\ldots, cmin_M, cr_M\}, \{cmax_1, cr_1, \ldots, cmax_M, cr_M\})$. The MINDIST computation is the same as in the exact case. It can be shown that MINDIST(Q,R) of the query from the above MBR (Definition 4.1) lower bounds $D_{AE}(Q,C)$, therefore ensuring retrieval of APCA points in the order of their distances D_{AE}(Q,C). Since these MBRs are always smaller than the MBRs in Definition 4.2, the MINDISTs will be larger resulting in fewer node accesses of the index structure compared to approximate search using the same index as the exact search and hence even better performance. To exploit this optimization, one can maintain two separate indices (one with MBRs as defined in Definition 4.2 and one with that defined in Definition 4.1) for exact and approximate searches respectively.

5. Experimental evaluation

In this section, we present the results of an extensive empirical study we have conducted to (1) evaluate APCA in terms of cost of computation (2) compare APCA with other dimensionality reduction techniques in terms of pruning power and query response times and (3) ascertain the ability of APCA to support approximate search. The major findings of our study can be summarized as follows:

(1) **Fast Computation:** APCA can be computed efficiently and is hence tractable for indexing.
(2) **High Pruning Power:** APCA has significantly higher pruning power compared to DFT, DWT, PAA and FastMap i.e., fewer false alarms.
(3) **Low Query Cost:** APCA outperforms other indexing techniques, namely DFT, DWT and linear scan, in terms of query cost, often by one to two orders of magnitude.
(4) **Fast Approximate Search:** APCA can support fast approximate search with a high level of accuracy ($\geq$ 80%).

Thus, our experimental results validate the thesis of the paper that APCA is an effective dimensionality reduction technique for time series databases.

5.1 Experiment methodology

We experimentally compare *all* the state of the art indexing techniques with our proposed method. We have taken great care to create high quality implementations of all competing techniques as discussed in detail below.

(1) **SVD**: Each time series of length n (n-dimensional point) is reduced to a point in an N-dimensional space using Singular Value Decomposition [24].

(2) FastMap: Each time series of length n is mapped to a point in an N-dimensional space using FastMap as proposed in [51].

(3) **DFT:** DFT is applied individually to each time series (of length n) as proposed in [1]. Since we are computing the DFT of a real signal, the first imaginary coefficient is zero, and because all objects in our database have had their mean value subtracted, the first real coefficient is also zero. These coefficients do not need to be retained, making room for additional coefficients. We further optimize the representation by utilizing the symmetric properties of the DFT as suggested in [39], i.e., we can simply double the magnitude of the 2nd to (N+1)th real coefficients and use it as the N-dimensional point.

(4) **DWT:** Each time series (of length n) is individually decomposed using the Haar wavelet decomposition as proposed in [9]. Since the objects have zero mean, the first Haar coefficient is always zero. We use the magnitudes of the 2nd to (N+1)th Haar coefficients to construct the N-dimensional point.

(5) **PAA** (also known as Segmented Means [52]): The N-dimensional representation is computed by dividing the time series (of length n) into N-equal length segments and recording the means of those segments as proposed in [24] and [52].

(6) **APCA:** The N-dimensional representation is computed by running Compute_APCA(C, $\lfloor N/2 \rfloor$) on each time series C (of length n).

We performed tests over a range of original dimensionalities (n) varying from 256 to 1024 and reduced dimensionalities (N) varying from 16 to 64. We used two datasets, one chosen because it is very heterogeneous and the other because it is very homogenous.

- **Homogenous Data: Electrocardiogram.** This dataset is taken from the MIT Research Resource for Complex Physiologic Signals [32]. It is a "*relatively clean and uncomplicated*" electrocardiogram. We generated data of 3 different dimensionalities: n=1024, n=512 and n=256.[4] In each case, the dataset consisted of 100,000 points/sequences.
- **Heterogeneous Data: Mixed Bag.** This dataset we created by combining 7 datasets with widely varying properties of shape, structure, noise etc. The only preprocessing performed was to insure that each time series had a mean of zero and a standard deviation of one (otherwise many queries become pathologically easy). The 7 datasets are, Space Shuttle STS-57 [27, 25], Arrhythmia [32], Random Walk [46, 34, 52, 24], INTERBALL Plasma processes (figure 4) [43], Astrophysical data (figure 1) [47], Pseudo Periodic Synthetic Time Series [4]. Exchange rate (figure 4) [47]. Once again, we generated data of 3 different dimensionalities: n=1024, n=512 and n=256 and in each case, the dataset consisted of 100,000 points.

To perform realistic testing we need queries that do not have exact matches in the database but have similar properties of shape, structure, spectral signature, variance etc. To achieve this we used cross validation. We removed 10% of the dataset, and build the index with the remaining 90%. The queries are then randomly taken from the withheld subsection. For each result reported for a particular dimensionality and query length, we averaged the results of 50 experiments.

[4] Because we wanted to include the DWT in our experiments, we chose n to be an integer power of two. We consider a length of 1024 to be the longest query likely to be encountered (by analogy, one might query a text database with a word, a phrase or a complete sentence, but the would be little utility in a paragraph-length text query. A time series query of length 1024 corresponds approximately with sentence length text query).

