A Multilevel Distance-based Index Structure for Multivariate Time Series

Kiyoung Yang
Computer Science Department
University of Southern California
Los Angeles, CA 90089–0781
kiyoungy@usc.edu

Cyrus Shahabi
Computer Science Department
University of Southern California
Los Angeles, CA 90089–0781
shahabi@usc.edu

Abstract

Multivariate time series (MTS) datasets are common in various multimedia, medical and financial applications. In previous work, we introduced a similarity measure for MTS datasets, termed Eros (Extended Frobenius norm), which is based on the Frobenius Norm and Principal Component Analysis (PCA). Eros computes the similarity between two MTS items by measuring how close the corresponding principal components (PCs) are using the eigenvalues as weights. Since the weights are based on the data items in the database, they change whenever data are inserted into or removed from the database. In this paper, we propose a distance-based index structure, Muse (Multilevel distance-based index structure for Eros), for efficient retrieval of MTS items using Eros. Muse constructs each level as a distance-based index structure without using the weights, up to z levels. At the query time, Muse combines the z levels with the weights, which enables the weights to change without the need to rebuild the index structure. In order to show the efficiency of Muse, we performed several experiments on a set of synthetically generated clustered datasets. The results show the superiority of Muse as compared to Sequential Scan and M-tree in performance.

1 INTRODUCTION

A time series is a series of observations, $x_i(t); [i = 1, \cdots, n; t = 1, \cdots, m]$, made sequentially over time where i indexes the measurements made at each time point t [13]. It is called a univariate time series when n is equal to 1, and a multivariate time series (MTS) when n is equal to, or greater than 2.

MTS datasets are common in various fields, such as in multimedia, medicine and finance. For example, in multimedia, Cybergloves used in the Human and Computer Interface applications have about 20 sensors, each of which generates 50∼100 values in a second [11]. In medicine, Electro Encephalogram (EEG) from 64 electrodes placed on the scalp are measured to examine the correlation of genetic predisposition to alcoholism [16]. Functional Magnetic Resonance Imaging (fMRI) from 696 voxels out of 4391 has been used to detect similarities in activation between voxels in [5].

In our previous work [14], we proposed a similarity measure *Eros* (*E*xtended F*ro*benius *s* norm) for efficient similarity searches in MTS databases. *Eros* is based on the Frobenius norm that is used to compute the matrix norm [9], and Principal Component Analysis (PCA) [6]. *Eros* computes the similarity between two MTS items by measuring how close their corresponding principal components (PCs), i.e., the eigenvectors from their covariance matrices, are using the eigenvalues as weights. The weights are aggregated from the eigenvalues of all the MTS items in the database. Hence, the weights change whenever data are inserted into or removed from the database. Empirically, we showed that *Eros* outperforms Euclidean Distance (ED), Weighted Sum SVD (WSSVD) [12], Dynamic Time Warping (DTW) [10] and PCA similarity factor (S_{PCA}) [8] in terms of precision/recall.

In this paper, we propose an index structure termed *Muse* (*Mu*ltilevel distance-based index *s*tructure for *Eros*) for efficient retrieval of MTS items using *Eros*. Distance-based index structures, such as iDistance [15] and M-tree [3], have been shown to outperform feature-based index structures, such as R-tree and its variants, for high dimensional datasets. Hence, we extend a distance-based index structure so that the similarity search using *Eros* whose lower bound is obtained using the weighted Euclidean distance, can be performed efficiently. To explain *Muse*, let us denote the principal component (PC) of an MTS item whose corresponding eigenvalue is the largest as *the first PC* of an MTS item, and that whose corresponding eigenvalue is the second largest as *the second PC* of an MTS item. We subsequently call the first PCs of all the MTS items as *the first PC group*, and the second PCs of all the MTS items as *the second PC group*. For *Muse*, we thusly construct a number

of *levels*, each of which is a distance-based index structure corresponding to one PC group *without using the weights*. In order to build a z-level *Muse*, we utilize the first z PC groups of all the data items. At the query time when a similarity search is performed, *Muse* combines the z levels of the distance-based index structures with the weights to yield the lower bounds of the similarities between the query MTS item and the MTS items in the database. Since the weights are employed not when constructing the index structure, but when performing a similarity search, *Muse* also allows the weights to change without the need to re-construct the index structure, even when data are added into or removed from the database. In order to show the efficiency of *Muse*, we compare the performance of *Muse* to those of M-tree [3] and sequential scan in terms of efficiency and scalability.

The remainder of this paper is organized as follows. Section 2 discusses the background. Our proposed method is described in Section 3. This is followed by the experiments and results in Section 4. Conclusions and future work are presented in Section 5.

2 BACKGROUND

Muse is the extension of a distance-based index structure designed for our previously introduced similarity measure *Eros* [14]. In this section, we briefly describe *Eros* and the distance-based index structures.

2.1 Eros

In [14], we proposed *Eros* as a similarity measure for multivariate time series. Intuitively, *Eros* computes the similarity between two matrices using the principal components (PCs), i.e., the eigenvectors from the covariance matrices, and the eigenvalues as weights. The weights are aggregated from the eigenvalues of all the MTS items in the database. Hence, the weights change whenever data are inserted into or removed from the database.

