An Efficient Implementation of Anytime K-medoids Clustering for Time Series under Dynamic Time Warping

Van The Huy, Duong Tuan Anh
Faculty of Computer Science and Engineering
Ho Chi Minh City University of Technology
268 Ly Thuong Kiet, District 10, Ho Chi Minh City, Vietnam
huyvanthe@yahoo.com.vn, dtanh@cse.hcmut.edu.vn

ABSTRACT
Time series clustering is one of the crucial tasks in time series data mining. So far, time series clustering has been most used with Euclidean distance. Dynamic Time Warping (DTW) distance measure has increasingly been used as a similarity measurement for various data mining tasks in place of traditional Euclidean distance due to its superiority in sequence-alignment flexibility. However, there exist some difficulties in clustering time series with DTW distance, for example, the problem of speeding up DTW distance calculation in the context of clustering. So far, there have been two proposed methods for time series clustering with DTW and both of them work in *batch* scheme. Recently, Zhu et al. proposed a framework of *anytime* clustering for time series with DTW which uses a data-adaptive approximation to DTW. In this paper, we present an efficient implementation of anytime K-medoids clustering for time series data with DTW distance. In our method, we exploit the anytime clustering framework with DTW proposed by Zhu et al., apply a method for medoid initialization, and develop a multithreading technique to speed-up DTW distance calculation. Experimental results on benchmark datasets validate our proposed implementation method for anytime K-medoids clustering for time series with DTW.

CCS Concepts
• Information systems→Information system applications →Data mining.

Keywords
K-medoids clustering; time series; anytime algorithm; dynamic time warping; medoid initialization; multithreading technique.

1. INTRODUCTION
Time series data arise in so many applications of various areas ranging from science, engineering, business, finance, economy, medicine to government. Besides similarity search, another crucial task in time series data mining which has received an increasing amount of attention lately is time series clustering. The most popular method in time series clustering is K-means algorithm due to its simplicity and flexibility (Lin et al., 2004 [16]). So far, time series clustering has been most achieved with K-means algorithm and under Euclidean distance. However, experiments have shown that the traditional Euclidean distance is not an accurate similarity measure for time series. A similarity measure called dynamic time warping (DTW) has been proposed and studied (Berndt, 1994 [1]; Sakoe and Chiba, 1978 [21]; Keogh and Ratanamahatana, 2002 [11]). DTW distance shows great superiority in accuracy of many tasks, e.g. classification and clustering over Euclidean distance. DTW has been used in various domains, especially applications related to multimedia data.

As for clustering time series with DTW distance, there exist some major difficulties, for example, the problem of shape averaging in DTW (Gupta et al., 1996 [2]; Niennattrakul and Ratanamahatana, 2007 [17]; Niennattrakul and Ratanamahatana, 2009 [18]; Petitjean et al., 2011 [20]) or the problem of speeding up DTW distance calculation in the context of clustering since DTW distance incurs high computational cost. Notice that lower bound techniques (e.g., LB_Keogh [11], LB_Kim [13], LB_Improved [14], LB_Yi [23]), which can help to accelerate DTW calculation in similarity search task, do not directly apply to clustering.

There have been so few works on clustering time series with DTW. Hautamaki et al., 2008 [5] proposed a method of time series clustering with DTW which combines hierarchical agglomerative clustering and K-means. Since this method applies a randomized local search to find cluster prototypes, it incurs high computational cost and therefore can be used for time series with very short lengths. Petitjean et al., 2011 [20] proposed a method of time series clustering with DTW which uses K-means for clustering and DTW barycenter technique for shape averaging. However, the two existing methods work in a *batch* scheme, i.e., they run till the end and do not allow user interaction during their execution. In contrast, *anytime* algorithms [25] quickly produce an approximate result and continuously refine it during their runtime. Moreover, they allow users to suspend them for examining the result and to resume them for finding better results until a satisfactory result is reached. Due to this scheme, anytime algorithms become a useful approach and are widely employed in many fields, especially when the user has to work in an environment with limited time or computational resources [25]. For example, if asked to cluster time series taken from a streaming environment, we may have anywhere from several minutes to several days to return a clustering result. For such problems, an anytime algorithm may be especially useful.

Zhu et al. (2012) [24] have recently proposed a framework of *anytime* clustering for time series with DTW which uses a data-adaptive approximation to DTW. The anytime clustering framework with DTW proposed by Zhu et al. (2012) consists of

SoICT 2016, December 08–09, 2016, Ho Chi Minh, Viet Nam.

http://dx.doi.org/10.1145/3011077.3011128

two important techniques: a novel approximation to DTW which is both fast to compute and accurate, and a heuristic ordering that gives the anytime algorithm the best order in which it should calculate the exact DTW distances.

Motivated by the framework proposed by Zhu et al., in this work, we develop an efficient method to implement an anytime K-medoids clustering for time series data with DTW distance. In our method, we (i) apply the anytime clustering framework with DTW in [24] by using a particular clustering algorithm, *K-medoids*; (ii) exploit a method for medoid initialization; and (iii) develop a multithreading technique to speed-up DTW distance calculation.

Extensive experimental study on several benchmark datasets shows that our proposed anytime K-medoids clustering method for time series under DTW can not only perform very fast but also bring out clustering results with high accuracy.

