Available online at www.sciencedirect.com

ScienceDirect

Information and Computation 205 (2007) 65–98

Information and Computation

www.elsevier.com/locate/ic

An efficient *k* nearest neighbor search for multivariate time series

Kiyoung Yang [*], Cyrus Shahabi

University of Southern California, Department of Computer Science, Information Laboratory (InfoLab), Los Angeles, CA 90089-0781, USA

Received 1 December 2005; revised 15 June 2006
Available online 18 October 2006
Communicated by Jan Chomicki and David Toma

Abstract

Multivariate time series (MTS) datasets are common in various multimedia, medical and financial applications. In order to efficiently perform *k* nearest neighbor searches for MTS datasets, we present a similarity measure, *Eros* (extended frobenius norm), an index structure, *Muse* (multilevel distance-based index structure for *Eros*), and a feature subset selection technique, *Ropes* (recursive feature elimination on common principal components for *Eros*). *Eros* is based on principal component analysis, and computes the similarity between two MTS items by measuring how close the corresponding principal components are using the eigenvalues as *weights*. *Muse* constructs each *level* as a distance-based index structure without using the weights, up to *z* levels, which are combined at the query time with the weights. *Ropes* utilizes both the common principal components and the weights recursively in order to select a subset of features for Eros. The experimental results show the superiority of our techniques as compared to earlier approaches.
© 2006 Elsevier Inc. All rights reserved.

Keywords: Multivariate time series; Distance-based index structure; Similarity measure; Principal component analysis; Feature subset selection

[*] Corresponding author. Fax: +1 213 821 1739.
E-mail addresses: kiyoungy@usc.edu (K. Yang), shahabi@usc.edu (C. Shahabi).

0890-5401/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2006.08.004

1. Introduction

A time series is a series of observations, $x_i(t)$; $[i = 1, \ldots, n; t = 1, \ldots, m]$, made sequentially over time where i indexes the measurements made at each time point t [1]. 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. An MTS item is naturally represented in an $m \times n$ matrix, where m is the number of observations and n is the number of *variables*, e.g., sensors.

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 [2]. In medicine, electro encephalogram (EEG) from 64 electrodes placed on the scalp are measured to examine the correlation of genetic predisposition to alcoholism [3]. Functional magnetic resonance imaging (fMRI) from 696 voxels out of 4391 has been used to detect similarities in activation between voxels in [4].

In this paper, we present three techniques in order to perform efficient k nearest neighbor (kNN) searches for MTS datasets, i.e., a similarity measure to compute the similarity between two MTS items, an index structure for efficient retrieval of MTS items, and a feature subset selection technique in order to compactly represent MTS items. The similarity measure, termed *Eros* (extended Frobenius norm), is based on the Frobenius norm that is used to compute the matrix norm [5], 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 either their covariance or correlation coefficients matrices, are using the eigenvalues as weights. The weights are aggregated over the eigenvalues of all the MTS items in the dataset. Hence, the weights change whenever data are inserted into or removed from the dataset. Though *Eros* in itself does not satisfy the triangle inequality, we can obtain the lower and upper bounds using the weighted Euclidean distance, which satisfies the triangle inequality. Without this property, the filter and refinement phases of existing multidimensional indexing techniques cannot be utilized, because false dismissals may occur in the filter phase [7–9].

In order to efficiently perform kNN searches using *Eros*, we present an index structure, termed *Muse* (multilevel distance-based index structure for *Eros*). *Muse* constructs each *level* as a distance-based index structure without using the weights, up to z levels, which are combined at the query time with the weights. The distance-based index structures, such as iDistance [10] and M-tree [11], typically utilize the Euclidean distance when building the index structures. Even though the weighted Euclidean distance metric can be used for iDistance and M-tree, the index structure should be rebuilt whenever 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 are incorporated later, 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 is described in Section 4.2.

The size of an MTS dataset can become very large quickly. For example, the EEG dataset utilizes tens of electrodes and the sampling rate is 256 Hz. In order to process MTS datasets efficiently, it is inevitable to preprocess the datasets to obtain the relevant subset of features which will be subsequently employed for further processing. Intuitively, *Eros* can be considered as first representing an MTS item using either the covariance or the correlation coefficient matrix, and then applying SVD in order to obtain the principal components. The correlation coefficient representation of an

MTS item reduces the dimension of the original MTS item. For example, an original MTS item for the AUSLAN dataset used in the experiments in Section 5 has 22 variables and 60 observations on the average, which may be considered as a point in a 22×60 dimensional space. With the correlation coefficient representation, the dimension is reduced to 22×22. In order to more compactly represent an MTS item for *Eros*, we present a feature subset selection (FSS) technique, termed *Ropes* (recursive feature elimination on common principal components for *Eros*). *Ropes* selects a subset of the original *variables* and helps *Eros* improve the processing time as well as the precision/recall by eliminating irrelevant and/or redundant *variables*, which also results in more compact representation of MTS datasets. Intuitively, *Ropes* extends **CL***e***V***er* proposed in [12], and utilizes both the common principal components (CPCs) [13] and the weights, and *recursively* selects a subset of features based on the CPCs after applying the same weights obtained for Eros to the CPCs.

In order to evaluate the performances of our proposed approaches, we conducted several experiments on three real-world datasets and two synthetic datasets. The experimental results in Section 5.3 show that *Eros* performs better than traditional techniques, such as Euclidean distance (ED) and dynamic time warping (DTW), in terms of precision/recall; *Muse* outperforms M-tree [11] and sequential scan in terms of efficiency and scalability; and *Ropes* outperforms the **CL***e***V***er* family [12] in terms of classification accuracy.

Eros and *Muse* have been originally proposed in [14] and [15], respectively. In this paper, we integrate the two and propose a novel feature subset selection technique, called *Ropes*, for *Eros* with more exhaustive performance evaluation. Hence, this paper subsumes our previous work in [14] and [15].

The remainder of this paper is organized as follows. In Section 2, the related work is discussed. Section 3 discusses the background. Our proposed methods are described in Section 4, which is followed by the experiments and results in Section 5. Conclusions and future work are presented in Section 6.

2. Related work

Traditional techniques for computing the similarity between two time series include Euclidean distance (ED) and dynamic time warping (DTW) [16]. DTW is a technique for performing time-alignment and has been extensively employed in various applications, such as speech recognition [16] and time series similarity search [9]. DTW, however, requires quadratic computation cost $O(n^2)$, where n is the length of the time series. Hence, DTW typically sets the global limit on the maximum amount of warping to less than 10% of the length of the time series, which reduces the processing time of DTW. These two techniques, however, do not consider the correlation information among the variables within an MTS item.

A couple of techniques to compute the similarities between trajectories are proposed in [17,18], where the trajectories may be represented as 2 variate or 3 variate time series. In [17], the authors presented EDR (edit distance on real sequences), which has been shown to work similarly as dynamic time warping (DTW), when no noise and no time shifting are artificially added to the dataset. However, the computation cost of EDR is $O(n \times m^2)$, where n is the number of dimensions/variables and m is the length of the trajectory. Furthermore, their datasets have only 2 dimensions/variables, and it remains to be seen if this technique works well for datasets with up to 66 variables as in our datasets. In [18], the authors presented a similarity measure for trajectories, which is LCSS-variant. For efficient retrieval, a clustering based index structure is also described. However, their datasets have only 2 dimensions/variables. Also, as observed in [17], LCSS is not a metric, which makes it rather not clear if *good* clustering result can be obtained for its index structure.

Index structures for high dimensional datasets have been extensively explored [7,8,19], and many techniques have been proposed, such as A-tree [20] which is a feature-based index structures, and M-tree [11] and iDistance [10] which are distance-based index structures. It has been shown that the distance-based index structures outperform the feature-based index structures [11,10].

In [21], the authors employed Chebyshev polynomials to lower-bound the Euclidean distance between two trajectories, in order to index trajectories. That is, they utilize Euclidean distance as their similarity measure. Moreover, their datasets have only up to 4 dimensions/variables, and it remains to be seen if this technique works well for datasets with up to 66 variables as in our datasets. In [22], the authors proposed an index structure that supports LCSS, DTW, and ED. Their technique is based on R-tree, which stores segmented trajectories. The dataset has only 2 dimensions/variables. Moreover, R-tree does not work well for high dimensional datasets; our datasets have up to 66 variables.

The spatial transform technique (STT) proposed in [23] is one of the approaches to addressing the weight change problem. A multidimensional index structure, A-tree, is first constructed using Euclidean distance. When a kNN search is issued, the MBRs of the index structure are transformed by STT based on the weights. The MBRs for the transformed MBRs are found and then kNNs are searched for. In [10], it has been shown that A-tree, which STT is based upon, is outperformed by iDistance, which is a distance-based index structure. Hence, we utilize the distance-based index scheme for our similarity measure.

In [24], QIC-M-tree is presented, where the similarity measure at the query time may be different from the similarity measure at the index creation time. That is, when constructing a tree, the un-weighted Euclidean distance is utilized ($d_I = L_2$), while, when querying, the weighted Euclidean Distance is employed ($d_Q = L_{qf}$), where the weights are placed on the diagonal of a matrix $\mathbf{A}$. However, in order to obtain the lower-bound of d_Q, the smallest eigenvalue of $\mathbf{A}$ is utilized. As the results on the Airphoto8 dataset and the Corel dataset show, the overhead may be overwhelming when only the smallest eigenvalue of $\mathbf{A}$ is used, since the difference between d_I and d_Q would be large, which would result in performance degradation.

There have been a lot of efforts in the field of feature subset selection (FSS) and feature extraction [25–27]. For the multivariate time series, however, the feature subset selection technique has not been extensively explored, especially for the similarity measures. In [28], Guyon et al. proposed recursive feature elimination (RFE) using support vector machine, whose procedure can be briefly described as follows: (1) train the classifier, (2) compute the ranking criterion for all features, and (3) remove the feature with lowest ranking criterion. This procedure is then repeated until the required number of features remain. Though RFE has been shown to perform well, RFE cannot be applied directly to MTS datasets; each MTS item should be represented as one row vector.

Extending RFE for MTS datasets, FSS is performed on the EEG dataset with 39 channels in [29]. In order to apply RFE to MTS datasets, each EEG item is first broken into 39 separate channels, and for each channel, autoregressive (AR) fit of order 3 [5] is computed. Subsequently, each channel is thus represented by 3 autoregressive coefficients. RFE is then performed on this transformed dataset to select significant channels. However, by considering the channels separately, they lose the correlation information among channels.

3. Background

In this section, we will briefly discuss principal component analysis (PCA), common principal components (CPCs), the distance-based index structure, and the feature subset selection techniques, on which our proposed techniques are based. For details, please refer to [6,10,12,13].

3.1. Principal component analysis

Principal component analysis (PCA) has been widely used for multivariate data analysis and dimension reduction [6]. Fig. 1a illustrates principal components obtained from a simple multivariate dataset with only two variables (x_1, x_2) measured on 30 observations. Geometrically, the principal component is a linear transformation of original variables and the coefficients defining this transformation are called *loadings*. For example, the first principal component (PC1) in Fig. 1a can be described as a linear combination of original variables x_1 and x_2, and the two coefficients (loadings) defining PC1 are the cosines of the angles between PC1 and variables x_1 and x_2, respectively. The loadings are thus interpreted as the contributions or weights on determining the directions.

In practice, PCA is performed by applying singular value decomposition (SVD) to either a covariance matrix or a correlation matrix of an MTS item depending on the dataset. That is, when a covariance matrix A is decomposed by SVD, i.e.,

$$A = U\Lambda U^{\mathrm{T}}, \tag{1}$$

a matrix U contains the variables' loadings for the principal components, and a matrix Λ has the corresponding variances along the diagonal [6].

3.2. Common principal component

Common principal component analysis (CPCA) is a generalization of PCA for N (≥ 2) multivariate data items, where the ith multivariate data item, ($1 \leq i \leq N$), is represented in an $m_i \times n$ matrix. That is, all the data items have the same number of variables, n, while each data item may have different number of observations. CPCA is based on the assumption that there exists a common subspace across all multivariate data items and this subspace should be spanned by the orthogonal components. Various efforts have been made to find the common components defining this common subspace [13,30–32].