Definition 1 *Eros (Extended Frobenius norm). Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be two MTS items of size* $m_A \times n$ *and* $m_B \times n$*, respectively*[1]*. Let* $\mathbf{V_A}$ *and* $\mathbf{V_B}$ *be two right eigenvector matrices obtained by applying SVD to the covariance matrices,* $\mathbf{M_A}$ *and* $\mathbf{M_B}$*, respectively. Let* $\mathbf{V_A} = [a_1, \cdots, a_n]$ *and* $\mathbf{V_B} = [b_1, \cdots, b_n]$*, where* a_i *and* b_i *are column orthonormal vectors of size* n*. The Eros similarity of* $\mathbf{A}$ *and* $\mathbf{B}$ *is then defined as*

$$Eros(\mathbf{A},\mathbf{B},w) = \sum_{i=1}^{n} w_i |< a_i, b_i >| = \sum_{i=1}^{n} w_i |\cos\theta_i| \qquad (1)$$

where $< a_i, b_i >$ *is the inner product of* a_i *and* b_i*,* w *is a weight vector which is based on the eigenvalues of the MTS dataset (see Section 3.3),* $\sum_{i=1}^{n} w_i = 1$ *and* $\cos\theta_i$ *is the angle between* a_i *and*[2] b_i*. The range of Eros is between 0 and 1, with 1 being the most similar.*

Eros does not satisfy the triangle inequality [4]. Hence, we obtained the lower and upper bounds of *Eros* as follows, so that efficient retrievals of MTS items can be performed using an index structure. We first defined D_{Eros} which reversely preserves the similarity relation of *Eros*, and obtained the lower and upper bounds of D_{Eros}, i.e., D_{min} and D_{max}, respectively. For notations used in the remainder of this paper, please refer to Table 1.

$$D_{Eros}(\mathbf{A},\mathbf{B},w) = \sqrt{2 - 2\sum_{i=1}^{n} w_i |< v_i^A, v_i^B >|} = \sqrt{2 - 2\sum_{i=1}^{n} w_i |\sum_{j=1}^{n} v_{ij}^A \times v_{ij}^B|}$$

$$D_{max}(\mathbf{A},\mathbf{B},w) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} w_i (v_{ij}^A - v_{ij}^B)^2} = \sqrt{2 - 2\sum_{i=1}^{n} w_i \sum_{j=1}^{n} v_{ij}^A v_{ij}^B}$$

$$D_{min}(\mathbf{A},\mathbf{B},w) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} w_i (|v_{ij}^A| - |v_{ij}^B|)^2} = \sqrt{2 - 2\sum_{i=1}^{n} w_i \sum_{j=1}^{n} |v_{ij}^A v_{ij}^B|}$$

Note that D_{max} and D_{min} are distance metrics that satisfy the triangle inequality. *Muse* utilizes the lower bound of *Eros*, i.e., D_{min} to filter out those MTS items that are not to be in the result set.

2.2 Distance-Based Index Structures

Intuitively, the distance-based index techniques, such as iDistance [15] and M-tree [3], work as in Algorithms 1 and 2. The distance-based index structures have been shown to dominate feature-based index structures, such as R-tree and its variants, for high dimensional datasets. iDistance and M-tree utilize the Euclidean distance. Though the weighted Euclidean distance can be employed for iDistance and M-tree, the index structures should then be reconstructed whenever there is a change in the weight, i.e., in the database. This reconstruction would be very costly when there are a lot of items in the dataset, and the items are continuously generated.

Algorithm 1 Distance-based Index : Construction

1: Partition the data or the data space.
2: Choose one reference point for each partition.
3: Compute the distance between all the data within the partition and its reference point.
4: Compute the max and min radii of each partition, i.e., the distances between the reference point and the farthest data item and the closest data item in the partition, respectively.

[1] MTS items have the same number of columns (e.g., sensors), but may have different number of rows (e.g., time samples).

[2] For simplicity, it is assumed that the covariance matrices are of full rank. In general, the summations in Equation (1) should be from 1 to min(r_A, r_B), where r_A is the rank of $\mathbf{M_A}$ and r_B the rank of $\mathbf{M_B}$.

Algorithm 2 Distance-based Index : kNN Search

1: Given a query item Q for which the kNN search is performed, compute the distances between the query item and all the reference points.
2: Sort the partitions based on the distances to Q in non-decreasing order.
3: Search for kNNs of Q using the triangle inequality from within the closest partition.

Table 1. Notations used in this paper

Symbol	Definition
$\mathbf{A}$	an $m \times n$ matrix representing an MTS item
$\mathbf{A}^T$	the transpose of $\mathbf{A}$
$\mathbf{M_A}$	the covariance matrix of size $n \times n$ for $\mathbf{A}$
$\mathbf{V_A}$	the right eigenvector matrix of size $n \times n$ for $\mathbf{M_A}$, i.e., $\mathbf{V_A} = [\ v_1^A,\ v_2^A,\ \cdots,\ v_n^A\ ]$
$\mathbf{\Sigma_A}$	an $n \times n$ diagonal matrix that has all the eigenvalues for $\mathbf{M_A}$ obtained by SVD
v_i^A	a column orthonormal eigenvector of size n for $\mathbf{V_A}$
v_{ij}^A	jth value of v_i^A, i.e., a value at the ith column and the jth row of $\mathbf{V_A}$
v_{*j}^A	all the values at the jth row of $\mathbf{V_A}$
w	a weight vector of size n, such that $\sum_{i=1}^{r} w_i = 1, \forall i\ w_i \geq 0$
n	number of variables, i.e., number of columns of a matrix
m_A	number of samples, i.e., number of rows of a matrix $\mathbf{A}$

3 *Muse*: *Mu*ltilevel distance-based index *s*tructure for *Eros*

In this section, we describe *Muse* (*Mu*ltilevel distance-based index *s*tructure for *Eros*). *Muse* constructs one *level* of distance-based index structure using each PC group, up to z levels. Note that the weights are not utilized when *Muse* is constructed. When performing a similarity search for a given query item Q, *Muse* combines the z levels constructed *a priori* with the weights to yield the lower bounds of the similarities between Q and all the MTS items in the database. That is, the weights are applied not when constructing a z-level *Muse*, but when performing a similarity search, which allows the weights to get updated without the need to reconstruct the index structure. Since *Eros* does not satisfy the triangle inequality, when performing a similarity search, *Muse* utilizes the lower bound of *Eros*, i.e., D_{min}, to filter out those MTS items that are not to be in the result set. For the MTS items that are not filtered out, *Muse* employs D_{Eros} to obtain the final result set.