For simplicity we only show results for nearest neighbor queries, however we obtained similar results for range queries.

5.2 Experimental results: Computing the dimensionality reduced representation

We begin our experiments by measuring the time taken to compute the reduced dimensionality representation for each of the suggested approaches. We did this for query lengths (n) from 32 to 1024 and database sizes of 40KB (20,000 objects) to 640 KB (320,000 objects). The relatively small databases were necessary to include SVD in the experiments. We used a Pentium PC 400 with 256 megs of ram. Experimental runs requiring more than 1,000 seconds were abandoned as indicated by the black-topped histogram bars in Figure 12.

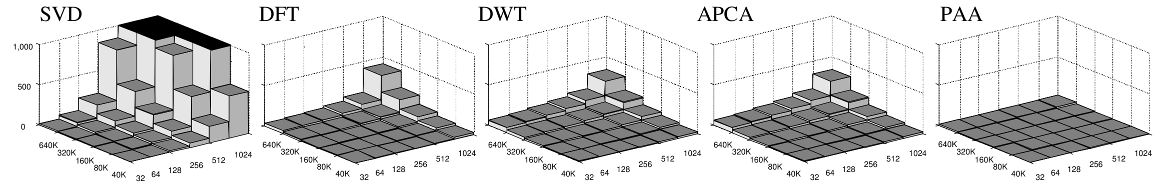


Figure 12: The time taken (in seconds) to compute the reduced representation using various techniques over a range of query lengths (n varying from 32 to 1024) and database sizes (S varying from 40KB (20,000 objects) to 640KB (320,000 objects)). The black topped histogram bars indicate that an experimental run was abandoned at 1,000 seconds.

We can see that SVD, being O(Sn^2), is simply intractable for even moderately sized databases with high query length. We extrapolated from these experiments that it would take several months of CPU time to include SVD in all the experiments in this paper. For this reason we shall exclude SVD from the rest of the experiments (in Section 6 we will discuss more reasons why SVD is not a practical approach). The results for DWT and APCA are virtually indistinguishable, which is to be expected given that the algorithm used to create the APCA spends most of its time in a subroutine call to the DWT. The main conclusion of this experiment is that APCA is tractable for indexing.

5.3 Experimental results: Pruning power

In comparing the four competing techniques (DFT, DWT, APCA and PAA) there exists a danger of implementation bias. That is, consciously or unconsciously implementing the code such that some approach is favored. As an example of the potential for implementation bias in this work consider the following. At query time DFT must do a Fourier transform of the query. We could use the naïve algorithm which is O(n^2) or the faster radix-2 algorithm (padding the query with zeros for $n \neq 2^{\text{integer}}$) which is O($nlogn$). If we implemented the simple algorithm it would make the other indexing methods appear to perform better relative to DFT. While we do present detailed experimental evaluation of an implemented system in the next section, we also present experiments in this section which are free of the possibility of implementation basis. We achieve this by comparing the pruning power of the various approaches. [5]

To compare the pruning power of the four techniques under consideration we measure *P*, the fraction of the database that must be examined before we can guarantee that we have found the nearest match to a 1-NN query.

$$P = \frac{\textit{Number of objects that must be examined}}{\textit{Number of objects in database}} \qquad (13)$$

To calculate *P* we do the following. Random queries are generated (as described above).

[5] We also include FastMap [51] (along with DFT, DWT, APCA and PAA) in our pruning power experiments for completeness.

Objects in the database are examined in order of increasing (feature space) distance from the query until the distance in feature space of the next unexamined object is greater than minimum actual distance of the best match so far. The number of objects examined at this point is the absolute minimum in order to guarantee no false dismissals.

Note the value of P for any transformation depends only on the data and is completely independent of any implementation choices, including spatial access method, page size, computer language or hardware. A similar idea for evaluating indexing schemes appears in [18].

Figure 13 shows the value of P over a range of query lengths and dimensionalities for the experiments that were conducted the Mixed Bag dataset.

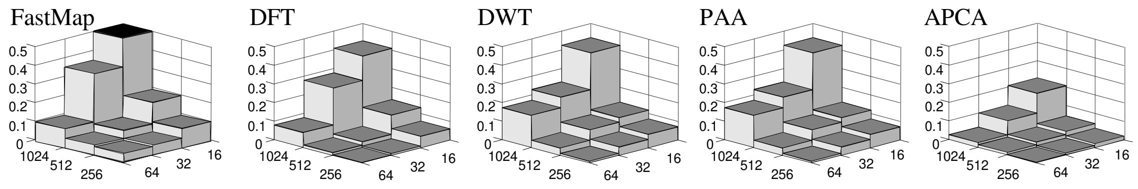


Figure 13: The fraction P, of the Mixed Bag database that must be examined by the five dimensionality reduction techniques being compared, over a range of original dimensionalities (n varying from 256 to 1024) and reduced dimensionalities (N varying from 16 to 64). To preserve a meaningful scale for the graphs, we truncated (poor) values at a certain threshold, the truncated values are shown as black-topped histogram bars.