The rest of the paper is organized as follows. In Section 2 we explain briefly some basic backgrounds on DTW, the anytime clustering framework and an approximation technique to DTW. Section 3 introduces our proposed method to implement anytime K-medoids clustering for time series with DTW. Section 4 reports the experiments on the proposed method over some benchmark datasets. Section 5 gives some conclusions and remarks on future work.

2. BACKGROUND

In this section, we provide essential background, including Dynamic Time Warping distance measure and the anytime clustering framework with DTW.

2.1 Dynamic Time Warping

DTW is a distance measure between two time series in which we can match one data point in a time series with more than one data points in the other time series. Therefore, DTW allows computing the distance between two time series with different lengths.

Suppose we have two time series, a sequence U of length n, and a sequence V of length m, where

$$U = u_1, u_2, \ldots, u_i, \ldots, u_n.$$
$$V = v_1, v_2, \ldots, v_j, \ldots, v_m.$$

To compute the DTW between these two series, we construct an $n \times m$ matrix where the element $D_{ij} = d(u_i , v_j)$ is the distance between two points u_i and v_j (e.g. $d(u_i , v_j) = (u_i - v_j)^2$). To find the best match between these two sequences, we find a path through the matrix that minimizes the total cumulative distance between them. A warping path W is a contiguous set of matrix elements that characterizes a mapping between U and V. The k-th element of W is defined as $w_k = (i, j)_k$. So, we have:

$W = w_1, w_2, \ldots, w_k, \ldots, w_c$ $\quad (\max(n, m) \le c \le m + n + 1)$ as shown in Figure 1.

Among the warping paths that satisfy the above conditions, we concern with the optimal path that minimizes the warping cost:

$$DTW(U,V) = \min \sum_{k=1}^{c} w_k$$

This path can be found by using dynamic programming to evaluate the following recurrence which defines the cumulative distance $\gamma(i, j)$ as the distance $d(i, j)$ found in the current cell and the minimum of the cumulative distances of the adjacent elements:

$$\gamma(i, j) = d(u_i, v_j) + \min\{\gamma(i-1, j), \gamma(i, j-1), \gamma(i-1, j-1)\}$$

The DTW distance between two time series U and V is the square root of the cumulative distance at cell (m, n).

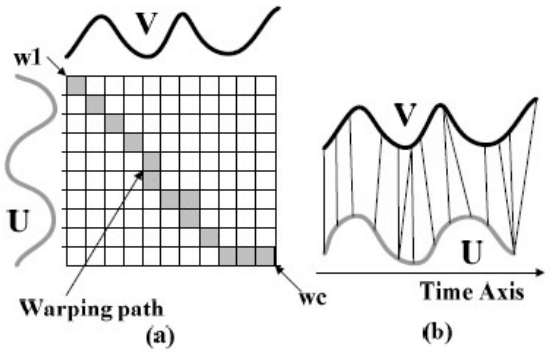

Figure 1. Illustration of DTW calculation (a) Build warping matrix and search for the minimal warping path, (b) The DTW alignments between time series.

The Euclidean distance between two sequences can be seen as a special case of DTW where the k-th element of W is constrained such that $w_k = (i, j)_k$, $i = j = k$. Note that it is only defined in the special case where the two sequences have the same length.

To speed up the DTW distance calculation, all practitioners using DTW constrain the warping path in a global manner by limiting how far it may stray from the diagonal. The subset of matrix that the warping path is allowed to visit is called *warping window* or a *band*. Two of the most frequently used global constraints in the literature are the Sakoe-Chiba band proposed by Sakoe and Chiba in 1978 [21] and Itakura Parallelogram proposed by Itakura in 1975 [8]. Sakoe-Chiba band is the area defined by two straight lines in parallel with the diagonal and Itakura Parallelogram is the area defined by the parallelogram which is symmetric over the diagonal.

Besides using global constraints, another approach to accelerate DTW calculation is replacing the vast majority of expensive DTW calculations with cheap-to-compute *lower bound* calculations. Some well-known lower bound techniques include LB_Keogh ([11]), LB_Kim ([13]), LB_Yi ([23]) and LB_Improved ([14]). The distance measured by these lower-bound techniques are smaller than that by DTW. Suppose a lower-bound technique for DTW is denoted as $LB(U, V)$, then we have $LB(U, V) \le DTW(U, V)$. Lower-bound techniques are used to prune the dissimilar objects from time series databases so as to retain the candidates for further measuring their similarity with DTW. Besides lower bound techniques, Li and Wang (2009) [15] proposed an *early abandoning* technique to speed-up DTW calculations.

2.2 Anytime Clustering Framework with DTW

The basic outline of the anytime clustering framework with DTW, proposed by Zhu et al. (2012) [24] is described in Figure 2.

The first three lines correspond to the setup phase of the anytime algorithm. Note that the algorithm cannot be interrupted during this phase of the algorithm. After this setup time, the algorithm can be interrupted at any point. The algorithm first builds an approximation of the DTW distance matrix (denoted as *aDTW*) in line 1. Once the distance matrix is available, clustering is

performed to obtain the first approximation solution that the algorithm can report to the user (line 2).

Input: *Dataset*: the input set of *n* time series.

