A Simple Dimensionality Reduction Technique for Fast Similarity Search in Large Time Series Databases

Eamonn J. Keogh and Michael J. Pazzani

Department of Information and Computer Science
University of California, Irvine, California 92697 USA
{eamonn,pazzani}@ics.uci.edu

Abstract. We address the problem of similarity search in large time series databases. We introduce a novel-dimensionality reduction technique that supports an indexing algorithm that is more than an order of magnitude faster than the previous best known method. In addition to being much faster our approach has numerous other advantages. It is simple to understand and implement, allows more flexible distance measures including weighted Euclidean queries and the index can be built in linear time. We call our approach PCA-indexing (Piecewise Constant Approximation) and experimentally validate it on space telemetry, financial, astronomical, medical and synthetic data.

1 Introduction

Recently there has been much interest in the problem of similarity search in time series databases. This is hardly surprising given that time series account for much of the data stored in business, medical and scientific databases. Similarity search is useful in its own right as a tool for exploring time series databases, and it is also an important subroutine in many KDD applications such as clustering [6], classification [14] and mining of association rules [5].

Time series databases are often extremely large. Given the magnitude of many time series databases, much research has been devoted to speeding up the search process [23,1,15,19,4,11]. The most promising methods are techniques that perform dimensionality reduction on the data, then use spatial access methods to index the data in the transformed space. The technique introduced in [1] and extended in [8, 21,23]. The original work by Agrawal et al. utilizes the Discrete Fourier Transform (DFT) to perform the dimensionality reduction, but other techniques have been suggested, most notably the wavelet transform [4].

In this paper we introduce a novel transform to achieve dimensionality reduction. The method is motivated by the simple observation that for most time series datasets we can approximate the data by segmenting the sequences into equi-length sections and recording the mean value of these sections. These mean values can be indexed efficiently in a lower dimensionality space. We compare our method to DFT, the only

obvious competitor, and demonstrate a one to two order of magnitude speedup on four natural and two synthetic datasets.

In addition to being much faster. We demonstrate that our approach has numerous other advantages over DFT. It is simple to understand and implement, allows more flexible queries including the weighted Euclidean distance measure, and the index can be built in linear time. In addition our method also allows queries which are shorter than length for which the index was built. This very desirable feature is impossible in DFT and wavelet transforms due to translation invariance [20].

The rest of the paper is organized as follows. In Section 2, we state the similarity search problem more formally and survey related work. In Section 3, we introduce our method. Section 4 contains extensive empirical evaluation of our technique. In Section 5, we demonstrate how our technique allows more flexible distance measures. Section 6 offers concluding remarks and directions for future work.

2 Background and Related Work

Given two sequences $X = x_1 \ldots x_n$ and $Y = y_1 \ldots y_m$ with $n = m$, their Euclidean distance is defined as:

$$D(X,Y) \equiv \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \qquad (1)$$

There are essentially two ways the data might be organized [8]:

- *Whole Matching.* Here it assumed that all sequences to be compared are the same length.
- *Subsequence Matching.* Here we have a query sequence X, and a longer sequence Y. The task is to find the subsequence in Y, beginning at Y_i, which best matches X, and report its offset within Y.

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 X be placed at every possible offset within the longer sequence Y. Note it is possible to convert subsequence

Figure 1: 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

matching to whole matching by sliding a "window" of length n across Y, and making copies of the m-n windows. Figure 1 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.
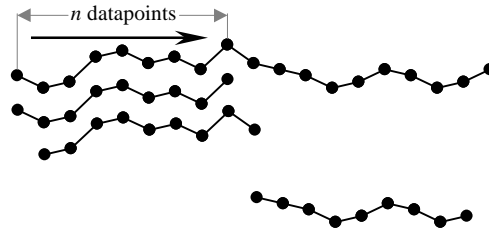
There are several kinds of queries that could be made of a database, such as range queries, all-pairs and nearest neighbor. For simplicity, we will concentrate just on nearest neighbor. The other kinds of queries can always be built using nearest neighbor, and the extensions are trivial.