Note that the results for PAA and DWT are identical. This because the pruning power of DWT and PAA are identical when $N = 2^{\text{integer}}$ [24]. Having empirically shown this fact which was proved in [24, 52] we have excluded PAA from future experiments for clarity.

We repeated the experiment for the Electrocardiogram data, the results are shown in Figure 14.

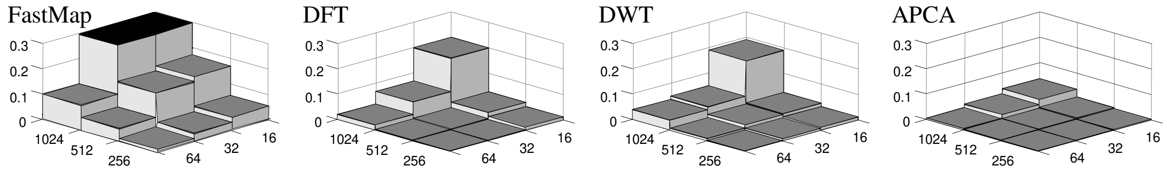


Figure 14: The fraction P, of the Electrocardiogram database that must be examined by the three dimensionality reduction techniques being compared over a range of original dimensionalities (n varying from 256 to 1024) and reduced dimensionalities (N varying from 16 to 64). As in Figure 13, we truncated (poor) values at a certain threshold, the truncated values are shown as black-topped histogram bars.

As shown in Figures 13 and 14, FastMap has significantly lower pruning power compared to the other techniques. This can be explained by the way FastMap works: while other approaches take the high-dimensional representation of the time series and project them to an N-dimensional space, FastMap attempts to model the dataset in an N-dimensional space such that the distances between objects are approximately preserved. That is to say, FastMap considers *only* the distances between the sequences, and disregards the actual shape of the sequences. This attribute of FastMap makes it an ideal technique for situations where the objects to be indexed have no natural features, but there exists a method to compute the distances between them (e.g., indexing strings under edit distance or time series under dynamic time warping [51]). In the case where high quality natural features can be extracted, FastMap is outperformed by techniques (like DFT, DWT and APCA) that use those features to obtain the reduced representation. Having shown this in the pruning power experiments, we exclude FastMap from further experiments for clarity. [6]

[6] Note that for fairness we did not include FastMap in our experiments on the time required to build compute the reduced representation (shown in Figure 12). This is because its implementation requires several user-defined

In both Figure 13 and 14 we can see that APCA outperforms DFT and DWT significantly, generally by an order of magnitude. These experiments indicate that the APCA technique has fewer false alarms, hence lower query cost as confirmed by the experiments below.

5.4 Experimental results: Implemented system

Although the pruning power experiments are powerful predictors of the (relative) performance of indexing systems using the various dimensionality reduction schemes, we include a comparison of implemented systems for completeness. We implemented four indexing techniques: linear scan, DFT-index, DWT-index and APCA-index. We compare the four techniques in terms of the I/O and CPU costs incurred to retrieve the exact nearest neighbor of a query time series. All the experiments reported in this subsection were conducted on a Sun Ultra Enterprise 450 machine with 4 296MHz CPUs, 1 GB of physical memory and several GB of secondary storage, running Solaris 2.6.

Cost Measurements:
We measured the I/O and CPU costs of the four techniques as follows:

(1) *Linear Scan (LS):* In this technique, we perform a simple linear scan on the original *n*-dimensional dataset and determine the exact nearest neighbor of the query. The I/O cost in terms of sequential disk accesses is (S*(n*sizeof(float) + sizeof(id)))/(PageSize). Since sizeof(id) << (n*sizeof(float)), we will ignore the sizeof(id) henceforth. Assuming sequential I/O is about 10 times faster than random I/O, the cost in terms of random accesses is (S*sizeof(float)*n)/(PageSize*10). The CPU cost is the cost of computing the distance D(Q,C) of the query Q from each time series $C = \{c_1, \ldots, c_n\}$ in the database.

(2) *DFT-index (DFT):* In this technique, we reduce the dimensionality of the data from *n* to *N* using DFT and build an index on the reduced space using a multidimensional index structure. We use the hybrid tree as the index structure. The I/O cost of a query has two components: (1) the cost of accessing the nodes of the index structure and (2) the cost of accessing the pages to retrieve the full time series from the database for each indexed item retrieved (cf. Table 6). For the second component, we assume that a full time series access costs one random disk access. The total I/O cost (in terms of random disk accesses) is the number of index nodes accessed plus the number of indexed items retrieved by the K-NN algorithm before the algorithm stopped (i.e. before the distance of the next unexamined object in the indexed space is greater than the minimum of the actual distances of items retrieved so far). The CPU cost also has two components: (1) the CPU time (excluding the I/O wait) taken by the K-NN algorithm to navigate the index and retrieve the indexed items and (2) the CPU time to compute the exact distance *D*(Q,C) of the query Q from the original time series C of each indexed item *C* retrieved (Line 12 in Table 6). The total CPU cost is the sum of the two costs.

(3) *DWT-index (DWT):* In this technique, we reduce the dimensionality of the data from *n* to *N* using DWT and build the index on the reduced space using the hybrid tree index structure. The I/O and CPU costs are computed in the same way as in DFT.