One approach proposed in [13] obtained the common principal components (CPCs) by bisecting the angles between their principal components after each multivariate data item undergoes PCA. These CPCs define the common subspace that *agrees most closely* with every subspace of the multivariate data items. Fig. 1b gives a plot of two multivariate data items A and B. Let A and B be denoted as a swarm of white points and black points, respectively, and have the same number of variables, i.e., x_1 and x_2, measured on 20 and 30 observations, respectively. The first principal component of each dataset is obtained using PCA and the common component is obtained by bisecting the angle

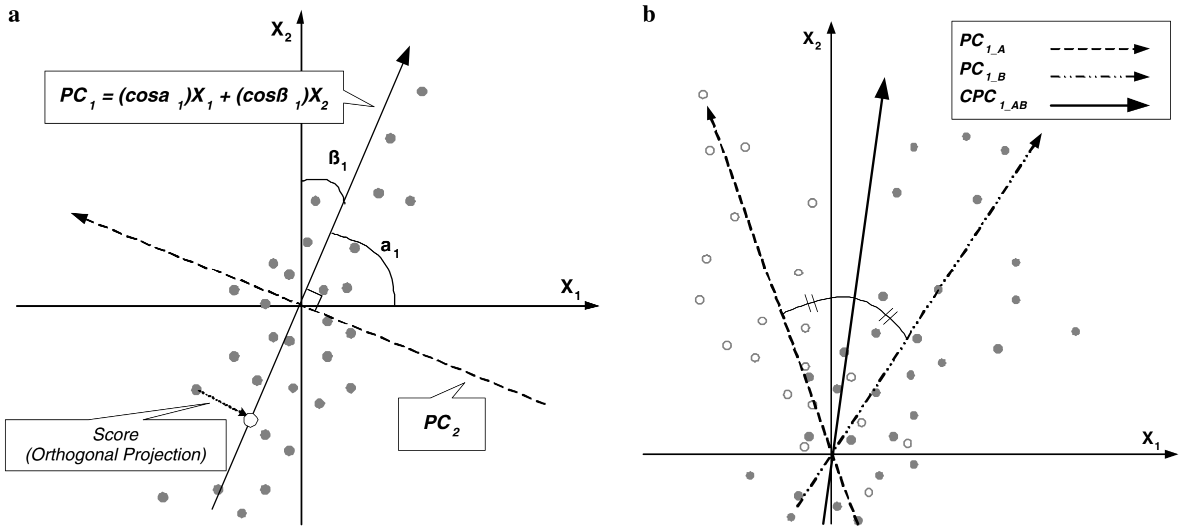


Fig. 1. (a) Two principal components obtained for one multivariate data with two variables x_1 and x_2 measured on 30 observations. (b) A common principal component of two multivariate data items with the same variables x_1 and x_2 measured on 20 and 30 observations, respectively.

between those two principal components. We refer to this CPC model as *descriptive common principal component* (DCPC), based on which we proposed a family of FSS techniques in [12], which is briefly described in the following section.

3.3. Feature subset selection

Feature subset selection (FSS) is one of the techniques to pre-process the data before we perform any data mining tasks, e.g., classification and clustering. FSS is to identify a subset of original features from a given dataset while removing irrelevant and/or redundant features [33]. The objectives of FSS are [26]:

– to improve the prediction performance of the predictors
– to provide faster and more cost-effective predictors
– to provide a better understanding of the underlying process that generated the data

FSS methods choose a subset of the original features to be used for the subsequent processes. Hence, only the data generated from those features need to be collected.

Based on the DCPCs described in the previous section, we proposed in [12] a family of three *unsupervised* feature subset selection methods for MTS datasets, which are termed **CL***e***V***er-Rank*, **CL***e***V***er-Cluster*, and **CL***e***V***er-Hybrid*. After obtaining the DCPCs for an MTS dataset, **CL***e***V***er-Rank* ranks each variable based on its contribution to the DCPCs. Subsequently, FSS is done by choosing the top K ranked variables. **CL***e***V***er-Cluster* clusters the DCPC loadings to identify the variables that have similar contributions to each of the DCPCs. For each cluster, we obtain the *centroid* variable, eliminating all the *similar* variables within the cluster. These centroid variables form the selected subset of variables. **CL*e*V***er-Hybrid* first clusters the DCPCs, and ranks the variables within each cluster based on their contribution to the DCPCs. Subsequently, the top ranked variables from all the clusters are chosen to be a part of the selected subset of variables.

3.4. Distance-based index structures

Intuitively, the distance-based index techniques, such as iDistance [10] and M-tree [11], 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 may employ any distance metric. Assume that the distance metric for, e.g., M-tree is a weighted Euclidean distance, whose weight changes adaptively based on the characteristics of datasets. Although the weighted Euclidean distance can be utilized for M-tree, the index structures should then be reconstructed whenever there is a change in the weight, i.e., in the dataset. This reconstruction would be very costly when there are a lot of items in the dataset, and the datasets are frequently updated.

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.
```

Algorithm 2 Distance-based index : kNN search

```
1: Given a query item $Q$ for which the $k$NN 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 $k$NNs of $Q$ using the triangle inequality from within the closest partition.
```

4. The proposed methods

In this section, we describe our proposed similarity measure, an index structure for MTS datasets, and a feature subset selection technique. Table 1 lists the notations used in the remainder of this paper, if not specified otherwise.

4.1. Eros: extended frobenius norm

We first formally define our proposed similarity measure, *Eros*. Next, we provide the intuitions behind it.

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. Let $\mathbf{V_A}$ and $\mathbf{V_B}$ be two right eigenvector matrices by applying SVD to the

Table 1
Notations used in this paper

Symbol	Definition
$\mathbf{A}$	an $m \times n$ matrix representing an MTS item
$\mathbf{A}^{\mathrm{T}}$	the transpose of $\mathbf{A}$
$\mathbf{M}_{\mathbf{A}}$	the covariance matrix of size $n \times n$ for $\mathbf{A}$
$\mathbf{V}_{\mathbf{A}}$	the right eigenvector matrix of size $n \times n$ for $\mathbf{M}_{\mathbf{A}}$, i.e., $\mathbf{V}_{\mathbf{A}} = [\, v_1^A, v_2^A, \ldots, v_n^A \,]$
$\Sigma_{\mathbf{A}}$	an $n \times n$ diagonal matrix that has all the eigenvalues for $\mathbf{M}_{\mathbf{A}}$ obtained by SVD
v_i^A	a column orthonormal eigenvector of size n for $\mathbf{V}_{\mathbf{A}}$
v_{ij}^A	jth value of v_i^A, i.e., a value at the ith column and the jth row of $\mathbf{V}_{\mathbf{A}}$
v_{*j}^A	all the values at the jth row of $\mathbf{V}_{\mathbf{A}}$
w	a weight vector of size n, such that $\sum_{i=1}^{r} w_i = 1, \forall i\ w_i \geqslant 0$
n	number of variables, i.e., number of columns of a matrix
m_A	number of observations, i.e., number of rows of a matrix $\mathbf{A}$
δ	threshold to determine the subspace for DCPC

covariance matrices, $\mathbf{M}_{\mathbf{A}}$ and $\mathbf{M}_{\mathbf{B}}$, respectively. Let $\mathbf{V}_{\mathbf{A}} = [a_1, \ldots, a_n]$ and $\mathbf{V}_{\mathbf{B}} = [b_1, \ldots, b_n]$, 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

$$Eros(\mathbf{A}, \mathbf{B}, w) = \sum_{i=1}^{n} w_i | < a_i, b_i > | = \sum_{i=1}^{n} w_i |\cos \theta_i| \tag{2}$$

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 4.1.1), $\sum_{i=1}^{n} w_i = 1$ and $\cos \theta_i$ is the angle between a_i and b_i. The range of *Eros* is between 0 and 1, with 1 being the most similar.

We now discuss the intuition behind the *Eros* equation. Our intuition is that the principal components (PCs) and the associated eigenvalues, i.e., the variances for the principal components, represent the *characteristics* of a matrix. Hence, *Eros* measures the similarity between two MTSs by comparing the corresponding principal components using the associated aggregated eigenvalues as weights. Using the PCs for similarity computation has the following advantages:

– Same size for all the MTS data items : in general, MTS items of a given application will have the same number of variables n, i.e., sensors, but may have different number of observations m. For example, the AUSLAN dataset [34] utilizes 22 sensors for all the MTS items, but the number of observations vary from 45 to 136. Comparing two MTS items with different sizes is a challenge. The size of a right eigenvector matrix U, which contains all the PCs (see Eq. (1)), however, is fixed at $n \times n$. Thus, the problem of different lengths is resolved.

Table 2
Computational costs for Eros, ED, and DTW

Eros	ED	DTW
$O(m \times n^2 + n^3)$	$O(m \times n)$	$O(m^2 \times n)$

– Dimension reduction: for MTS items, the number of observations m, is usually far greater than the number of variables, n. Considering that the size of a right eigenvector matrix is $n \times n$, the size of the data to be dealt with is greatly reduced.

Note that *Eros* only considers the *acute* angle that the two corresponding PCs generate. The PCs represent the *axes* with the maximum variances, not the direction [35]. Therefore, *Eros* only considers the acute angle between the corresponding PCs by taking the absolute value of the inner product as in Eq. (2).

Table 2 shows the comparison of the computational cost of *Eros* to those of Euclidean distance (ED) and dynamic time warping (DTW). *Eros* first computes the covariance matrix, which is to fill an n by n matrix whose entry requires m multiplications. Subsequently, the covariance matrix is decomposed by singular value decomposition (SVD), which requires n^3 multiplications. Hence, the first term for *Eros* in Table 2 represents the cost for computing the covariance matrix, and the second term, $O(n^3)$, represents the cost for computing SVD. The costs for ED and DTW are obtained from [17], multiplied by the number of variables n. Note that in [17], the trajectories are represented by either 2 or 3 variables. Hence, the number of variables can be considered as constants. However, in our case, the number of variables should be considered explicitly in the cost computation, since the number of variables vary depending on the datasets. For example, the datasets considered in this paper have up to 66 variables, which may not be considered as constants. Consider the case for dynamic time warping (DTW). Using the dynamic programming technique, DTW needs to fill a an m by m matrix, where m is the number of observations, i.e., the length of the time series. Moreover, each entry in the matrix requires n multiplications, where n is the number of variables/dimensions. Hence, the computation cost for DTW is $O(m \times n^2)$. In general, for an MTS item, the number of observations m is far greater than the number of variables n. Hence, asymptotically, *Eros* is slower than ED, while faster than DTW.

Fig. 2 depicts the preprocessing stage for *Eros*. That is, given an MTS dataset, the principal components and their corresponding variances are computed for each of the MTS item. Note that this preprocessing stage is common to all the subsequent techniques, as well. Based on the variances obtained from all the MTS items in the dataset, the weight vector is computed, which is described in the following section.

4.1.1. Computing weights

When comparing two MTS items, *Eros* considers both the PCs and the eigenvalues. In this section, we propose two heuristics for computing the weight vector w for *Eros* based on the eigenvalues obtained from the MTS dataset satisfying the following conditions: (1) $\sum_i^n w_i = 1$ and (2) $w_i \geqslant 0$ for all i .

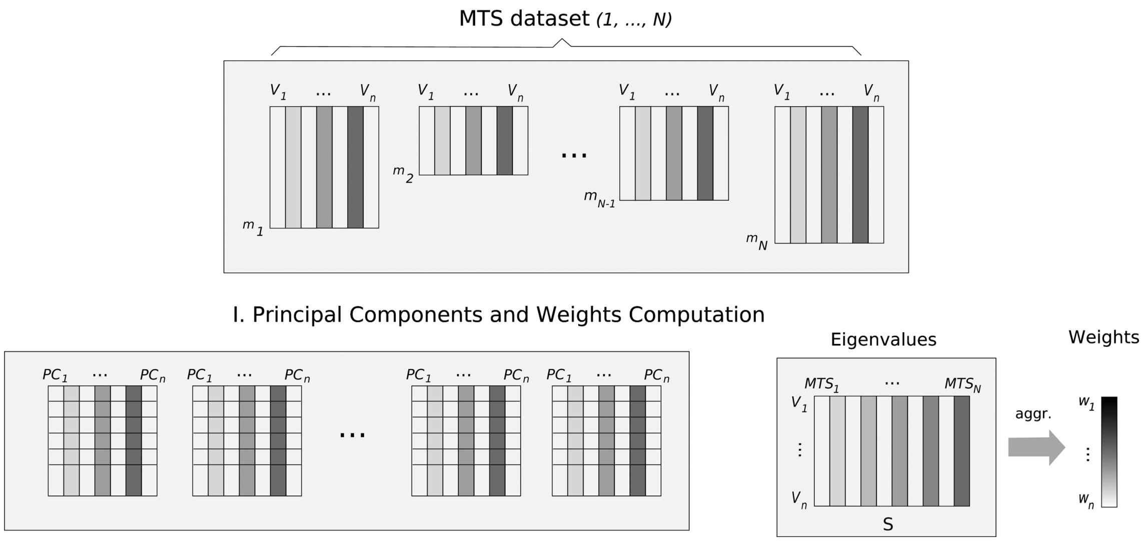


Fig. 2. The preprocessing processes for *Eros*.

The same weight vector should be used for all the similarity measure computations for each *k*NN search. Hence, the eigenvalues obtained from all the MTS items in the dataset are aggregated into one weight vector as in Algorithm 3. Algorithm 3 computes the weight vector w based on the distribution of *raw* eigenvalues. Function $f()$ in Line 3 of Algorithm 3 can be any aggregating function, e.g., min, mean or max. Intuitively, each w_i in the weight vector represents the aggregated variance for all the ith PCs for the dataset. The weights are then normalized so that $\sum_{i=1}^{n} w_i = 1$. The weight vector can also be obtained by first normalizing the matrix S that contains all the eigenvalues from the dataset (See Fig. 2), and then applying Algorithm 3.

Algorithm 3 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 $i$th MTS item in the dataset. $s_{ij}$ is a value at column $i$ and row $j$ in S. $s_{*i}$ is the $i$th row in S. $s_{i*}$ is the $i$th column, i.e, $s_i$.
2: for i=1 to n
3:    $w_i \leftarrow f(s_{*i})$;
4: end for
5: $sum\_weight \leftarrow \sum_{j=1}^{n} w_j$
6: for i=1 to n
7:    $w_i \leftarrow w_i / sum\_weight$;
8: end for
```