Given a query X and a database consisting of K time series Y_i ($1 \leq i \leq K$), we want to find the time series Y_i such that $D(Y_i, X)$ is minimized. The brute force approach, sequential scanning, requires comparing every time series Y to X. 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. In contrast, false dismissals, when qualifying objects are missed because they appear distant in index space, are usually unacceptable. In this work we will focus on admissible searching, indexing techniques that guarantee no false dismissals.

2.1 Related Work

A time series X can be considered as a point in n-dimensional space. This immediately suggests that time series could be indexed by Spatial Access Methods (SAMs) such as the R-tree and its many variants [9,3]. However SAMs begin to degrade rapidly at dimensionalities greater than 8-10 [12], and realistic queries typically contain 20 to 1,000 datapoints. In order to utilize SAMs it is necessary to first perform dimensionality reduction. Several dimensionality reduction schemes have been proposed. The first of these *F-index*, was introduced in [1] and extended in [8,23,21]. Because this is the current state-of-the-art for time series indexing we will consider it in some detail.

An important result in [8] is that the authors proved that in order to guarantee no false dismissals, the distance in the index space must satisfy the following condition

$$D_{\text{true}}(A,B) \geq D_{\text{index space}}(A,B) \tag{2}$$

Given this fact, and the ready availability of off-the-shelf SAMs, a generic technique for building an admissible index suggests itself. Given the true distance metric (in this case Euclidean) defined on n datapoints, it is sufficient to do the following:

- 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 SAM.
- Produce a distance measure defined on the N dimensional representation of the data, and prove that it obeys $D_{\text{true}}(A,B) \geq D_{\text{index space}}(A,B)$.

In [8] the dimensionality reduction technique chosen was the Discrete Fourier Transform (DFT). Each of the time series are transformed by the DFT. The Fourier representation is truncated, that is, only the first k coefficients are retained ($1 \leq k < n$), and the rest discarded. The k coefficients can then be mapped into $2k$ space ($2k$ because each coefficient has a real and imaginary component) and indexed by an R* tree.

An important property of the Fourier Transform is Parseval's Theorem, which states that the energy in Euclidean space is conserved in Fourier space [18]. Because of the truncation of positive terms the distance in the transformed space is guaranteed to underestimate the true distance. This property is exploited by mapping the query into the same $2k$ space and examining the nearest neighbors. The theorem guarantees underestimation of distance, so it is possible that some apparently close neighbors are actually poor matches. These false alarms can be detected by examining the corresponding original time series in a post processing stage.

Many other schemes have been proposed for similarity search in time series databases. As they focus on speeding up search by sacrificing the guarantee of no false dismissals [11, 15, 19], and/or allowing more flexible distances measures [2,11, 15, 14, 13, 23, 16, 21] we will not discuss them further.

3 Our Approach

As noted by Faloutsos et al. [8], there are several highly desirable properties for any indexing scheme:

- It should be much faster than sequential scanning.
- The method should require little space overhead.
- The method should be able to handle queries of various lengths.
- The method should be allow insertions and deletions without requiring the index to be rebuilt.
- It should be correct, i.e. there should be no false dismissals.

We will now introduce the *PCA* indexing scheme and demonstrate that it has all the above properties.

3.1 Dimensionality Reduction

We denote a time series query as $X = x_1,\ldots,x_n$, and the set of time series which constitute the database as $Y = \{Y_1,\ldots Y_K\}$. Without loss of generality, we assume each sequence in Y is n units long. Let N be the dimensionality of the transformed space we wish to index ($1 \leq N \leq n$). For convenience, we assume that N is a factor of n. This is not a requirement of our approach, however it does simplify notation.

