Knowledge-Based Systems 35 (2012) 361–368

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Short communication

# Selective Subsequence Time Series clustering

Sura Rodpongpun, Vit Niennattrakul, Chotirat Ann Ratanamahatana *

*Department of Computer Engineering, Chulalongkorn University, 254 Phayathai Road, Pathumwan, Bangkok 10330, Thailand*

ARTICLE INFO

*Article history:*
Received 11 December 2011
Received in revised form 20 April 2012
Accepted 22 April 2012
Available online 27 April 2012

*Keywords:*
Time series
Subsequence clustering
STS clustering
Meaningful time series clustering
Time series mining

ABSTRACT

Subsequence Time Series (STS) Clustering is a time series mining task used to discover clusters of interesting subsequences in time series data. Many research works had used this algorithm as a subroutine in rule discovery, indexing, classification and anomaly detection. Unfortunately, recent work has demonstrated that almost all of the STS clustering algorithms give meaningless results, as their outputs are always produced in sine wave form, and do not associate with actual patterns of the input data. Consequently, algorithms that use the results from the STS clustering as their input will fail to produce its meaningful output. In this work, we propose a new STS clustering framework for time series data called Selective Subsequence Time Series (SSTS) clustering which provides meaningful results by using an idea of data encoding to cluster only essential subsequences. Furthermore, our algorithm also automatically determines an appropriate number of clusters without user's intervention.



## 1. Introduction

Time series clustering [2,6,15,13,24] is one of the most popular tasks in time series data mining community [5,16,18,25,26,20]. Most algorithms generally perform whole time series clustering [24,15]. More specifically, those algorithms try to group individual time series instances to a set of clusters. On the other hand, Subsequence Time Series (STS) clustering [2,13,6], which will be considered in this work, has been gaining more popularity. STS clustering algorithm discovers clusters of interesting subsequences within a single time series data stream. This algorithm can be used as a subroutine of other data mining tasks, such as rule discovery [28,4,12], indexing [17], classification [3], and anomaly detection [28].

Unfortunately, it has been demonstrated that these STS clustering algorithms produce meaningless results [13]. Because most algorithms use a sliding window to extract subsequences and try to cluster them all, the resulting cluster centers turn out to be some forms of sine waves regardless of the original shape of the patterns in the input data. Therefore, every algorithm that uses this meaningless STS clustering as a subroutine will in turn fail to produce meaningful results as well.

The cause of producing sine waves as outputs has been analyzed by many authors [8,11,21]. They have shown that clustering of every single subsequence leads to meaningless outputs. In fact, some subsequences such as noises or outliers should not be clustered. For instance, consider a speech recognition problem, non-speech segments in a source data has to be determined and removed. Similarly, in sign language recognition, transitions between consecutive signs, called movement epenthesis [29,22], has to be discarded. We show in Fig. 1 that meaningful STS clustering can be achieved by ignoring some subsequences. We use an ECG data from [9] to demonstrate that it is not necessary to include some trivial subsequences in a cluster.

In this work, we propose a new STS clustering framework called Selective Subsequence Time Series (SSTS) clustering, which performs subsequence clustering to produce meaningful cluster centers. We will show that the cluster centers from our algorithm do represent the actual patterns within the input data, instead of producing sine waves. In essence, we adopt an idea of data encoding to determine proper clusters by clustering only important subsequences. Some subsequences that are not significant will be discarded. On the other hand, because it is hard to exactly specify window size of the subsequences, our approach allows window size to be varied. The appropriate sliding window size, $w$, depends on types of data and application requirement. In practice, a user only need to roughly estimate a value of $w$, and then our algorithm will determine an appropriate value. However, due to the flexible window length $w$, the members of clusters could be of different lengths. Moreover, different types of data need different predefined number of clusters $k$, so our algorithm automatically determines an appropriate number of clusters depending on characteristics of input data.

The rest of this paper is organized as follows. In Section 2, we provide review and discussion of some related works. Section 3 offers background knowledge used in this work. Detail of our approach are described in Section 4. Section 5 shows essential experiments in various domains including real and synthetic data. Finally, conclusion and discussion about future research direction are discussed in Section 6.

* Corresponding author. Tel.: +66 8 9499 9400; fax: +66 2 218 6955.
*E-mail addresses:* g53srd@cp.eng.chula.ac.th (S. Rodpongpun), g49vnn@cp.eng.chula.ac.th (V. Niennattrakul), ann@cp.eng.chula.ac.th (C.A. Ratanamahatana).

0950-7051/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.knosys.2012.04.022

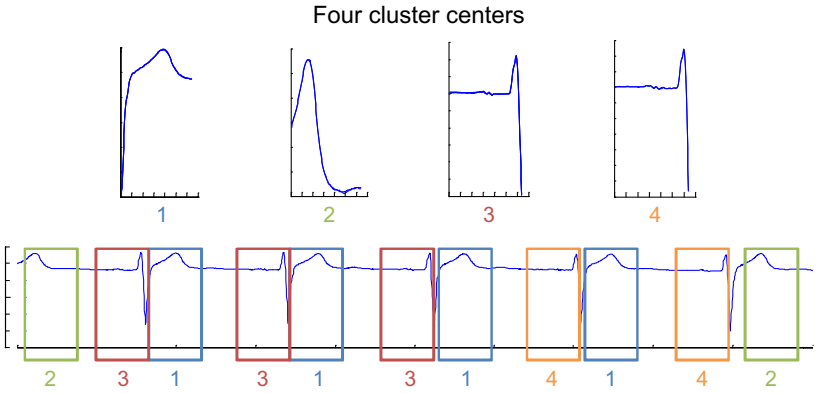


**Fig. 1.** Meaningful STS clustering achieved by ignoring some subsequences.

## 2. Related work

In time series mining communities, clustering task has always been receiving much attention. However, most works focus on clustering individual time series whereas clustering subsequences of a single long time series, the problem considered in this work, is not well resolved. The most referenced work is the one that uses STS clustering as a subroutine for rule discovery [4]. Until recently, it has been discovered that the STS clustering used in that work is meaningless [13]. Because they try to cluster every single extracted subsequences, their output turns out to always be in sine waves regardless of what the input sequence looks like. This problem has then been analyzed by many authors [8,11,21]. It now comes to a conclusion that clustering every subsequence extracted by moving a fix sliding window leads the cluster centers to converge to sine waves. Many authors have proposed algorithms to solve this problem [2,6], but those algorithms require a fixed value of number of clusters $k$ and length of subsequences $w$, which are not suitable in real world problems.

