Bandwidth-Efficient Distributed k-Nearest-Neighbor Search with Dynamic Time Warping

Chin-Chi Hsu[*], Perng-Hwa Kung[†], Mi-Yen Yeh[*], Shou-De Lin[‡] and Phillip B. Gibbons[§]
[*]*Institute of Information Science, Academia Sinica, Taiwan*
Email: chinchi@iis.sinica.edu.tw, miyen@iis.sinica.edu.tw
[†]*The Media Lab, Massachusetts Institute of Technology*
Email: pernghwa@media.mit.edu
[‡]*Department of Computer Science and Information Engineering, National Taiwan University, Taiwan*
Email: sdlin@csie.ntu.edu.tw
[§]*Computer Science Department and Electrical & Computer Engineering Department, Carnegie Mellon University*
Email: gibbons@cs.cmu.edu

***Abstract*—We study the fundamental k-nearest neighbor (kNN) search problem on distributed time series. A server has constantly received various reference time series Q of length X and seeks the exact kNN over a collection of time series distributed across a set of M local sites. When X and M are large, and when the amount of query increases, simply sending each Q to all M sites incurs high communication bandwidth costs, which we would like to avoid. Prior work has presented a communication-efficient kNN algorithm for the Euclidean distance similarity measure. In this paper, we present the first communication-efficient kNN algorithm for the dynamic time warping (DTW) similarity measure, which is generally believed a better measure for time series. To handle the complexities of DTW, we design a new multi-resolution structure for the reference time series, and multi-resolution lower bounds that can effectively prune the search space. We present a new protocol between the server and the local sites that leverages multi-resolution pruning for communication efficiency and cascading lower bounds for computational efficiency. Empirical studies on both real-world and synthetic data sets show that our method reduces communication bandwidth by up to 92%.**

I. Introduction

This paper studies the kNN problem for distributed time series. The system model is shown in Figure 1. Here we have a server that can communicate to all M geographically distributed sites, each of which collects multiple data readings from the sensors it monitors. The server, acting like a time-series search engine, accepts query references Q in a time series form from users and for each single query searches for the k nearest time series across all sites. Taking a large-scale environment monitoring system for example, it is composed of a central server and many local stations with many sensors detecting certain environmental level. Suppose a scientist detects several unusual and potentially dangerous events such as the dramatic oscillation of CO_2 level or the increasing ambient concentration of some hazardous material, he or she may want to issues warnings to the location where similar events had happened.

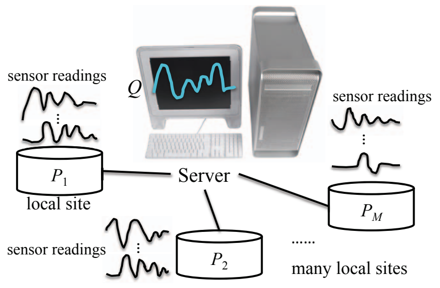


Figure 1. The system model

Under the system model, the main goal is to answer exact kNN queries in a communication efficient manner. Our primary cost metric is the total number of bytes exchanged between the server and the local-to-the-sensors sites to answer the query. An obvious solution is to send the entire reference Q to all M local sites. After receiving Q, each local site computes the distances of all the candidate time series it monitored to the reference and sends these values back to the server. The server can then determine the global k nearest and further retrieve those time series. However, in real applications, the number of sites M can be huge. For example, a city-wide environment monitoring system can have thousands of sensors distributed in different locations. Also, the length of the reference time series Q can be long as well. For example, a short period of observations can have hundreds to thousands of readings if the sampling rate is high. More importantly, the server usually will receive many requests, i.e., the number of queries will be large. Under such conditions, sending all Qs to all the local sites, which then respond with distances for all the candidates for all these Qs, will consume huge bandwidth.

People might argue that one can always compress the query before sending it. We will show through experiments that our method is non-exclusive to any loseless compression, and thus brings further bandwidth saving after compression.

Prior work has considered exact kNN search for the same system model, presenting a communication-efficient approach for the Euclidean distance similarity measure [1][2]. In the work, the server decomposes the reference time series into multi-resolution Haar wavelet coefficients and sends these coefficients from the coarsest level to the finest to the distributed sites. As more levels are transmitted, an increasingly tighter similarity range, bounded by upper and lower bounds, can be derived for each candidate to the reference at either the server or the local sites, until the exact kNN are determined. Although Haar wavelet is good for decomposing the query reference, their solution is very much tied to the Euclidean distance measure.

In this paper, we consider the dynamic time warping (DTW) distance, which is generally believed a better measure for the similarity between two time series. It is widely applied in many applications such as speech recognition, information retrieval on music and motion, and various mining tasks [3], [4], [5], to name but a few. In essence, the DTW distance finds the optimal match between two time series by warping them non-linearly, both stretching and shifting, in the time dimension. Although the DTW measure is powerful, its computation complexity is quadratic in the length of the time series. To accelerate the task of finding kNN using the expensive DTW measure, a common way is to do early pruning of unqualified candidates using computationally cheaper lower bounds [6], [7], [8], [9], [5], [10]. Rakthanmanon et al. [5] have pointed out several important lower bounds for the DTW distance and discussed the tradeoff between their tightness and computational complexity. They further proposed a cascading lower bounding technique to prune the unpromising candidates more efficiently: loose but computationally-cheap bounds are used to do the initial pruning, followed-by tight but computationally-costly bounds. To our knowledge, all existing work is based on a centralized computing environment, where all candidates are in the same location.