A time series X of length n is represented in N space by a vector $\overline{X} = \overline{x}_1,\ldots,\overline{x}_N$. The ith element of $\overline{X}$ is calculated by the following equation:

$$\overline{x}_i = \frac{N}{n} \sum_{j=\frac{n}{N}(i-1)+1}^{\frac{n}{N}i} x_j \tag{3}$$

Simply stated, to reduce the data from n dimensions to N dimensions, the data is divided into N equi-sized "frames". The mean value of the data falling within a frame is calculated and a vector of these values becomes the data reduced representation. Figure 2 illustrates this notation. The complicated subscripting in Eq. 3 is just to insure that the original sequence is divided into the correct number and size of frames.
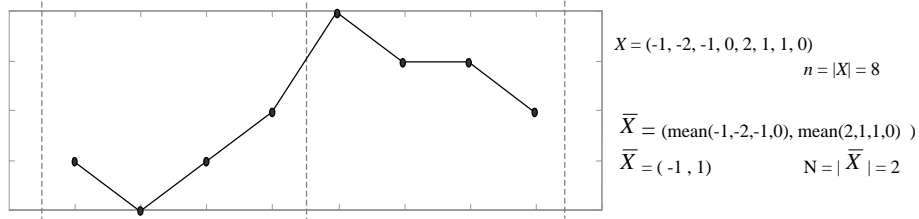
$X = (-1, -2, -1, 0, 2, 1, 1, 0)$

$n = |X| = 8$

$\overline{X} = (\text{mean}(-1,-2,-1,0), \text{mean}(2,1,1,0)\ )$

$\overline{X} = (\ -1\ ,\ 1)$ $N = |\ \overline{X}\ | = 2$

Figure 2: An illustration of the data reduction technique utilized in this paper. A time series consisting of eight (*n*) points is projected into two (*N*) dimensions. The time series is divided into two (*N*) frames and the mean of each frame is calculated. A vector of these means becomes the data reduced representation

Two special cases worth noting are when $N = n$ the transformed representation is identical to the original representation. When $N = 1$ the transformed representation is simply the mean of the original sequence. More generally the transformation produces a piecewise constant approximation of the original sequence.

3.2 Building the index

Table 1 contains an outline of the indexing algorithm. We are deliberately non-committal about the particular indexing structure used. This is to reinforce the fact the dimensionality reduction technique proposed is independent of the indexing structure. All sequences in *Y* are transformed by Eq. 3 and indexed by the spatial access method 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.

```
for i = 1 to K                    // For each sequence to be indexed
    Y_i  ← Y_i − mean(Y_i);       // Optional: remove the mean of Y_i
    Ȳ_i ← transformed(Y_i);       // As in eq. 3
    Insert  Ȳ_i into the indexing structure with a pointer to Y_i
on disk;
end;
```
Table 1: An outline of the 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 optional. We include it because we want to compare our results directly to *F-index*, and *F-index* discards information about offset. For some applications this step is undesirable and can be omitted [13]. Note that the transformation for a single sequence takes $O(n)$ time, thus the entire index can be built in $O(Kn)$. This contrasts well to *F-index* which requires $O(KnLogn)$ time.

3.3 Searching the Index

As mentioned in Section 2, in order to guarantee no false dismissals we must produce a distance measure *DR*, defined in index space, which has the following property: $D(X,Y) \geq DR(\overline{X},\overline{Y})$. The following distance measure has this property:

$$DR(\overline{X},\overline{Y}) \equiv \sqrt{\tfrac{n}{N}} \sqrt{\sum_{i=1}^{N}(\overline{x}_i - \overline{y}_i)^2} \qquad (4)$$

The proof that $D(X,Y) \geq DR(\overline{X},\overline{Y})$ is straightforward but long. We omit it for brevity. Table 2 below contains an outline of the nearest neighbor search algorithm. Once a query *X* is obtained a transformed copy of it $\overline{X}$ is produced. The indexing structure is searched for the nearest neighbor of $\overline{X}$. The original sequence pointed to by this nearest neighbor is retrieved from disk and the true Euclidean distance is calculated. If the second closest neighbor in the index is further than this true Euclidean distance, we can abandon the search, because we are guaranteed its distance in the index is an underestimate of its true distance to the query. Failing that, the algorithm repeatedly retrieves the sequence pointed to by the next most promising item in the index and tests if its true distance is greater than the current best so far. As soon as that happens the search is abandoned.

```
best-so-far   ← infinity;
done          ← FALSE;
i             ← 1;
X̄             ← transformed(X);                  // Using eq.2
while i ≤ G AND NOT(done)
  Find X̄'s iᵗʰ nearest neighbor in the index;  // Using DR (eq.3)
  Retrieve sequence represented by the iᵗʰ nearest neighbor;
  if D(original-sequenceᵢ,X) < best-so-far  //D is defined in eq.1
          best-so-far ← D(original-sequenceᵢ,X);
  end;
  if best-so-far ≤ iᵗʰ+1 nearest neighbor in the index
          done    ← TRUE;
          Display('Sequence ',i,' is the nearest neighbor to Query');
          Display('At a distance of ',best-so-far );
  end;
    i ← i + 1;
  end;
```