## 3. Background

In this section, we provide definitions and background knowledge used in this work.

### 3.1. Definition and notation

**Definition 1.** A *time series* $T$ of size $m$ is an ordered sequence of real value data, where $T = (t_1, t_2, \ldots, t_m)$.

Our approach takes a sequence of time series $T$ as an input and extracts it to a set of *subsequences*.

**Definition 2.** A *subsequence* of length $n$ of time series $T$ is $T_{i,n} = (t_i, t_{i+1}, \ldots, t_{i+n-1})$, where $1 \leqslant i \leqslant m - n + 1$, $n < m$.

As mentioned in Section 1, we adopt an idea of simple data encryption to determine proper clusters by emulating the clusters as a *codebook*.

**Definition 3.** A *codebook* is a data structure used to store codewords, representing repeating parts in an input data. The input data can be compressed by substituting the repeating parts with smaller codeword symbols. In this work, we emulate cluster centers as the codewords used to represent their member subsequences. Performance of the encoding can be measured by using *Compression ratio* and *Error* defined below.

**Definition 4.** *Compression ratio* is a ratio of the data size between after and before compression, including an overhead of construction of a codebook and codeword symbols. For example, given a 16-charecter string $S =$ "*ABCDEFGHIJKLMNOP*". Suppose that substrings "*ABC*" and "*HIJ*" are similar, we can substitute them with a symbol *x*, therefore the encoded string $S' =$ "*xDEFGxKLMNOP*". In this case, we can eliminate 6 characters ("*ABC*" and "*HIJ*"), but a codeword of size 3, and two *x*'s must be created; thus, the compression is $6 - (3 + 2) = 1$ character, and the *compression ratio* $\widetilde{R} = \frac{16-1}{16} = 0.94$.

**Definition 5.** *Error* can be obtained by calculating summation of distances from cluster centers to their cluster members. For example, the two substrings "*ABC*" and "*HIJ*", which are mentioned in the previous definition, are grouped into a cluster $C$, then a codeword (a cluster center) $\overline{C}$ is created. The *Error* of creating a cluster from those substrings is $\widetilde{E}(C) = Dist(\overline{C}, ABC) + Dist(\overline{C}, HIJ)$.

Next, we summarize background knowledge used in this paper.

### 3.2. Euclidean distance measure

To measure distance between two subsequences, we use Euclidean distance that has been widely used in time series domain. The distance is as shown

$$Dist(X_{i,n}, X_{j,n}) = \sqrt{\sum_{k=0}^{n-1} (x_{i+k} - x_{j+k})^2} \quad (1)$$

Before the distance calculation, all subsequences must be normalized. We use $Z$-normalization [10] that makes the value of mean and standard deviation of a time series to be zero and one, respectively. Given a subsequence $T_{i,n} = (t_i, t_{i+1}, \ldots, t_{i+n-1})$ whose mean is $\mu$ and standard deviation is $\sigma$. The normalized time series is $T'_{i,n} = (t'_i, t'_{i+1}, \ldots, t'_{i+n-1})$, where $t'_k = \frac{t_k - \mu}{\sigma}$.

To create a cluster center, we use amplitude averaging approach to average two sequences. Given subsequences $P = (p_1, \ldots, p_i, \ldots, p_n)$ and $Q = (q_1, \ldots, q_i, \ldots, q_n)$, a new subsequence $R = (r_1, \ldots, r_i, \ldots, r_n)$ is produced by $r_i = \frac{\omega_p p_i + \omega_q q_i}{\omega_p + \omega_q}$, where $\omega_p$ and $\omega_q$ are weight of $P$ and $Q$.

### 3.3. Uniform scaling

Many research works show that uniform scaling technique can improve performance in terms of accuracy [7,30]. Specifically, a subsequence $T = (t_1, \ldots, t_i, \ldots, t_m)$ can be shrunk/stretched, by specifying a scaling factor $f \geqslant 1$, to a new time series $T' = \left(t'_1, \ldots, t'_j, \ldots, t'_n\right)$, where $t'_j = t_{\lfloor j \cdot n/m \rfloor}$, $\lceil m/f \rceil \leqslant n \leqslant \lfloor m \cdot f \rfloor$. We

extract input time series to subsequences of different lengths. In detail, clustering algorithm takes two parameters, $w$ and $f$, window length and scaling factor, respectively. Subsequences of length from $\lceil w/f \rceil$ to $\lfloor w \cdot f \rfloor$ are extracted, then we use uniform scaling to make them the same length of $w$ before clustering them.

### 3.4. Subsequence matching

Subsequence matching algorithm [27] is usually used as a subroutine in many data mining tasks. By giving a query sequence, we can retrieve a subsequence, which is the most similar to the query, from a longer time series. In this work, we use the Euclidean distance as a distance measure to compare the query sequence with all the extracted subsequences.

### 3.5. Subsequence motif discovery

A subsequence motif [23] is the most similar pair of subsequences in a time series data. Many research works have proposed motif discovery algorithms trying to improve performance in terms of speed and accuracy. In this paper, we use the MK algorithm in [19], which is considered the fastest algorithm to find a pair of motif by using the Euclidean distance.

## 4. Selective Subsequence Time Series clustering framework

This section provides details of our approach called Selective Subsequence Time Series (SSTS) clustering framework. Firstly, we begin by stating the problem definition.