4.1.2. Bounding Eros

Eros in itself does not satisfy the triangle inequality. Hence, we obtain the lower and upper bounds using the weighted Euclidean distance, which satisfies the triangle inequality. The lower bound of *Eros* is to be utilized for efficient retrieval of MTS items in Section 4.2.

We first define the *Eros distance measure*, D_{Eros}, which preserves the similarity relation of *Eros*. Subsequently, we describe two weighted Euclidean distances between two eigenvector matrices, which are used as the upper and lower bounds of D_{Eros}.

Definition 2. D_{Eros} is defined as:

$$D_{Eros}(\mathbf{A},\mathbf{B},w) = \sqrt{2 - 2\sum_{i=1}^{n} w_i| < v_i^A, v_i^B > |} = \sqrt{2 - 2\sum_{i=1}^{n} w_i| \sum_{j=1}^{n} v_{ij}^A \times v_{ij}^B|} \qquad (3)$$

D_{Eros} preserves the similarity relation of *Eros*. That is, if the *Eros* similarity between **A** and **B** is greater than that of **B** and **C**, the *Eros* distance between **A** and **B** is then shorter than that of **B** and **C**. This relation is formally stated in the following lemma.

Lemma 1. *The similarity relation with Eros is reversely preserved with D_{Eros}.*

Proof 1. Plug in the definition of *Eros*$(\mathbf{A},\mathbf{B},w)$ into Eq. (3). □

Now, we employ the weighted Euclidean distance to bound D_{Eros}. Let us consider the weighted Euclidean distance, called $D_{\max}$, between **A** and **B**

$$D_{\max}(\mathbf{A},\mathbf{B},w) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} w_i(v_{ij}^A - v_{ij}^B)^2} = \sqrt{2 - 2\sum_{i=1}^{n} w_i \sum_{j=1}^{n} v_{ij}^A \times v_{ij}^B} \qquad (4)$$

The distance measure $D_{\max}$, which is a weighted Euclidean distance metric, satisfies the triangle inequality. Let us consider one more distance metric.

$$D_{\min}(\mathbf{A},\mathbf{B},w) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} w_i(|v_{ij}^A| - |v_{ij}^B|)^2} = \sqrt{2 - 2\sum_{i=1}^{n} w_i \sum_{j=1}^{n} |v_{ij}^A \times v_{ij}^B|} \qquad (5)$$

$D_{\min}$ is hence similar to $D_{\max}$ except that the absolute values of v_i^A and v_i^B are used. Similarly, $D_{\min}$ satisfies the triangle inequality. The upper and lower bounds of D_{Eros} are then obtained as in the following lemma.

Lemma 2. *The upper and lower bounds of D_{Eros} are D_{max} and D_{min}, respectively.*

Proof 2. Consider the summation parts in D_{Eros}, $D_{\max}$ and $D_{\min}$ in Eqs. (3), (4), and (5), respectively. Thus, we can find the inequalities among them as follows :

$$\sum_{j=1}^{n} |v_{ij}^A \times v_{ij}^B| \quad \geqslant \quad |\sum_{j=1}^{n} v_{ij}^A \times v_{ij}^B| \quad \geqslant \quad \sum_{j=1}^{n} v_{ij}^A \times v_{ij}^B$$

Thus, we conclude $D_{\min} \leqslant D_{Eros} \leqslant D_{\max}$. □

Hence, when performing kNN searches, $D_{\min}$ is used in the filter phase, and then D_{Eros} is employed in the refinement phase as described in the following section.

4.2. Muse: multilevel distance-based index structure 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 dataset. 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 4.2.1 and 4.2.2, respectively. Updating *Muse* is presented in Section 4.2.3, followed by the discussions on the selection of reference points for *Muse* in Section 4.2.4. The discussion on the number of levels is presented in Section 4.2.5.

4.2.1. Construction

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 thus 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 3. A distance metric $D_{|\cdot|,k}$ between two MTS items, $\mathbf{A}$ and $\mathbf{B}$, using the kth PC is defined as follows:

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

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 3 using matrix S in Fig. 3.

Algorithm 4 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 4 *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 $1 \leqslant l \leqslant n_R$ and $1 \leqslant j \leqslant z$;
1: for j=1 to $z$
2:   for i=1 to $N$
3:     E ← the $j$th PC of the $i$th MTS item in the dataset;
4:     $l$ ← the ID of the partition closest to E;
5:     $dist[i,j] \leftarrow D_{|\cdot|}(R_{jl}, E)$;
6:     $partitionID[i,j] \leftarrow l$;
7:     if $r_{jl} \leqslant dist[i,j]$
8:       $r_{jl} \leftarrow dist[i,j]$;
9:     end if
10:  end for
11: end for
12: Compute the weight vector $w$ using Algorithm 3;
```

Fig. 3 represents the process of *Muse* construction, 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 ($1 \leqslant j \leqslant z$), the jth PC group is divided into u_j partitions. The lth partition ($1 \leqslant l \leqslant u_j$) 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 dataset. Note that in Algorithm 4, it is assumed for simplicity that the number of reference points, i.e., n_R, is the same for all the levels.

4.2.2. kNN search using Muse

Given a query object Q, a set of objects $\mathcal{D}$, and an integer $k \geqslant 1$, k nearest neighbor (kNN) search is to find k objects in $\mathcal{D}$ which have the shortest distance from Q [11]. Range queries, which retrieve all the items within a fixed distance from a query object, would be simpler than kNN searches [10]. 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 the gray rectangle in Fig. 3. That is, we pre-compute all the distances between

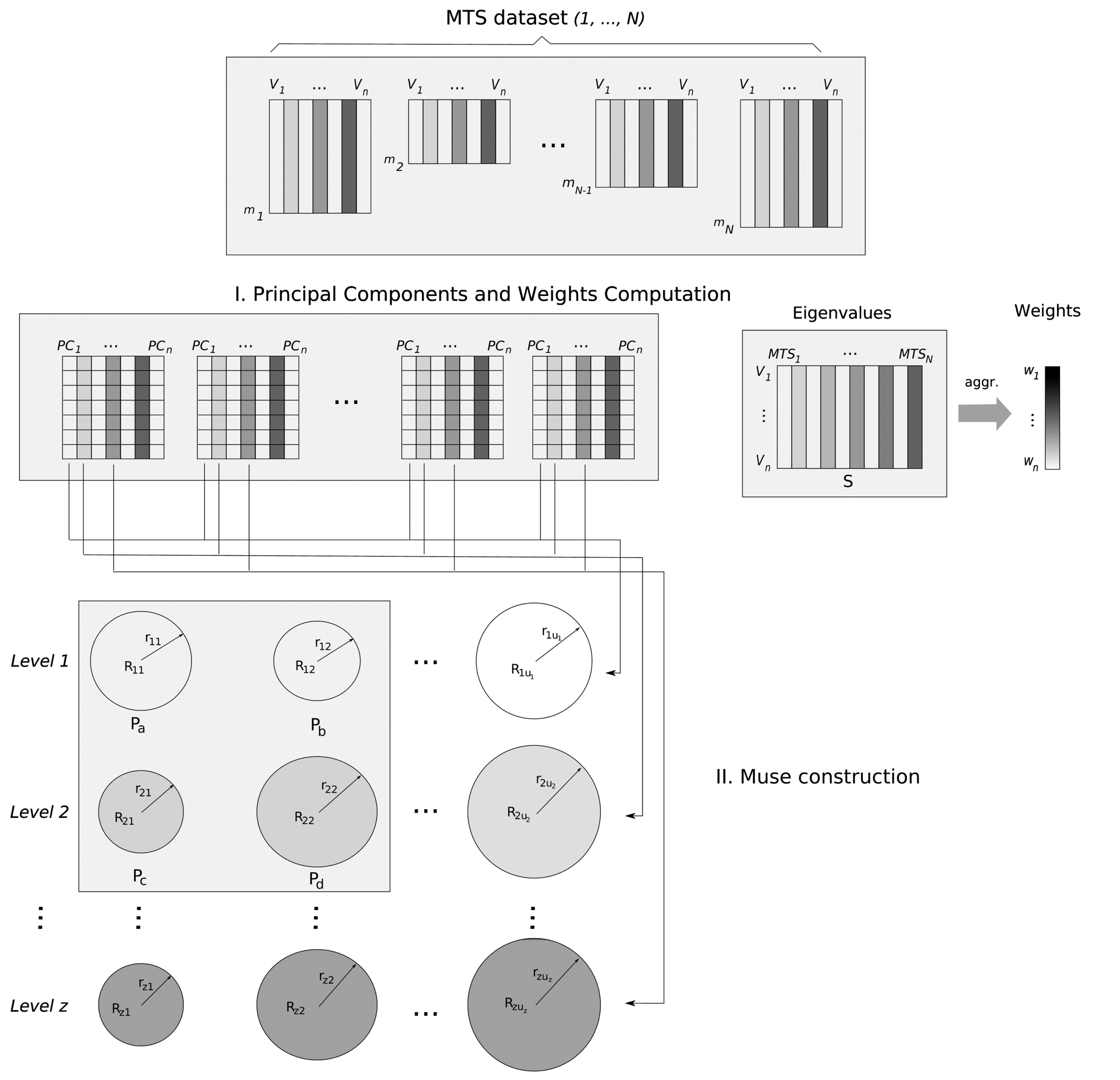


Fig. 3. The process of *Muse* construction.

the first 2 PCs of all the MTS items and their reference points. Using the first 2 PCs, we then have the following inequalities:

$$\begin{aligned} D_{\min}(\mathbf{A},\mathbf{Q},w) &\geqslant D^{1-2}_{\min}(\mathbf{A},\mathbf{Q},w) \\ &\geqslant D^{1-2}_{\min}(\mathbf{C},\mathbf{A},w) - D^{1-2}_{\min}(\mathbf{Q},\mathbf{C},w) \\ &\quad \text{(triangle inequality)} \end{aligned} \tag{6}$$

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:

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

and

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

$D_{|\cdot|,i}(\mathbf{C}, \mathbf{A})$ 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 4. The lower bound LB_{Naive} between two MTS items, $\mathbf{A}$ and $\mathbf{Q}$, using a z-level *Muse* is defined as follows:

$$LB_{Naive,z}(\mathbf{A}, \mathbf{Q}, w) = \sqrt{\sum_{i=1}^{z} w_i (D_{|\cdot|,i}(\mathbf{C}, \mathbf{A}))^2} - \sqrt{\sum_{i=1}^{z} w_i (D_{|\cdot|,i}(\mathbf{Q}, \mathbf{C}))^2} \tag{7}$$

where $\mathbf{C} = [c_1, \ldots, c_z]$ 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 Eq. (6), which can be expanded as follows:

$$D_{\min}^{1-2}(\mathbf{A}, \mathbf{Q}, w) = \sqrt{w_1 \sum_{j=1}^{n} (|v_{1j}^A| - |v_{1j}^Q|)^2 + w_2 \sum_{j=1}^{n} (|v_{2j}^A| - |v_{2j}^Q|)^2} \tag{8}$$

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

$$\sqrt{w_1 \sum_{j=1}^{n} (|v_{1j}^A| - |v_{1j}^Q|)^2} \geqslant \sqrt{w_1 \sum_{j=1}^{n} (|c_{1j}| - |v_{1j}^A|)^2} - \sqrt{w_1 \sum_{j=1}^{n} (|v_{1j}^Q| - |c_{1j}|)^2} \tag{9}$$

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

$$\sqrt{w_2 \sum_{j=1}^{n} (|v_{2j}^A| - |v_{2j}^Q|)^2} \geqslant \sqrt{w_2 \sum_{j=1}^{n} (|c_{2j}| - |v_{2j}^A|)^2} - \sqrt{w_2 \sum_{j=1}^{n} (|v_{2j}^Q| - |c_{2j}|)^2} \tag{10}$$

By squaring and summing up Eqs. (9) and (10) and then taking its square root, we obtain the lower bound of Eq. (8). The lower bound described above is defined as follows:

Definition 5. The lower bound LB_{Muse} between two MTS items, $\mathbf{A}$ and $\mathbf{Q}$, using a z-level *Muse* is defined as follows:

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

where $\mathbf{C} = [c_1,\ldots,c_z]$ 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, as in the following lemma. Hence, we use $LB_{Muse,z}$ when performing kNN searches with *Muse*.

Lemma 3. *For the lower bounds LB_{Naive} and LB_{Muse}, the following inequality holds*: $LB_{Muse,z}(\mathbf{A},\mathbf{Q},w) \geqslant LB_{Naive,z}(\mathbf{A},\mathbf{Q},w)$.

Proof 3. 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,

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

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

$$\begin{aligned} &LB_{Muse,z}(\mathbf{A},\mathbf{Q},w)^2 = \sum_{i=1}^{z}A_i^2 + \sum_{i=1}^{z}B_i^2 - 2\sum_{i=1}^{z}A_iB_i \\ \textit{and}\quad & \\ &LB_{Naive,z}(\mathbf{A},\mathbf{Q},w)^2 = \sum_{i=1}^{z}A_i^2 + \sum_{i=1}^{z}B_i^2 - 2\sqrt{\sum_{i=1}^{z}A_i^2}\sqrt{\sum_{i=1}^{z}B_i^2}\end{aligned} \tag{11}$$

Hence, the inequality between $LB_{Muse,z}$ and $LB_{Naive,z}$ depends on the last term of Eqs. (11) and (11). Recall that Hölder's Inequality [5] states that

$$\sum_{k=1}^{n}|a_kb_k| \leqslant \left(\sum_{k=1}^{n}|a_k|^p\right)^{1/p}\left(\sum_{k=1}^{n}|b_k|^q\right)^{1/q}$$

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

$$\sum_{i=1}^{z}|A_iB_i| \leqslant \left(\sum_{i=1}^{z}|A_i|^2\right)^{1/2}\left(\sum_{i=1}^{z}|B_i|^2\right)^{1/2}$$

Hence, by Eqs. (11), (11) and Hölder's Inequality[1] ,

$$LB_{Muse,z}(\mathbf{A},\mathbf{Q},w) \geqslant LB_{Naive,z}(\mathbf{A},\mathbf{Q},w) \qquad \square$$

A kNN search using *Muse* is performed as in Algorithm 5. 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 the gray rectangle in Fig. 3, which is a 2-level *Muse* with two reference

[1] Note that $A_i \geqslant 0$ and $B_i \geqslant 0$. Hence, $|A_i| = A_i$, $|B_i| = B_i$ and $|A_iB_i| = A_iB_i$.

points for each level. Given a query item **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 5 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 15–17. The data structures *knnDistance* and *knnID* are updated so that they are sorted in non-decreasing order of D_{Eros} in *knnDistance*.

Algorithm 5 *Muse* : kNN search