Table 2: An outline of the indexing searching algorithm.

3.4 Handling queries of various lengths

In the previous section we showed how to handle queries of length *n*, the length for which the index structure was built. However, it is possible that a user might wish to query the index with a query which is longer or shorter that *n*. For example a user might normally be interested in monthly patterns in the stock market, but occasionally wish to search for weekly patterns. Naturally we wish to avoid building an index for

every possible length of query. In this section we will demonstrate how we can execute queries of different lengths on a single fixed-length index. For convenience we will denote queries longer than n as *XL* and queries shorter than n as *XS*, with $|XL| = n_{XL}$ and $|XS| = n_{XS}$.

3.4.1 Handling short queries

Queries shorter than n can be dealt with in two ways. If the SAM used supports dimension weighting (for example the hybrid tree [3]) one can simply weigh all the dimensions from $ceiling(\frac{N\,n_{XS}}{n})$ to N as zero. Alternatively, the distance calculation in Eq. 4 can have the upper bound of its summation modified to:

$$Nshort = \left\lceil \tfrac{Nn_{XS}}{n} \right\rceil, \qquad \sqrt{\tfrac{n}{N}}\sqrt{\sum_{i=1}^{Nshort}\left(\bar{x}_i - \bar{y}_i\right)^2} \tag{5}$$

The modification does not affect the admissibility of the no false dismissal condition in eq. 2. Because the distance measure is the same as Eq. 4 which we proved, except we are summing over an extra 0 to $\frac{n}{N}-1$ nonnegative terms on the larger side of the inequality. Apart from making either one of these changes, the nearest neighbor search algorithm given in table 2 is used unmodified. This ability of *PCA-index* to handle short queries is an attractive feature not shared by *F-index*, which must resort to sequential scanning in this case [8], as must indexing schemes based on wavelets [20].

3.4.2 Handling longer queries

Handling long queries is a little more difficult than the short query case. Our index only contains information about sequences of length n (projected into N dimensions) yet the query *XL* is of length n_{XL} with $n_{XL} > n$. However we can regard the index as containing information about the prefixes of potential matches to the longer sequence. In particular we note that the distance in index space between the prefix of the query and the prefix of any potential match is always less than or equal to the true Euclidean distance between the query and the corresponding original sequence. Given this fact we can use the nearest neighbor algorithm outlined in table 2 with just two minor modifications. In line four, the query is transformed into the representation used in the index, here we need to replace *X* with *XL*[1:*n*]. The remainder of the sequence, *XL*[*n*+1:*n*$_{XL}$], is ignored during this operation.

In line seven, the original data sequence pointed most promising object in the index is retrieved. For long queries, the original data sequence retrieved and subsequently compared to *XL* must be of length $n_{XL,}$ not *n*.

4 Experimental Results

To demonstrate the generality of our method we tested it on five datasets with widely varying properties.

- **Random Walk:** The sequence is a random walk $x_t = x_{t-1} + z_t$ Where $z_t (t = 1,2,\dots)$ are independent identically distributed (uniformly) random variables in the range (-500,500) [1]. (100,000 datapoints).

- **Astronomical:** A dataset that describes the rate of photon arrivals [17]. (28,904 datapoints).
- **Financial:** The US Daily 5-Year Treasury Constant Maturity Rate, 1972 - 1996 [15]. (8,749 datapoints).
- **Space Shuttle:** This dataset consists of ten time series that describe the orientation of the Space Shuttle during the first eight hours of mission STS-57 [14,15]. (100,000 datapoints).
- **Control Chart:** This dataset consists of the Cyclic pattern subset of the control chart data from the UCI KDD archive (kdd.ics.uci.edu). The data is essentially a sine wave with noise. (6,000 datapoints).