Output: *Clusters*: the result of clustering.

```
1.   aDTW = BuildApproDistMatrix(Dataset);
2.   Clusters = Clustering(aDTW, Dataset);
3.   Disp('Setup is done, interruption is possible');
4.
5.   O = OrderToUpdateDTW(Dataset);
6.   for i = 1:Length(O) do
7.       aDTW(O(i)) = DTW(Dataset, O(i));
8.       if UserInterruptIsTrue()
9.           Clusters = Clustering(aDTW, Dataset);
10.          if UserTerminateIsTrue(Clusters)
11.              return;
12.          end if
13.      end if
14.  end for
15.  Clusters = Clustering(aDTW);
```

Figure 2. Basic framework for anytime clustering.

Lines 6 to 14 calculate the exact DTW distance to incrementally replace the values in *aDTW*, in the order specified in *O* obtained in line 5. During this phase, clustering is performed (line 9) only if the user requests an answer (line 8). If the user interrupts the algorithm in this manner, he or she chooses to terminate the algorithm after checking the current clustering result (line 10). When all $(n\times(n-1))/2$ distances have been updated, the *aDTW* becomes the exact DTW matrix, and we have the same answer (line 15) as the batch algorithm.

Notice that the DTW updates can be performed gradually many times before the clustering algorithm is performed due to the user interruption request.

Approximation of the DTW Distance Matrix

In order to build an approximation of the DTW distance matrix (*aDTW*) as required in line 1 of the algorithm in Figure 2, we can initialize *aDTW* with the Euclidean distance, or with one of the many lower bounds to DTW proposed previously such as LB_Keogh, LB_Kim and LB_Yi. The most important requirement on so many choices we have is that the time to build *aDTW* must be a small fraction of the time to calculate all DTW distances. Zhu et al. (2012) used LB_Keogh (denoted as LB) as a tight lower bound and Euclidean distance (denoted as ED) as a tight upper bound to DTW. Both methods can be computed in $O(N)$ where N is the length of time series.

Besides, Zhu et al. suggested that the two bounds should be used together.

Zhu et al. define the *DTW_Ratio* for each pair of time series (T_1, T_2) as:

$$DTW\_Ratio(T_1,T_2)=\frac{DTW(T_1,T_2)-LB(T_1,T_2)}{ED(T_1,T_2)-LB(T_1,T_2)} \quad (1)$$

DTW_Ratio(T_1, T_2) gives the relative position of *DTW*(T_1, T_2) between its lower and upper bounds. We can notice from (1) that it equals zero if *DTW*(T_1, T_2) = *LB*(T_1, T_2) and it equals one if *DTW*(T_1,T_2) = *ED*(T_1, T_2) and the range of *DTW_Ratio*(T_1, T_2) is in [0, 1].

Assume that the *DTW_Ratio* for two time series is known, and that we have calculated the corresponding LB and ED, then we can rearrange (1) to solve for the DTW distance:

$$DTW(T_1, T_2) = LB(T_1, T_2) + DTW\_Ratio(T_1, T_2)\times(ED(T_1, T_2) - LB(T_1, T_2)) \quad (2)$$

Estimating the DTW_Ratio

Based on (2) we have converted the problem of estimating DTW distance for each pair of time series to the problem of estimating the appropriate *DTW_Ratio*.

Zhu et al. (2012) [24] proposed a method for estimating *DTW_Ratio* for each pair of time series. The method consists of the following steps.

Input: A set of time series

Output: Appoximate DTW_Ratios for all pairs of time series

1. Sort all pairs of time series in ascending order of ED(T_1, T_2)/LB(T_1, T_2)
2. Divide the sorted list of all pairs of time series into *M* groups.
3. Estimate the mean of DTW_Ratios for each group by *sampling*. The mean of DTW_Ratios of each group can be used as appropriate DTW_Ratio for each pair of time series in this group.

The rationale for each step in this method is explained in details in [24].

Ordering Heuristic

Having initialized with a good approximation of the DTW distance matrix, the next step is to update each value in *aDTW*, one by one, until it becomes the matrix of true DTW distances (denoted as *tDTW*). We would like to find a good updating order *O*. Therefore, Zhu et al. [24] focus on how to reduce the approximate error of *aDTW* quickly. The authors define the *Normalized DTW Approximation Error* (NDAE) for a pair of time series (T_1, T_2) as:

$$NDAE(T_1,T_2)=\frac{|DTW(T_1,T_2)-DTW'(T_1,T_2)|}{DTW(T_1,T_2)} \quad (3)$$

NDAE(T_1, T_2) equals to zero if the approximation DTW distance (denoted as *DTW'*(T_1, T_2)) is identical to the *DTW*(T_1, T_2), in which case we do not need to update the distance for that pair, while we should give priority to update DTW for those pairs with a larger *NDAE*.

By replacing DTW in (3) by (2), the formula for *NDAE* obtains another equivalent expression:

$$NDAE(T_1,T_2)=\frac{|DTW\_Ratio(T_1,T_2)-DTW\_Ratio'(T_1,T_2)|}{1/(\frac{ED(T_1,T_2)}{LB(T_1,T_2)}-1)+DTW\_Ratio(T_1,T_2)} \quad (4)$$

We can not compare the numerator for pairs of time series (since the *DTW_Ratio*(T_1, T_2) is unknown); however based on experiments, Zhu et al. pointed out that its expected value is very small. So, we can approximately sort the denominator. Since ED and LB are known, and through experiments, Zhu et al. (2012) found out that a pair of time series with a larger ED/LB is more likely to have a smaller *DTW_Ratio*.