(4) *APCA-index (APCA):* In this technique, we reduce the dimensionality of the data from *n* to *N* using APCA and build the index on the reduced space using the hybrid tree index structure. The I/O and CPU costs are computed in the same way as in DFT and DWT.

We chose the hybrid tree as the index structure for our experiments since it is a space

parameters and we could not guarantee our implementation was the most efficient. We note that its time complexity is O(S*nN*) while that of APCA is O(Snlog(n)).

partitioning index structure ("dimensionality-independent" fanout) and has been shown to scale to high dimensionalities [6, 37, 24]. Since we had access to the source code of the index structure (http://www-db.ics.uci.edu) we implemented the optimization discussed in Section 4 (i.e. to increase leaf node fanout) for our experiments. We used a page size of 4KB for all our experiments.

Dataset: We used the Electrocardiogram (ECG) database for these experiments. We created 3 datasets from the ECG database by choosing 3 different values of query length *n* (256, 512 and 1024). For each dataset, we reduced the dimensionality to $N = 16$, $N = 32$ and $N = 64$ using each of the 3 dimensionality reduction techniques (DFT, DWT and APCA) and built the hybrid tree indices on the reduced spaces (resulting a total of 9 indices for each technique). As mentioned before, the queries were chosen randomly from the withheld section of the dataset. All our measurements are averaged over 50 queries.

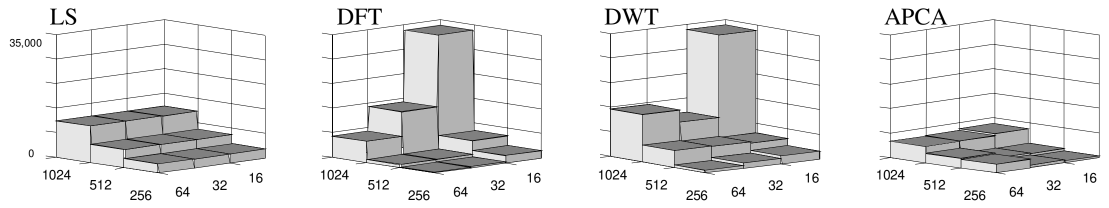


Figure 15: Comparison of LS, DFT, DWT and APCA techniques in terms of I/O cost (number of random disk accesses) over a range of original dimensionalities (n varying from 256 to 1024) and reduced dimensionalities (N varying from 16 to 64). For LS, the cost is computed as number_sequential_disk_accesses/10. We used the ECG dataset for this experiement.

Figure 15 compares the LS, DFT, DWT and APCA techniques in terms of I/O cost (measured by the number of random disk accesses) for the 3 datasets ($n = 256$, 512 and 1024) and 3 different dimensionalities of the index ($N = 16$, 32 and 64). The APCA technique significantly outperforms the other 3 techniques in terms of I/O cost. The LS technique suffers due to the large database size (e.g., 100,000 sequential disk accesses for $n = 1024$ which is equivalent to 10,000 random disk accesses). Although LS is not considerably worse than APCA in terms of I/O cost, it is significantly worse in terms of the overall cost due to its high CPU cost component (see Figure 16). The DFT and DWT suffer mainly due to low pruning power (cf. Figure 14). Since DFT and DWT retrieve a large number of indexed items before it can guaranteed that the exact nearest neighbor is among the retrieved items, the second component of the I/O cost (that of retrieving full time series from the database) tends to be high. The DFT and DWT costs are the highest for large *n* and small *N* (e.g., $n = 1024$, N=16) as the pruning power is the lowest for those values (cf. Figure 14). The DWT technique shows a U-shaped curve for $n = 1024$: when the reduced dimensionality is low ($N = 16$), the second component of the I/O cost is high due to low pruning power, while when *N* is high ($N = 64$), the first component of the I/O cost (index node accesses) becomes large due to dimensionality curse. We did not observe such U-shaped behavior in the other techniques as their costs were either dominated entirely by the first component (e.g., $n = 256$ and $n = 512$ cases of APCA) or by the second component (all of DFT and $n = 1024$ case of APCA).

Figure 16 compares the LS, DFT, DWT and APCA techniques in terms of CPU cost (measured in seconds) for the 3 datasets ($n = 256$, 512 and 1024) and 3 different dimensionalities of the index ($N = 16$, 32 and 64). Once again, the APCA technique significantly outperforms the other 3 techniques in terms of CPU cost. The LS technique is the worst in terms of CPU cost as it computes the exact (*n*-dimensional) distance D(Q,C) of the query Q from every time series C in the database. The DFT and DWT techniques suffer again due to their low pruning power (cf. Figure 14), causing the second component of the CPU cost (i.e. the time to compute the exact

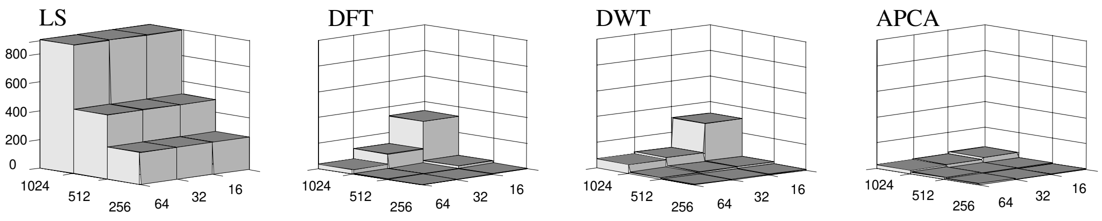


Figure 16: Comparison of LS, DFT, DWT and APCA techniques in terms of CPU cost (seconds) over a range of original dimensionalities (n varying from 256 to 1024) and reduced dimensionalities (N varying from 16 to 64). We used the ECG dataset for this experiement.

distances *D*(Q,C) of the original time series of the retrieved APCA points from the query) to become high.

