Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases

Eamonn Keogh Kaushik Chakrabarti Sharad Mehrotra Michael Pazzani
Department of Information and Computer Science
University of California, Irvine, California 92697 USA
(949) 824-7210

eamonn@ics.uci.edu kaushik@ics.uci.edu sharad@ics.uci.edu 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, Content-Based 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].

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.

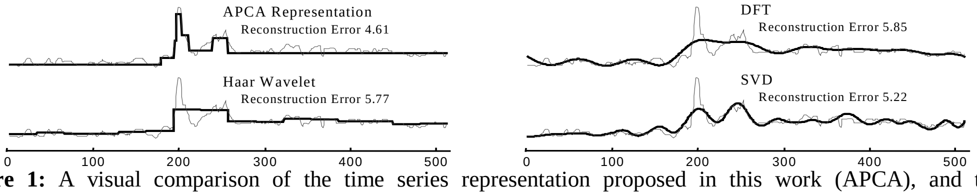


Figure 1: A visual comparison of the time series representation proposed in this work (APCA), and the 3 other representations advocated in the literature. For fair comparison, all representations have the same compression ratio.

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.

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 and the 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 some conclusions.

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:

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

Figure 2 shows the intuition behind the Euclidean distance.

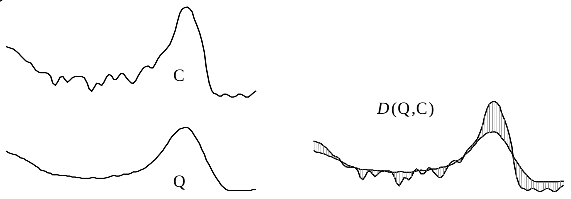


Figure 2: The intuition behind the Euclidean distance. The Euclidean distance can be visualized as the square root of the sum of the squared lengths of the gray lines.

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

1. **Whole Matching**. Here it assumed that all sequences to be compared are the same length n.

2. **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 ci, 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.

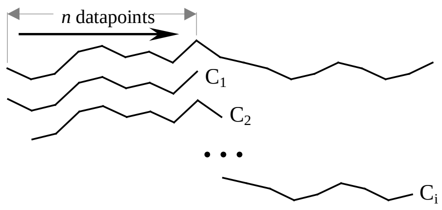


Figure 3: The subsequence matching problem can be converted into the whole matching problem by sliding a "window" of length *n* across the long sequence and making copies of the data falling within the windows.

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. 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.

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 a 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:

$$D_{\text{index space}}(A,B) \leq D_{\text{true}}(A,B) \quad (2)$$

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

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 $D_{\text{index space}}(A,B) \leq D_{\text{true}}(A,B)$ 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 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, for clarity we will refer to it as Piecewise Aggregate Approximation (PAA). 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.

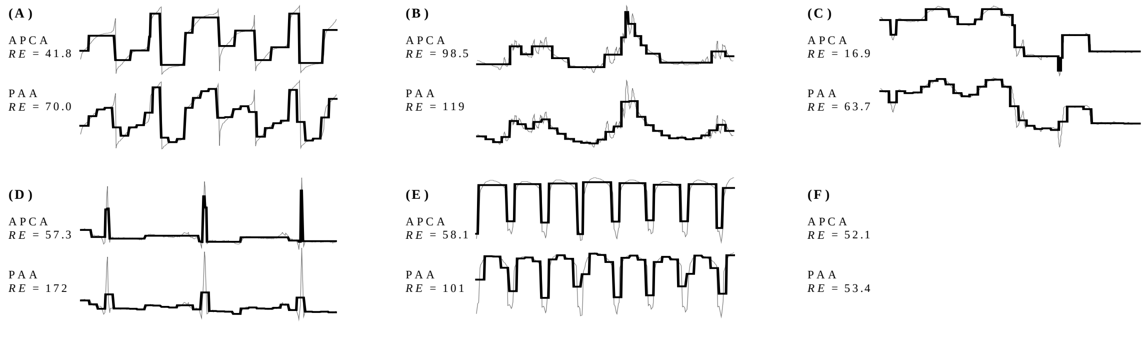
Figure 4 illustrates a fair comparison between the two techniques on several real datasets. Note that for the task of indexing, subjective feelings about which technique "looks better" are irrelevant. All that matters is the quality of the approximation, which is given by the reconstruction error (because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

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

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

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 ith 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 ith segment can be calculated as $cr_i - cr_{i-1}$. Figure 5 illustrates this notation.