Therefore, Zhu et al. (2012) concluded that *a pair of time series with a large ED/LB is more likely to have a larger NDAE, and*

therefore should be given priority to update its true DTW distance.

3. ANYTIME K-MEDOIDS CLUSTERING UNDER DTW

The anytime clustering framework with DTW proposed by Zhu et al. is quite general. There are some components in this framework we have to determine before applying the framework in a real world application. These components are

- The clustering algorithm
- The method to compute approximate DTW distances.
- The method to compute exact DTW distance.

In this section we describe the details of our method for an anytime *K*-medoids clustering of time series with DTW distance. In this work, when exploiting the framework suggested by Zhu et al., we do not use K-means for time series clustering under DTW because of two reasons. First, we would like to avoid the problem of shape averaging with DTW which is still a challenging problem (Gupta et al., 1996 [2]; Niennattrakul and Ratanamahatana, 2007 [17]; Niennattrakul and Ratanamahatana, 2009 [18]; Petitjean et al., 2011 [20]). Second, K-medoids is more robust to noise and outliers than K-means.

K-medoids clustering belongs to the category of clustering called *partitional* clustering. Instead of averaging data objects as K-means, K-medoids chooses a data object from the cluster as its representative (called *medoid*). The medoid of a cluster is the most centrally located object of the cluster. There exist several variants of K-medoids based algorithms, and we have chosen the classic PAM (Partitioning Around Medoids) algorithm (Kaufman and Rousseeuw, 1990 [9]) since it does not requires any parameter settings beyond *k*.

Input: *k*: the number of clusters, *D*: the data set containing n objects.

Output: A set of *k* clusters

1. arbitrarily choose *k* objects in *D* as the initial medoids.
2. **repeat**
3. assign each remaining object to the cluster with the nearest medoid;
4. randomly select a non-medoid object, o_{random};
5. compute the new cost of the clustering when swapping the medoid o_j with o_{random};
6. **if** the new cost is less than the previous cost after the swapping **then** swap o_j with o_{random} to form a new set of *k* medoids
7. **until** no change.

Figure 3. PAM, a K-medoids partitioning algorithm.

The PAM algorithm is described briefly as in Figure 3. The algorithm begins by selecting randomly the representative objects for the clusters. Then it assigns each remaining object to the cluster with the nearest representative object. The iterative process of replacing representative objects by nonrepresentative objects continues as long as the quality of the resulting clustering is improved.

Notice that the cost of a clustering is computed as the average distance between an object to the medoid of its cluster.

To improve the anytime K-medoids clustering for time series under DTW, we apply two important techniques: (i) how to choose the initial medoids for K-medoids clustering and (ii) how to speed up the DTW computation.

3.1 Medoid Initialization

The performance of K-medoids algorithm may vary according to the method of selecting the initial medoids. There exist some possibilities of choosing the initial medoids which are listed as follows.

- Random selection: select *k* objects randomly from all objects.
- Sort all objects in the order of values of the chosen attributes. Divide the range of the above values into *k* equal intervals and select one object randomly from each interval.
- Sampling: take 10% random sampling from all objects and perform a preliminary clustering phase on these sampled objects using the proposed algorithm. The clustering result is used as the initial medoids.
- Outmost objects: Select *k* objects which are farthest from the center.

In this work, we apply the method of selecting the initial medoids proposed by Park and Jun (2009) [19] which is simple and effective. The technique is described as follows.

Input: $S = \{T_1, T_2, \ldots, T_n\}$ is the set of *n* time series that will be grouped into *k* ($k < n$} clusters and *d* is the distance matrix.

Output: *medoids* is the list of time series which are chosen as representatives of *k* clusters.

1. Using DTW distance as a dissimilarity measure, compute the distance between every pair of all time series in S as follows:
$$d_{ij} = DTW(T_i, T_j)$$
2. Calculate p_{ij} to make an initial guess at the centers of the clusters (this ratio is smaller if *i*-th time series is closer to the *j*-th time series).
$$p_{ij} = \frac{d_{ij}}{\sum_{l=1}^{nl} d_{il}} \quad i = 1,\ldots,n; j = 1,\ldots,n$$
3. Calculate $v_j = \sum_{i=1}^{n} p_{ij} \quad (j = 1,\ldots,n)$ at each time series *j* and sort them in ascending order, select *k* objects having the first *k* smallest values as initial cluster medoids.

This medoid initialization technique is easy to implement, efficient and works on a distance matrix. Park and Jun (2009) [19] empirically compared the proposed medoid initialization method to the four above-mentioned methods of medoid initialization (random, sorting, sampling, outmost objects) and the experimental results revealed that the proposed medoid initialization method outperforms the four other methods.

Notice that in this work, we employ the medoid initialization technique proposed by Park and Jun, but we do not use the variant of K-medoids algorithm proposed by them in [19]. In this work, we use the traditional K-medoids algorithm given by Kaufman and Rousseeuw [9].

3.2 Multithreading Technique for Speeding up DTW

In this work, we develop a user-level multithreading technique to speed up DTTW calculation. This technique is very effective in a multicore system. The main idea of this multithreading method is that each cell in the distance matrix for two time series represents

a computation which is dependent on the computational results of its upper cell and its left cells. Exploiting this data dependency, we apply multithreading combined with task parallelism to speed-up the DTW computation in the warping matrix. This technique requires a strategy of assigning threads to groups of matrix cells and let these threads working simultaneously. We have to restructure the DTW algorithm so as to extract parallelism from it.