5.5 Experimental results: Scalability Experiments

In the previous subsection, we reported the I/O and CPU costs instead of the actual wall clock times required to answer the queries. We did that for two reasons: first, it gives us information about the two components of the cost individually enabling us to do a better comparison and second, the wall clock times would have been misleading as it would not have included the I/O cost component at all (because the index always fit in main memory (1GB of physical memory) and there was no actual I/O involved). To measure wall clock times, we ran experiments where the index did not fit in main memory (by using larger datasets (upto 500,000 objects) and running them on a machine with less physical memory); we report those results in this subsection. We compare the four indexing techniques, namely, linear scan (LS), DFT-index (DFT), DWT-index (DWT) and APCA-index (APCA), in terms of the wall clock time (elapsed time) required to retrieve the exact nearest neighbor of a query time series. The wall clock time includes both the I/O wait time and the compute time (CPU cost). As before, we used the hybrid tree index structure for our experiments, the page size was 4KB. Due to unavailability of larger real-life datasets, we generated 200,000 and 500,000 object databases from the 100,000 object ECG database by slightly perturbing the datapoints in the sequence to generate the neighboring datapoints. For this experiment, we fixed the original dimensionality to n=1024, i.e., we generated 3 datasets containing 100,000, 200,000 and 500,000 1024-dimensional points respectively. We then reduced the dimensionality to N=64 using each of the 3 dimensionality

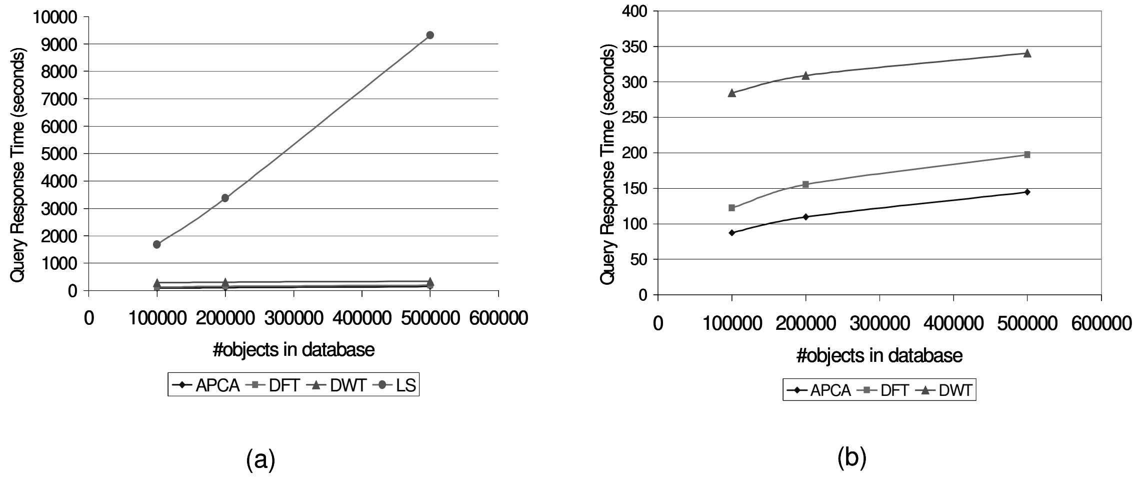


Figure 17: (a) Comparison of LS, DFT, DWT and APCA techniques in terms of query response time (wall clock time in seconds) with database size varying from 100,000 objects to 500,000 objects (n=1024, N=64). (b) Same as (a) showing just DFT, DWT and APCA for better comparison. We synthetically generated the dataset from the ECG dataset.

reduction techniques and built hybrid tree indices on the reduced space. The queries were chosen randomly as discussed before and the measurements are averaged over 50 queries. The experiments were conducted on a machine Sun Ultra 2 machine with 2 168MHz CPUs and 256MB of physical memory.

Figure 17 compares the LS, DFT, DWT and APCA techniques in terms of wall clock time required to retrieve the exact nearest neighbor of a query time series. As shown in Figure 17(a), LS is much slower compared to other 3 techniques mainly due to its high computational cost (as it computes the exact 1024-dimensional distance D(Q,C) of the query Q from every time series C in the database). Since the plots of APCA, DFT and DWT are very close to each other in 17(a), we replot them in Figure 17(b) for a better comparison. As before, APCA significantly outperforms the other techniques. DFT and DWT suffer due to their low pruning power leading to longer I/O wait times as well as more distance computations and hence higher wall clock times.