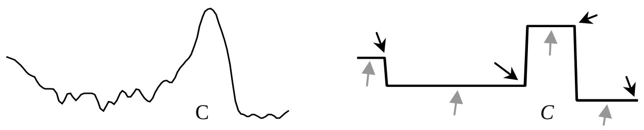


Figure 5: A time series C and its APCA representation *C*, with $M = 4$

In general, finding the optimal piecewise polynomial representation of a time series requires a $O(Nn^2)$ dynamic programming algorithm [15, 35]. For most purposes, however, an optimal representation is not required. Most researchers, therefore, use a greedy suboptimal approach instead [42, 27, 46]. In this work we utilize an original algorithm which produces high quality approximations in $O(n\log(n))$. 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.

3.2 Distance Measures Defined for APCA

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 6 illustrates the intuition behind the formulas.

3.2.1 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:

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

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.

Consider the time series A = {-1, -1, -2, 1, 2}, B = {1, 1, 0, -1, -1} and C = {0, 1, 0, 1, -2} their APCA representations *A* = {<-1,2>, <$\frac{1}{3}$,5>}, *B* = {<$\frac{2}{3}$,3>,<-1,5>}, C = {<$\frac{1}{2}$,2>,<-$\frac{1}{3}$,5>}

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

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

Assume that D_{AE}(Q,*C*) does satisfy the triangular inequality, then we can write

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

Apply Equation 3 $\quad 5.0662 \le 3.8079 + 1.2247$

But this implies $\quad 5.0662 \le 5.0326$

So the assumption was wrong and 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.

3.2.2 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 3.1 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 6 III.

Q' is defined as:

$Q' = \{<qv_1,qr_1>,\ldots,<qv_M,qr_M>\}$,

$$\text{where } qr_i = cr_i \text{ and } qv_i = mean(q_{cr_{i-1}+1,\ldots,}q_{cr_i}) \tag{4}$$

$D_{LB}(Q',C)$ is defined as:

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

This distance measure does lower bound the Euclidean distance. For brevity we omit the proof which is a generalization of the proof for the special case of equal length segments in [24].

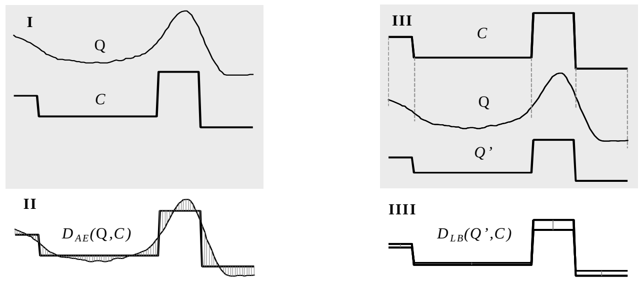


Figure 6: A visualization of the two distance measures define on the APCA representation. **I**) A query time series Q and a APCA object *C*. **II**) The D_{AE} measure can be visualized as the Euclidean distance between Q and the reconstruction of *C*. **III**) *Q'* is obtained by projecting the endpoints 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.