### 4.1. Problem definition

Input of our algorithm is a single time series data. The problem is to first determine a number of clusters $n$, and then to group subsequences into proper clusters; some subsequences can be discarded without being assigned to any cluster. The subsequences are extracted using a sliding window approach. The sliding window can be varied in a range specified by a user. For example, we demonstrate by using a 16-character string $S$ = "*ABCDEFGHIJKLMNOP*" as an input. We use a sliding window $w$ of size 3, and a scaling factor $f$ = 1.5; therefore, the length of the subsequences is varied from 2 to 4. The subsequences are extracted into a set $S'$ = {"*AB*","*BC*",…, "*OP*","*ABC*","*BCD*",…,"*NOP*","*ABCD*","*BCDE*",…,"*MNOP*"}. The algorithm should produce a set of clusters $\widetilde{C} = \{C_1, \ldots, C_i, \ldots, C_n\}$. Each cluster consists of its members and a cluster center: $C_i = \{t_{i_1}, t_{i_2}, \ldots, t_{i_n}, \overline{C}_i\}$, where $t_{i_j}$ is the $j$th member of the $i$th cluster, and the $\overline{C}_i$ is the cluster center of the $i$th cluster.

### 4.2. Clustering method

To form clusters from a set of subsequences, we must iteratively pick one subsequence and assign it to a cluster. However, in the first place, we do not have any predefined cluster yet, and we must make a decision as follows. Intuitively, we can choose two subsequences which are the most similar, to create the first cluster, then the first cluster center is produced. As a result, we can choose other subsequences, which are the most similar to the already created cluster, to be added to the existing cluster; therefore, the cluster center is then updated. Nevertheless, it is better to create a new cluster if there exist two subsequences that are similar to each other more than to the existing cluster center. Moreover, if there are two clusters that can be grouped together, we can decide to merge them to create a new cluster. Thus, we define three operations for producing clusters from a set of subsequences; those are *Create*, *Add* and *Merge* to iteratively select two subsequences to create a new cluster, to assign a subsequence to an existing cluster, and to merge two clusters into a new cluster, respectively.

Our approach iteratively selects an operation, which are *Create*, *Add* and *Merge* to produce a set of clusters. Accordingly, we adopt an idea of data encoding as a heuristic function to choose an optimal operation in each step of clusters construction. We emulate a set of cluster centers as a codebook, where each cluster center is a codeword used to encode the input time series. Some subsequences from the input time series, which are members of a cluster, will be substituted by a small codeword symbol. *Error* of a cluster is determined by a summation of Euclidean distance from the codeword, which is the cluster center, to their member subsequences.

$$\widetilde{E}(C_i) = \sum_{j=1}^{m} Dist(t_{i_j}, \overline{C}_i) \tag{2}$$

where $m$ is a number of members in the $i$th cluster.

*Increased error* $\Delta\widetilde{E}$ is obtained after a cluster update.

$$\Delta\widetilde{E} = \widetilde{E}_{after} - \widetilde{E}_{before} \tag{3}$$

*Compression ratio* $\widetilde{R}$ is determined by calculating data reduction of the original subsequence including overhead from codeword construction and codeword symbol substitution.

In detail, *Compression ratio* and *Increased Error* for each operation are described below.

1. *Create*: Create a new cluster $C$ from two subsequences $P$ of length $u$, and $Q$ of length $v$. A new codeword $\overline{C}$ of length $w$ is obtained by merging $P$ and $Q$. The length of input time series $l$ is reduced by $u + v$. The overhead is added by the codeword construction and the substitution of $P$ and $Q$ by two of a codeword symbol "$x$" of size 1.

$$\Delta\widetilde{E} = \widetilde{E}(C) \tag{4}$$

$$\widetilde{R} = [(u+v) - (w+2)]/l \tag{5}$$

2. *Add*: Update an existing cluster $C$ to a new cluster $C'$ by adding a subsequence $P$ of length $u$, and update the codeword $\overline{C}$ to $\overline{C}'$. This operation reduces the length of input time series $l$ by $u$. The overhead is added by substituting $P$ by "$x$" of size 1.

$$\Delta\widetilde{E} = \widetilde{E}(C') - \widetilde{E}(C) \tag{6}$$

$$\widetilde{R} = (u-1)/l \tag{7}$$

3. *Merge*: two clusters $C_i$ and $C_j$ are merged into a new cluster $C'$. A codeword of length $w$ is reduced.

$$\Delta\widetilde{E} = \widetilde{E}(C') - [\widetilde{E}(C_i) + \widetilde{E}(C_j)] \tag{8}$$

$$\widetilde{R} = w/l \tag{9}$$

The problem can be considered as a search space consisting of nodes of the three operations, which is illustrated in Fig. 2. Our approach uses greedy method to iteratively select a node that has minimal *Increased Error*. To do this, as shown in Fig. 3, we apply the MK motif discovery algorithm [19] to discover a pair of subsequences, which has minimal Euclidean distance, to be the best node for the *Create* operation. To search for the optimal *Add* node, all codewords are used as queries for the subsequence matching algorithm to locate the best subsequence to be added to an existing cluster. The optimal *Merge* node can be determined by searching all nodes, due to its small number of nodes. Note that the subsequences can be of different lengths, so we use a uniform scaling technique to make them the same length $w$ before applying the motif discovery and the subsequence matching algorithms.