We start by first describing how *Muse* is constructed, and how the similarity search for a given MTS item Q using *Muse* is performed in Sections 3.1 and 3.2, respectively. Updating *Muse* is presented in Section 3.3, followed by the discussions on the selection of reference points for *Muse* in Section 3.4. The discussion on the number of levels is presented in Section 3.5. Table 1 lists the notations used in the remainder of this paper, if not specified otherwise.

3.1 Construction

Recall that for *Eros*, each principal component (PC) is assigned a weight which may change whenever items are inserted into or removed from the database. The distance-based index structures, such as iDistance [15] and M-tree [3], utilize the Euclidean distance when building the index structures. Though the weighted Euclidean distance metric, e.g., D_{min}, may be employed for iDistance and M-tree, the index structure should be rebuilt when the weights change. This is due to the fact that the weight is applied when constructing the index structure. If an index structure is constructed without using the weights, and the weights can later be incorporated into the index structure constructed *a priori*, then the weights can change without having to rebuild the index structure. That is, the index construction and the weight application should be *independent*. This is exactly how *Muse* works for *Eros*, which will be subsequently described.

Let us denote the PC of an MTS item whose corresponding eigenvalue is the largest as *the first PC* of an MTS item, and that whose corresponding eigenvalue is the second largest as *the second PC* of an MTS item. Let us subsequently call the first PCs of all the MTS items as *the first PC group*, and the second PCs of all the MTS items as *the second PC group*. For *Muse*, we thusly construct one distance-based index structure, which is called a *level*, for each PC group of the dataset without using the weights. In order to build a z-level *Muse*, we utilize the first z PC groups of all the data items. Let us first define two more distance metrics to be used for building *Muse*.

Definition 2 *A distance metric $D_{|\cdot|,k}$ between two MTS items, $\mathbf{A}$ and $\mathbf{B}$, using the kth PC is defined as follows:*

$$D_{|\cdot|,k}(\mathbf{A},\mathbf{B}) = \sqrt{\sum_{j=1}^{n} (|v_{kj}^A| - |v_{kj}^B|)^2}$$

and a distance metric $D_{|\cdot|}$ between two n dimensional vectors, a and b, is defined as follows:

$$D_{|\cdot|}(a,b) = \sqrt{\sum_{j=1}^{n} (|a_j| - |b_j|)^2}$$

Note that *Eros* assigns a weight to *each* PC. Hence, *Muse* constructs one level of a distance-based index structure using each PC group. For each PC group, we first partition the PCs and assign one reference point for each partition. We then compute the distances between each of the reference points and all the PCs which belong to that partition using $D_{|\cdot|}$. The weights are computed separately by Algorithms 6 or 7 (Refer to [14] for details).

Algorithm 3 describes how to construct a z-level *Muse*, where R_{jl} represents the lth reference point at the jth level; r_{jl} is the farthest distance from the lth reference point at the jth level; n is the number of the PCs of an MTS item and n_R is the number of reference points. $dist[i,j]$ stores

the distance between the jth PC of the ith MTS item and its reference point. $partitionID[i,j]$ stores the ID of the partition to which the jth PC of the ith MTS item belongs in the jth level. Intuitively, we store the ID of the partition to which each PC of an MTS item belongs and compute the distance between each PC and the reference point of its partition.

Algorithm 3 *Muse* : Construction

Require: the number of all the MTS items in the dataset N, the number of levels to be built for *Muse*, $z \ll n$, reference points R_{jl} where $1 \leq l \leq n_R$ and $1 \leq j \leq z$;

```
for j=1 to z do
  for i=1 to N do
    E ← the jth PC of the ith MTS item in the database;
    l ← the ID of the partition closest to E;
    dist[i,j] ← D_{|·|}(R_{jl}, E);
    partitionID[i,j] ← l;
    if r_{jl} ≤ dist[i,j] then
      r_{jl} ← dist[i,j];
    end if
  end for
end for
Compute the weight vector w using Algorithm 6 or 7;
```

Figure 1(a) represents the conceptual diagram of *Muse*, where u_j is the number of reference points at the jth level. In this figure, the lth partition at the jth level is represented by a reference point R_{jl} and its radius r_{jl}. At the jth level ($1 \leq j \leq z$), the jth PC group is divided into u_j partitions. The lth partition ($1 \leq l \leq u_j$) at the jth level contains all the IDs of the MTS items whose jth PCs belong to this partition as well as the distances between R_{jl} and all the jth PCs of the MTS items in this partition. When a similarity search is issued for a given MTS item Q, the pre-computed distances in the z-level *Muse* are then combined together with the weights to find the lower bounds, i.e., D_{min} of the similarities between Q and all the MTS items in the database. Note that in Algorithm 3, it is assumed for simplicity that the number of reference points, i.e., n_R, is the same for all the levels.

3.2 kNN search using *Muse*

Given a query object Q, a set of objects $\mathcal{D}$, and an integer $k \geq 1$, k Nearest Neighbor (kNN) search is to find k objects in $\mathcal{D}$ which have the shortest distance from Q [3]. Range queries, which retrieve all the items within a fixed distance from a query object, would be simpler than kNN searches [15]. Hence, we concentrate on kNN searches in this paper.

Muse stores the ID of the partition to which each PC of an MTS item belongs and the distance between each PC and its partition's reference point. When a kNN search is issued given a query item $\mathbf{Q}$, we need to combine all the pre-computed distances between the PCs of an MTS item $\mathbf{A}$ and the reference points as well as the weights w to obtain the lower bound of $D_{min}(\mathbf{A},\mathbf{Q},w)$. There are two ways to compute the lower bound of $D_{min}(\mathbf{A},\mathbf{Q},w)$ using a z-level *Muse*. We first describe the *naive* way followed by the *Muse* way which generates a tighter lower bound.