4. INDEXING APCA

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;
 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) ≤ top.dist
 5.       Remove C from temp;
 6.       Add C to result;
 7.       if |result| = K return result;
 8.  queue.Pop();
 9.   if top is an APCA point $C$
 10.    Retrieve full time series C from database;
 11.      temp.insert(C, $D$(Q,C));
 12.  else if top is a leaf node
 13.      for each data item C in top
 14.          queue.push(C, $D_{LB}(Q',C)$);
 15.  else                  // top is a non-leaf node
 16.      for each child node U in top
 17.             queue.push(U, MINDIST(Q,R))  // R
                is MBR associated with U
```

Table 5: 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 5. 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 3.2.2) 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 popping out the item from the top of queue at each step (Line 8). 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 9-11). 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 12-17). 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 ∉ *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 ∪ result). If we can guarantee that ∀ C ∈ **I**, ∀ E ∉ **I**, $D_{LB}(Q',C) \leq D(Q,E)$, then the condition "D(Q,C) ≤ 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 ∀ C ∈ **I**, ∀ E ∉ **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. ∀ C ∈ **I**, ∀ E ∉ **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)

$$l_i = \min_{C\,in\,U} cv_{(i+1)/2} \quad \text{if i is odd} \tag{6}$$

$$= \min^{L}_{C\,in\,U} cr_{i/2} \quad \text{if i is even}$$

$$h_i = \max^{L}_{C\,in\,U} cv_{(i+1)/2} \quad \text{if i is odd}$$

$$= \max^{L}_{C\,in\,U} cr_{i/2} \quad \text{if i is even}$$

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

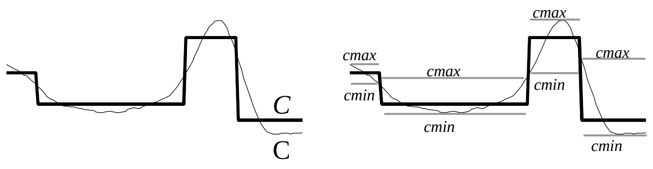


Figure 7: Definition of *cmax*i and *cmin*i for computing MBRs

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 $cmax_i$ and $cmin_i$ denote the maximum and minimum values of the corresponding time series C among the datapoints in the ith segment i.e.

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

$$cmin_i = \min_{t=cr_{i-1}+1}^{cr_i}(c_t)$$

The $cmax_i$ and $cmin_i$ of a simple time series with 4 segments is shown in Figure 7.

We define the MBR R = (L, H) associated with **U** as follows:

Definition 4.2 (New definition of MBR)

$$l_i = \min_{C\,in\,U} cmin_{(i+1)/2} \quad \text{if i is odd} \tag{8}$$

$$= \min_{C\,in\,U} cr_{i/2} \quad \text{if i is even}$$

$$h_i = \max_{C\,in\,U} cmax_{(i+1)/2} \quad \text{if i is odd}$$

$$= \max_{C\,in\,U} cr_{i/2} \quad \text{if i is even}$$

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. The same is true for MBRs associated with non-leaf nodes. Since we use one of the existing multidimensional index structures for this purpose, 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 14 of the K-NN algorithm in Table 5). 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**, $D_{LB}(Q',C) \le D(Q,E)$ (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.

Lemma 1:

If MINDIST(Q,R) $\le$ D(Q,C) for any time series C placed under **U**, the algorithm in Table 5 is correct i.e. $\forall$ C $\in$ **I**, $\forall$ E $\notin$ **I**, $D_{LB}(Q',C) \le D(Q,E)$ 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**, $D_{LB}(Q', C) \le D_{LB}(Q',E)$
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**, $D_{LB}(Q',C) \le$ MINDIST(Q,R).

Since $D_{LB}(Q',E) \le D(Q,E)$ (see Section 3.2.2), (1) implies $\forall$ C $\in$ **I**, $D_{LB}(Q',C) \le D(Q,E)$. If MINDIST(Q,R) $\le$ D(Q,E) for any time series E under **U**, (2) implies that $\forall$ C $\in$ **I**, $D_{LB}(Q', C) \le D(Q,E)$. Since either (1) or (2) must be true for any item E $\notin$ **I**, $\forall$ C $\in$ **I**, $\forall$ E $\notin$ **I**, $D_{LB}(Q',C) \le D(Q,E)$.■

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.

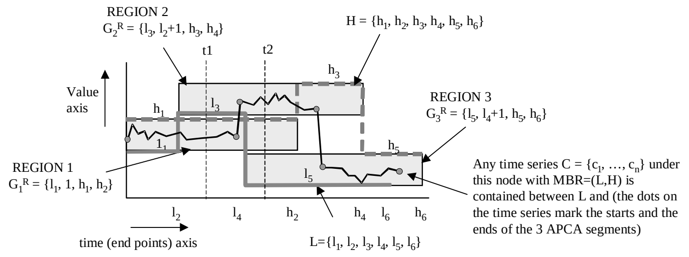

REGION 2
$G_2^R = \{l_3, l_2+1, h_3, h_4\}$
$H = \{h_1, h_2, h_3, h_4, h_5, h_6\}$
t1
t2
h_3
Value axis
h_1
l_3
REGION 3
$G_3^R = \{l_5, l_4+1, h_5, h_6\}$
l_1
h_5
REGION 1
$G_1^R = \{l_1, 1, h_1, h_2\}$
l_5
Any time series C = {c_1, …, c_n} under this node with MBR=(L,H) is contained between L and (the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

Figure 8: 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]}

Let us consider a node **U** with MBR R = (L,H). We can view the MBR as two APCA representations $L=\{l_1, l_2, …, l_N\}$ and $H = \{h_1, h_2, …, h_N\}$. The view of a 6-dimensional MBR $(\{l_1,l_2,..,l_6\}, \{h_1,h_2,…,h_6\})$ as two APCA representations $\{l_1, l_2, …, l_6\}$ and $\{h_1, h_2, …, h_6\}$ is shown in Figure 8. Any time series $C = \{c_1, c_2,…, c_n\}$ under the node **U** is "contained" within the two bounding time series L and H (as shown in Figure 9). In order to formalize this notion of containment, we define a set of M regions associated with R. The ith 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 ith segment of all time series stored under **U**. The boundary of a region G, being 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,

$$G_i^R[1] = \min_{C\ under\,U}\ (cmin_i) \tag{9}$$

$$G_i^R[2] = \min_{C\ under\ U}\ (cr_{i-1}+1)$$

$$G_i^R[3] = \max_{C\ under\ U}\ (cmax_i)$$

$$G_i^R[4] = \max_{C\ under\ U}\ (cr_i)$$

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)

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

Figure 8 shows the 3 regions associated with the 6-dimensional MBR ($\{l_1,l_2,..,l_6\}$, $\{h_1,h_2,\ldots,h_6\}$). 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 8, 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. $\bigvee_{G_i^R\ is\ active} G_i^R[1] \le c_t \le G_i^R[3]$.

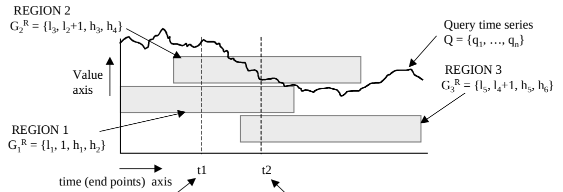
2.1 Lemma 2:The value c_t of C under U at time instant t must lie within one of the regions active at t.

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. $\bigvee_{G_i^R\ is\ active} G_i^R[1] \le c_t \le G_i^R[3]$.■


REGION 2
$G_2^R = \{l_3, l_2+1, h_3, h_4\}$
Query time series
$Q = \{q_1, \ldots, q_n\}$
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 (end points) axis
t1
t2

MINDIST(Q,R,t_1)
= min (MINDIST(Q, G_1^R, t1), MINDIST(Q, G_2^R, t1))
= min ($(q_{t1}-h1)^2$, $(q_{t1}-h3)^2$)
= $(q_{t1}-h3)^2$

MINDIST(Q,R,t_2)
= min (MINDIST(Q, G_1^R, t2), MINDIST(Q, G_2^R, t2), MINDIST(Q, G_3^R, t2))
= min ($(q_{t2}-h1)^2$, 0, $(q_{t2}-h3)^2$)
= 0

Figure 9: Computation of MINDIST

Given a query time series Q = $\{q_1, q_2, \ldots, q_n\}$, the minimum distance MINDIST(Q,R,t) of Q from R at time instant t (cf. Figure 9) is given by $\min_{region\ G\ is\ active\ at\ t}$ MINDIST(Q,G,t) where

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

MINDIST(Q,R) is defined as follows:

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

Lemma3: 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) = $(q_t-c_t)^2$ 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. $G[1] \le c_t \le G[3]$. 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)
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. else                             // T is a leaf node
5.   for each APCA point C in T
6.     if D_LB(Q',C)≤ ε
7.     Retrieve full time series C from database;
8.     if D(Q,C) ≤ ε  Add C to result;
```