We present the first communication-efficient distributed kNN algorithm for the DTW similarity measure. We cannot leverage the Haar wavelet approach of the prior work for the Euclidean distance measure, because Haar wavelet coefficients can only be used to compute the approximate DTW distance without any bound guarantee [11]. To handle the complexity of DTW, we instead present a framework based on the following insights. First, *we design a new multi-resolution structure for the reference time series.* At the coarsest level, the time series is divided into several disjoint segments, each of which records only its maximum and minimum values. Then, each segment is further divided into a fixed number of segments to become the finer level to represent more detailed information of the reference time series. Based on the multi-resolution structure, the server iteratively sends the information of Q in a level-wise manner starting with the coarsest resolution (fewest bytes sent) and continuing with increasingly finer resolutions (more bytes sent). Second, *we further derive three lower bounds* to be used for cascaded pruning. Two of them are multi-resolution versions of existing DTW lower bounds, while the third is *a newly designed lower bound.* We prove that all three bounds will become tighter and close to the exact DTW distance as we move from the coarser level to the finer one. Third, *we design a new protocol between the server and the local sites* that leverages multi-resolution pruning for communication efficiency and cascading lower bounds for computational efficiency. Experiment results on both real-world and synthetic data show that our solution significantly saves the total bandwidth consumption in a variety of different setups for searching kNN using the DTW distance. Note that all the supplemental materials, including test data and codes, can be downloaded from [12].

Our main contributions can be summarized as follows. (1) We present a communication-efficient framework to process the exact kNN query using the DTW distance in a distributed environment. To our knowledge, this is the first solution proposed for such purpose. (2) We propose a new multi-resolution structure for decomposing the query reference. Based on the structure, cascading lower bounds, including the existing lower bounds and our newly designed one, can then be exploited to prune candidates for more efficient search without compromising correctness. (3) We conduct extensive experiments that demonstrate the significant bandwidth savings of our framework.

II. Preliminaries

Dynamic Time Warping. Suppose we have a reference $Q = \{q_1, q_2, ..., q_X\}$ of length X and a candidate time series $C = \{c_1, c_2, ..., c_Y\}$ of length Y, their *dynamic time warping (DTW) distance* is computed as follows.

$$DTW(Q, C) = f(X, Y), \tag{1}$$

where

$$f(i,j) = \begin{cases} 0, & \text{if } i = j = 0, \\ \infty, & \text{if } i = 0 \text{ or } j = 0, \\ d(q_i, c_j) + min\{f(i, j-1), & \\ f(i-1, j), f(i-1, j-1)\}, & \text{otherwise,} \end{cases} \tag{2}$$

$$d(q_i, c_j) = (q_i - c_j)^2, i = 1, 2, ..., X, \text{ and } j = 1, 2, ..., Y.$$

In essence, the DTW distance finds the optimal match between two time series by warping them non-linearly in the time dimension with the quadratic computation complexity of $O(XY)$.

III. Methodology

A. Overview

Figure 2 demonstrates the outline of our framework. Our key ideas of reducing the bandwidth consumption for the distributed kNN query with the DTW distance contains three parts: first, we quickly identify a good initial threshold

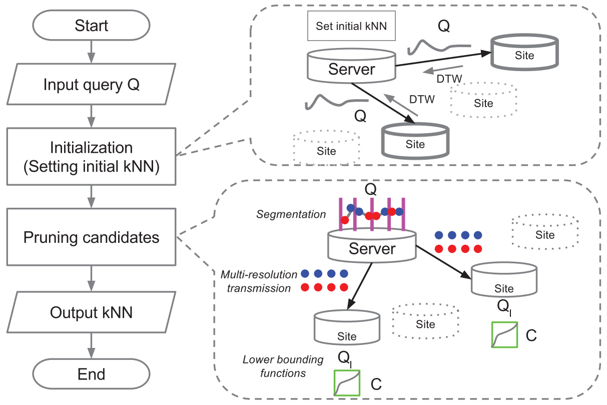


Figure 2. Big picture of the framework

for pruning unpromising candidates; second, we send the current threshold and only the partial information of the reference, i.e., the coarse $Q^{(l)}$ at certain resolution l, to a portion of local sites at each communication round; third, we let local sites to compute the lower bounds of the DTW distance between each candidate and the reference reconstructed by the $Q^{(l)}$ they have received so far so that the candidate pruning can be achieved without causing any false dismissals. By few iterations of the second and third steps, the candidate time series and even the local sites can be quickly pruned and thus the final k can be obtained without consuming much bandwidth.

In the following, we first describe the concept of multi-resolution structure used for our framework in Sec. III-B. Then we depict how the multi-resolution data are communicated between the server and the local sites in Sec. III-C, how the lower bounds, which are used to prune candidates, are computed based on the multi-resolution representation of Q in Sec. III-D, and finally the entire flow of the proposed framework in Sec. III-E.

B. Multi-Resolution Segmentation and Reconstruction

Here we introduce the way of decomposing the reference Q into a multi-resolution representation at the server and the way of constructing the reference given the partial information of Q at a local site.

Basically, we transform Q into multiple levels of disjoint segments. Each segment is denoted as $seg_{(l,p)} = [max, min, \ len]$, where l represents the level number, p represents its position number in that level, max (min) are the maximum (minimum) values of the original time series covered by the segment, and len is the length of the segment. A predefined number N determines the number of segments at the first level. Then, we cut each segment evenly into R disjoint equal-size ones and get the next level. An example given in Figure 3, where $N = 2$, and $R = 2$, shows the multi-representation of Q={5,2,4,3,4,7,1,3}. Without loss of generality, in the following examples we always show

Q	5	2	4	3	4	7	1	3
$l = 1$	seg$_{(1,1)}$=[5,2,4]				seg$_{(1,2)}$=[7,1,4]			
$l = 2$	seg$_{(2,1)}$=[5,2,2]		seg$_{(2,2)}$=[4,3,2]		seg$_{(2,3)}$=[7,4,2]		seg$_{(2,4)}$=[3,1,2]	
$l = 3$	seg$_{(3,1)}$= [5,5,1]	seg$_{(3,2)}$= [2,2,1]	seg$_{(3,3)}$= [4,4,1]	seg$_{(3,4)}$= [3,3,1]	seg$_{(3,5)}$= [4,4,1]	seg$_{(3,6)}$= [7,7,1]	seg$_{(3,7)}$= [1,1,1]	seg$_{(3,8)}$= [3,3,1]

Figure 3. The multi-resolution representation of Q, where the triples are $[max, min, len]$

Q	{5,2,4,3,4,7,1,3}
$Q_U^{(1)}$ $Q_L^{(1)}$	{5,5,5,5,7,7,7,7} {2,2,2,2,1,1,1,1}
$Q_U^{(2)}$ $Q_L^{(2)}$	{5,5,4,4,7,7,3,3} {2,2,3,3,4,4,1,1}
$Q_U^{(3)}$ $Q_L^{(3)}$	{5,2,4,3,4,7,1,3} {5,2,4,3,4,7,1,3}

Figure 4. Reconstructed upper and lower query references at different levels

$N = 2$ and $R = 2$ for ease of exposition. However, our framework also works when N and R equal to other values. Please refer to [12] for these possible cases.