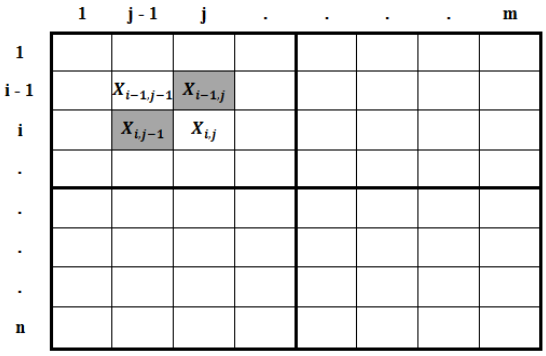

Figure 4. Illustration of multithreading technique to speed up DTW computation.

Figure 4 illustrates the main idea of the multithreading technique. We are given two time series with arbitrary lengths; the warping matrix is divided into 4 blocks, each block consists of 16 cells and the result of cell [i, j] depends on the results of the cell [i, j-1] and the cell [i -1, j]. Instead of computing each individual cell, we divide distance matrix into several blocks and apply task parallelism on these blocks and each block is viewed as a task or an execution thread. Hence, each block can be computed using the results from its upper block and its left block.

The details of the multithreading algorithm for speeding up DTW computation are left out due to space limitation. Interested reader can find them in [7].

Thank to this technique of thread-level parallelism, we can calculate DTW distance as exactly as the classical DTW way but with shorter running time.

4. EXPERIMENTAL EVALUATION

We implemented all the comparative methods with Microsoft Visual C# and conducted the experiments on an Intel(R) Core(TM) i3-2310M CPU @ 2.10 GHz (4 CPUs), 4GB RAM PC. The experiments aim to four purposes. First, we empirically study the effects of medoid initialization in clustering efficiency and the effects of multithreading technique in DTW computation. Second, we empirically study the effects of approximation technique in building DTW distance matrix. Third, we compare clustering quality of K-medoids algorithm in the four anytime methods. Finally, we compare the time efficiency of K-medoids in the four anytime methods.

For convenience, in this section, we denote the four anytime methods in K-medoid clustering as follows.

- Method 1: We initialize *a*DTW by using *DTW_Ratio* and arrange DTW updating order by *ED/LB*. This method is denoted as *DTW_Ratio + ED/LB*.
- Method 2: We initialize *a*DTW by using *DTW_Ratio* and arrange DTW updating order randomly. This method is denoted as *DTW_Ratio + Random*.
- Method 3: We initialize *a*DTW by using *ED* and arrange DTW updating order by *ED/LB*. This method is denoted as *ED + ED/LB*.
- Method 4: We initialize *a*DTW by using *ED* and arrange DTW updating order randomly. This method is denoted as: *ED + Random*.

4.1 Clustering Quality Evaluation Criteria

Evaluating clustering quality is not a trivial task since clustering is an unsupervised learning process. We use five classified datasets and one unclassified dataset in the experiment. We apply two different ways to evaluate clustering quality for two different kinds of datasets. As for classified datasets, we apply four objective clustering evaluation criteria: Jaccard, Rand, Folkes and Mallow (FM), and Adjusted Rand Index (ARI). The definition of ARI can be found in Hubert and Arabie (1995) [6]. The definitions of Jaccard, Rand, Folkes and Mallows can be found in (Halkidi et al., 2001 [3]). All the four criteria take value in the range [0, 1]; higher value means better clustering quality.

As for non-classified datasets, since M-medoids seeks to optimize the objective function by minimizing the sum of squared intra-cluster error, we can evaluate the quality of clustering by using the following objective function:

$$F = \sum_{i=1}^{M} \sum_{k \in A_i} \left\| x_k - c_i \right\|$$

where x_k is the objects and c_i are the cluster medoids.

4.2 Data Description

We conducted the experiment on six datasets: five classified datasets (Heterogeneous, Synthetic Chart Control, Cylinder-Bell-Funnel, FourFace, Trace) and one unclassified dataset (Stock).

Heterogeneous: This dataset is generated from a mixture of 10 real time series data from the UCR Time Series Data Mining Archive, 2015 [10]. Using the 10 time-series as seeds, we produced variation of the original patterns by adding small time shifting (2-3 % of the series length), and interpolated Gaussian noise. Gaussian noisy peaks are interpolated using splines to create smooth random variations. The length of each time series is 100.

Synthetic Control Chart Time Series (CC): This dataset has 100 instances from each of the six different classes of control charts. The length of each time series is 60. This data set is from the UCR Time Series Classification/Clustering Homepage, 2015 [12].

Cylinder-Bell-Funnel (CBF): This dataset contains three types of time series: cylinder (c), bell (b) and funnel (f). This is an artificial dataset originally proposed in Saito, 1994 [22]. The dataset consists of 128 time series for each class. The length of each time series is 128.

Trace: This dataset has four classes, each contains 25 instances. The length of each time series is 275.

FaceFour: This dataset has four classes, each contains 22 instances. The length of each time series is 350. Both Trace dataset and FaceFour dataset are from the UCR Time Series Data Mining Archive, 2015 [10].