4.1 Building queries

Choosing queries that actually appear in the indexed database will always produce optimistic results. On the other hand, some indexing schemes can do well if the query is greatly different from any sequence in the dataset. 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 do the following. We extract a sequence from the database then flip it either backwards or upside-down depending on the outcome of a fair coin toss. The flipped sequence then becomes our query.

For every combination of dataset, number of dimensions, and query length we performed 1,000 random queries and report the average result.

4.2 Evaluation

In previous work on indexing of time series, indexing schemes have been evaluated by comparing the time taken to execute a query. However this method has the disadvantage of being sensitive to the implementation of the various indexing schemes being compared. For example in [1], the authors carefully state that they use the branch and bound optimization for the Sequential-Scan (a standard indexing strawman). However, in [11] and [23] the authors do not tell us whether they are comparing their indexing schemes to optimized or unoptimized Sequential-Scan. This is a problem because the effect of the optimization can be as much as two orders of magnitude, which is far greater than the speedup reported.

As an example of the potential for implementation bias in this work consider the following. At query time *F-index* 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 $n \neq 2^{integer}$ [18]) which is $O(n\log n)$. If we implemented the simple algorithm it would make our indexing method perform better relative to *F-index*.

To prevent implementation bias we will compare our indexing scheme to *F-index* by reporting the *P*, the fraction of the database that must be examined before we can guarantee that we have found the nearest match to our query.

$$P = \frac{Number\ of\ objects\ retreived}{Number\ of\ objects\ in\ database} \tag{6}$$

Note the value of P depends only on the data and the queries and is completely independent of any implementation choices, including spatial access method, page size, computer language or hardware. It is a fair evaluation metric because it corresponds to the minimum number of disk accesses the indexing scheme must make, and disk time dominates CPU time. A similar idea for evaluating indexing appears in [10].

4.3 Experimental results

Figure 3 shows the results of comparing *PCA-index* to *F-index* on a variety of datasets. Experiments in [1,8] suggest a dimensionality of 6 for *F-index*. For completeness we tested of a range of dimensionalities, however only even numbers of dimensions are used because *F-index* (unlike *PCA-index*) is only defined for even numbers. We also tested over a variety of query lengths. Naturally, one would expect both approaches to do better with more dimensions and shorter queries, and the results generally confirm this.
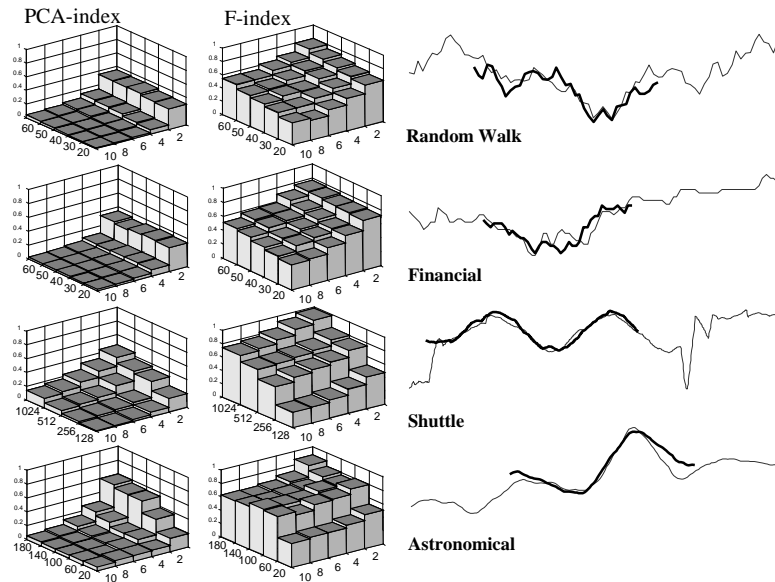