There are some constraints on the relations of different resolution levels. First, segments at the same level are all disjoint, i.e., no data point belongs to more than one segment at the same level. Second, every boundary between two segments at the resolution level-l has to be preserved at the resolution level-$(l + 1)$ as shown in Figure 3. Third, the length/size of the segments at the first level can be either equal or unequal. For ease of exposition, we show the equal size case in Figure 3. However, we may also apply an existing time series segmentation algorithm such as [13]. The intuition of unequal-sized segmentation is to minimize the different between the maximum and minimum value of each segment so that we can have a better lower bounding effect and thus save more bandwidth. We will show different pruning effects of equal and unequal segmentations in the experiments. Finally, for the N value, we can set it as $N = \sqrt{\frac{|Q|}{2}}$ as a rule of thumb.

Given the level-l representation, we can reconstruct the query reference at resolution l, denoted as $Q^{(l)}$, which is composed of an upper bound $Q_{ub}^{(l)} = \{q_{ub,1}^{(l)}, ..., q_{ub,X}^{(l)}\}$ and a lower bound $Q_{lb}^{(l)} = \{q_{lb,1}^{(l)}, ..., q_{lb,X}^{(l)}\}$. Basically, $Q_{ub}^{(l)}$ and $Q_{lb}^{(l)}$ are constructed by repeating the max and min values by len times of each segment at level l, respectively. For example, given the level-1 representation shown in Figure 3, $Q_{ub}^{(1)}$ is constructed by repeating the $max = 5$ value $len = 4$ times for $seg_{(1,1)}$ and the $max = 7$ value $len = 4$ times for $seg_{(1,2)}$. The reconstruction results for all levels is shown in Figure 4.

Obviously, $Q_{ub}^{(l)}$ and $Q_{lb}^{(l)}$ have the same length as the original reference Q and we have

$$q_{lb,i}^{(l)} \leq q_i \leq q_{ub,i}^{(l)}, \forall 1 \leq i \leq X \text{ and } \forall 1 \leq l \leq L,$$

where L is the number of levels of the multi-resolution representation. In addition, the two sequences form an *envelope* of Q, which leads to the following several theorems about the lower bounding distance between data points of Q and a candidate time series C. For ease of exposition, let $q_i^{(l)} = (q_{lb,i}^{(l)}, q_{ub,i}^{(l)})$ and we have the following theorems.

Theorem 1. *The width of envelope is more narrow at a higher resolution level. That is,*

$$q_{lb,i}^{(l+1)} \geq q_{lb,i}^{(l)} \text{ and } q_{ub,i}^{(l+1)} \leq q_{ub,i}^{(l)} \tag{3}$$

for any resolution level l.

Proof: Suppose $q_{lb,i}^{(l+1)}$ and $q_{ub,i}^{(l+1)}$ are in $seg_{(l+1,p_x)}$ and $q_{lb,i}^{(l)}$ and $q_{ub,i}^{(l)}$ are in $seg_{(l,p_y)}$. According to our segmentation, $seg_{(l+1,p_x)}$ at the level $l+1$ is contained in $seg_{(l,p_y)}$ at the level l. Therefore the min and max values of $seg_{(l,p_y)}$ are lower and upper bounds of the min and max values of $seg_{(l+1,p_x)}$, respectively. Since $q_{lb,i}^{(l+1)}$ and $q_{ub,i}^{(l+1)}$ are the duplications of min and max value in $seg_{(l+1,p_x)}$ and $q_{lb,i}^{(l)}$, $q_{ub,i}^{(l)}$ are the duplications of min and max values of $seg_{(l,p_y)}$, the inequalities hold. ■

Theorem 2. *There is a lower bound d_{lb} of the square difference between q_i, the i-th data point of Q, and c_j, the j-th data point of a candidate time series C. That is,*

$$\begin{aligned} d_{lb}(q_i^{(l)}, c_j) &= \begin{cases} (q_{ub,i}^{(l)} - c_j)^2, & \text{if } q_{ub,i}^{(l)} < c_j, \\ (q_{lb,i}^{(l)} - c_j)^2, & \text{if } q_{lb,i}^{(l)} > c_j, \\ 0, & \text{otherwise.} \end{cases} \\ &\leq (q_i - c_j)^2 \\ &= d(q_i, c_j) \end{aligned} \tag{4}$$

for any resolution level l.

Proof:

$$\begin{cases} (q_{ub,i}^{(l)} - c_j)^2 \leq (q_i - c_j)^2, & \text{if } q_i \leq q_{ub,i}^{(l)} < c_j, \\ (q_{lb,i}^{(l)} - c_j)^2 \leq (q_i - c_j)^2, & \text{if } q_i \geq q_{lb,i}^{(l)} > c_j, \\ 0 \leq (q_i - c_j)^2, & \text{otherwise.} \end{cases} \tag{5}$$

■

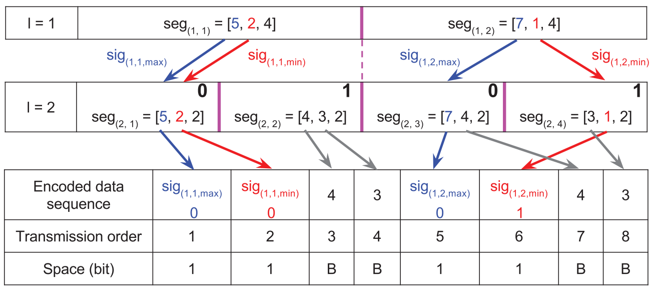
Theorem 3. *The lower bound distance between two data points is increasing at higher resolution levels. That is,*

$$d_{lb}(q_i^{(l+1)}, c_j) \geq d_{lb}(q_i^{(l)}, c_j). \tag{6}$$

Proof: By Theorem 1, we have $q_{lb,i}^{(l+1)} \geq q_{lb,i}^{(l)}$ and $q_{ub,i}^{(l+1)} \leq q_{ub,i}^{(l)}$. The following cases prove the inequality.