Stock: This dataset has 1000 instances, the length of each time series is 128. This dataset is from the Historical Data for S&P 500 Stocks website [http://kumo.swcp.com/stocks/].

For normalization of the time series datasets, in this work, we apply *min-max normalization* (Han et al., 2012 [4]).

4.3 Effects of Medoid Initialization

To evaluate the effects of the medoid initialization technique to clustering process, we compare clustering performance in two cases: with or without initialization technique. We compare their performances in terms of run time over three datasets: Heterogeneous, Synthetic Control Chart and Trace. For K-medoids without initialization technique, we run the algorithm 100 times and take the average of their runtimes. We show the experimental results on the three datasets in Table 1.

Table 1. Running times of clustering with two medoid initialization ways

Dataset	Method 1	Method 2	Method 3	Method 4	Initialization
Heterogeneous	11.10	10.93	10.63	10.60	Park-based
	15.70	14.47	16.54	15.43	Random
CC	15.58	17.68	22.72	23.48	Park-based
	20.52	22.20	23.09	24.59	Random
Trace	0.118	0.118	0.093	0.096	Park-based
	0.146	0.12	0.096	0.101	Random

4.4 Effects of Multithreading Technique in DTW Calculation

To evaluate the effects of the multithreading technique to clustering process, we compare the running time of exact DTW distance matrix computation in two cases: DTW with or without multithreading technique. We compare the running time over five datasets: CBF, FaceFour, Heterogeneous, Synthetic Control Chart and Trace. We show the experimental results on the five datasets in Table 2.

Table 2. Running times of building exact DTW distance matrix with two ways to compute DTW distance

	CBF	Face-four	Hetero-geneous	CC	Trace
DTW_ Multithread	**27.66**	**68.82**	**92.65**	**190.62**	**56.14**
DTW_ Standard	52.75	141.21	139.39	236.74	116.19

From Table 2, we can see that running times (in seconds) of computing DTW distance matrix with multithreading technique (in bold) are always shorter than those with classical DTW computation. The experimental results indicate that multithreading technique improves the time efficiency of computing distance matrix remarkably. In average, the speedup of DTW with multithreading technique to classical DTW is about 1.75.

4.5 Effects of Approximation Technique in Building DTW Distance Matrix

In this experiment we compare the mean NDAE (the smaller the better, see Eq. (3) in Section 2.2) of all pairwise approximations by three methods: LB, ED and DTW_Ratio based approximation. Recall that the method DTW_Ratio needs to randomly sample a small fraction of true DTW distances. For each dataset, we calculated mean NDAE of all $(n\times(n\text{-}1))/2$ pairs approximations, repeating each test 100 times, and calculated the mean, as in Table 3. The sampling rate for each dataset varies from 10% to 20%.

Table 3. Mean of *NDAE* of all pairwise approximations by three DTW approximation methods, tested on 5 datasets

Dataset	Method	*n*%		
		10%	15%	20%
CBF	DTW_Ratio	**0.0599**	**0.0524**	**0.0492**
	ED	0.8347	0.7853	0.7363
	LB_Keogh	0.4179	0.3929	0.3681
Face-four	DTW_Ratio	**0.0725**	0.0683	**0.0627**
	ED	1.1464	1.0844	1.0212
	LB_Keogh	0.1565	0.1474	0.1383
Hetero-geneous	DTW_Ratio	**0.0872**	**0.0826**	**0.0777**
	ED	0.7789	0.7364	0.6938
	LB_Keogh	0.3048	0.2876	0.2706
CC	DTW_Ratio	**0.1438**	**0.1355**	**0.1272**
	ED	0.7180	0.6773	0.6367
	LB_Keogh	0.3304	0.3117	0.2930
Trace	DTW_Ratio	**0.1764**	**0.1665**	**0.1528**
	ED	1.3746	1.3150	1.1942
	LB_Keogh	0.4781	0.4588	0.4166

Table 4. Running time of three DTW approximation methods in building the distance matrix

Dataset	Method	*n*%		
		10%	15%	20%
CBF	DTW_Ratio	5.99	9.16	12.82
	ED	0.25	0.25	0.25
	LB_Keogh	1.45	1.45	1.45
Face-four	DTW_Ratio	6.91	14.56	20.53
	ED	0.23	0.23	0.2
	LB_Keogh	1.33	1.42	1.33
Hetero-geneous	DTW_Ratio	18.10	26.77	39.05
	ED	0.72	1.0	0.83
	LB_Keogh	4.56	6.21	5.54
CC	DTW_Ratio	49.61	69.58	91.71
	ED	1.92	1.78	2.04
	LB_Keogh	10.76	10.53	12.31
Trace	DTW_Ratio	9.22	18.60	31.47
	ED	0.22	0.23	0.31
	LB_Keogh	1.48	1.54	2.0

Experimental results in Table 3 show that the method DTW_Ratio performs the best. Besides, when the sampling rate increases, the mean NDAE decreases and the computational time increases. Table 4 reports the running time (in seconds) of each approximation method in building the distance matrix.

From Table 4, we can see that the DTW_Ratio method requires more time than the two other methods and ED is the fastest method.

4.6 Clustering Quality

We conducted an experiment on clustering quality of all four anytime methods over five datasets (CBF, FaceFour, Heterogeneous, Stock and CC) with four criteria (ARI, Rand, Jacckard and FM). But for brevity, in this subsection, we reported only the experimental results over one dataset and with two criteria (ARI and Jaccard).