Figure 3: The fraction of the database which must be retrieved from disk using the two indexing schemes compared in this paper, together with sample queries and a section containing the corresponding best match. Each pair of 3d histograms represents a different dataset. Each bar in the 3d histogram represents P, the fraction of the database that must be retrieved from disk for a particular combination of index dimensionality and query length (averaged over 1,000 trials)

For low dimensionalities, say 2-4, *PCA-index* generally outperforms *F-index* by about a factor of two. However as the dimensionality increases the difference between the approaches grows dramatically. At a dimensionality of ten, *PCA-index* outperforms *F-index* by a factor of 81.4 (averaged over the 5 datasets in Fig 3). Competitive index
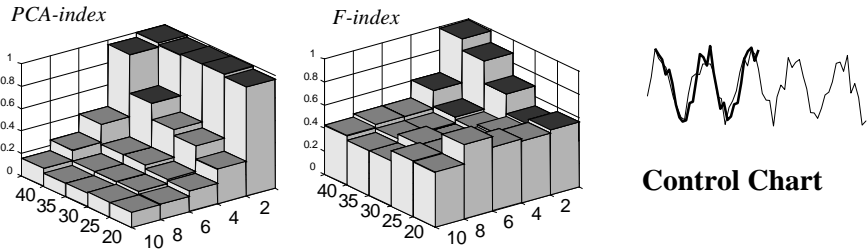
Figure 4: The result of experiments on the Control Dataset, with a sample query and a section containing the corresponding best match. The black topped 3d histogram bars indicate where *F-index* outperforms *PCA-index*

trees can easily handle dimensionalities of ten or greater [3,12].

The Control dataset shown in Fig. 4 contains the only instances where *F-index* outperformed *PCA-index,* so we will consider it in more detail. This dataset is a sine wave with noise. With just two dimensions (corresponding to the real and imaginary parts of a single Fourier coefficient) *F-index* can model a sine wave very well. In contrast, at the same dimensionality *PCA-index* has several entire periods contained within a single frame, thus all frames have approximately the same value and *PCA-index* has little discriminating power. However the situation changes dramatically as the dimensionality increases. Because most of the energy is concentrated in the first coefficient, adding more dimensions does not improve *F-index's* performance. In contrast *PCA-index* extracts great benefit from the extra dimensions. Once the frame size is less than a single period of the sine wave its performance increases dramatically.

This special case clearly illustrates a fact that can also be observed in all the other experiments, *PCA-index* is able to take advantage of extra dimensions much more that *F-index*.

5 Generalizing the Distance Measure

Although the Euclidean distance measure is optimal under several restrictive assumptions [1], there are many situations where a more flexible distance measure is desired [13]. The ability to use these different distance measures can be particularly useful for incorporating domain knowledge into the search process.   One of the advantages of the indexing scheme proposed in this paper is that it can handle many different distance measures, with a variety of useful properties. In this section we will consider one very important example, weighted Euclidean distance. To the author's knowledge, this is the first time an indexing scheme for weighted Euclidean distance has been proposed.

5.1 Weighted Euclidean distance

It is well known in the machine learning community that weighting of features can greatly increase classification accuracy [22]. In [14] we demonstrated for the first time that weighing features in time series queries can increase accuracy in time series classification problems. In addition in [13], we demonstrated that weighting features (to-

gether with a method for combining queries) allows relevance feedback in time series databases. Both [14,13] illustrate the utility of weighted Euclidean metrics, however no indexing scheme was suggested. We will now show that *PCA-index* can be easily modified to support of weighted Euclidean distance.