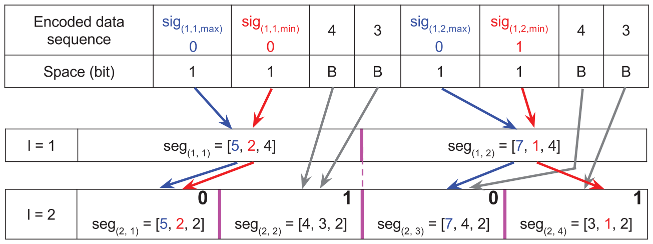
$$\begin{cases} (q_{ub,i}^{(l+1)} - c_j)^2 \geq (q_{ub,i}^{(l)} - c_j)^2, & \text{if } q_{ub,i}^{(l+1)} \leq q_{ub,i}^{(l)} < c_j, \\ (q_{lb,i}^{(l+1)} - c_j)^2 \geq (q_{lb,i}^{(l)} - c_j)^2, & \text{if } q_{lb,i}^{(l+1)} \geq q_{lb,i}^{(l)} > c_j, \\ (q_{ub,i}^{(l+1)} - c_j)^2 \geq 0, & \text{if } q_{ub,i}^{(l+1)} < c_j \leq q_{ub,i}^{(l)}, \\ (q_{lb,i}^{(l+1)} - c_j)^2 \geq 0, & \text{if } q_{lb,i}^{(l+1)} > c_j \geq q_{lb,i}^{(l)}, \\ 0 \geq 0, & \text{if } q_{lb,i}^{(l+1)} \leq c_j \leq q_{ub,i}^{(l+1)}. \end{cases} \tag{7}$$

■



Encoded data sequence	sig(1,1,max) 0	sig(1,1,min) 0	4	3	sig(1,2,max) 0	sig(1,2,min) 1	4	3
Transmission order	1	2	3	4	5	6	7	8
Space (bit)	1	1	B	B	1	1	B	B

Figure 5. Data encoding in the server



Encoded data sequence	sig(1,1,max) 0	sig(1,1,min) 0	4	3	sig(1,2,max) 0	sig(1,2,min) 1	4	3
Space (bit)	1	1	B	B	1	1	B	B

Figure 6. Data decoding in sites

C. Multi-resolution Transmission

The transmission of the query reference Q starts from level 1 to level L. At level 1, the server sends $seg_{(1,p)} = [min, max, len]$, where $1 \leq p \leq N$. When sending the reference from level-l to level-$(l+1)$ to the same site, where $l \geq 1$, we can find there are redundancies between the two levels. For example, in Figure 3, the $max = 5$ and $min = 2$ are shared between $seg_{(1,1)}$ and $seg_{(2,1)}$. In this case, we only need to send the *additional information* between two consecutive levels. Therefore, we propose how to encode the information between different levels at the server and the way of encoding it at the local sites.

To encode the multi-resolution representation of Q at level $l+1$, here we introduce a $\lceil \log_2 R \rceil$-bit signal sig to indicate that the max and min values in the segments at level l should be copied to which newly generated segments at level $l+1$. We first encode R signals and then the sequences of new max and min values. We show an example of encoding level 2 of Q in Figure 5, where $R = 2$ and $\lceil \log_2 R \rceil = 1$. In this case, as one segment at level 1 is divided into two segments at level 2, $sig = 0$ indicates the first such segment and $sig = 1$ indicates the second. As the $seg_{(2,1)}$ segment shares the same max and min value from $seg_{(1,1)}$, the first two signals in the sequence is all zero, followed by the new values $max = 4$ and $min = 3$. As the max and min values in $seg_{(1,2)}$ are shared with the first divided segment $seg_{(2,3)}$ and the second one $seg_{(2,4)}$, the next two signals are 0 and 1, followed by the new values $max = 4$ and $min = 3$. Suppose for each data value we use B bits to encode it, say 32 bits in usual, we save much space for encoding each

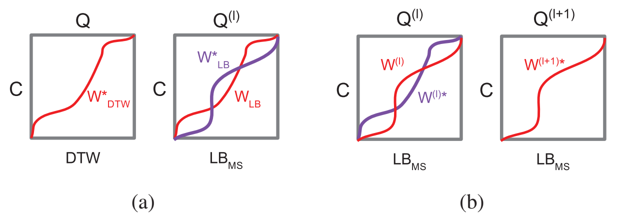


Figure 7. LB_{MS} (a) is a lower bounding function, (b)increases as the level increases

duplication from $B = 32$ bits to $\lceil \log_2 R \rceil = 1$ bit.

Suppose a site can know the R, N and B values at the first time when it communicates with the server. After receiving the encoded data sequence at level l, the local site can decode the sequence by reading it in a sequential way. Continue the last example, the decoding is shown in Figure 6. As the site attempts to split $seg_{(1,1)}$ into $seg_{(2,1)}$ and $seg_{(2,2)}$, it reads two $\lceil \log_2 R \rceil = 1$-bit signals, all equals to 0, and knows to assign the $max = 5$ and $min = 2$ to corresponding sub-segments $seg_{(2,1)}$. Then the site continues to read two data points of B bits to fill in the unknown values of $seg_{(2,2)}$. In addition, lengths of segments at level 2 can be easily calculated given the lengths of segments at level 1. Consequently, the site can obtain and reconstruct the level-2 query reference.