```
Require: the number of levels built for Muse, $z \ll n$, reference points $R_{jl}$ where $1 \leqslant l \leqslant n_R$ and $1 \leqslant j \leqslant z$,
    a query MTS item $Q$ and $k$;
1: for i=1 to $k$
2:    $knnDistance[i] \leftarrow \infty$;
3: end for
4: for t=1 to $z$
5:    for i=1 to $n_R$
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 $N$
11:    $\mathbf{A} \leftarrow$ the $i$th sorted MTS item;
12:    if $LB_{Muse,z}(\mathbf{A},\mathbf{Q}) \geqslant knnDistance[k]$
13:       continue;
14:    end if
15:    if $D_{Eros}(\mathbf{A},\mathbf{Q}) \leqslant knnDistance[k]$
16:       Update knnDistance and knnID using $D_{Eros}(\mathbf{A},\mathbf{Q})$ and id of A, respectively;
17:    end if
18: end for
```

4.2.3. 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 4.1.1, 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 dataset, the weights for *Eros* can thus be updated as in Algorithm 6. The MTS item can be inserted into *Muse* following

Lines 3–9 of Algorithm 4. When MTS items are removed from the dataset, *Muse* can be updated similarly.

Algorithm 6 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($S_{new}$);
```

4.2.4. Reference points

The choice of reference points affects the performance of the distance-based index structures [10] 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 Fig. 4 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 [10], this would in general 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 $argmax_j|v_{ij}^{\mathbf{A}}|$, 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.

4.2.5. 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. On 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 5. 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 dataset. 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 PCA to choose the number of principal components for dimension reduction [6]. The number of levels, z, is chosen such that the sum of the first z weights is greater than 0.99 for the first time.

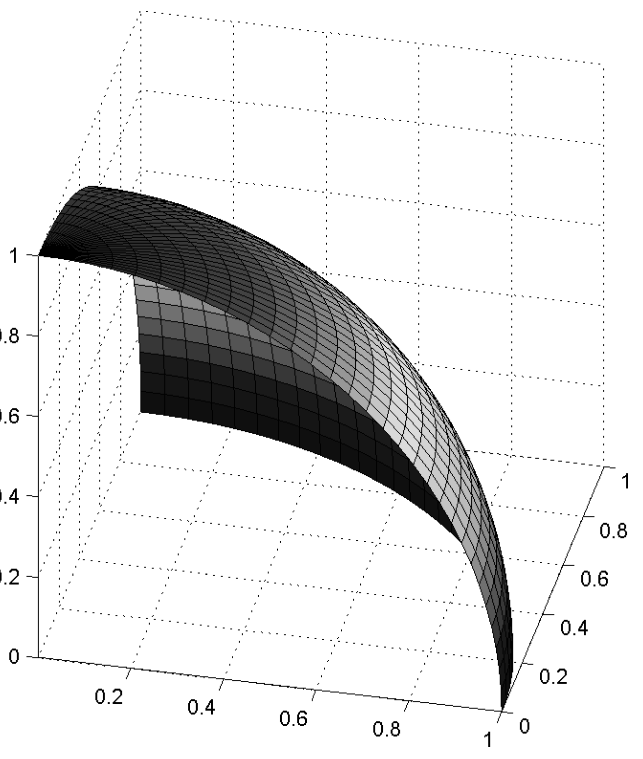


Fig. 4. The data space of the first PC group for *Muse* in 3D.

4.3. Ropes: recursive feature elimination on common principal components for Eros

Ropes extends **CL*e*V*er*** [12], and *recursively* selects a subset of variables/features for *Eros* utilizing both the common principal components and the *weights*. Fig. 5 depicts the processes of *Ropes*. As for **CL*e*V*er***, *Ropes* first computes the principal components and their variances for all the MTS items in a given data set. Using the principal components from all the MTS items, the Descriptive Common Principal Components (DCPCs) are computed (See Section 3.2), while using the variances, the *weights* are calculated as in Section 4.1.1.

The intuition to use the PCs and the DCPCs as a basis for *variable* subset selection is that they keep the most compact overview of the MTS items in a dramatically reduced space, while retaining both the correspondence to the original variables and the correlation among the variables. The DCPC loadings show how much each variable contributes to each of the DCPCs.

CL*e*V*er* also utilizes the PCs and DCPCs. The differences between **CL*e*V*er*** and *Ropes* are two folds:

- **CL*e*V*er*** utilizes the DCPCs obtained using all the n variables, while *Ropes* utilizes the DCPCs obtained using k variables, where $1 \leqslant k \leqslant n$. That is, at each recursion, one variable is eliminated, and at the next recursion, the DCPCs are obtained using the remaining variables.