5.6 Experimental results: Approximate queries

In this section we evaluate the ability of APCA to support approximate search. To evaluate the quality of the returned answer set we used the precision concept from information retrieval [25].

$$precision \equiv \frac{relevant\ and\ retrieved}{retrieved} \qquad (14)$$

The ideal value precision is 1, indicating that all the items retrieved are relevant.

We compared D_{AE} to D_{LB} on the Mixed Bag dataset for K-NN queries. We use a combination of long queries, n = 1024 and few dimensions, N = 16, because this is when all techniques have the worst query response time.

For simplicity, we only consider the second component of the I/O cost (i.e. that of retrieving the full time series from the database (Line 11 in Table 6)). This is reasonable since for n = 1024 and N = 16, the second component dominates the total I/O cost (cf. Figure 15). For D_{AE} we adopted the following simple protocol. We retrieved 3K objects from the database, then used the true Euclidean distance to prune away 2K objects. We then measured the precision of the remaining K items. Here the precision is simply the fraction of items that D_{AE} reported as being in the set of the top K neighbors, which actually belong that set. The results for each value of K are averaged over 50 runs. The results are reported in Table 8.

K	Disk Accesses D_{LB}	Disk Accesses D_{AE}	Precision
5	3,212.3	15	0.82
10	3,174.3	30	0.84
20	3,412.1	60	0.81

Table 8: The precision of D_{AE} for various values of K, with the number of disk accesses required by both D_{AE} (approximate search) and D_{LB} (exact search.).

We can see that D_{AE} is useful for very fast queries that give approximately the same answer set as exact search. This feature is very useful interactive exploratory analysis of massive datasets.

6. Discussion

Now that the reader is more familiar with the contribution of this paper we will briefly revisit related work. We believe that this paper is the first to suggest locally adaptive indexing time series indexing. A locally adaptive representation for 2-dimensional shapes was suggested in [8] but no indexing technique was proposed. Also in the context of images, it was noted by [50] that the use of the first N Fourier coefficients does not guarantee the optimal pruning power. They introduced a technique where they adaptively choose which coefficients to keep after looking at the data. However, the choice of coefficients was based upon a global view of the data. Later work [49] in the context of time series noted that the policy of using the first *N* wavelet coefficients [9, 49, 22] is not generally optimal, but "*keeping the largest coefficients needs additional indexing space and (more complex) indexing structures*". Singular value decomposition is also a data adaptive technique used for time series [28, 24, 23], but it is globally, not locally, adaptive. Recent work [7] has suggested first clustering a multi-dimensional space and then doing SVD on local clusters, making it a semi-local approach. It is not clear however that this approach can be made work for time series. Finally a representation similar to APCA was introduced in [15] (under the name "piecewise flat approximation") but no indexing technique was suggested.

6.1 Other factors in choosing a representation to support indexing.

Although we have experimentally demonstrated that the APCA representation is superior to other approaches in terms of query response time, there are other factors which one may wish to consider when choosing a representation to support indexing. We will briefly consider some of these issues here.

One important issue is the length of queries allowed. For example the wavelet approach only allows queries with lengths that are an integer power of two [24]. This problem could be addressed by having the system pad in zeros up to the next power of two, then filter out the additional false hits. However this will severely degrade performance. The APCA approach, in contrast, allows arbitrary length queries.

Another important point to consider are the set of distance measures supported by a representation. It has been argued that for many applications, distance measures other than Euclidean distance are required. For example in [52] the authors noted that the PAA representation can support queries where the distance measure is an arbitrary Lp norm (i.e. p = 1,2,.., ∞). We refer the interested reader to that paper for a discussion of the utility of these distance metrics, but note that the APCA representation can easily handle such queries by trivial generalizations of Equations 3 and 5 to Equations 16 and 17.

$$D_{AE}(Q,C) \equiv \sqrt[p]{\sum_{i=1}^{M} \sum_{k=1}^{cl_i - cl_{i-1}} \left(c_i - q_{k+cl_{k-1}}\right)^p} \quad (16)$$

$$D_{LB}(Q',C) \equiv \sqrt[p]{\sum_{i=1}^{M} cl_i (q_i - c_i)^p} \quad (17)$$

Note that as with the approach of [52] we can reuse the same index for any Lp norm.

Almost all time series databases are dynamic. For example, NASA updates its archive of Space Shuttle telemetry data after each mission. Some databases are updated continuously, for example financial datasets are updated (at least) at the end of each business day. It is therefore important that any indexing technique be able to support dynamic inserts. Our proposed approach (along with DWT, DFT and PAA) has this property. However dynamic insertion is the Achilles heel of SVD, a single insertion requires recomputing the entire index. Faster methods do exist for incremental updates, but they introduce the possibility of false dismissals [10].

7. Conclusions and directions for future work

The main contribution of this paper is to show that a simple, novel dimensionality reduction technique, namely APCA, can outperform more sophisticated transforms by one to two orders of magnitude. In contrast to popular belief [52, 15], we have shown that the APCA representation can be indexed using a multidimensional index structure. In addition to fast exact queries, the approach also allows even faster approximate querying on the same index structure. We have also shown that our approach can support arbitrary Lp norms, again using a single index structure.