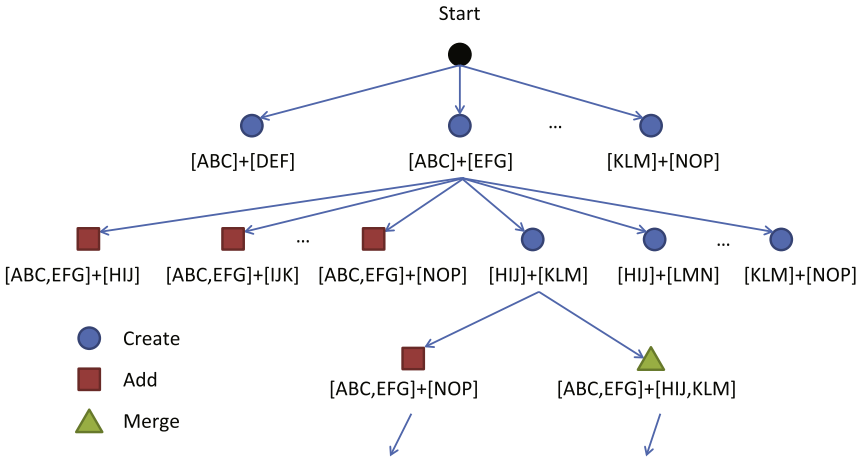


**Fig. 2.** The search space consists of *Create*, *Add* and *Merge* operations.

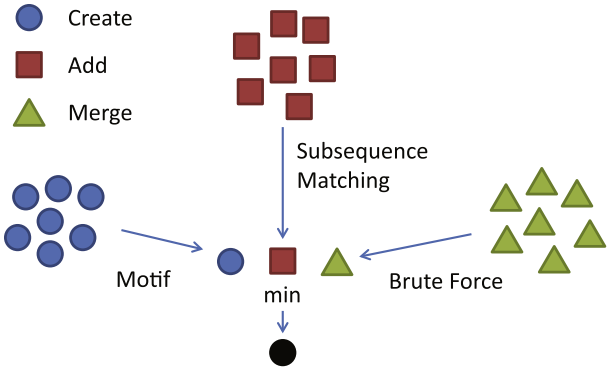


**Fig. 3.** The optimal node can be determined by using motif discovery and subsequence matching algorithms.

As a running example, we use the character string $S$ mentioned in Section 4.1 to demonstrate our algorithm step by step, as shown in Fig. 4. For brevity, we set a window length $w$ to 3 and a scaling factor to 1; so the subsequences can only be of length 3. First, none of the cluster exists. The *Create* operation must be chosen to create the first cluster $C_1$. The motif discovery is applied then we get "*ABC*" and "*EFG*" as its results. A codeword $\overline{C}_1$ is produced by merging "*ABC*" and "*EFG*" together. Second, after creating the first cluster, there are two choices that are *Create* and *Add* operations. We use $\overline{C}_1$ as a query to the subsequence matching algorithm, then we obtain "*NOP*" as the result; after removing "*ABC*" and "*EFG*" from the subsequence set, the motif algorithm returns "*HIJ*" and "*KLM*". In this case, suppose the latter case gives smaller $\Delta\widetilde{E}$, we choose the *Create* operation to create a new cluster $C_2$ from "*HIJ*" and "*KLM*", then $\overline{C}_2$ is produced. Next, the choices are *create*, *add*, and *merge*. Suppose the smallest $\Delta\widetilde{E}$ is obtained from the *Merge* operation of $C_1$ and $C_2$, so "*ABC*", "*EFG*", "*HIJ*" and "*KLM*" are merged into the same cluster, and $\overline{C}_1$ and $\overline{C}_2$ are combined. Finally, only $C_1$ remains.

To determine a proper number of clusters, we must choose a state of creating clusters that provides large *compression ratio* while producing less *error*. From the compression–error graph shown in Fig. 5a, it is obvious that there is a *knee point* in the graph where *errors* are dramatically increased. It means applying an operation after that point will lose the clustering accuracies. Thus, we return clusters in that state as a result of the algorithm. To find that point, we determine linear fitting function to the compression–error graph and choose a point that gives minimum residual value to the fitting function as shown in Fig. 5b. Consider a special case that a user want to specify the number of clusters $k$, our algorithm can effortlessly handle it by choosing a latest state that has the number of clusters equal to the one specified by the user.

Table 1 illustrates our main algorithm. The algorithm starts by extracting subsequences from the input sequence by running SubsequenceExtractor function. After that, it enters a loop to iteratively

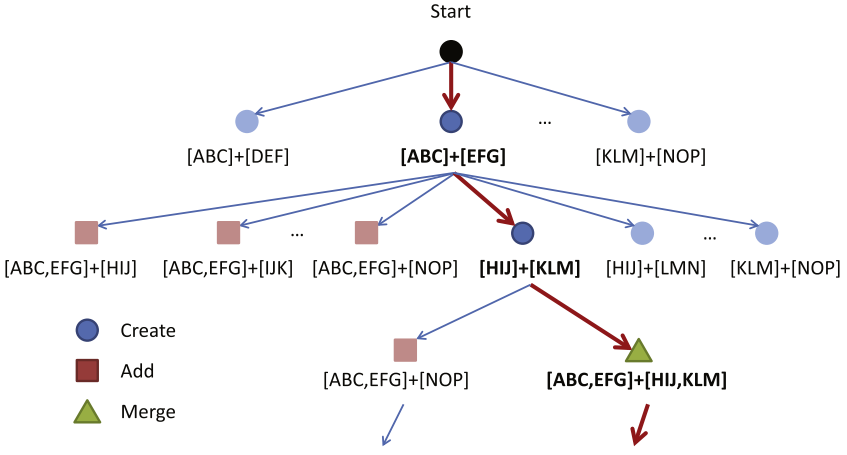


**Fig. 4.** Greedy search example: an optimal node will be chosen in each step.

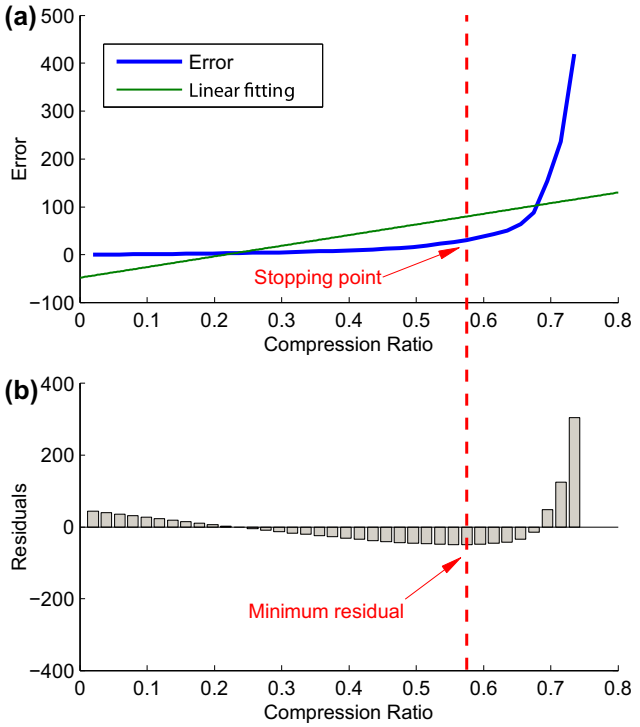


**Fig. 5.** The stopping point can be found at the state that has minimum value of residual. (*a*) A linear estimation line of the *error* line helps find the *knee point* in the *error* line. (*b*) The stopping point is determined at the minimum residuals value to the linear fitting function.