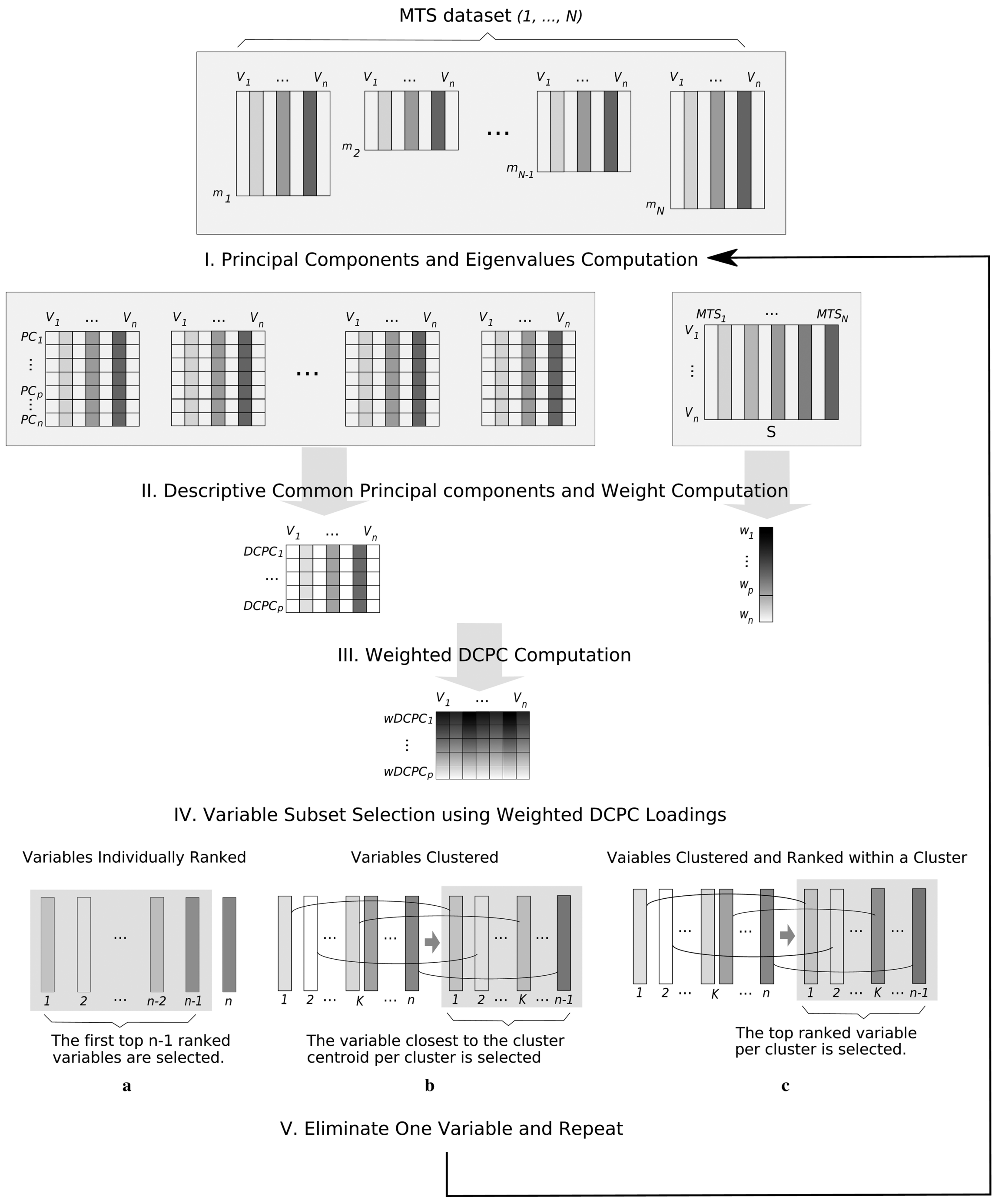

MTS dataset (1, ..., N)
V1 ... Vn
m1
m2
mN-1
mN
I. Principal Components and Eigenvalues Computation
PC1
PCp
PCn
MTS1 ... MTSN
S
II. Descriptive Common Principal components and Weight Computation
DCPC1
DCPCp
w1
wp
wn
III. Weighted DCPC Computation
wDCPC1
wDCPCp
IV. Variable Subset Selection using Weighted DCPC Loadings
Variables Individually Ranked
Variables Clustered
Vaiables Clustered and Ranked within a Cluster
1 2 ... n-2 n-1 n
1 2 ... K ... n
1 2 ... K ... n-1
The first top n-1 ranked variables are selected.
The variable closest to the cluster centroid per cluster is selected
The top ranked variable per cluster is selected.
a
b
c
V. Eliminate One Variable and Repeat

Fig. 5. The processes of *Ropes*.

– **CL***e***V***er* utilizes only the common principal components, while *Ropes* utilizes both the common principal components and their variances.

The intuition of recursively selecting variables for *Eros* is that after eliminating one variable, the PCs and DCPCs using the remaining variables would be different from the PCs and DCPCs using all the variables. Hence, the *contributions* of each variables to the PCs and DCPCs should change with one less variable. This observation leads us to *Ropes* which extends **CL***e***V***er*, and recursively selects a subset of variables. That is, at each recursion, the PCs and DCPCs from the remaining variables are computed again.

Furthermore, *Ropes* utilizes the *weighted* DCPCs, while **CL***e***V***er* employs the *un-weighted* DCPCs. The intuition is that the *loadings* of the PCs and DCPCs represent the contribution of each variables to the PCs and DCPCs. However, each PC and each DCPC represents different variances; the first few PCs and DCPCs represent most of the variances within the dataset. Hence, instead of giving the same weight to all the variables, more weights are given to the variables that contribute much to the first few PCs and DCPCs than the variables that do not contribute much to the first few PCs and DCPCs. Note that this intuition is also utilized for *Eros* when computing the similarity between two MTS items.

Algorithm 7 *Ropes-Rank*

```
Require: an MTS dataset, $K$ {the number of variables to select}, and $\delta$ {a predefined threshold}, $n$ {the
    number of variables in the MTS dataset}, $N$ {the number of MTS items in the dataset}
1: for $j = n$:-1:$K$
2:   $updatedMTS \leftarrow$ extract remaining $j$ variables from each MTS items;
3:   $DCPC \leftarrow computeDCPC(updatedMTS, \delta)$;
4:   $S \leftarrow$ extract all the eigenvalues from each of MTS items in $updatedMTS$; {the size of $S$ will be $j$ by $N$};
5:   $weight \leftarrow computeWeight(S)$;
6:   $weightedDCPC \leftarrow$ apply $weight$ to $DCPC$;
7:   for $i$=1 to $j$
8:     $score(i) \leftarrow$ norm($weightedDCPC$ loadings of the $i$th variable);
9:   end for
10:  $[IDX, score] \leftarrow$ sort $score$ in a non-increasing order;
11:  Remove one feature/variable with the least contribution to the common principal components.
12: end for
```

Extending each family member of **CL***e***V***er*, we propose three family members of *Ropes*; *Ropes*-Rank, *Ropes*-Cluster and *Ropes*-Hybrid, which are described in Algorithms 7, 8, and 9, respectively. All the *Ropes* variants utilize the weighted DCPCs, which are obtained by Lines 2–6 in all the algorithms. Using the weighted DCPCs, *Ropes*-Rank ranks the variables based on the contributions of each variable to the weighted DCPCs. The contribution of a variable is computed as follows. Let

Algorithm 8 *Ropes-Cluster*

```
Require: an MTS dataset, $K$ {the number of clusters}, and $\delta$ {a predefined threshold}, $n$ {the number of
    variables in the MTS dataset}, $N$ {the number of MTS items in the dataset}
 1: for $j = n - 1$:-1:$K$ do
 2:    $updatedMTS \leftarrow$ extract remaining $j$ variables from each MTS items;
 3:    $CPC \leftarrow computeDCPC(updatedMTS, \delta)$;
 4:    $S \leftarrow$ extract all the eigenvalues from each of MTS items in $updatedMTS$; {the size of $S$ will be $j$ by $N$};
 5:    $weight \leftarrow computeWeight(S)$;
 6:    $weightedDCPC \leftarrow$ apply $weight$ to $CPC$;
 7:    for $i$=1 to 20 do
 8:       $[index(i), centroid(i)] \leftarrow K$means($weightedDCPC$ loadings, $j$);
 9:    end for
10:    $[IDX, C] \leftarrow$ choose the clustering result whose sum of the distances between the centroid and its points
       are minimum;
11:    For the two variables that belong to one cluster, arbitrarily choose one;
12: end for
```

Algorithm 9 *Ropes-Hybrid*

```
Require: an MTS dataset, $K$ {the number of clusters}, and $\delta$ {a predefined threshold}, $n$ {the number of
    variables in the MTS dataset}, $N$ {the number of MTS items in the dataset}
 1: for $j = n - 1$:-1:$K$ do
 2:    $updatedMTS \leftarrow$ extract remaining $j$ variables from each MTS items;
 3:    $CPC \leftarrow computeDCPC(updatedMTS, \delta)$;
 4:    $S \leftarrow$ extract all the eigenvalues from each of MTS items in $updatedMTS$; {the size of $S$ will be $j$ by $N$};
 5:    $weight \leftarrow computeWeight(S)$;
 6:    $weightedDCPC \leftarrow$ apply $weight$ to $CPC$;
 7:    for $i$=1 to 20 do
 8:       $[index(i), centroid(i)] \leftarrow K$means($weightedDCPC$ loadings, $j$);
 9:    end for
10:    $[IDX, C] \leftarrow$ choose the clustering result whose sum of the distances between the centroid and its points
       are minimum;
11:    For the two variables that belong to one cluster, compute the contributions of them to the DCPCs and
       choose the one variable with the least contribution, and eliminate it;
12: end for
```

one of the variables be denoted as a vector v_1 consisting of $p(\leqslant n)$ weighted DCPC loadings, that is, $v_1 = (l_1, l_2, \ldots, l_p)$. Then, the score of v_1 is defined by

$$|v_1| = \sqrt{l_1^2 + l_2^2 + \cdots + l_p^2} \tag{12}$$

The intuition of using ℓ^2-norm to measure the contribution of each of variables is based on the observation that the stronger impact a variable has on the common principal components, the larger absolute DCPC loading value it has, as well as the further away it lies from the origin. Therefore, the norm describing the length of a vector in general can be used as a valid score in this context. In Lines 7–9, the score of each variable, i.e., the ℓ^2-norm of weighted DCPC loadings of each variable. Subsequently, the variable with the least contribution to the weighted DCPCs is identified and eliminated. This process is repeated until the desired number of variables are obtained.

Note that the DCPCs obtain the most common *subspace* to all the MTS items, and only p out of n PCs are employed when computing DCPCs. p should be heuristically determined, just as the number of PCs are heuristically determined when PCA is utilized for dimension reduction, e.g., based on the accumulated variances. In Algorithms 7, 8, and 9, the parameter δ will determine p. For details, please refer to [12,13].

Using the weighted DCPCs, *Ropes*-Cluster performs k-means clustering with k equal to the remaining number of variables minus one. As a result, only one cluster out of k clusters will have two variables in it. These two variables may be considered as having *similar* contributions to the weighted DCPCs. Out of these two variables, *Ropes*-Cluster arbitrarily chooses one. After eliminating one variable, this process repeats until the specified number of variables remain.

Ropes-Hybrid works similarly as *Ropes*-Cluster. That is, *Ropes*-Hybrid performs k-means clustering on the weighted DCPCs, and as a result, only one cluster out of k clusters will have two variables in it. These two variables are then ranked based on their contribution to the weighted DCPCs using Eq. (12), and *Ropes*-Hybrid eliminates the variable with the least contribution. This process repeats until the specified number of variables are retained.

5. Performance evaluation

In order to evaluate the performance of our proposed techniques, we performed experiments on three real-world datasets, and two synthetic datasets. In this section, we first describe the data sets used in the experiments, and the experiments methods followed by the results.

5.1. Datasets

The Australian Sign Language (AUSLAN) dataset [34] 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 size of the right eigenvector is 22×22 and the average length is around 60.

The HumanGait dataset from [36] has been used for identifying a person by gait recognition at a distance. In order to capture the gait data, a twelve-camera VICON system was utilized with 22 reflective markers attached to each subject. For each reflective marker, 3D position, i.e., x, y, and z, are acquired at 120Hz, generating 66 values at each timestamp. 15 subjects, which are the labels assigned to the dataset, participated in the experiments and were required to walk at four different speeds, nine times for each speed. The total number of data items is 540 ($15 \times 4 \times 9$) and the average length is 133.

The brain computer interface (BCI) dataset [37] was collected during the BCI experiment, where a subject had to perform imagined movements of either the left small finger or the tongue. The time series of the electrical brain activity was collected during these trials using 64 ECoG platinum electrodes. All recordings were gathered at 1000Hz. The total number of data items is 378 and the average length is 3000.

Table 3
Summary of datasets used in the experiments.

	AUSLAN	HumanGait	BCI	TRACE16	SYNTH
# of variables	22	66	64	4	22
average length	60	133	3000	250	60
# of labels	95	15	2	16	95
# of items per label	27	36	189	100	324–1080
total # of items	2565	540	378	1600	30–100 K

The transient classification benchmark (TRACE) datasets have been used in [38] for plant diagnostics. For the TRACE dataset with 16 classes (TRACE16), each class has 100 examples. There are 5 variables, out of which the first 4 variables are the four signals and the 5th one is the class label. The change of the class label from 0 to a class number (from 1 to 16) means the start of the transient. Using this information, we first located exactly where the transient starts and ends and removed those signals with class label 0. The size of the right eigenvector is 4×4 and the average length is 250. Note that in [38], the focus is to find the transition over the continuous data stream where the starting point is unknown, while the focus of this paper is to find k nearest neighbors given a query MTS item, assuming that the endpoints of all MTS items are accurately located. Finding k nearest neighbors over the continuous data stream is part of our future research directions (see Section 6).

In order to show the efficiency of *Muse*, we create a set of clustered synthetic dataset, *SYNTH*, as in [39], by adding a small random variation to each of the AUSLAN MTS item. 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.