To discuss the computation of the naive lower bound, assume a 2-level *Muse* with two partitions for each level as in Figure 1(b). That is, we pre-compute all the distances between the first 2 PCs of all the MTS items and their reference points. Using the first 2 PCs, we then have the following inequalities:

$$\begin{aligned} D_{min}(\mathbf{A},\mathbf{Q},w) &= \sqrt{\textstyle\sum_{i=1}^{n}\sum_{j=1}^{n} w_i(|v_{ij}^A|-|v_{ij}^Q|)^2} \\ &\geq D_{min}^{1-2}(\mathbf{A},\mathbf{Q},w) \\ &\geq D_{min}^{1-2}(\mathbf{C},\mathbf{A},w) - D_{min}^{1-2}(\mathbf{Q},\mathbf{C},w) \\ &\quad (triangle\ inequality) \end{aligned} \tag{2}$$

where $\mathbf{C} = [c_1, c_2]$, c_i is the reference point of v_i^A, i.e., the ith PC of $\mathbf{A}$, and D_{min}^{1-2} is to use the first 2 PCs to compute D_{min}. $D_{min}^{1-2}(\mathbf{C},\mathbf{A},w)$ and $D_{min}^{1-2}(\mathbf{Q},\mathbf{C},w)$ can be expanded and represented as follows:

$$\begin{aligned} D_{min}^{1-2}(\mathbf{C},\mathbf{A},w) &= \sqrt{w_1\textstyle\sum_{j=1}^{n}(|c_{1j}|-|v_{1j}^A|)^2 + w_2\sum_{j=1}^{n}(|c_{2j}|-|v_{2j}^A|)^2} \\ &= \sqrt{w_1(D_{|\cdot|,1}(\mathbf{C},\mathbf{A}))^2 + w_2(D_{|\cdot|,2}(\mathbf{C},\mathbf{A}))^2} \end{aligned}$$

and

$$\begin{aligned} D_{min}^{1-2}(\mathbf{Q},\mathbf{C},w) &= \sqrt{w_1\textstyle\sum_{j=1}^{n}(|v_{1j}^Q|-|c_{1j}|)^2 + w_2\sum_{j=1}^{n}(|v_{2j}^Q|-|c_{2j}|)^2} \\ &= \sqrt{w_1(D_{|\cdot|,1}(\mathbf{Q},\mathbf{C}))^2 + w_2(D_{|\cdot|,2}(\mathbf{Q},\mathbf{C}))^2} \end{aligned}$$

$D_{|\cdot|,i}(\mathbf{C},\mathbf{A})$ is computed *a priori* at the time of a z-level *Muse* construction and $D_{|\cdot|,i}(\mathbf{Q},\mathbf{C})$ is computed once online when $\mathbf{Q}$ is provided. Hence, we can immediately obtain the lower bound of $D_{min}(\mathbf{A},\mathbf{Q},w)$ using *Muse* when performing a kNN search. Note that the pre-computed distance $D_{|\cdot|,i}(\mathbf{C},\mathbf{A})$ is not affected even when the weight w_i changes. Consequently, the weight vector w can change without the need to rebuild the index. Let us formally define the lower bound described above, termed LB_{Naive}.

Definition 3 *The lower bound LB_{Naive} between two MTS items, $\mathbf{A}$ and $\mathbf{Q}$, using a z-level Muse is defined as follows:*

$$\begin{aligned} LB_{Naive,z}(\mathbf{A},\mathbf{Q},w) &= \sqrt{\textstyle\sum_{i=1}^{z} w_i(D_{|\cdot|,i}(\mathbf{C},\mathbf{A}))^2} \\ &\quad - \sqrt{\textstyle\sum_{i=1}^{z} w_i(D_{|\cdot|,i}(\mathbf{Q},\mathbf{C}))^2} \end{aligned} \tag{3}$$

where $\mathbf{C} = [c_1, \cdots, c_z]$ and c_i is the reference points of v_i^A.

Yet, using *Muse*, we can obtain a tighter lower bound of $D_{min}(\mathbf{A},\mathbf{Q},w)$ than LB_{Naive}. Consider $D_{min}^{1-2}(\mathbf{A},\mathbf{Q},w)$ in Equation (2), which can be expanded as follows:

$$D_{min}^{1-2}(\mathbf{A},\mathbf{Q},w) = \sqrt{\begin{aligned} &w_1\textstyle\sum_{j=1}^{n}(|v_{1j}^A|-|v_{1j}^Q|)^2 \\ &+w_2\textstyle\sum_{j=1}^{n}(|v_{2j}^A|-|v_{2j}^Q|)^2 \end{aligned}} \tag{4}$$

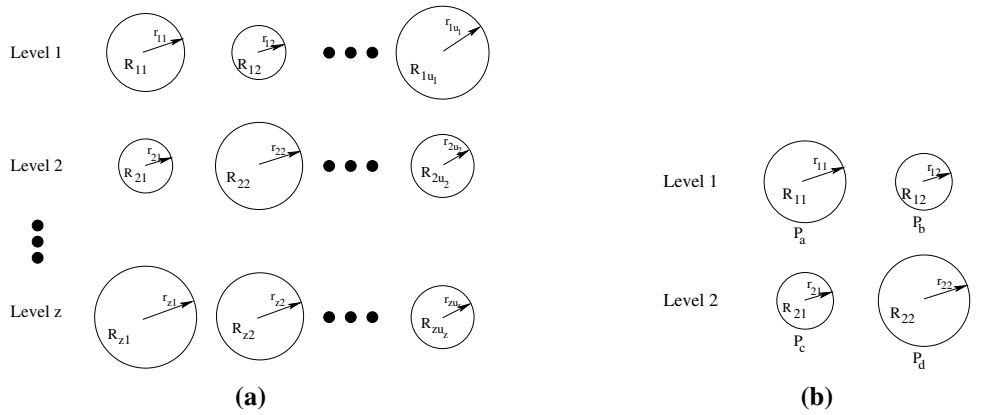


Figure 1. (a) Conceptual Diagram of *Muse* (b) 2-level *Muse*

Then at the first level of *Muse*, i.e., using the first PCs, we have the following inequality:

$$\sqrt{w_1 \sum_{j=1}^{n} (|v_{1j}^A| - |v_{1j}^Q|)^2} \;\geq\; \sqrt{w_1 \sum_{j=1}^{n} (|c_{1j}| - |v_{1j}^A|)^2} - \sqrt{w_1 \sum_{j=1}^{n} (|v_{1j}^Q| - |c_{1j}|)^2} \tag{5}$$

and at the second level, using the second PCs, we have

$$\sqrt{w_2 \sum_{j=1}^{n} (|v_{2j}^A| - |v_{2j}^Q|)^2} \;\geq\; \sqrt{w_2 \sum_{j=1}^{n} (|c_{2j}| - |v_{2j}^A|)^2} - \sqrt{w_2 \sum_{j=1}^{n} (|v_{2j}^Q| - |c_{2j}|)^2} \tag{6}$$