**Table 1**
SSTS clustering.

| Function $[\widetilde{C}]$ = SSTS($T,w,f$) | |
|---|---|
| 1 | $S$ = SubsequenceExtractor($T,w,f$) |
| 2 | while there is an operation left |
| 3 | $[C'[1],S'[1]]$ = Create Cluster($\widetilde{C},S$) |
| 4 | $[C'[2],S'[2]]$ = UpdateCluster($\widetilde{C},S$) |
| 5 | $[C'[3],S'[3]]$ = MergeClusters($\widetilde{C},S$) |
| 6 | $m$ = ArgMinError($C'$) |
| 7 | $\widetilde{C} = C'[m]$ |
| 8 | $S = S'[m]$ |
| 9 | $P.add(\widetilde{C})$ |
| 10 | return $P$.at(StoppingState($P$)) |

select an operator to create clusters until there is no subsequence left. The *Create*, *Add* and *Merge* operations are applied, then the best one, which gives minimum *error*, is selected in each iteration. Every cluster construction state is kept in a list $P$ for determining the best state later. After breaking the loop, a proper cluster state will be chosen by using StoppingState function.

Details of SubsequenceExtractor function are shown in Table 2. The function extracts subsequences of length varied from $w_{min}$ to $w_{max}$, and makes it the same length by using UniformScaling function. Consequently, the extracted subsequences are normalized by z-Normalize function, and they are stored in a list of subsequences $S$.

Table 3 shows CreateCluster function in details. It starts by executing MotifDiscovery to find a motif pair. After that, a cluster is created from the motif pair, and the motif pair and the subsequences that overlap with them are removed from the list of subsequences $S$.

Table 4 explains details of UpdateCluster function. Every cluster center of all created clusters is used as a query sequence for SubsequenceMatching function. The function returns a subsequence

**Table 2**
Subsequence extractor.

| Function $[S]$ = SubsequenceExtractor($T,w,f$) | |
|---|---|
| 1 | $w_{min} = \lceil w/f \rceil$ |
| 2 | $w_{max} = \lfloor w \cdot f \rfloor$ |
| 3 | $l$ = Length($T$) |
| 4 | for $i = w_{min}:w_{max}$ |
| 5 | for $j = 1:l$ |
| 6 | $t$ = UniformScaling($s[j:j+i-1]$) |
| 7 | z-Normalize($t$) |
| 8 | $t.start = j$ |
| 9 | $t.end = j+i-1$ |
| 10 | $S.add(t)$ |
| 11 | return $S$ |

**Table 3**
Create operation.

| Function $[\widetilde{C}',S']$ CreateCluster($\widetilde{C},S$) | |
|---|---|
| 1 | $[l_1,l_2]$ = MotifDiscovery($S$) |
| 2 | $C.\overline{C}$ = Average($S[l_1],S[l_2]$) |
| 3 | $C.addMember(S[l_1])$ |
| 4 | $C.addMember(S[l_2])$ |
| 5 | $\widetilde{C}.add(C)$ |
| 6 | remove $S[l_1]$ and $S[l_2]$ and subsequences that overlap $S[l_1]$ and $S[l_2]$ from $S$ |
| 7 | return $\widetilde{C},S$ |

**Table 4**
Add operation.

| Function $[\widetilde{C}',S']$ = UpdateCluster($\widetilde{C},S$) | |
|---|---|
| 1 | for $i = 1:\widetilde{C}.numberOfCluster()$ |
| 2 | $C = \widetilde{C}[i]$ |
| 3 | $t$ = SubsequenceMatching($S, C.\overline{C}$) |
| 4 | $C.\overline{C}$ = Average($C.\overline{C}, S[t]$) |
| 5 | $C.addMember(S[t])$ |
| 6 | if $error_{BSF} > C.error()$ |
| 7 | $C' = C$ |
| 8 | $i_{BSF} = i$ |
| 9 | $error_{BSF} = C.error()$ |
| 10 | $t' = t$ |
| 11 | $C[i_{BSF}] = C'$ |
| 12 | remove $S[t']$ and subsequences that overlap $S[t']$ from $S$ |
| 13 | return $\widetilde{C},S$ |

**Table 5**
Merge operation.

| Function $[\widetilde{C}',S']$ = MergeClusters($\widetilde{C},S$) | |
|---|---|
| 1 | $n$ = number of clusters in $\widetilde{C}$ |
| 2 | for $i = 1:n-1$ |
| 3 | for $j = i+1:n$ |
| 4 | $C_1 = \widetilde{C}[i]$ |
| 5 | $C_1 = \widetilde{C}[j]$ |
| 6 | $C_1.\overline{C}$ = Average($C_1.\overline{C}, C_2.\overline{C}$) |
| 7 | add all members of $C_2$ to $C_1$ |
| 8 | if $error_{BSF} > C_1.error()$ |
| 9 | $C' = C_1$ |
| 10 | $l_1 = i$ |
| 11 | $l_2 = j$ |
| 12 | $\widetilde{C}[l_1] = C'$ |
| 13 | $\widetilde{C}.remove(l_2)$ |
| 14 | return $\widetilde{C},S$ |

from $S$ that is the most similar to the query. The subsequence that produces the least *error* is chosen to be added to the cluster that holds cluster center that was used as the query. The cluster center