Table 6: 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 6. 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 3.2.1, 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 14 of the K-NN algorithm (Table 5) 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 10 of Table 5), 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 7), 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 will experimentally demonstrate the superiority of APCA in terms of query response time.

For completeness 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. For example we utilized the symmetric properties of the DFT as suggested in [39]. Additionally when taking 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. We do not include these constants in the index, making room instead for two additional coefficients that carry information. All other approaches are similarly optimized.

5.1 Experiment methodology

We performed all tests over a range of reduced dimensionalities (*N*) and query lengths (i.e original dimensionalities, *n*). Because we wanted to include the DWT in our experiments, we are limited to query lengths that are 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).

We tested on two datasets, one chosen because it is very heterogeneous and one chosen 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. The total size of the data is 100,000 objects.

- **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. 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]. The total size of the data is 100,000 objects.

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.

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

5.2 Experimental results: Pruning power

In comparing the four competing techniques 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 $O(n\log n)$. 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.

To compare the pruning power of the four techniques under

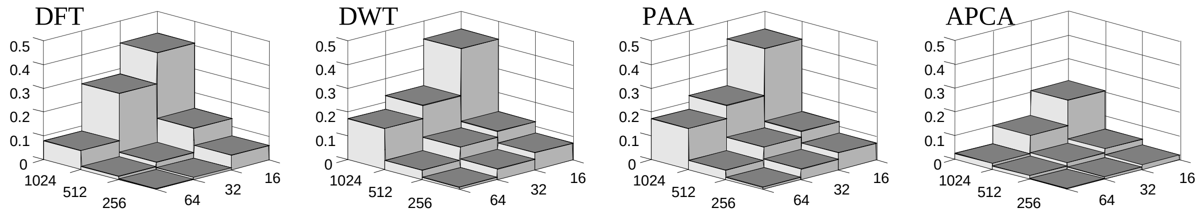


Figure 10: The fraction *P*, of the Mixed Bag database that must be examined by the four dimensionality reduction techniques being compared, over a range of query lengths (256-1024) and dimensionalities (16-64).