By squaring and summing up Equations (5) and (6) and then taking its square root, we obtain the lower bound of Equation (4). The lower bound described above is defined as follows:

Definition 4 *The lower bound LB_{Muse} between two MTS items, $\mathbf{A}$ and $\mathbf{Q}$, using a z-level Muse is defined as follows:*

$$LB_{Muse,z}(\mathbf{A},\mathbf{Q},w) = \sqrt{\sum_{i=1}^{z} \left\{ \begin{array}{l} \sqrt{w_i (D_{|\cdot|,i}(\mathbf{C},\mathbf{A}))^2} \\ -\sqrt{w_i (D_{|\cdot|,i}(\mathbf{Q},\mathbf{C}))^2} \end{array} \right\}^2}$$

where $\mathbf{C} = [c_1, \cdots, c_z]$ and c_i is the reference points of v_i^A, i.e., the ith PC of $\mathbf{A}$.

That is, LB_{Muse} computes the lower bounds at each level, and sums up the lower bounds to yield the lower bound of D_{min} between the MTS items and the query item. $LB_{Muse,z}$ yields tighter lower bound of D_{min} than $LB_{Naive,z}$ meaning that the former generates less false hits. Hence, we use $LB_{Muse,z}$ when performing kNN searches with *Muse*.

Lemma 1 *For the lower bounds LB_{Naive} and LB_{Muse}, the following inequality holds : $LB_{Muse,z}(\mathbf{A},\mathbf{Q},w) \geq LB_{Naive,z}(\mathbf{A},\mathbf{Q},w)$.*

Proof 1 *Substitute $\sqrt{w_i (D_{|\cdot|,i}(\mathbf{C},\mathbf{Q}))^2}$ with A_i and $\sqrt{w_i (D_{|\cdot|,i}(\mathbf{C},\mathbf{A}))^2}$ with B_i. Then,*

$$LB_{Muse,z}(\mathbf{A},\mathbf{Q},w) = \sqrt{\sum_{i=1}^{z} \{A_i - B_i\}^2}$$

and

$$LB_{Naive,z}(\mathbf{A},\mathbf{Q},w) = \sqrt{\sum_{i=1}^{z} A_i^2} - \sqrt{\sum_{i=1}^{z} B_i^2}$$

Square $LB_{Muse,z}(\mathbf{A},\mathbf{Q},w)$ and $LB_{Naive,z}(\mathbf{A},\mathbf{Q},w)$ and we get

$$LB_{Muse,z}(\mathbf{A},\mathbf{Q},w)^2 = \sum_{i=1}^{z} A_i^2 + \sum_{i=1}^{z} B_i^2 - 2\sum_{i=1}^{z} A_i B_i \tag{7}$$

and

$$LB_{Naive,z}(\mathbf{A},\mathbf{Q},w)^2 = \sum_{i=1}^{z} A_i^2 + \sum_{i=1}^{z} B_i^2 - 2\sqrt{\sum_{i=1}^{z} A_i^2}\sqrt{\sum_{i=1}^{z} B_i^2} \tag{8}$$

Hence, the inequality between $LB_{Muse,z}$ and $LB_{Naive,z}$ depends on the last term of Equations (7) and (8). Recall that Hölder's Inequality [9] states that

$$\sum_{k=1}^{n} |a_k b_k| \leq (\sum_{k=1}^{n} |a_k|^p)^{1/p} (\sum_{k=1}^{n} |b_k|^q)^{1/q}$$

By Hölder's Inequality where $p = q = 2$, we find the following inequality

$$\sum_{i=1}^{z} |A_i B_i| \leq (\sum_{i=1}^{z} |A_i|^2)^{1/2} (\sum_{i=1}^{z} |B_i|^2)^{1/2}$$

Hence, by Equations (7), (8) and Hölder's Inequality[3]*,*

$$LB_{Muse,z}(\mathbf{A},\mathbf{Q},w) \geq LB_{Naive,z}(\mathbf{A},\mathbf{Q},w)$$

A kNN search using *Muse* is performed as in Algorithm 4. Intuitively, we first sort the MTS items so that those in the partitions closer to the given query item $\mathbf{Q}$ will be examined first (Line 9). This can be done as follows: See Figure 1(b), which is a 2-level *Muse* with two reference points

[3]Note that $A_i \geq 0$ and $B_i \geq 0$. Hence, $|A_i| = A_i$, $|B_i| = B_i$ and $|A_i B_i| = A_i B_i$.

for each level. Given a query item $\mathbf{Q}$, assume that $v_1^{\mathbf{Q}}$ is closer to partition P_a than P_b, and $v_2^{\mathbf{Q}}$ is closer to partition P_d than P_c. First, all the MTS items whose first PCs belong to partition P_a are identified. Among these items, the MTS items whose second PCs belong to partition P_d are examined first, and then the items whose second PCs belong to partition P_c. Similarly, the MTS items whose first PCs belong to partition P_b are examined. This process is repeated for all the z levels of *Muse*. The rest of the Algorithm 4 is described as follows: In Lines 1∼3, $knnDistance$ array is initialized. The distances between the first z PCs of Q and the reference points are computed (Lines 4∼8), and kNNs of Q are searched from the closest partition to Q (Line 9). $LB_{Muse,z}$ is used to filter out the MTS items that are not to be in the candidate set (Line 12) and the refinement phase using D_{Eros} is performed in Lines 14∼18. The data structures $knnDistance$ and $knnID$ are updated so that they are sorted in non-decreasing order of D_{Eros} in $knnDistance$.