D. Lower Bounding Functions

In this section, we show how to derive various lower bounds using the query reference at different resolution levels at the local sites. First, we introduce a DTW-based multi-resolution distance function of the query reference at resolution l and a candidate C, denoted as $LB_{MS} = (Q^{(l)}, C)$, as follows.

$$\begin{aligned} LB_{MS}(Q^{(l)}, C) &= D_{lb}(q_X^{(l)}, c_Y) \\ D_{lb}(q_i^{(l)}, c_j) &= d_{lb}(q_i^{(l)}, c_j) + \min \left\{ \begin{array}{l} D_{lb}(q_{i-1}^{(l)}, c_j) \\ D_{lb}(q_i^{(l)}, c_{j-1}) \\ D_{lb}(q_{i-1}^{(l)}, c_{j-1}) \end{array} \right. , \end{aligned} \tag{8}$$

where $d_{lb}(q_i^{(l)}, c_j)$ is defined in Eq. (4).

Theorem 4. *$LB_{MS}(Q^{(l)}, C)$ is a lower bound of $DTW(Q, C)$. That is,*

$$LB_{MS}(Q^{(l)}, C) \leq DTW(Q, C), \forall 1 \leq l \leq L \tag{9}$$

, where L is the total level number of the multi-resolution representation of Q.

Proof: Suppose W^*_{DTW} is the optimal warping path for computing $DTW(Q, C)$ and we copy the same path to the warping table of $LB_{MS}(Q^{(l)}, C)$ and denote it as W_{LB} as shown in Figure 7(a). Let $LB_{MS}(W_{LB})$ denote the summation of $d_{lb}(q_i^{(l)}, c_j)$ of all cells on the warping path W_{LB} and $DTW(W^*_{DTW})$ denote the summation of $d(q_i, c_j)$ of all cells along the warping path W^*_{DTW}. Apparently,

$$LB_{MS}(W_{LB}) \leq DTW(W^*_{DTW}) \tag{10}$$

because $d_{lb}(q_i^{(l)}, c_j) \leq d(q_i, c_j)$ for each corresponding pair (q_i, c_j), by Theorem 2, on the warping path.

Let W^*_{LB} be the optimal warping path for $LB_{MS}(Q^{(l)}, C)$. Clearly,

$$\begin{aligned} LB_{MS}(Q^{(l)}, C) &= LB_{MS}(W^*_{LB}) \\ &\leq LB_{MS}(W_{LB}) \\ &\leq DTW(W^*_{DTW}) \\ &= DTW(Q, C) \end{aligned} \tag{11}$$

■

Theorem 5. *$LB_{MS}(Q^{(l)}, C)$ is increasing as l increases. That is,*

$$LB_{MS}(Q^{(l)}, C) \leq LB_{MS}(Q^{(l+1)}, C) \tag{12}$$

Proof:

As shown in Figure 7(b), let W^*_l be the optimal warping path for $LB_{MS}(Q^{(l)}, C)$, and $W^{(l+1)*}$ for $LB_{MS}(Q^{(l+1)}, C)$. In addition, let $W^{(l)}$ be the same path copied from $W^{(l+1)*}$ to the warping table of $LB_{MS}(Q^{(l)}, C)$. Then we have

$$LB_{MS}(W^{(l)}) \leq LB_{MS}(W^{(l+1)*}), \tag{13}$$

because $d_{lb}(q_i^{(l)}, c_j) \leq d_{lb}(q_i^{(l+1)}, c_j)$ for each corresponding pair (q_i, c_j), by Theorem 3, on the warping path. Therefore,

$$\begin{aligned} LB_{MS}(Q^{(l)}, C) &= LB_{MS}(W^{(l)*}) \\ &\leq LB_{MS}(W^{(l)}) \\ &\leq LB_{MS}(W^{(l+1)*}) \\ &= LB_{MS}(Q^{(l+1)}, C) \end{aligned} \tag{14}$$

■

Theorem 5 guarantees that LB_{MS} is increasing and thus close to the exact DTW distance if higher resolution data of Q are transmitted.

In fact, the LB_{MS} bound is a special case of LB_{FTW} proposed by Sakurai et al. [10]. There is a main difference between these two bounds. We do the multi-resolution segmentation on the query reference only, while in [10] both the reference and the candidate are decomposed to a coarse version. Since we are allowed to access all data points of candidates in local machines, LB_{MS} utilizing full candidate information has higher chance to achieve a tighter bound than LB_{FTW} that does not exploit all data points of candidates. Under the main goal of bandwidth saving, we have to prune candidates as many as possible at a given resolution level of query reference, and hence a tighter lower bounding function is what we need.

The computation cost of LB_{MS} is O(XY), which is high. Rakthanmanon et al. [5] suggested to use cascading lower bounds to save the computation cost. The idea is to leverage lower bounds that are loose but have low computation costs to do the early pruning first, and then to use lower

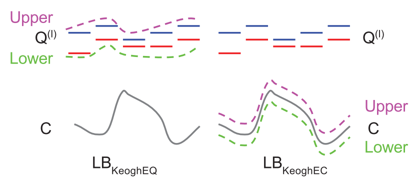


Figure 8. $LB_{KeoghEQ}$ and $LB_{KeoghEC}$

bounds that are tight but have high computation costs to do subsequent pruning. According to [5], LB_{Keogh} and LB_{KimFL} are two other possible choices of lower bounds, of which the computation cost is $O(X+Y)$ and $O(1)$, respectively.

In the remainder of this section, we show that the lower bounds LB_{Keogh} and LB_{KimFL} can be used in our multi-resolution Q framework. The required condition is that $LB(Q^{(l)}, C)$ is a lower bound for $DTW(Q, C)$ that increases as the resolution of Q gets higher. We prove that both lower bounds satisfy this condition.