The idea of locally adaptive representation is applicable not just to time-series data but to sequence data in general (one-dimensional as well as multidimensional sequences). For example, we applied such a representation for 2-dimensional shapes in [8]. As future work, we intend to apply this idea of locally adaptive representation to more such application domains. We also intend to increase the speedup of our method even further by exploiting the similarity of adjacent sequences (in a similar spirit to the "trail indexing" technique introduced in [16]).

Acknowledgements

We thank Christos Faloutsos providing us with the FastMap code and helpful suggestions on the paper.

References

[1] Agrawal, R., Faloutsos, C., & Swami, A. (1993). Efficient similarity search in sequence databases. *Proceedings of the 4th Conference on Foundations of Data Organization and Algorithms*.

[2] Agrawal, R., Psaila, G., Wimmers, E. L., & Zait, M. (1995). Querying shapes of histories. *Proceedings of the 21st International Conference on Very Large Databases*.

[3] Agrawal, R., Lin, K. I., Sawhney, H. S., & Shim, K. (1995). Fast similarity search in the presence of noise, scaling, and translation in times-series databases. *Proceedings of 21th International Conference on Very Large Data Bases*. Zurich. pp 490-50.

[4] Bay, S. D. (2000). The UCI KDD Archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science.

[5] Bennett, K., Fayyad, U. & Geiger. D. (1999). Density-based indexing for approximate nearest-neighbor queries. *Proceedings 5th International Conference on Knowledge Discovery and Data Mining*. pp. 233-243, ACM Press, New York.

[6] Chakrabarti, K & Mehrotra, S. (1999). The Hybrid Tree: An index structure for high dimensional feature spaces. *Proceedings of the 15th IEEE International Conference on Data Engineering*.

[7] Chakrabarti, K & Mehrotra, S (2000). Local dimensionality reduction: A new approach to indexing high dimensional spaces. *Proceedings of the 26th Conference on Very Large Databases, Cairo, Egypt*.

[8] Chakrabarti, K., Ortega-Binderberger, M., Porkaew, K & Mehrotra, S. (2000) Similar shape retrieval in MARS. *Proceeding of IEEE International Conference on Multimedia and Expo.*

[9] Chan, K. & Fu, W. (1999). Efficient time series matching by wavelets. *Proceedings of the 15th IEEE International Conference on Data Engineering*.

[10] Chandrasekaran, S., Manjunath, B.S., Wang, Y. F. Winkeler, J. & Zhang. H. (1997). An eigenspace update algorithm for image analysis. *Graphical Models and Image Processing*, Vol. 59, No. 5, pp. 321-332.

[11] Chu, K & Wong, M. (1999). Fast time-series searching with scaling and shifting. *Proceedings of the 18th ACM Symposium on Principles of Database Systems*, Philadelphia.

[12] Das, G., Lin, K. Mannila, H., Renganathan, G., & Smyth, P. (1998). Rule discovery from time series. *Proceedings of the 3rd International Conference of Knowledge Discovery and Data Mining*. pp 16-22.

[13] Debregeas, A. & Hebrail, G. (1998). Interactive interpretation of Kohonen maps applied to curves. *Proceedings of the 4th International Conference of Knowledge Discovery and Data Mining*. pp 179-183.

[14] Evangelidis, G., Lomet, D. & Salzberg B (1997). The hB-Pi-Tree: A multi-attribute index supporting concurrency, recovery and node consolidation. VLDB Journal 6(1): 1-25.

[15] Faloutsos, C., Jagadish, H., Mendelzon, A. & Milo, T. (1997). A signature technique for similarity-based queries. SEQUENCES 97, Positano-Salerno, Italy.

[16] Faloutsos, C., Ranganathan, M., & Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data. Minneapolis.

[17] Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. *Proceedings ACM SIGMOD Conference*. pp 47-57.

[18] Hellerstein, J. M., Papadimitriou, C. H., & Koutsoupias, E. (1997). Towards an analysis of indexing schemes. *Sixteenth ACM Symposium on Principles of Database Systems*.

[19] Hjaltason, G., Samet, H (1995). Ranking in spatial databases. Symposium on Large Spatial Databases. pp 83-95.

[20] Huang, Y. W., Yu, P. (1999). Adaptive Query processing for time-series data. *Proceedings of the 5th International Conference of Knowledge Discovery and Data Mining*. pp 282-286.

[21] Jonsson. H., & Badal. D. (1997). Using signature files for querying time-series data. *First European Symposium on Principles of Data Mining and Knowledge Discovery*.

[22] Kahveci, T. & Singh, A (2001). Variable length queries for time series data. *Proceedings 17th International Conference on Data Engineering*. Heidelberg, Germany.

[23] Kanth, K.V., Agrawal, D., & Singh, A. (1998). Dimensionality reduction for similarity searching in dynamic databases. *Proceedings ACM SIGMOD Conf.*, pp. 166-176.

[24] Keogh, E,. Chakrabarti, K,. Pazzani, M. & Mehrotra (2000) Dimensionality reduction for fast similarity search in large time series databases. Journal of Knowledge and Information Systems.