Algorithm 4 *Muse* : kNN search

Require: the number of levels built for *Muse*, $z \ll n$, reference points R_{jl} where $1 \leq l \leq n_R$ and $1 \leq j \leq z$, a query MTS item Q and k;
1: **for** i=1 to k **do**
2: $knnDistance[i] \leftarrow \infty$;
3: **end for**
4: **for** t=1 to z **do**
5: **for** i=1 to n_R **do**
6: $distRQ[t,i] \leftarrow D_{|\cdot|}(R_{t1}, v_t^Q)$;
7: **end for**
8: **end for**
9: Sort MTS items using $distRQ$ so that the MTS items which belong to the partition closest to Q be checked first;
10: **for** i=1 to N **do**
11: $\mathbf{A} \leftarrow$ the ith sorted MTS item;
12: **if** $LB_{Muse,z}(\mathbf{A}, \mathbf{Q}) \geq knnDistance[k]$ **then**
13: continue;
14: **end if**
15: **if** $D_{Eros}(\mathbf{A}, \mathbf{Q}) \leq knnDistance[k]$ **then**
16: $knnDistance[k] \leftarrow D_{Eros}(\mathbf{A}, \mathbf{Q})$;
17: $knnID[k] \leftarrow$ id of $\mathbf{A}$;
18: Update $knnDistance$ and $knnID$;
19: **end if**
20: **end for**

3.3 Updating *Muse*

Recall that *Muse* considers the computation of distances between reference points and the MTS items, and the computation of weights separately. As described in Section 3.2 of [14], all the eigenvalues from the dataset are stored in an $n \times N$ matrix S, where n is the number of variables and N is the number of MTS items in the dataset. When a new MTS item is added into the database, the weights for *Eros* can thusly be updated as in Algorithm 5. The MTS item can be inserted into *Muse* following Lines 3∼9 of Algorithm 3. When MTS items are removed from the database, *Muse* can be updated similarly.

Algorithm 5 Update weights for *Muse*

1: function updateWeight(S,S')
Require: an $n \times N$ matrix S, where n is the number of variables for the dataset and N is the number of MTS items in the dataset. An $n \times N'$ matrix S', where n is the number of variables for the dataset and N' is the number MTS items newly added to the dataset.
2: $S_{new} \leftarrow$ concatenate S and S';
3: computeWeightRaw(S_{new}) or computeWeightRatio(S_{new});

Algorithm 6 Computing a weight vector w based on the distribution of raw eigenvalues

1: function computeWeightRaw(S)
Require: an $n \times N$ matrix S, where n is the number of variables for the dataset and N is the number of MTS items in the dataset. Each column vector s_i in S represents all the eigenvalues for the ith MTS item in the dataset. s_{ij} is a value at column i and row j in S. s_{*i} is the ith row in S. s_{i*} is the ith column, i.e, s_i.
2: **for** i=1 to n **do**
3: $w_i \leftarrow f(s_{*i})$;
4: **end for**
5: **for** i=1 to n **do**
6: $w_i \leftarrow w_i / \sum_{j=1}^{n} w_j$;
7: **end for**

Algorithm 7 Computing a weight vector w based on the distribution of normalized eigenvalues

1: function computeWeightRatio(S)
Require: the same as Algorithm 6.
2: **for** i=1 to N **do**
3: $s_i \leftarrow s_i / \sum_{j=1}^{n} s_{ij}$;
4: **end for**
5: computeWeightRaw(S);

3.4 Reference Points

The choice of reference points affects the performance of the distance-based index structures [15] and the index should be re-built when the reference points change. Therefore, the reference points should be chosen with care. First, let us consider the data, i.e., the PCs, for which an index structure is built. The PCs are normal vectors, whose norms are 1s. Hence, they can be represented as points in a hypersphere whose radius is 1. If we take the absolute values of the PCs in order to compute $D_{|\cdot|}$, the resultant PCs are represented as points on the hypersphere in the first quadrant (see Figure 2 for an example in 3D space).

Based on this observation, for our experiments, the edge points where the hypersphere meets each axis are chosen as the reference points at each level. And the reference points are the same for all the levels. There are two advantages to this heuristic: 1) As observed in [15], this would in general

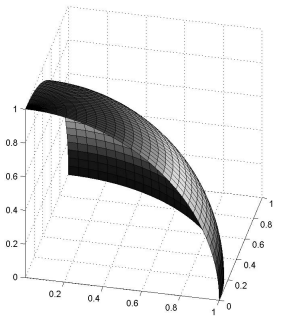


Figure 2. The data space of the first PC group for *Muse* in 3D

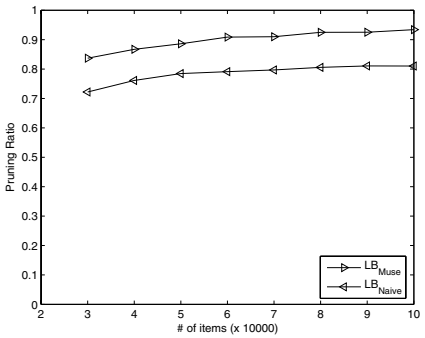


Figure 3. Pruning ratios of LB_{Muse} with LB_{Naive}

reduce the amount of overlap among partitions, and 2) It is easy to find the partition to which a PC belongs. The partition ID of a PC $v_i^{\mathbf{A}}$ is $\underset{j}{\operatorname{argmax}}|v_{ij}^{\mathbf{A}}|$, i.e., the dimension whose absolute value is the maximum. Hence, we do not have to compute the distances between a PC and reference points to find out to which partition a PC belongs.