Table 3 shows the summary of the datasets used in the experiments.

5.2. Methods

In order to validate our proposed similarity measure *Eros*, we performed leave-one-out kNN searches. Hence, we take one MTS item out as a query item **Q**, and construct *Muse* using the rest of the dataset, and perform kNN search varying k until we obtain at least 10 *relevant* items with the same label as **Q**. Recall that each dataset used in the experiments has more than 10 *relevant* items as shown in Table 3. For example, AUSLAN has 95 labels and each label has 27 items. The recall-precision graph [40] is then plotted, which has been frequently used to measure the performance of content-based image retrieval (CBIR) systems [41,42] as well as information retrieval (IR) systems.

For *Eros*, the weight vector w is computed using Algorithm 3 with and without normalizing the matrix S (See Section 4.1.1) excluding the eigenvalues of the query item. We employ three different aggregating functions, i.e., mean, min and max. Subsequently, the one with the best performance will be presented for *Eros*. We then compare the performance of *Eros* with those of 4 other distance measures, i.e., the Euclidean distance (ED), dynamic time warping (DTW), principal component analysis (PCA) similarity factor (S_{PCA}) and weighted sum SVD (WSSVD).

DTW is a technique for performing time-alignment [16]. Though DTW can be applied to 2 MTS items regardless of the items' lengths, the performance is shown to be the best when the ratio of the items' lengths is close to 1 [43], and the indexing technique for DTW is available only when the two items are of the same length [9,44]. Also, ED is not defined for 2 MTS items with different lengths. Hence, before applying ED and DTW, all the MTS items are linearly interpolated to be of identical length. We chose this length to be the average length for each dataset as in [43]. For DTW, the *MATLAB*^{TM} DTW code in [5] is used with slight modifications. A global limit on the maximum amount of warping Q is set to 10% of the length, and the distance between points, i.e., the local distance, is modified to be the square of the Euclidean distance, as in [9]. In addition, for ED and DTW, the experiments were performed with and without z-normalization which renders the data zero mean and unit variance. The better performance between the two are then presented. The ED and DTW with z-normalization are denoted as EDZ and DTWZ, respectively.

S_{PCA} is a similarity measure for MTS datasets [13,45]. It first finds k principal components (PCs), such that the corresponding k eigenvalues describe more than, e.g., 95% of the total variance. Only the k PCs are then used to compare the similarities between MTS items, and the eigenvalues are not utilized. For S_{PCA} similarity measure, we tried both 95% and 99% of the total variance for each dataset, which are denoted as $S_{PCA}95$ and $S_{PCA}99$, respectively.

In [46], we proposed *WSSVD*, which employs the eigenvectors and eigenvalues of MTS items to compute the similarities between items using the inner product of the eigenvectors with the eigenvalues as weights. Note that the eigenvectors utilized in *WSSVD* are not principal components, since SVD is applied on the transpose-multiplication of the data matrix, not on the covariance matrix. We tried both transpose-multiplication as in [46] and covariance matrix for *WSSVD*, denoted as *WSSVD* and $WSSVD_{COV}$, respectively.

For *Muse*, we performed experiments on the SYNTH dataset, and compared the performance of *Muse* to that of 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 dataset [9]. 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 dataset once the index structures are constructed. Page sizes of 8 and 16 KB are employed for M-tree. We only show the result of 16 KB M-tree, which is better than that of 8 KB M-tree. For the reference points of *Muse*, as described in Section 4.2.4, the edge points where the hypersphere meets each axis are chosen at each level.

For *Ropes*, we performed experiments on two real-world datasets, the AUSLAN dataset and the BCI dataset, and compared the performances of *Ropes* to those of others in terms of classification accuracy, using the 1 Nearest Neighbor Classifier [47]. That is, we take one MTS item out as a query item Q, and perform the feature subset selection in order to choose a specified number of variables. Subsequently, using *Eros* with the selected variables, we retrieve the nearest neighbor of Q in the dataset. We consequently check if the label of the query item Q is the same as that of the retrieved item, and repeat this for all the items in the dataset, while varying the number of selected features from 3 to $n-1$, where n is the number of variables in the dataset. In order to compute the weight vector, we employ three different aggregating functions, i.e., mean, min and max, as for *Eros*. Subsequently, the one with the best performance will be presented for *Ropes*. For δ when computing DCPCs, 7 different values are employed, i.e., 70%, 75%, 80%, 85%, 90%, 95%, and 99% of the total variance, and the one which produces the best classification accuracy is presented. We consequently compare *Ropes* with its counterpart for **CL**e**V**er. That is, *Ropes-Rank* is compared with **CL**e**V***er-Rank*, *Ropes-Cluster* with **CL**e**V***er-Cluster*, and *Ropes-Hybrid* with **CL**e**V***er-Hybrid*.

Eros and *Ropes* are implemented in Matlab and *Muse* is implemented in both C++ and Matlab. The experiments are performed on a Pentium IV 3.2 GHz machine with 3 GB of main memory, unless specified otherwise.

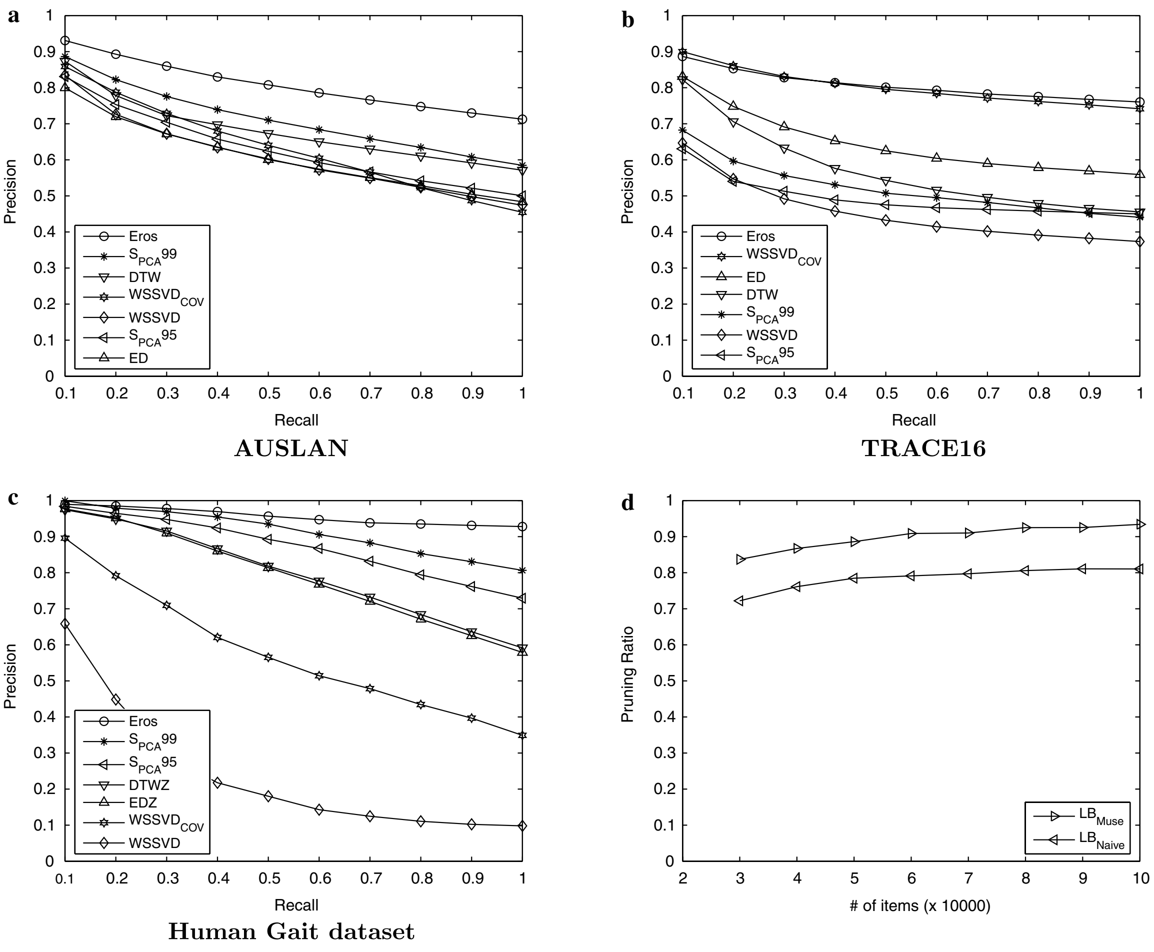
5.3. Results

The experimental results for *Eros* is first presented. Figs. 6a–c are the precision/recall graphs on the AUSLAN, TRACE16 and HumanGait datasets, respectively. Fig. 6a shows that *Eros* gives the best recall-precision ratio for the AUSLAN dataset. Poor performances of ED and DTW may indicate that there are correlations among variables, which are not considered by those two similarity measures. Note also that the performances of EDZ and DTWZ are worse than those of ED and DTW. There may be a couple of reasons for this. One of them would be that, as observed in [48], the normalization does not always lead to the optimal results when it comes to similarity. The other may be that some of the items in the dataset contains very small standard deviations close to 0, or sometimes even 0, in which case the normalization may result in distorting the data. It should be further investigated how the normalization is to be performed for a dataset which may contain small standard deviation. For *Eros*, S_{PCA} and $WSSVD_{COV}$, the covariance matrices are used. S_{PCA} performs similarly as *Eros*. However, the full sequential scan should be performed for S_{PCA}, since, to the best of our knowledge, there is no indexing technique for S_{PCA}.

Fig. 6b shows that for TRACE16, *Eros* and $WSSVD_{COV}$ perform similarly. However, as is the case for S_{PCA}, $WSSVD_{COV}$ requires full scan of the whole data for similarity searches. *Eros* outperforms all the other similarity measures. When recall is equal to 1.0, *Eros* outperforms ED, DTW, S_{PCA} and WSSVD by more than 33%, 64%, 69%, and 100%, respectively. For TRACE16, ED and DTW again outperform EDZ and DTWZ, and the results of ED and DTW are presented. Notice that the performance of S_{PCA} is worse than ED and DTW. The results show that for TRACE16, the principal components by themselves do not distinctively represent the characteristics of MTS items. Combined with the eigenvalues, i.e., the variances that the principal components represent, however, the principal components well distinguish MTS items, as can be seen by the good performances of *Eros* and $WSSVD_{COV}$.

Fig. 6c represents similar results for the HumanGait dataset. *Eros* and S_{PCA} yield similar performances when the recall is less than 0.5. However, as the recall increases, the performance of S_{PCA} degrades faster than *Eros*. Note also that for *Eros*, an indexing structure can be utilized for efficient retrieval, while S_{PCA} cannot. For HumanGait, EDZ and DTWZ perform much better than ED and DTW. *Eros* and S_{PCA} also produces better results with the correlation coefficient matrices, when obtaining the principal components.

Intuitively, S_{PCA} considers only the angles between the corresponding principal components, *WSSVD* in [46] and $WSSVD_{COV}$ consider both the angles between the corresponding principal components and the variance for each principal component and *Eros* considers both the *acute* angles and



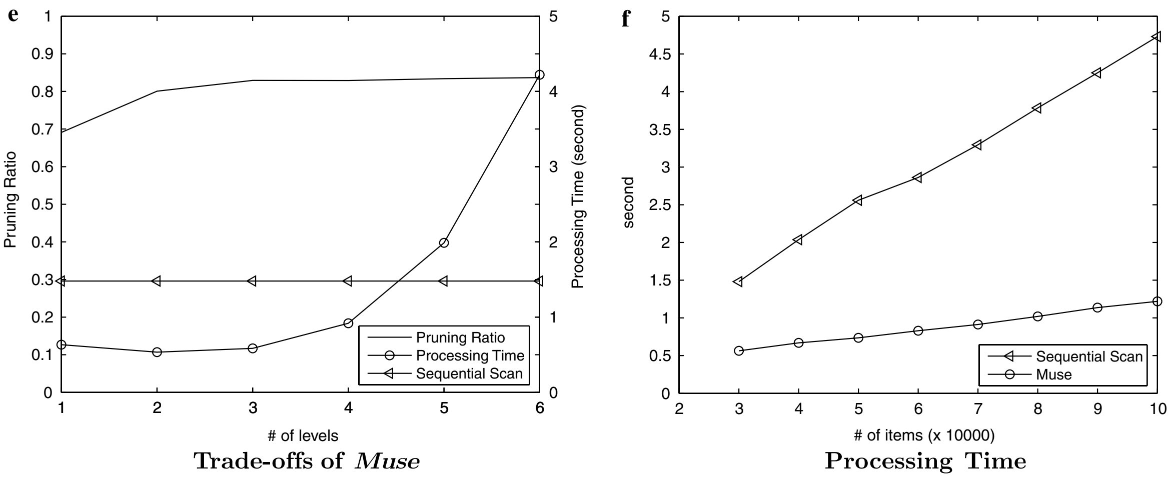


Fig. 6. Experimental results for *Eros* and *Muse*.