[25] Keogh, E. & Pazzani, M. (1999). Relevance feedback retrieval of time series data. *Proceedings of the 22th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval.*

[26] Keogh, E., & Pazzani, M. (1998). An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. *Proceedings of the 4th International Conference of Knowledge Discovery and Data Mining*. pp 239-241, AAAI Press.

[27] Keogh, E., & Smyth, P. (1997). A probabilistic approach to fast pattern matching in time series databases. *Proceedings of the 3rd International Conference of Knowledge Discovery and Data Mining*. pp 24-20.

[28] Korn, F., Jagadish, H & Faloutsos. C. (1997). Efficiently supporting ad hoc queries in large datasets of time sequences. *Proceedings of* SIGMOD '97, Tucson, AZ, pp 289-300.

[29] Lam, S., & Wong, M (1998) A fast projection algorithm for sequence data searching. Data & Knowledge Engineering 28(3): 321-339.

[30] Li, C,. Yu, P. & Castelli V.(1998). MALM: A framework for mining sequence database at multiple abstraction levels. CIKM. pp 267-272.

[31] Loh, W., Kim, S & Whang, K. (2000). Index interpolation: an approach to subsequence matching supporting normalization transform in time-series databases. *Proceedings 9th International Conference on Information and Knowledge Management.*

[32] Moody, G. (2000). MIT-BIH Database Distribution [http://ecg.mit.edu/index.html]. Cambridge, MA.

[33] Ng, M. K., Huang, Z., & Hegland, M. (1998). Data-mining massive time series astronomical data sets - a case study. *Proceedings of the 2nd Pacific-Asia Conference on Knowledge Discovery and Data Mining*. pp 401-402

[34] Park, S., Lee, D., & Chu, W. (1999). Fast retrieval of similar subsequences in long sequence databases. *In 3rd IEEE Knowledge and Data Engineering Exchange Workshop*.

[35] Pavlidis, T. (1976). Waveform segmentation through functional approximation. IEEE Transcations on Computers, Vol C-22, NO. 7 July.

[36] Perng, C., Wang, H., Zhang, S., & Parker, S. (2000). Landmarks: a new model for similarity-based pattern querying in time series databases. *Proceedings 16th International Conference on Data Engineering*. San Diego, USA.

[37] Porkaew, K., Chakrabarti, K. & Mehrotra, S. (1999). Query refinement for multimedia similarity retrieval in MARS. Proceedings of the ACM International Multimedia Conference, Orlando, Florida, pp 235-238

[38] Qu, Y., Wang, C. & Wang, S. (1998). Supporting fast search in time series for movement patterns in multiples scales. *Proceedings 7th International Conference on Information and Knowledge Management*. Washington, DC.

[39] Refiei, D. (1999). On similarity-based queries for time series data. *Proc of the 15th IEEE International Conference on Data Engineering*. Sydney, Australia.

[40] Roussopoulos, N., Kelley, S. & Vincent, F. (1995). Nearest neighbor queries. SIGMOD Conference 1995: 71-79.

[41] Seidl, T. & Kriegel, H. (1998). Optimal multi-step k-nearest neighbor search. SIGMOD Conference: pp 154-165.

[42] Shatkay, H., & Zdonik, S. (1996). Approximate queries and representations for large data sequences. *Proceedings 12th IEEE International Conference on Data Engineering*. pp 546-553.

[43] Shevchenko, M. (2000). [http://www.iki.rssi.ru/] Space Research Institute. Moscow, Russia.

[44] Stollnitz, E., DeRose, T., & Salesin, D. (1995). Wavelets for computer graphics A primer: IEEE Computer Graphics and Applications.

[45] Struzik, Z. & Siebes, A. (1999). The Haar wavelet transform in the time series similarity paradigm. *Proceedings 3rd European Conference on Principles and Practice of Knowledge Discovery in Databases*. pp 12-22.

[46] Wang, C. & Wang, S. (2000). Supporting content-based searches on time Series via approximation. *International Conference on Scientific and Statistical Database Management*.

[47] Weigend, A. (1994). The Santa Fe Time Series Competition Data [http://www.stern.nyu.edu/~aweigend/Time-Series/SantaFe.html]

[48] Welch. D. & Quinn. P (1999). http://wwwmacho.mcmaster.ca/Project/Overview/status.html

[49] Wu, Y., Agrawal, D. & Abbadi, A.(2000). A Comparison of DFT and DWT based Similarity Search in Time-Series Databases. *Proceedings of the 9th International Conference on Information and Knowledge Management*.

[50] Wu, D., Agrawal, D., El Abbadi, A. Singh, A. & Smith, T. R. (1996). Efficient retrieval for browsing large image databases. *Proc of the 5th International Conference on Knowledge Information*. pp 11-18, Rockville, MD.

[51] Yi, B,K., Jagadish, H., & Faloutsos, C. (1998). Efficient retrieval of similar time sequences under time warping. *IEEEE International Conference on Data Engineering*. pp 201-208.

[52] Yi, B,K., & Faloutsos, C.(2000). Fast time sequence indexing for arbitrary Lp norms. *Proceedings of the 26st International Conference on Very Large Databases*, Cairo, Egypt.