According to [5], LB_{KimFL} of the query reference Q at level l and the candidate C can be computed as follows.

$$LB_{KimFL}(Q^{(l)}, C) = \max\{d_{lb}(q_1^{(l)}, c_1), d_{lb}(q_X^{(l)}, c_Y)\}. \tag{15}$$

We prove that $LB_{KimFL}(Q^{(l)}, C)$ is a lower bound of $DTW(Q, C)$ and it increases as l increases. Due to page limits, please refer to our supplemental material [12].

To compute LB_{Keogh}, an envelope, composed of an $Upper$ sequence and a $Lower$ sequences, should be built around the query reference, denoted as $LB_{KeoghEQ}$, or the candidate, denoted as $LB_{KeoghEC}$. We show how to compute these two values using $Q^{(l)}$ and C, which is illustrated in Figure 8.

The computation of $LB_{KeoghEQ}$ is as follows.

$$\begin{aligned}
&upper_i^{(l)} = \max_{t=i-r\ldots i+r} q_{ub,t}^{(l)}\\
&lower_i^{(l)} = \min_{t=i-r\ldots i+r} q_{lb,t}^{(l)}\\
&LB_{KeoghEQ}(Q_l, C)\\
&= \sum_{j=1}^{X} \begin{cases} (c_j - upper_j^{(l)})^2 & \text{if } upper_j^{(l)} < c_j \\ (c_j - lower_j^{(l)})^2 & \text{if } lower_j^{(l)} > c_j \\ 0 & \text{otherwise} \end{cases}\\
&= \sum_{j=1}^{X} d_{lb}(qe_j^{(l)}, c_j), \text{ where } qe_j^{(l)} = (upper_j^{(l)}, lower_j^{(l)})
\end{aligned} \tag{16}$$

The computation of $LB_{KeoghEC}$ is as follows. $LB_{KeoghEC}$

$$\begin{aligned}
&upper_j = \max_{t=j-r\ldots j+r} c_t\\
&lower_j = \min_{t=j-r\ldots j+r} c_t\\
&LB_{KeoghEC}(Q^{(l)}, C)\\
&= \sum_{i=1}^{X} \begin{cases} (q_{lb,i}^{(l)} - upper_i)^2 & \text{if } upper_i < q_{lb,i}^{(l)} \\ (q_{ub,i}^{(l)} - lower_i)^2 & \text{if } lower_i > q_{ub,i}^{(l)} \\ 0 & \text{otherwise} \end{cases}\\
&= \sum_{i=1}^{X} d_{lb}(q_i^{(l)}, ce_i), \text{ where } ce_i = (upper_i, lower_i)
\end{aligned} \tag{17}$$

The parameter r is related to constraints and the detail is referred to the paper [7]. Similarly, we prove that LB_{Keogh} can be well included in our framework in [12].

Finally, we merge the two versions of LB_{Keogh} into one:

$$LB_{Keogh} = \max\{LB_{KeoghEQ}, LB_{KeoghEC}\} \tag{18}$$

E. The Communication Protocol

Now we can describe the flow of our proposed framework to solve the distributed kNN queries using the DTW distance. The pseudocode of our framework is given in Table I. We will introduce the communication protocol between the server and the local sites in two parts, the initialization and the pruning by cascade lower bounds.

Initialization. Steps 1–5 in Table I initialize the algorithm. To find an initial threshold, denoted as θ, for pruning the candidates, we propose to sample the k actual DTW distances from the candidates and use the largest value, i.e., the k^{th} smallest, as the threshold. The intuition of sampling the exact DTW distance is to avoid choosing a too loose lower bound that hardly has a pruning power. To achieve this, we let the server select just a limited number (at most k) of the sites and transmit the entire reference Q to them. The selection of these sites can be done by sending segments of $Q^{(1)}$, of which the size is much smaller compared to Q, to all sites. Each local site can reconstruct $Q^{(1)}$ and computes $LB(Q^{(1)}, C)$ for every candidate time series C. For $LB(Q^{(1)}, C)$ here, we can just use LB_{MS}. The server gathers these lower bound distances from all sites and selects the sites having the top k smallest lower bound distances. These selected sites will then receive the exact query Q from the server, and thus have the ability to compute the exact DTW distances $DTW(Q, C)$ for every candidate time series they have. After collecting the exact DTW distances from these initially selected sites, the server assigns the k smallest DTW distance as the threshold θ.

Pruning by cascade of lower bounds. Steps 6–12 perform the pruning until the exact kNN is determined. Recall that the server has the lower bound distance $LB(Q^{(1)}, C)$ of *all* candidate time series C to $Q^{(1)}$ after the initialization. If all the lower bound distances from a site are greater than the threshold θ, it means that all candidates in that site will never be the final kNN for the query. Therefore, the server no longer needs to transmit any further data to the site. In other words, the site is *pruned*. The server searches through all the sites until every site has either received the entire Q or has been pruned. It does this in a series of rounds with m sites per round. At each round (steps 6–11), the server sends to a set of m local sites the current θ and the reference level-by-level. For each level, the local site performs a cascade of lower bounds computation for $LB(Q^{(l)}, C)$, which is set to $LB_{KimFL}(Q^{(l)}, C)$ first, then $LB_{Keogh}(Q^{(l)}, C)$, and then $LB_{MS}(Q^{(l)}, C)$. For each candidate, as soon as a lower bound value exceeds θ, the corresponding candidate will be