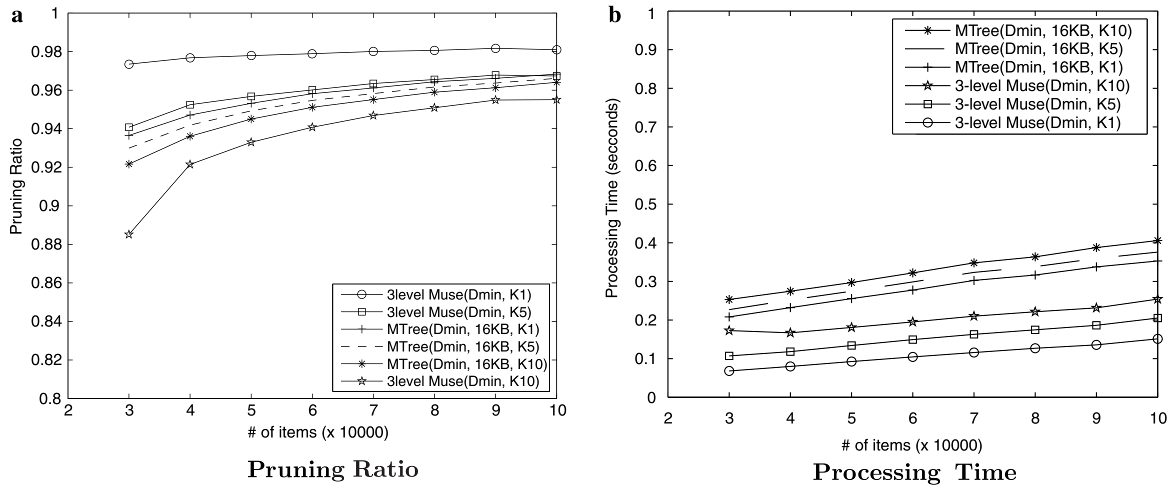


Fig. 7. For the SYNTH dataset, (a) shows the pruning ratios of 3-level *Muse* utilizing 66 dimensions and M-tree with page size of 16 KB employing 484 dimensions, and (b) depicts the processing times of both methods. The distance metric D_{min} is employed for both *Muse* and M-tree for the sake of comparison.

the variances for the principal components (see Section 4.1), when comparing two MTS items. Fig. 6a shows that for the AUSLAN dataset, the acute angles and the variances should be considered. Fig. 6b indicates that for the TRACE16 dataset, it does not matter whether the angles are acute or blunt, but the variances should be taken into account. Fig. 6c represents that for the HumanGait dataset, the acute angles should be considered, but maybe not the variances. These results show that for S_{PCA} and *WSSVD*, the characteristics of the dataset should be examined first, so that it can be decided whether S_{PCA} or *WSSVD* can be applied to the dataset. However, with *Eros* the prior examination of the dataset's characteristics may not be required, as shown in Figs. 6a–c.

Subsequently, we represent the experimental results for *Muse*. Fig. 6d depicts the pruning ratios of LB_{Muse} and LB_{Naive} using the SYNTH dataset. This figure confirms Lemma 3 as LB_{Muse} yields tighter lower bound than LB_{Naive} resulting in higher pruning ratio than LB_{Naive} by more than 10%.

Fig. 6e shows the tradeoff among the number of levels and the pruning ratio and the processing time using the SYNTH dataset with 30 K items. That is, 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. 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 5 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.

Fig. 6f 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. Three-level *Muse*, which utilizes the first 3 PCs, has 66 dimensions. Hence, Fig. 6f also 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 [39].

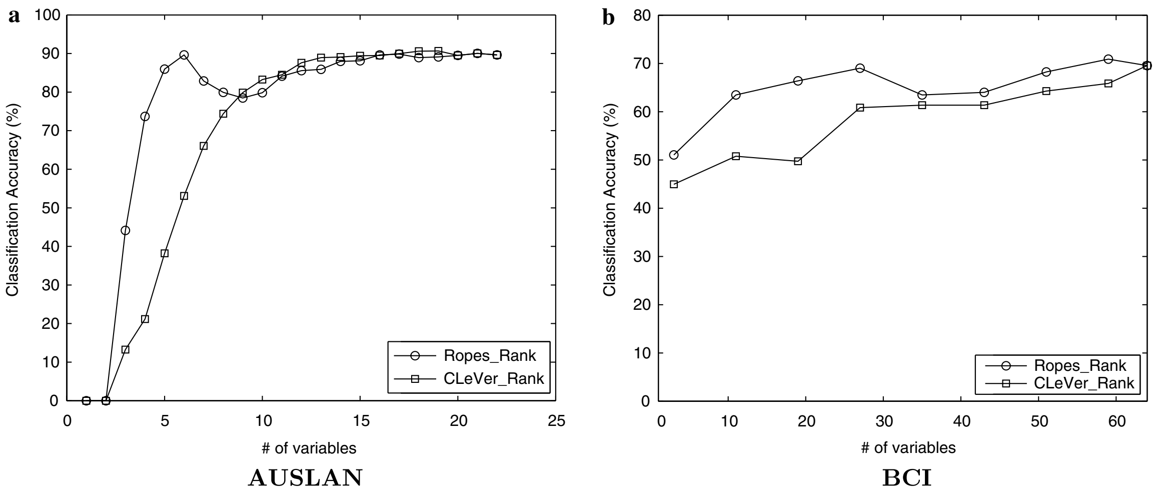


Fig. 8. *Ropes-Rank* vs. **CL***e***V***er-Rank* for the (a) AUSLAN and (b) BCI datasets.

The pruning ratios of M-tree and 3-level *Muse* when D_{min} distance metric is employed are shown in Fig. 7a, 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 dataset, 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), Fig. 7a 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 [49]. Fig. 7b shows that 3-level *Muse* outperforms M-tree in processing time, which may reconfirm the aforementioned limitation of M-tree.

Finally, we show the experimental results for *Ropes*. Figs. 8a and b depict the comparison between *Ropes-Rank* and **CL***e***V***er-Rank* for the AUSLAN dataset and the BCI dataset, respectively. In both cases, *Ropes* performs better than **CL***e***V***er* when fewer number of variables are employed for *Eros*. For example, for the AUSLAN dataset, when 6 variables are selected, the classification accuracy of *Ropes* is around 90%, while that of **CL***e***V***er* is less than 60%. Similarly, for the BCI dataset, when 19 variables out of 66 are selected, the classification accuracy of *Ropes* is around 75%, while that of **CL***e***V***er* is less than 50%. As the number of selected variables increases, however, the performances of both *Ropes* and **CL***e***V***er* are more or less the same. This trend is rather what is expected when the feature subset selection is performed. That is, when you have a small number of variables, these variables may be considered as non-redundant and relevant variables. As you increase the number of variables, redundant and/or non-relevant variables may be included, which

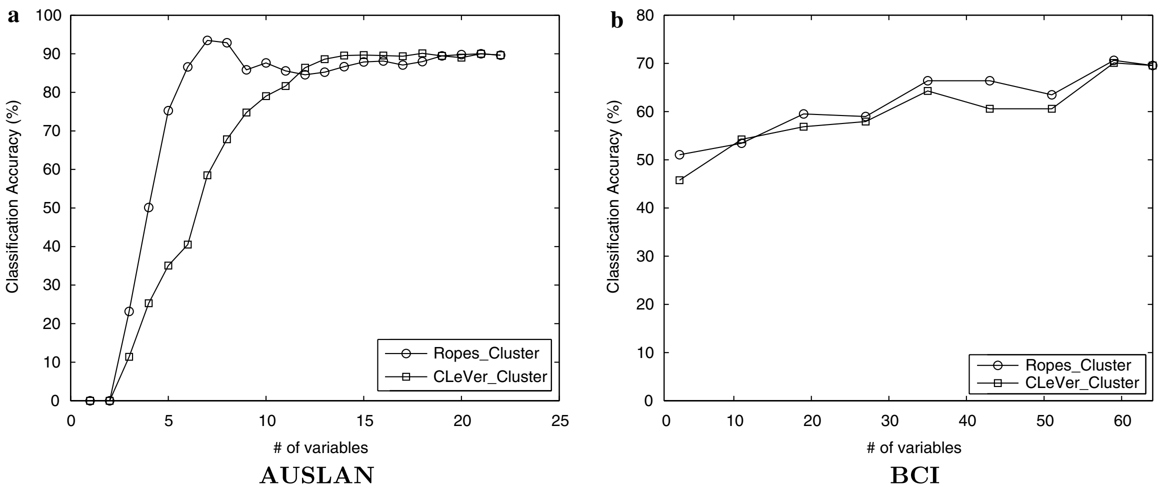
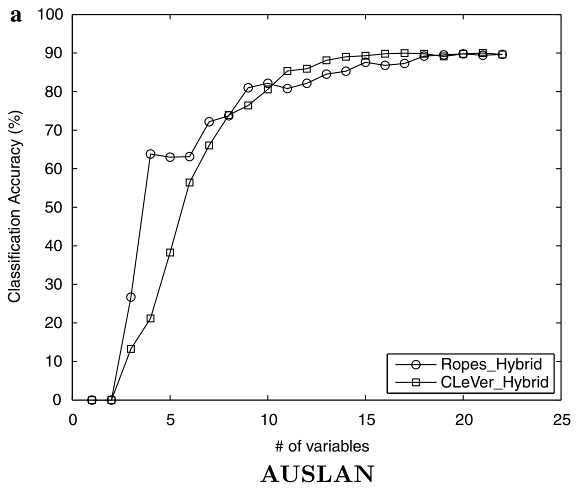


Fig. 9. *Ropes-Cluster* vs. **CL**e**V**er-*Cluster* for the (a) AUSLAN and (b) BCI datasets.

would degrade the performance of, e.g., the classifier. Intuitively, *Ropes-Rank* is similar to recursive feature elimination (RFE) [28] in that both techniques rank the features based on some criteria, eliminate the variable with the lowest rank, and repeat the process. One of the differences between *Ropes-Rank* and RFE is that the former is an *unsupervised* feature subset selection technique, while the latter is a *supervised* feature subset selection technique. That is, the former does not require the label information in the dataset, while the latter does require the label information.

Figs. 9a and b compare the performance of *Ropes-Cluster* and that of **CL***e***V***er-Cluster* for the AUSLAN and the BCI datasets, respectively. For the AUSLAN dataset, as in Fig. 9a *Ropes* performs better than **CL***e***V***er* when the number of selected variables is small, and both *Ropes* and **CL***e***V***er* perform similarly as the number of variables increases. However, for the BCI dataset, as shown in Fig. 9b, both *Ropes* and **CL***e***V***er* perform similarly. One of the reasons for this trend would be that, for the BCI dataset, the variables with similar contribution to the DCPCs may collectively play an important role when it comes to classification. Recall that the intuition of both *Ropes-Cluster* and **CL***e***V***er-Cluster* is that the variables with similar contribution to DCPCs which belong to one cluster would have redundant information, and we choose one variable for each cluster intending to eliminate the redundant information. For the AUSLAN dataset, this intuition seems to be working. However, for the BCI dataset, this intuition does not quite work and *Ropes-Rank* performs better than *Ropes-Cluster*, and *Ropes-Cluster* performs similarly as **CL***e***V***er-Cluster*.