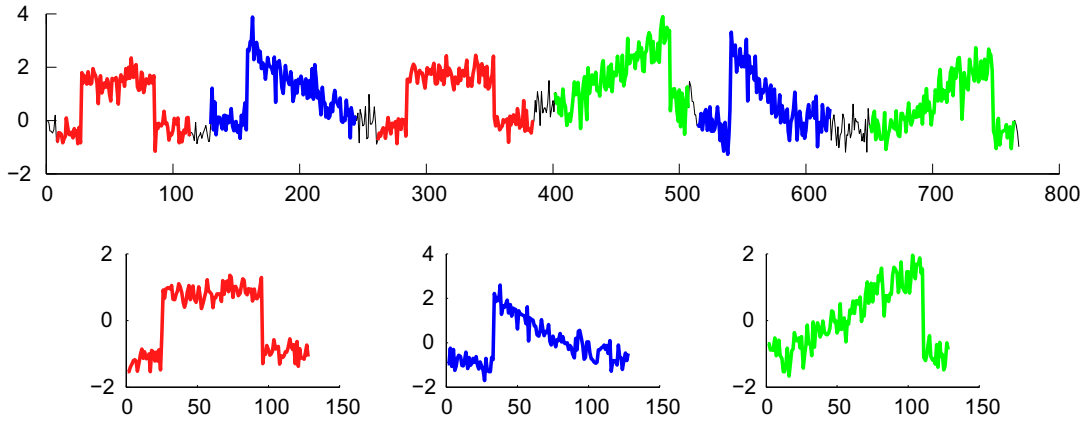

**Fig. 6.** (*top*) A sequence of CBF dataset. (*bottom*) Cluster centers of each class.

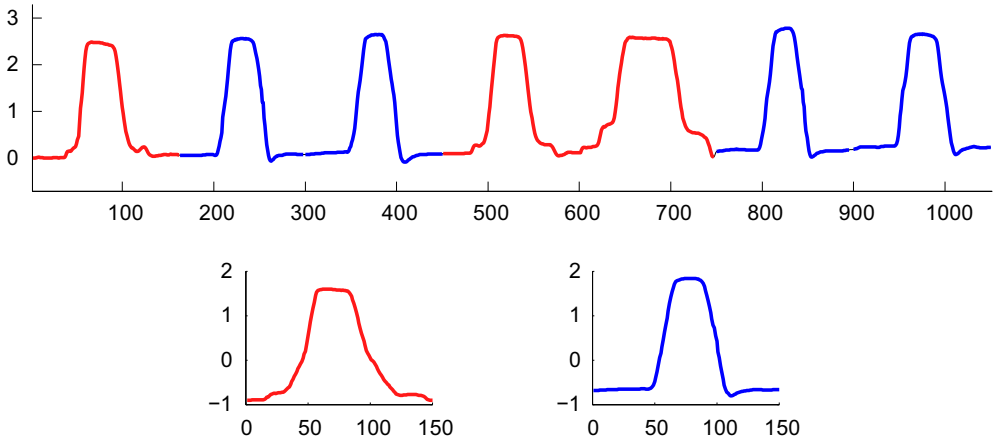

**Fig. 7.** (*top*) *Gun-Point* data extracted from a video surveillance camera. (*bottom*) Cluster centers of each class.

of that cluster are updated by averaging the old cluster center and the subsequence resulted from the subsequence matching function. After that, the resulted subsequence and its overlapping subsequences are removed from *S*.

The last operation, the *Merge*operation are described as the MergeClusters function shown in Table 5. All combination pairs of the existing clusters are examined, then a pair that gives minimum *error* will be merged.

## 5. Experimental results

This section provides experimental results of our proposed method on various data domains. We separate the experiments into two parts. First, we shows usefulness of our algorithm on real and synthetic datasets from various domains. Second, we demonstrate our search performance by comparing our algorithm with brute-force method on the defined search space.

### 5.1. Usefulness of our method

In this part, we demonstrate that our algorithm can be applied in many types of data domains, i.e., synthetic dataset, data extracted from video surveillance system and images, and real ECG data sequence.

#### 5.1.1. Synthetic data

We experiment on the *Cylinder–Bell–Funnel (CBF)* dataset from the UCR time series archive [14]. It has been shown that most STS clustering algorithms fail to produce meaningful result from this very simple dataset.

In our experiment, we randomly select data from each class, then concatenate them to a single time series, as shown in Fig. 6. The cluster results are illustrated as colored[1] subsequences in Fig. 6. The result shows that the key characteristics of each class are clustered correctly, and the cluster centers can represent the shape of their member subsequences.

#### 5.1.2. Video surveillance problem

In this experiment, we apply our algorithm on the video surveillance domain, which is the gun problem [14]. The time series data is captured from the centroid of each actor's right hand performing two actions: *Gun-Draw* and *Point*. The motion of the two classes of action are very similar and hard to distinguish.

Result of our proposed method, as illustrated in Fig. 7, shows that all subsequences of motions are clustered correctly to their classes. Furthermore, the cluster centers from our method can preserve the important features and shapes in the data.

#### 5.1.3. Time series data extracted from images

This experiment shows the result from clustering data extracted from images, which are created by tracing the local angles from the centroid of an image to its perimeter. We make the input time series by choosing the dataset that has different *complexities* [1]. The

[1] For interpretation of color in Figs. 1–10, the reader is referred to the web version of this article.