In Section 3.2, we denoted a time series query as a vector $X = x_1,\ldots,x_n$. More generally we can denote a time series query as a tuple of equi-length vectors $\{X = x_1,\ldots,x_n , W = w_1,\ldots,w_n\}$ where *X* contains information about the shape of the query and *W* contains the relative importance of the different parts of the shape to the query. Using this definition the Euclidean distance metric in Eq. 1 can be extended to the weighted Euclidean distance metric *DW*:

$$DW([X,W],Y) = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2} \qquad (7)$$

We can perform weighted Euclidean queries on our index by making two simple modifications to the algorithm outlined in Table 2. We replace the two distance measures *D* and *DR* with *DW* and *DRW* respectively. *DW* is defined in Eq. 7 and *DRW* is defined as:

$$\overline{w}_i = \min(w_{\frac{n}{N}(i-1)+1},\ldots,w_{\frac{n}{N}i}) , \qquad DRW([\overline{X},\overline{W}],\overline{Y}) = \sqrt{\frac{n}{N}}\sqrt{\sum_{i=1}^{N} \overline{w}_i (\overline{x}_i - \overline{y}_i)^2}$$

$$(8)$$

Note that it is not possible to modify *F-index* in a similar manner, because each coefficient represents amplitude and phase of a signal that is added along the entire length of the query

6 Conclusions

We have introduced a dimensionality reduction technique that allows fast indexing of time series. We performed extensive empirical evaluation and found our method outperforms the current best known approach by one to two orders of magnitude. We have also demonstrated that our technique can support weighted Euclidean queries.

In future work we intend to further increase the speed up of our method by exploiting the similarity of adjacent sequences (in a similar spirit to the "trail indexing" technique introduced in [8]). Additionally, we hope to show the speedup obtained by *PCA-index* will support a variety of time series datamining algorithms that scale poorly to large datasets, for example the rule induction algorithm proposed in [5].

References

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

2. 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. In *VLDB*.

3. Chakrabarti, K & Mehrotra, S. (1999). The Hybrid Tree: An Index Structure for High Dimensional Feature Spaces. *Proc of the IEEE International Conference on Data Engineering*.

4. Chan, K. & Fu, W. (1999). Efficient Time Series Matching by Wavelets. *Proceedings of the 15th International Conference on Data Engineering*.

5. Das, G., Lin, K. Mannila, H., Renganathan, G., & Smyth, P. (1998). Rule Discovery from Time Series. In *Proc of the 3rd Inter Conference of Knowledge Discovery and Data Mining.*

6. Debregeas, A. & Hebrail, G. (1998). Interactive interpretation of Kohonen maps applied to curves. *Proc of the 4th International Conference of Knowledge Discovery and Data Mining.*

7. Faloutsos, C. & Lin, K. (1995). Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proc. ACM SIGMOD Conf.*, pp 163-174.

8. Faloutsos, C., Ranganathan, M., & Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. In *Proc. ACM SIGMOD Conf.*, Minneapolis.

9. Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Conf.*, pp 47-57.

10. Hellerstein, J. M., Papadimitriou, C. H., & Koutsoupias, E. (1997). Towards an Analysis of Indexing Schemes. 16th ACM SIGACT- Symposium on Principles of Database Systems.

11. 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.

12. Kanth, K.V., Agrawal, D., & Singh, A. (1998). Dimensionality Reduction for Similarity Searching in Dynamic Databases. In *Proc. ACM SIGMOD Conf.*, pp. 166-176.

13. Keogh, E. & Pazzani, M. (1999). Relevance Feedback Retrieval of Time Series Data. *Proc. of the 22th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval.*

14. 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.

15. Keogh, E., & Smyth, P. (1997). A probabilistic approach to fast pattern matching in time series databases. *Proc. of the 3rd Inter Conference of Knowledge Discovery and Data Mining*

16. 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*.

17. Scargle, J. (1998). Studies in astronomical time series analysis: v. Bayesian blocks, a new method to analyze structure in photon counting data. Astrophysical Journal, Vol. 504.

18. Shatkay, H. (1995). The Fourier Transform - a Primer, Technical Report CS-95-37, Department of Computer Science, Brown University.

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

20. Struzik, Z. & Siebes, A. (1999). The Haar Wavelet Transform in the time series similarity paradigm. 3rd European Conference on Principles and Practice of KDD.

21. Refiei, D., & Mendelzon, A. (1997). Similarity-Based queries for time series data. In *Proc. ACM SIGMOD Conf.*, pp. 13-25.

22. Wettschereck, D., Aha, D. & Mohri, T. (1997). A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. AI Review, Vol 11, Issues 1-5.

23. 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.