Table I
PSEUDOCODE OF THE FRAMEWORK

Procedure: kNN search on distributed time series with the DTW distance	
Input: k, the query reference Q, a set of M sites	
Output: k most similar time series to Q	
Server	Sites
1. Do multi-resolution segmentation on Q. Transmit encoded $Q^{(1)}$ to all sites.	**2.** Reconstruct $Q^{(1)}$. Compute lower bounds $LB_{MS}(Q^{(1)}, C)$ for each candidate time series C and return to the server.
3. Sort the lower bounds. Select a set of initialization sites, M_{init}, that contain candidates with the k smallest lower bounds. Transmit Q to the sites in M_{init}.	**4.** If receive Q, compute the exact DTW distances $DTW(Q, C)$ for each candidate time series C and return them to the server.
5. Sort the DTW distances, initialize kNN, and set the k-th smallest DTW distance as the threshold θ.	
6. Prune candidates whose $LB_{MS}(Q^{(1)}, C) > \theta$. Select m sites that are not pruned and have not received the entire Q yet. Transmit θ to these sites. For level $l = 2, 3, ..., L\{$	
7. Transmit encoded $Q^{(l)}$ to the m sites *simultaneously*, ignoring sites that have dropped out.	**8.** If receive encoded $Q^{(l)}$, reconstruct it. Do cascade of lower bounds pruning for $LB(Q^{(l)}, C)$=$\{LB_{KimFL}(Q^{(l)}, C)$ then $LB_{Keogh}(Q^{(l)}, C)$ then $LB_{MS}(Q^{(l)}, C)\}$: Compute $LB(Q^{(l)}, C)$ for each candidate time series C and prune candidates whose $LB(Q^{(l)}, C) > \theta$. **9.** If all candidates are pruned, inform the server and drop out. **10.** If $l = L$ and there exist unpruned candidates, then compute their exact DTW distances and return them to the server.
11. If exact DTW distances are returned from local sites, update kNN and θ. $\}$ **12.** Repeat step 6 until all sites are searched.	

pruned. At the end of each level except for the final level L, the local sites will return a 1-bit signal indicating whether it has dropped out. At the final level L, if there are still some candidates left, the local site computes their $DTW(Q, C)$ distances and sends these values back to the server. After receiving these values, the server can update the current kNN list and set the threshold θ as the current k smallest DTW distance, completing the round. When all sites have either been pruned or reported their exact DTW distances of any unpruned candidates, the kNN can be reported.

IV. EXPERIMENTS

A. Data Description and Experiment Setup

In the experiments, we would like to verify the following hypotheses: (1) *The overall bandwidth saving*: Can our framework saves significant bandwidth usage comparing to the baseline approach? (2) *The effect of query reference length*: Is our framework in favor of long query reference? i.e., $|Q| = X$ is large. (3)*Equal-size or unequal-size segmentation on Q*: Does the unequal-size segmentation yield a more power pruning effect in our framework? (4) *Initialization*: Can the design of our initialization improve the bandwidth saving ratio? (5) *Cascade of Lower bounding functions*: How effective is the designed cascade of lower bounding technique? Can LB_{MS} provide better pruning power? (6) *The bandwidth saving with compressing Q*: Still obtain decent bandwidth saving if the compression is performed on the query Q?

We use both real datasets and a synthetic dataset in our experiments. For real datasets, we use the data from the UCR Time Series Classification/Clustering Page [14], where the statistics of each data set are given in details. For each dataset in the archive, we simply merge both the training and testing sets into a single dataset for kNN search. For the synthetic dataset, we apply the random walk generator as in [5], where the authors claimed that the random walk can well model the real-world financial data and are often used for similarity search. Each data point in a synthetic time series is generated by a random variable of standard normal distribution. We generate 12,500 time series of length 12,500, which follows the same way used in [1].

To our knowledge, there has not yet been any prior work dealing with the exact kNN queries with the DTW distance in the distributed environment, thus in the experiment we propose a simple baseline to compare with. In the baseline approach, the server sends the entire query reference to all local sites, and the local sites transmit the exact DTW distances of candidates back to the server. The server then sorts the distances to identify the exact kNN solutions.

We exploit the *Bandwidth ratio* to measure the bandwidth consumption. Bandwidth ratio represents the total number of bits transmitted in our framework divided by those transmitted in the baseline strategy. The smaller bandwidth ratio implies the better performance of our framework. We assume that a data point occupies $B = 32$ bits for all conducted experiments.

Without further specification of the parameters, we have the following settings by default. We use the bottom-up algorithm proposed in [13] to do the unequal-size segmentation on Q and set the number of segments at the first level

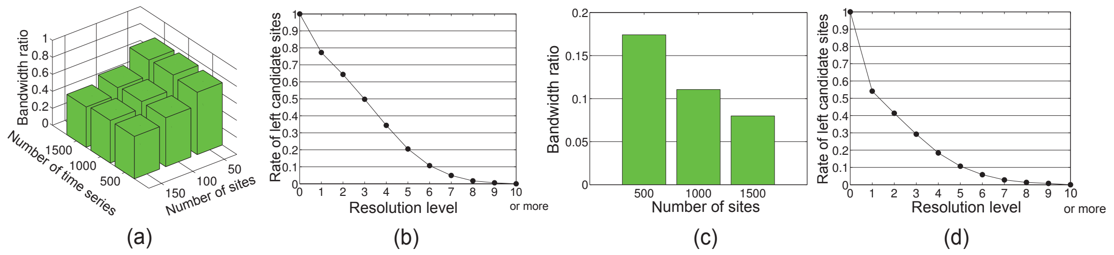


Figure 9. (a) Bandwidth ratio on the UCR datasets, $k = 10$; (b) Corresponding average rate of left candidate sites of (a) at different resolution levels; (c) Bandwidth ratio on the synthetic dataset, $k = 30$ and $S = 12499$; (d) Corresponding average rate of left candidate sites of (c) at different resolution levels.

as $N = \text{Round}(\sqrt{\frac{|Q|}{2}})$ according to the rule of thumb in [15]. For the R value, we have tried 2,4, and 8 but found no significant difference on the bandwidth usage. Thus, we report the results of $R = 2$. In addition, by default $k = 10, S = 500, M = 50$ where S is the number of time series and M is the number of sites. In addition, as suggested in [16], [5], we do z-normalization on the time series before running any kNN similarity search and we do apply the Sakoe-Chiba constraint with 5% width.

We select $(S + 1)$ time series extracted from the dataset for experiments. We use each of them as a query while the rest as the candidates, evenly distributed in all sites. Finally we report the average of the bandwidth consumed for these $(S + 1)$ cases.