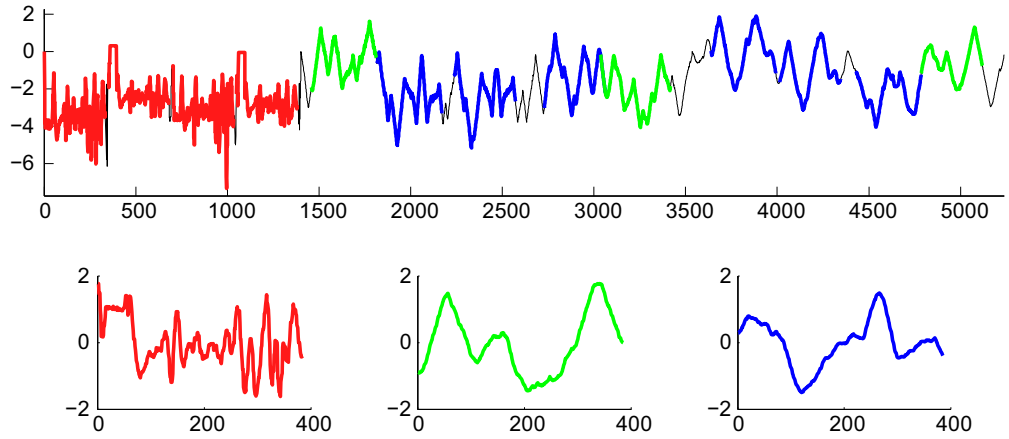

**Fig. 8.** (*top*) A sequence of data extracted from image of faces and leaves. (*bottom*) cluster centers of each class.

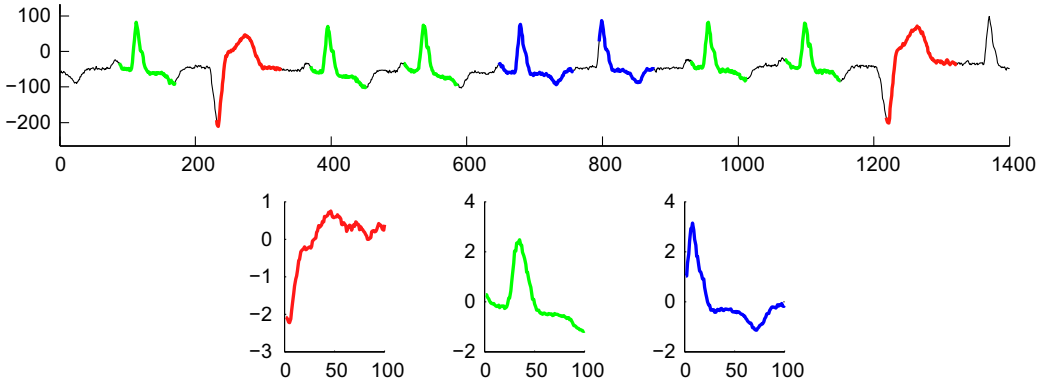

**Fig. 9.** (*top*) ECG sequences with abnormal heartbeats. (*bottom*) Cluster centers from our algorithm.

datasets used here are *Face-all* and *OSU-Leaf* [14], which are extracted from human faces with various expressions on the face, and from different species of leaf images.

Fig. 8 shows that our algorithm can cluster subsequences of the data even when the data has different *complexity* values. The subsequences of face data are grouped in a cluster, which is shown in red, and the leaf subsequences are separated into two subclasses that have the same shape.

### 5.1.4. ECG data

In this experiment, we run our algorithm on a medical dataset, which is an ECG data [9]. Fig. 9 shows that the beats are of different shapes. If we can separate the beats into clusters, the heart diseases will be diagnosed easier. From the result in Fig. 9, three groups of heartbeats are clustered. The normal beats are clustered within the same group as shown in green, the abnormal beats, as shown in red, are clustered into the same group, and the blue cluster contains the beats that have minor anomalies, and are clustered separately.

## 5.2. Comparison with the brute-force method

We will demonstrate our performance on searching through the search space. Fig. 10 illustrates the result of our algorithm comparing with the brute-force method tested on the ECG data used in Section 5.1.4. It roughly contains 6000 possible paths of the input time series of length 1400 data points and a window size of 100. The result of our method is shown in a thick blue line. Our main goal is to maintain *error* while maximizing the *compression ratio*. As shown in Fig. 10, our algorithm can search through the search space closely following the optimal path by examining just 1 out of 6000 possible paths.

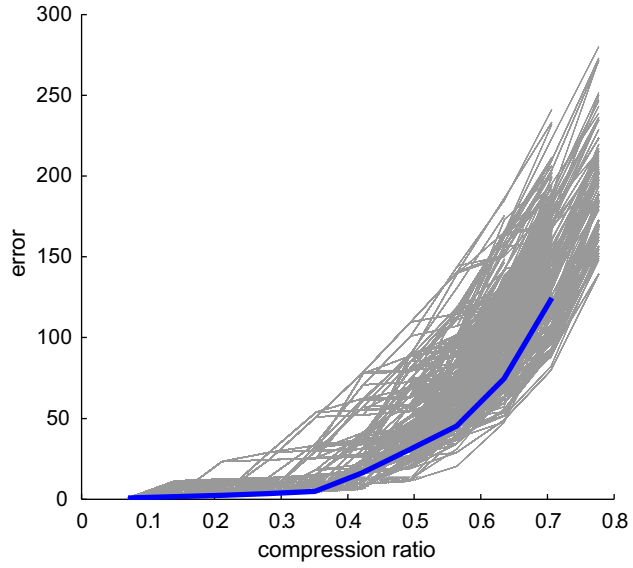


**Fig. 10.** Our algorithm (shown in blue) comparing with the brute-force method.

## 6. Conclusions

In this work, we propose a novel Subsequence Time Series (STS) clustering named Selective Subsequence Time Series (SSTS) clustering. We show that clustering on time series subsequences can be meaningful if some noise or unimportant subsequences are discarded, and different lengths of member subsequences are allowed. We assure the efficiency and usefulness of our algorithm by experimenting in various data domains. Furthermore, our method can perform clustering by requiring only a few parameters where users can easily and flexibly adjust.

## Acknowledgment

This research is partially supported by the Thailand Research Fund (Grant No. MRG5380130), Postdoctoral Fellowship (Ratchadaphiseksomphot Endowment Fund), and the Thailand Research Fund given through the Royal Golden Jubilee Ph.D. Program (PHD/0319/2551 to S. Rodpongpun).