3.5 Levels of *Muse*

Muse utilizes the first z PC groups to construct a z-level index structure. The computation of distances and weights are separately considered. When a kNN search is performed, the distances computed *a priori* are combined with the weights to yield a lower bound, i.e., LB_{Muse}. On one hand, the greater the number of levels of *Muse* is, the tighter the LB_{Muse} is. One the other hand, the greater the number of levels, the longer it would take to compute LB_{Muse} and to perform Line 9 of Algorithm 4. Hence, the number of levels of *Muse* should be decided with discretion.

Recall that the weights used for *Muse* are based on the distribution of the eigenvalues obtained from all the MTS items in the database. Hence, the weights will have similar characteristics as the eigenvalues, i.e., the first few weights have large values while the rest have small values close to zero. For our experiments, we employed similar heuristics used for S_{PCA} to choose the number of principal components. The number of levels, z, is chosen such that the sum of the first z weights is greater than 0.99.

4 PERFORMANCE EVALUATION

4.1 Datasets

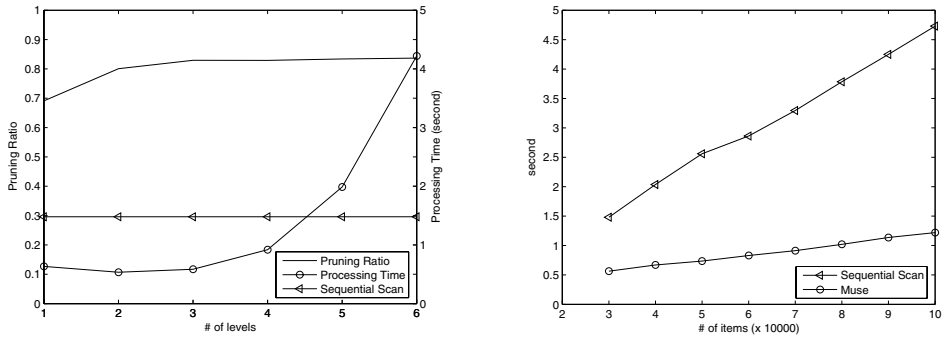
In order to show the efficiency of *Muse*, we create a set of clustered synthetic dataset, $SYNTH$, as in [1], by adding a small random variation to each of the AUSLAN MTS item. The Australian Sign Language (AUSLAN) dataset [4] uses 22 sensors on the hands to gather the datasets generated by signing of a native AUSLAN speaker. It contains 95 distinct signs, each of which has 27 examples. In total, the number of signs gathered is 2565. The average length is around 60.

The variation added to each of the AUSLAN MTS item is a vector whose values are chosen from the interval $[0, 1]$ and then processed so that its mean is 0 and its values are between $[-\epsilon, \epsilon]$. For experiments, ϵ is chosen to be 0.05. We use each AUSLAN data as seed and generate approximately 30,000 to 100,000 items.

4.2 Methods

We compare *Muse* to M-tree in terms of pruning ratio and processing time. Pruning Ratio is the ratio of the number of items pruned to the number of items in the database [7]. The processing time of *Muse* is also compared to that of sequential scan. Recall that *Eros*, in itself, is not a distance metric; *Eros* utilizes its lower bound, D_{min}, to perform similarity search efficiently. Hence, M-tree cannot be used with *Eros*. Moreover, M-tree cannot be used with weighted distance metrics whose weights may change frequently; M-tree should be reconstructed each time the weights change. Therefore, in order to compare *Muse* and M-tree, we modified both of them to compute the distance between two MTS items using D_{min} which is a distance metric, and assumed there would be no change in the database once the index structures are constructed. Page sizes of 8KB and 16KB are employed for M-tree. We only show the result of 16KB M-tree, which is better than that of 8KB M-tree. For the reference points of *Muse*, as described in Section 3.4, the edge points where the hypersphere meets each axis are chosen at each level. *Muse* is implemented in

[4] http://kdd.ics.uci.edu/



(a) Trade-offs of *Muse* **(b) Processing Time**

Figure 4. (a) shows the trade-offs of *Muse* : Pruning Ratio *vs* Processing Time *vs* Level. (b) depicts the Processing times of *Muse* and Sequential Scan using *Eros* as the similarity measure.

both C++ and Matlab. The experiments are performed on a Pentium IV 3.2GHz machine with 3GB of main memory.

4.3 RESULTS

The pruning ratios of LB_{Muse} and LB_{Naive} using the SYNTH dataset is presented in Figure 3. This figure confirms Lemma 1 as LB_{Muse} yields tighter lower bound than LB_{Naive} resulting in higher pruning ratio than LB_{Naive} by more than 10%.

For *Muse*, even though the number of levels increases, the pruning ratio does not improve much after level 3, while the processing time more than doubles when the number of level changes from 5 to 6, as Figure 4(a) shows using the SYNTH dataset with 30K items. Also, the performance is worse than sequential scan when the number of levels is greater than 5, where the overhead of computing LB_{Muse} and performing Line 9 of Algorithm 4 begin to overwhelm. Moreover, the number of partitions would be n^z for a z-level *Muse*. For the AUSLAN dataset, there would be 234256 (22^4) partitions for a 4-level *Muse*. Hence, we suggest no more than 4 levels for *Muse* in general.