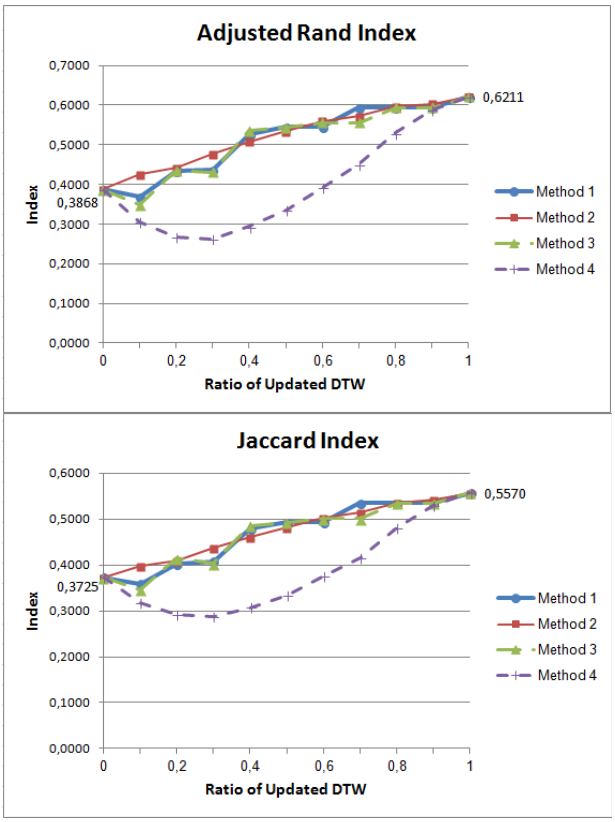

Figure 5. Clustering quality of four anytime methods on Trace dataset. Four anytime methods are compared by ARI and Jaccard.

Figure 5 shows the experimental results for the Trace dataset. We can see from Figure 5, Method 1 (DTW_Ratio + ED/LB) outperforms all the other method almost all time steps. The second finding is that the two methods using DTW_Ratio achieve a large improvement quite early, when just 60% of all DTW calculations have been performed. On the contrary, Method 4 (Random) is still very poor even after it has calculated 90% of all DTW distances. Another interesting finding is that Method 3 (ED/LB) still performs very well without boosting by DTW_Ratio; its clustering quality is sometimes better or very close to Method 2. Besides, the performance of Method 2 and Method 3 almost always resides between the other two methods.

For the Stock dataset, since it is not a classified dataset, we have to use objective function to evaluate clustering quality. Figure 6 shows the clustering quality of the four anytime methods on the Stock dataset. We can see that on this dataset, Method 1 (*DTW_Ratio + ED/LB*) and Method 3 (ED/LB) perform better than the two other anytime methods.

4.7 Running Time of Clustering

To evaluate the running times of the batch algorithm and the four methods of anytime K-medoids clustering, we conducted the experiment on the five datasets: Heterogeneous, Synthetic Control Chart, Cylinder-Bell-Funnel, FaceFour and Trace. Table 5 shows the running times (in seconds) of the five cases.

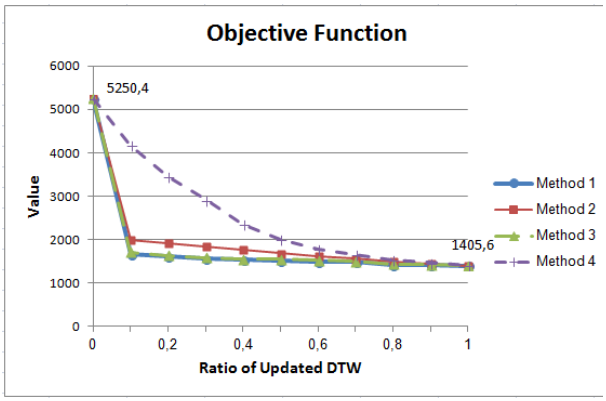

Figure 6. Clustering quality of four anytime methods on Stock dataset (by objective function).

From experimental results in Table 5, we can see that the running times (in seconds) of four anytime clustering methods are always slightly larger than the batch clustering method since the four anytime clustering methods have to do sampling to compute the mean DTW_Ratios and building the initial distance matrix *a*DTW. For larger datasets, for example *Heterogeneous* and *Control Chart*, this cost will increase. However, recall that the purpose of anytime algorithm is not to become faster than the batch algorithm, but to allow user interaction with the algorithm conveniently during its execution.

Table 5. Running times of clustering on five datasets

	CBF	Face-Four	Hetero-geneous	CC	Trace
Batch	28,052	69,295	104,516	224,872	57,135
Method 1	31,424	70,963	140,154	288,979	59,929
Method 2	31,233	72,184	141,932	288,162	61,52
Method 3	32,496	73,131	133,549	278,639	61,597
Method 4	33,591	70,911	138,265	244,406	61,012

5. CONCLUSIONS

There has been little research on time series clustering with DTW distance measure, especially the case of casting it into an *anytime* algorithm. In this paper, we propose an efficient method to implement an anytime K-medoids clustering for time series data with DTW distance. In our method, we exploit the anytime clustering framework with DTW proposed by Zhu et al. (2012), apply a method for medoid initialization, and develop a multithreading technique to speed-up DTW distance calculation. Experimental results on benchmark datasets reveal that our proposed implementation method for anytime time series K-medoids clustering with DTW can perform very fast and bring out high clustering quality.