Figs. 10a and b present the performance of *Ropes-Hybrid* and that of **CL***e***V***er-Hybrid* on the AUSLAN and the BCI datasets, respectively. For the AUSLAN dataset, when 4 variables are selected, *Ropes-Hybrid* achieves more than 60% of classification accuracy, while **CL***e***V***er-Hybrid* produces around 20% of classification accuracy. However, the performances of both techniques are more or less the same with more variables. For the BCI dataset as well, *Ropes-Hybrid* performs slightly better than **CL***e***V***er-Hybrid* with a small number of variables. As the number of selected variables increases, the performances of both *Ropes-Hybrid* and **CL***e***V***er-Hybrid* are more or less the same. This trend is similar to the case for *Ropes-Cluster* and **CL***e***V***er-Cluster*, and the performance of *Ropes-Hybrid* is rather similar to that of **CL***e***V***er-Hybrid.*



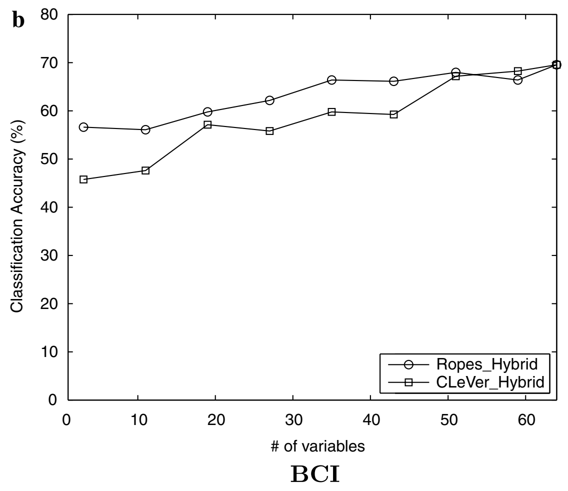


Fig. 10. *Ropes-Hybrid* vs. **CL**e**V**er-*Hybrid* for the (a) AUSLAN and (b) BCI datasets.

6. Conclusions and future work

In order to perform *k*NN searches efficiently for multivariate time series datasets, we proposed three techniques; *Eros* the similarity measure, *Muse* the index structure, and *Ropes* the feature subset selection technique.

In order to compute the similarity between two MTS items, *Eros* (*E*xtended F*ro*beniu*s* norm) compares the similarity between the corresponding principal components of the MTS items using the associated eigenvalues as weights. Experimentally, we show the validity of our proposed measure. In precision/recall, *Eros* outperforms ED, WSSVD, DTW and S_{PCA} by a minimum of 6.5% when recall is 0.1 and as much as 100% when recall is 1. In elapsed time, *Eros* also benefits from using the reduced dimension representations of MTS items and hence takes less time to compute than dynamic time warping.

Muse (*Mu*lti-level distance-based index *s*tructure for *Eros*) builds a number of *levels*, each of which is a distance-based index structure corresponding to one *principal component (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 the *z* levels with the weights to compute the lower bounds between the query item and the items in the dataset, and filter out those that should not 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.

In order to represent an MTS item more compactly by removing redundant and/or irrelevant features, as well as to improve the performance of *Eros*, we proposed a family of feature subset selection techniques for *Eros*, termed *Ropes* (recursive feature elimination on common principal components for Eros). *Ropes* extends **CL**e**V**er, and recursively eliminates features one by one using both the DCPCs and the weights. Experimental results show that *Ropes* outperforms **CL**e**V**er in terms of classification accuracy.

We intend to extend this work in three directions. First, we plan to extend our technique to continuous data streams generating the result as soon as new data arrives. Second, in this paper, we considered only the whole matching queries. Sub-sequence matching queries for MTS datasets is one of the interesting future directions of this work. Instead of using sliding window-based approaches for sub-sequence matching, we plan to use a change detection technique, such as in [50], to segment the whole sequence into multiple sub-sequences. This way, the time complexity for the sub-sequence matching would be much lower than that of the sliding window-based approaches. Finally, *Ropes* is an unsupervised technique, which does not utilize the label information of a dataset. A supervised feature subset selection technique would further improve the performance of *Eros*.

Acknowledgments

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 Google and 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] A. Tucker, S. Swift, X. Liu, Variable grouping in multivariate time series via correlation, IEEE Trans. Syst. Man Cybern. B 31 (2) (2001) 235–245.
[2] C. Shahabi, AIMS: an immersidata management system, in: VLDB CIDR, 2003.
[3] X.L. Zhang, H. Begleiter, B. Porjesz, W. Wang, A. Litke, Event related potentials during object recognition tasks, Brain Res. Bull. 38 (6) (1995) 531–538.
[4] C. Goutte, P. Toft, E. Rostrup, F.A. Nielsen, L.K. Hansen, On clustering fMRI time series, NeuroImage (9) (1999) 298–310.
[5] T.K. Moon, W.C. Stirling, Mathematical Methods and Algorithms for Signal Processing, Prentice Hall, Englewood Cliffs, NJ, 2000.
[6] I.T. Jolliffe, Principal Component Analysis, Springer, Berlin, 2002.
[7] C. Bohm, S. Berchtold, D.A. Keim, Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases, ACM Comput. Surveys 33 (3) (2001).
[8] E. Chavez, G. Navarro, R. Baeza-Yates, J.L. Marroqu, Searching in metric spaces, ACM Comput. Surveys 33 (3) (2001).
[9] E. Keogh, Exact indexing of dynamic time warping, in: Proc. of the VLDB Conference, 2002.
[10] C. Yu, B.C. Ooi, K.-L. Tan, H.V. Jagadish, Indexing the distance: an efficient method to KNN processing, in: Proc. of the VLDB Conference, 2001.
[11] P. Ciaccia, M. Patella, P. Zezula, M-tree: an efficient access method for similarity search in metric spaces, in: Proc. of the VLDB Conference, 1997.
[12] H. Yoon, K. Yang, C. Shahabi, Feature subset selection and feature ranking for multivariate time series, IEEE Trans. Knowl. Data Eng. 17 (9) (2005) 1186–1198.
[13] W. Krzanowski, Between-groups comparison of principal components, J. Am. Stat. Assoc. 74 (367) (1979).
[14] K. Yang, C. Shahabi, A pca-based similarity measure for multivariate time series, in: ACM MMDB, 2004.
[15] K. Yang, C. Shahabi, A multilevel distance-based index structure for multivariate time series, in: IEEE TIME, 2005.

[16] H. Sakoe, S. Chiba, Dynamic programming algorithm optimization for spoken word recognition, IEEE Trans. Acoust. Speech Signal Process. 26 (1) (1978).
[17] L. Chen, M.T. Ö;zsu, V. Oria, Robust and fast similarity search for moving object trajectories, in: Proc. of the ACM SIGMOD Conference, New York, NY, USA, 2005.
[18] M. Vlachos, G. Kollios, D. Gunopulos, Discovering similar multidimensional trajectories, in: Proc. of the Int. Conf. on Data Engineering, 2002.
[19] C. Yu, High-Dimensional Indexing : Transformational Approaches to High-Dimensional Range and Similarity Searches, ser. Lecture Notes in Computer Science, vol. 2341, Springer-Verlag, 2003.
[20] Y. Sakurai, M. Yoshikawa, S. Uemura, H. Kojima, The a-tree: an index structure for high-dimensional spaces using relative approximation, in: Proc. of the VLDB Conference, 2000.
[21] Y. Cai, R. Ng, Indexing spatio-temporal trajectories with chebyshev polynomials, in: Proc. of the ACM SIGMOD Conference, New York, NY, USA, 2004.
[22] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, E. Keogh, Indexing multi-dimensional time-series with support for multiple distance measures, in: Proc. of the ACM SIGKDD Conference, 2003.
[23] Y. Sakurai, M. Yoshikawa, R. Kataoka, S. Uemura, Similarity search for adaptive ellipsoid queries using spatial transformation, in: Proc. of the VLDB Conference, 2001.
[24] P. Ciaccia, M. Patella, Searching in metric spaces with user-defined and approximate distances, ACM Trans. Database Syst. 27 (4) (2002) 398–437.
[25] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Springer, Berlin, 2000, ch. 3, p. 121.
[26] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, J. Mach. Learn. Res. 3 (2003) 1157–1182.
[27] R. Kohavi, G.H. John, Wrappers for feature subset selection, Artif. Intell. 97 (1-2) (1997) 273–324.
[28] I. Guyon, J. Weston, S. Barnhill, V. Vapnik, Gene selection for cancer classification using support vector machines, Mach. Learn. 46 (1- 3) (2002) 389–422.
[29] T.N. Lal, M. Schröder, T. Hinterberger, J. Weston, M. Bogdan, N. Birbaumer, B. Schölkopf, Support vector channel selection in BCI, IEEE Trans. Biomed. Eng. 51 (6) (2004).
[30] W. Krzanowski, Orthogonal components for grouped data: review and applications, Stat. Trans. 5 (5) (2002) 759–777.
[31] B.N. Flury, Common principal components in *k* groups, J. Am. Stat. Assoc. 79 (388) (1984) 892–898.
[32] J.R. Schott, Some tests for common principal components in several groups, Biometrika 78 (1991) 771–777.
[33] H. Liu, L. Yu, M. Dash, H. Motoda, Active feature selection using classes, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2003.
[34] M.W. Kadous, Temporal classification: extending the classification paradigm to multivariate time series," Ph.D. dissertation, University of New South Wales, 2002.
[35] J.E. Jackson, A User's Guide to Principal Components, Wiley Interscience, 1991.
[36] R. Tanawongsuwan, A. Bobick, Performance analysis of time-distance gait parameters under different speeds, in: 4th Int. Conf. on Audio- and Video Based Biometric Person Authentication, Guildford, UK, June 2003..
[37] T.N. Lal, T. Hinterberger, G. Widman, M. Schröder, N.J. Hill, W. Rosenstiel, C.E. Elger, B. Schölkopf, N. Birbaumer, Methods towards invasive human brain computer interfaces, 2005, pp. 737–744.
[38] D. Roverso, Plant diagnostics by transient classification: the aladdin approach, Intl. J. Intell. Syst. (2002).
[39] T. Bozkaya, M. Ozsoyoglu, Indexing large metric spaces for similarity search queries, ACM Trans. Database Syst. 24 (3) (1999).
[40] W.B. Frakes, R. Baeza-Yates, Information Retrieval: Data Structures and Algorithms, Prentice-Hall, Englewood Cliffs, NJ, 1992.
[41] R.O. Stehling, M.A. Nascimento, A.X. Falcão, A compact and efficient image retrieval approach based on border/interior pixel classification, in: Proc. of the ACM-CIKM Conference, 2002.
[42] D.-H. Kim, C.-W. Chung, Qcluster: relevance feedback using adaptive clustering for content-based image retrieval, in: Proc. of the ACM SIGMOD Conference, 2003.
[43] C. Myers, L.R. Rabiner, A.E. Rosenberg, Performance tradeoffs in dynamic time warping algorithms for isolated word recognition, IEEE Trans. Acoust. Speech Signal Process. 28 (6) (1980).
[44] T.M. Rath, R. Manmatha, Lower-bounding of dynamic time warping distances for multivariate time series, 2002, TR MM-40, University of Massachusetts Amherst.

[45] A. Singhal, D. Seborg, Clustering of multivariate time-series data, in: Proc. of the American Control Conference, vol. 5, 2002.
[46] C. Shahabi, 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.
[47] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, Second ed., Wiley Interscience, 2001.
[48] D.Q. Goldin, T.D. Millstein, A. Kutlu, Bounded similarity querying for time-series data, Inf. Comput. 194 (2) (2004) 203–241.
[49] L. Chen, R. Ng, On the marriage of lp-norms and edit distance, in: Proc. of the VLDB Conference, 2004.
[50] D. Kifer, S. Ben-David, J. Gehrke, Detecting change in data streams, in: Proc. of the VLDB Conference, 2004.