## References

[1] G.E.A.P.A. Batista, X. Wang, E.J. Keogh, A complexity-invariant distance measure for time series, in: Proceedings of the 2011 SIAM International Conference on Data Mining (SDM'11), Arizona, USA, 2011, pp. 699–710.
[2] J.R. Chen, Making subsequence time series clustering meaningful, in: Proceedings of the 5th IEEE International Conference on Data Mining (ICDM'05). Texas, USA, 2005, pp. 114–121.
[3] P. Cotofrei, K. Stoffel, Classification rules + time = temporal rules, in: Proceedings of 2002 International Conference on Computational Science, Amsterdam, Netherlands, 2002, pp. 572–581.
[4] G. Das, K. Lin, H. Mannila, G. Renganathan, P. Smyth. Rule discovery from time series, in: 4th International Conference on Knowledge Discovery and Data Mining (KDD'98). New York, USA, 1998, pp. 16–22.
[5] de A.R. Araújo, A class of hybrid morphological perceptrons with application in time series forecasting, Knowledge-Based Systems 24 (4) (2011) 513–529.
[6] A.M. Denton, C.A. Besemann, D.H. Dorr, Pattern-based time-series subsequence clustering using radial distribution functions, Knowledge and Information Systems 18 (2009) 1–27.
[7] A.W.-C. Fu, E. Keogh, L.Y. Lau, C.A. Ratanamahatana, R.C.-W. Wong, Scaling and time warping in time series querying, The VLDB Journal 17 (2008) 899–921.
[8] R. Fujimaki, S. Hirose, T. Nakata, Theoretical analysis of subsequence time-series clustering from a frequency-analysis viewpoint, in: Proceedings of the 2008 SIAM International Conference on Data Mining (SDM'08), Georgia, USA, 2008, pp. 506–517.
[9] A.L. Goldberger, L.A.N. Amaral, L. Glass, J.M. Hausdorff, P.C. Ivanov, R.G. Mark, J.E. Mietus, G.B. Moody, C.-K. Peng, H.E. Stanley, PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals, Circulation 101 (23) (2000) e215–e220.
[10] J. Han, M. Kamber, J. Pei, Data Mining: Concepts and Techniques, second ed., The Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann, 2006.
[11] T. Idé, Why does subsequence time-series clustering produce sine waves? in: 10th Pacific–Asia Conference on Knowledge Discovery and Data Mining (PAKDD'06), Singapore, 2006, pp. 211–222.
[12] X. Jin, Y. Lu, C. Shi, Distribution discovery: local analysis of temporal rules, in: Proceedings of the 6th Pacific–Asia Conference on Advances in Knowledge Discovery and Data Mining (ICDM'02), London, UK, 2002, pp. 469–480.
[13] E. Keogh, J. Lin, Clustering of time-series subsequences is meaningless: implications for previous and future research, Knowledge and Information Systems 8 (2) (2005) 154–177.
[14] E. Keogh, X. Xi, L. Wei, C.A. Ratanamahatana, The UCR Time Series Classification/Clustering Homepage, 2008. <www.cs.ucr.edu/~eamonn/time_series_data>.
[15] C.-P. Lai, P.-C. Chung, V.S. Tseng, A novel two-level clustering method for time series data analysis, Expert Systems with Applications 37 (2010) 6319–6326.
[16] Y.-S. Lee, L.-l. Tong, Forecasting time series using a methodology based on autoregressive integrated moving average and genetic programming, Knowledge-Based Systems 24 (1) (2011) 66–72.
[17] C.-S. Li, P.S. Yu, V. Castelli, Malm: a framework for mining sequence database at multiple abstraction levels, in: Proceedings of the 7th international conference on Information and knowledge management (CIKM'98), New York, USA, 1998, pp. 267–272.
[18] H. Li, C. Guo, Piecewise cloud approximation for time series mining, Knowledge-Based Systems 24 (4) (2011) 492–500.
[19] A. Mueen, E.J. Keogh, Q. Zhu, S. Cash, B. Westover, Exact Discovery of Time Series Motifs, in: SIAM International Conference on Data Mining (SDM'09), Nevada, USA, 2009, pp. 473–484.
[20] V. Niennattrakul, D. Srisai, C.A. Ratanamahatana, Shape-based template matching for time series data, Knowledge-Based Systems 26 (2012) 1–8.
[21] M. Ohsaki, M. Nakase, S. Katagiri, Analysis of subsequence time-series clustering based on moving average, in: Proceedings of the 9th IEEE International Conference on Data Mining (ICDM'09), Washington, DC, USA, 2009, pp. 902–907.
[22] S.C.W. Ong, S. Ranganath, Automatic sign language analysis: a survey and the future beyond lexical meaning, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (2005) 873–891.
[23] H. Tang, S.S. Liao, Discovering original motifs with different lengths from time series, Knowledge-Based Systems 21 (7) (2008) 666–671.
[24] X. Wang, K. Smith, R. Hyndman, Characteristic-based clustering for time series data, Data Mining and Knowledge Discovery 13 (2006) 335–364.
[25] X. Weng, J. Shen, Classification of multivariate time series using locality preserving projections, Knowledge-Based Systems 21 (7) (2008) 581–587.
[26] X. Weng, J. Shen, Classification of multivariate time series using two-dimensional singular value decomposition, Knowledge-Based Systems 21 (7) (2008) 535–539.
[27] H. Wu, B. Salzberg, G.C. Sharp, S.B. Jiang, H. Shirato, D. Kaeli, Subsequence matching on structured time series data, in: Proceedings of the 2005 ACM SIGMOD international conference on Management of data (SIGMOD'05), New York, USA, 2005, pp. 682–693.
[28] T. Yairi, Y. Kato, K. Hori, Fault detection by mining association rules from house-keeping data, in: Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space, Montreal, Canada, 2001, pp. 18–21.
[29] R. Yang, S. Sarkar, B. Loeding, Handling movement epenthesis and hand segmentation ambiguities in continuous sign language recognition using nested dynamic programming, IEEE Transactions on Pattern Analysis and Machine Intelligence 32 (2010) 462–477.
[30] D. Yankov, E. Keogh, J. Medina, B. Chiu, V. Zordan, Detecting time series motifs under uniform scaling, in: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07), New York, USA, 2007, pp. 844–853.