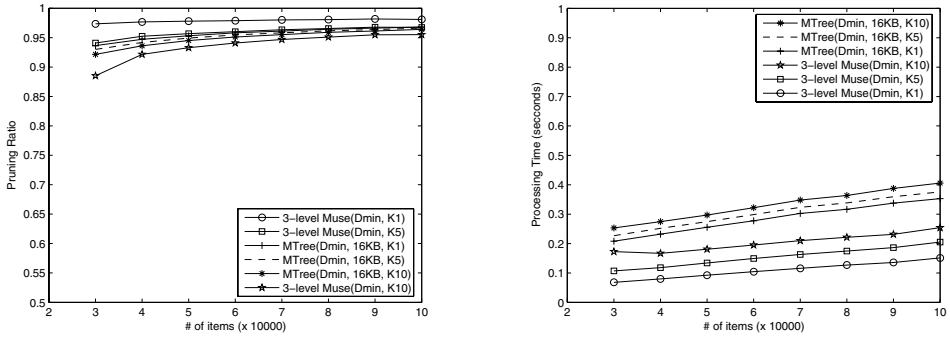
Figure 4(b) demonstrates that *Muse* is almost 4 times faster than sequential scan when the number of data items is 100,000. The length of a PC for each MTS item in the SYNTH dataset is 22. 3-level *Muse*, which utilizes the first 3 PCs, has 66 dimensions. Figure 4(b) shows that *Muse* works well for high dimensional datasets, while the feature-based index techniques, such as R-tree and its variants, become inefficient when the dimension is greater than 20 [1].

The pruning ratios of M-tree and 3-level *Muse* when D_{min} distance metric is employed are shown in Figure 5(a), for 1 NN searches (K1), 5 NN searches (K5) and 10 NN searches (K10). Recall that, unlike *Muse*, M-tree cannot be utilized, e.g., for *Eros*, when the weighted Euclidean distance is the distance metric to be used for the index structures, and the weight keeps changing whenever new data items are inserted into and/or removed from the database, which would probably be the usual case when dealing with real-world time series datasets.

Though 3-level *Muse* utilizes only the first three PC groups (i.e., 66 dimensions), and M-tree considers all the PC groups (i.e., $22 \times 22 = 484$ dimensions), Figure 5(a) represents that the pruning ratio of 3-level *Muse* is comparable to that of M-tree. When performing 1 NN searches, the pruning ratio of *Muse* is even higher than that of M-tree. This is due to the fact that the weights for *Eros* are based on the distributions of the eigenvalues, the first few of which represent more than 99% of the total variance. Another reason of M-tree's poor performance in processing time would be that M-tree does not utilize all the pre-computed distances immediately; M-tree can only utilize the distances of MTS items in the visited nodes. Similar result has also been observed in [2]. Figure 5(b) shows that 3-level *Muse* outperforms M-tree in processing time, which may reconfirm the aforementioned limitation of M-tree.

5 CONCLUSIONS AND FUTURE WORK

We proposed a multilevel distance-based index structure, termed *Muse* (*Mu*ltilevel distance-based index *s*tructure for *Eros*) for efficient retrieval of MTS items using *Eros* [14]. *Muse* builds a number of *levels*, each of which is a distance-based index structure corresponding to one *PC group*. Hence, for a z-level *Muse*, the first z PC groups are utilized. Since *Eros* assigns a weight to each PC group, each level of *Muse* can be constructed without considering the weight. When performing a similarity search, we combine



(a) Pruning Ratio **(b) Processing time**

Figure 5. For the SYNTH dataset, (a) shows the pruning ratios of 3-level *Muse* utilizing 66 dimensions and M-tree with page size of 16KB employing 484 dimensions, and (b) depicts the processing times of both methods. The distance metric D_{min} is employed for both *Muse* and M-tree for the sake of comparison.

the z levels with the weights to compute the lower bounds between the query item and the items in the database, and filter out those that are not to be in the result sets. For items that are not filtered out, the refinement is performed using D_{Eros} to obtain the result. Using a set of synthetically generated clustered datasets, we showed that *Muse* outperforms the sequential scan and M-tree in terms of pruning ratio and processing time.

We intend to extend this work to obtain possibly higher pruning ratio by utilizing the upper bound of D_{Eros}, i.e., D_{max}, as well as the lower bound of D_{Eros}, i.e., D_{min}. In addition, we also plan to figure out if some simpler structures, such as B+tree utilized in iDistance [15], can be employed for *Muse*. If this is plausible, *Muse* may be easily plugged into commercial database systems.

Acknowledgment

This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), IIS-0238560 (PECASE) and IIS-0307908, and unrestricted cash gifts from Microsoft. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The authors would also like to thank the anonymous reviewers for their valuable comments.

References

[1] T. Bozkaya and M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM TODS*, 24(3), 1999.

[2] L. Chen and R. Ng. On the marriage of lp-norms and edit distance. In *Proc. of the VLDB Conference*, 2004.

[3] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, 1997.

[4] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley Interscience, second edition, 2001.

[5] C. Goutte, P. Toft, E. Rostrup, F. A. Nielsen, and L. K. Hansen. On clustering fMRI time series. *NeuroImage*, 9(3):298–310, 1999.

[6] I. T. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[7] E. Keogh. Exact indexing of dynamic time warping. In *Proc. of the VLDB Conference*, 2002.

[8] W. Krzanowski. Between-groups comparison of principal components. *JASA*, 74(367), 1979.

[9] T. K. Moon and W. C. Stirling. *Mathematical Methods and Algorithms for Signal Processing*. Prentice Hall, 2000.

[10] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. Acoust., Speech, Signal Processing*, 26(1), 1978.

[11] C. Shahabi. AIMS: An immersidata management system. In *VLDB CIDR*, January 2003.

[12] C. Shahabi and D. Yan. Real-time pattern isolation and recognition over immersive sensor data streams. In *Proc. of the 9th Int. Conference On Multi-Media Modeling*, 2003.

[13] A. Tucker, S. Swift, and X. Liu. Variable grouping in multivariate time series via correlation. *IEEE Trans. Syst., Man, Cybern. B*, 31(2):235–245, 2001.

[14] K. Yang and C. Shahabi. A PCA-based similarity measure for multivariate time series. In *Proc. of the Second ACM International Workshop on Multimedia Databases*, 2004.

[15] C. Yu, B. C. Ooi, K.-L. Tan, and H. V. Jagadish. Indexing the distance: An efficient method to KNN processing. In *VLDB*, pages 421–430, September 2001.

[16] X. L. Zhang, H. Begleiter, B. Porjesz, W. Wang, and A. Litke. Event related potentials during object recognition tasks. *Brain Research Bulletin*, 38(6):531–538, 1995.