As for future work, we plan to apply some improved variant of K-medoids algorithm (e.g. by Park and Jun [19]) in our proposed method. Besides, we intend to apply anytime K-medoids clustering of time series with DTW in developing an anytime algorithm that can discover motif and anomaly at the same time in time series data with DTW distance.

6. ACKNOWLEDGMENTS

We are grateful to Prof. Eamonn J. Keogh for kindly providing necessary datasets for this research work.

This research is funded by Vietnam National University Ho Chi Minh City (VNU-HCM) under grant number C2016-20-04.

7. REFERENCES

[1] Berndt D. J. and Clifford J. 1994. Using Dynamic Time Warping to Find Patterns in Time Series. In *Proceedings of AAA1-94 Workshop on Knowledge Discovery in Databases*, Seattle, Washington, 229 – 248.

[2] Gupta L., Molfese, D. L., Tammana, R. and Simos, P.G. 1996. Nonlinear alignment and averaging for estimating the evoked potential. *IEEE Transactions on Biomedical Engineering*, vol.43, 348 – 356.

[3] Halkidi, M., Batistakis, Y., Vazirgiannis. 2001. On clustering validation techniques. *Journal of Intelligent Information Systems*, Vol. 17, No. 2-3, 107-145.

[4] Han, J., Kamber, M. and Pei, J. 2012. *Data Mining: Concepts and Techniques*, 3th Ed.. Morgan Kaufmann.

[5] Hautamaki, V., Nykanen, P., Franti, P., 2008. Time series clustering by approximate prototypes. In *Proc. of 19th International Conference on Pattern Recognition*.

[6] Hubert, L. and Arabie, P. 1985. Comparing partitions. *Journal of Classification*, vol.2, 193 – 218.

[7] Huy, V.T. 2015. *Anytime time series clustering under DTW based on an approximation method*. Master thesis, Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology.

[8] Itakura, F. 1975. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol.23, 67 – 72.

[9] Kaufman, L. and Rousseeuw, P.L. 1990. *Finding groups in data: An introduction to cluster analysis*. New York: Wiley.

[10] Keogh, E. and Folias, T. The UCR Time Series Data Mining Archive, [http://www.cs.ucr.edu/~eamonn/TSDMA/index.html]. Assessed in Jan. 2015.

[11] Keogh, E. and Ratanamahatana, C.A. 2002. Exact indexing of dynamic time warping. In *Proc. of 28th International Conference on Very Large Databases*, 406 – 417.

[12] Keogh, E., Zhu, Q., Hu, B., Hao, Y., Xi, X., Wei, L. and Ratanamahatana, C.A. The UCR Time Series Classification/ Clustering Homepage, [www.cs.ucr.edu/~eamonn/time_series_data/]. Accessed in Jan. 2015.

[13] Kim, S., Park, S. and Chu, W.W. 2001. An index-based approach for similarity search supporting time warping in large sequence databases. In *Proceedings of 17th International Conference on Data Engineering*, 607 – 614.

[14] Lemire, D. 2009. Faster retrieval with a two-pass dynamic-time-warping lower bound. *Pattern Recognition*, Vol. 42, No. 9, 2169-2180.

[15] Li, J., Wang, Y. 2009. Early Abandon to Accelerate Exact Dynamic Time Warping. *The International Arab Journal of Information Technology*, Vol. 6, No. 2, April.

[16] Lin, J., Vlachos, M., Keogh, E. and Gunopulos, D. 2004. Iterative Incremental Clustering of Time Series. In *Proc. of 9th International Conference on Extending Database Technology*, 106 – 122.

[17] Niennattrakul, V. and Ratanamahatana, C.A. 2007. Inaccuracies of shape averaging method using dynamic time warping for time series data. In *Proc. of 7th International Conference on Computational Science*, 513 – 520.

[18] Niennattrakul, V. and Ratanamahatana, C.A. 2009. Shape averaging under Time Warping. In *Proc. of 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, Pattaya, vol.02, 626 – 629.

[19] Park, H., Lee, J. and Jun C. 2009. A simple and fast algorithm for K-medoids clustering. *Expert Systems with Applications*, Vol. 36, 3336-3341.

[20] Petitjean, F., Ketterlin, A. and Gancarski P. 2011. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, vol. 44, 678 – 693.

[21] Sakoe, H. and Chiba, S. 1978. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol.26, 43 – 49.

[22] Saito, H. 1994. Local Feature Extraction and Its Application Using a Library of Bases. *PhD thesis*, Department of Mathematics, Yale University.

[23] Yi, B. K., Jagadish, H.V. and Faloutsos, C. 1998. Efficient retrieval of similar time sequences under time warping. In *Proc. of 14th International Conference on Data Engineering*, Florida, 201-208.

[24] Zhu, Q., Batista, G., Rakthanmanon, T. and Keogh, E. 2012 A Novel Approximation to Dynamic Time Warping allows Anytime Clustering of Massive Time Series Datasets. In *Proceedings of SDM*, 999-1010.

[25] Zilberstein, S. 1996. Using anytime algorithms in intelligent systems. *AI Magazine,* 17(3):73-83.