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.

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

To calculate *P* we do the following. Random queries are generated (as described above). 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 10 shows the value of *P* over a range of query lengths and dimensionalities for the experiments that were conducted the Mixed Bag dataset.

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 11.

In both Figure 10 and 11 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.

5.3 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 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, …, 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 5). 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

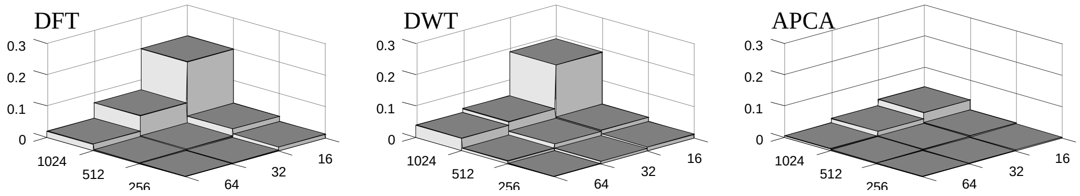


Figure 11: The fraction *P*, of the Electrocardiogram database that must be examined by the three dimensionality reduction techniques being compared over a range of query lengths (256-1024) and dimensionalities (16-64).

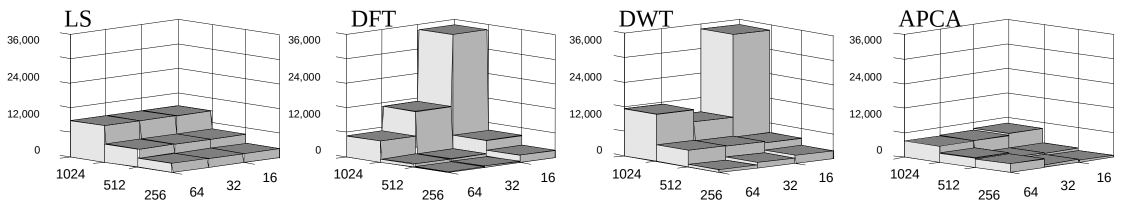


Figure 12: Comparison of LS, DFT, DWT and APCA techniques in terms of I/O cost (number of random disk accesses). For LS, the cost is computed as number_sequential_disk_accesses/10.

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 11 in Table 5). 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 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 12 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 13). The DFT and DWT suffer mainly due to low pruning power (cf. Figure 11). 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 11). 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 13 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 11), causing the second component of the CPU cost (i.e. the time to compute the exact distances D(Q,C) of the original time series of the retrieved APCA points from the query) to become high.

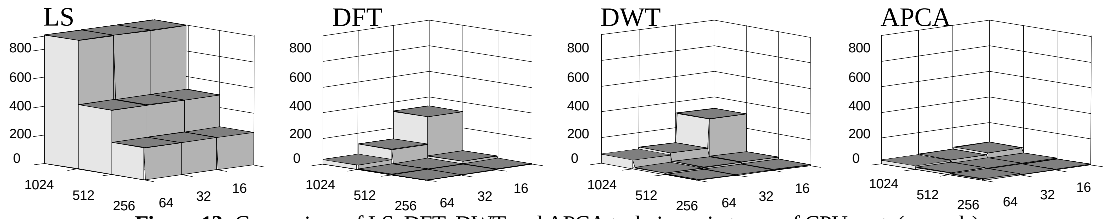


Figure 13: Comparison of LS, DFT, DWT and APCA techniques in terms of CPU cost (seconds).

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.

7. CONCLUSIONS

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 L*p* norms, again using a single index structure.
Future directions for research include further increasing the speedup of our method by exploiting the similarity of adjacent sequences (in a similar spirit to the "trail indexing" technique introduced in [16]). Additionally we intend to explore the possibility of local adaptability for other representations and problems.

ACKNOWLEDGMENTS
This work was supported in part by the National Science Foundation, Information Technology Research under Grant No. IIS-0086124 and in part by the Army Research Laboratory under Cooperative Agreement No. DAAL-01-96-2-0003 and Cooperative Agreement No. DAAD-19-00-1-0188.

8. 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.

[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*.

[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.