B. Results

Overall Bandwidth Consumption. Figure 9(a) shows the average bandwidth ratio on all the real UCR datasets, where the number of time series varies S is set to {500, 1000, 1500} and the number of sites M is set to {50, 100, 150}. The bandwidth ratio is always smaller than one, indicating the bandwidth saving capability of our framework. As the site number increases, the saving becomes even more significant. Our framework can save up to 65% of bandwidth usage (as the bandwidth ratio less than 35% when the site number is 150). Also, we can see that the bandwidth ratio is not highly correlated to the number of time series.

Figure 9(b) shows the corresponding site pruning effect at different level of query resolutions. We can see that on average about a half of candidates sites can be pruned after within three levels of Q and about 90% of them are pruned within the first six level communications. The results demonstrate that our framework can already prune many candidate sites with only the coarse information of Q.

On the synthetic dataset, the results of bandwidth ratio at $M = \{500, 1000, 1500\}$ and $k = 30$ are shown in Figure 9(c). We report the average results of 100 random queries. The results on the synthetic dataset demonstrate that our framework can save even more bandwidth on the larger-scale data, where the bandwidth ratio is less than 20%. Similarly, Figure 9(d) demonstrates that a half of candidate sites are pruned given only one level transmission of Q, and 90% of them are pruned within the first five levels. This shows the powerful pruning effect of our proposed framework on large-scale datasets.

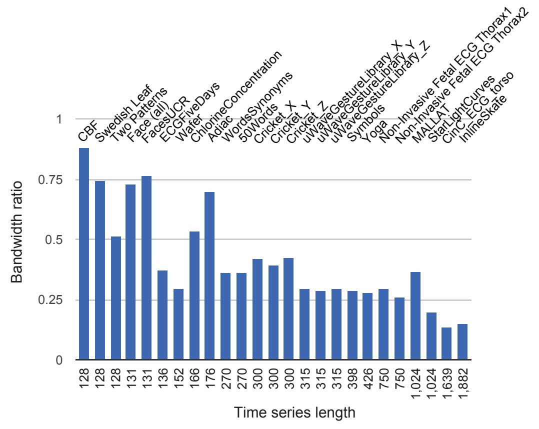


Figure 10. Bandwidth ratio at different time series length on the UCR datasets, where $k = 10$, $S = 500$ and $M = 150$

The effect of query reference length. In Figure 10, we list the average bandwidth ratio when $k = 10$, $S = 500$, and $M = 150$ on the UCR datasets. Datasets with time series length smaller than 100 are excluded. The results show that the performance of our work tends to be better when the time series becomes longer.

Equal-Sized or Unequal-Sized Segmentation. Previous experiment results are all based on the unequal-size segmentation at the first level. Here we would like to compare the outcome of unequal-size segmentation to the outcome of the equal-size segmentation. Figure 11 shows the bandwidth ratios of two different segmentations on Q at the first level, given the same number of segments. In general, the unequal-size segmentation can save 10% more bandwidth compared to the equal-size segmentation. This is because the unequal-size segmentation makes $Q^{(1)}$ is even closer to Q so that the lower bound is tighter.

Initialization. In the initialization step, we need to select some sites to transmit the exact query reference to obtain an

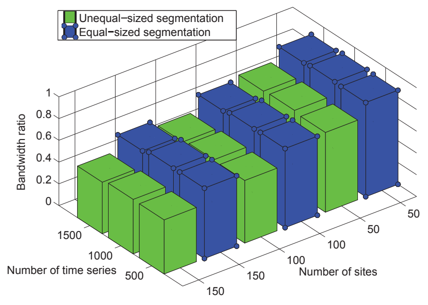


Figure 11. Bandwidth ratio for unequal-size/equal-size segmentations on the UCR datasets, $k = 10$

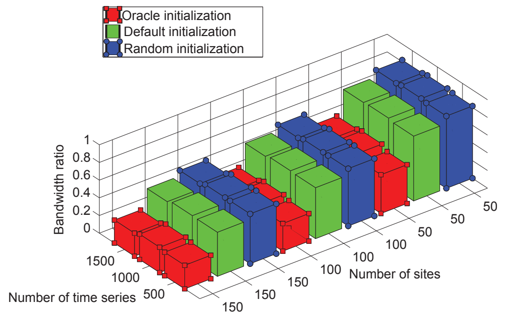


Figure 12. Bandwidth ratio with three initialization methods using oracle selection, default selection(our proposal for the framework) and random selection on the UCR datasets

initial threshold θ. The initialized sites are determined using the order of the level-1 lower bounds. We believe that this initialization does benefit the bandwidth savings.

Here we compare three possible ways of initialization. First, the *oracle* initialization assumes that we assume there is an oracle revealing which sites have the exact kNN, thus we can simply select these sites in the initialization step for the server to transmit the entire Q to them. It is the ideal situation as there is no need to consume bandwidth to perform the follow-up level-by-level pruning steps. On the opposite, we do the *random* initialization, which simulates the unfavourable scenario that we have no idea where the exact kNN locate and the server can only randomly pick k sites to transmit the entire Q. The *default* initialization is our proposed method by sending the level-1 coarse query reference to estimate where the possible kNN could be.

Figure 12 demonstrates the bandwidth ratio for our framework with three different initializations. Apparently, the oracle initialization consumes the least bandwidth. However, our framework only consumes 20% more bandwidth compared to the optimal approach. On the other hand, our initialization wins the random initialized by saving around 10% bandwidth usage, which shows the advantage of our initialization.

Cascade of lower bounding functions. Here we show how the cascade of lower bounds prune the candidates. In Table II, we present the average distribution of candidate time series processed in different steps of our framework on the UCR datasets. For example, at $M = 50$, there are 18.43% of candidates processed (i.e. either pruned or retained) in the initialization steps, 2.79% candidates are pruned by LB_{KimFL}, then 24.99% candidates are pruned by LB_{Keogh}, and lastly 34.44% candidates are pruned by LB_{MS}. The results show that all the lower bounding functions in the cascade structure have chances to prune time series. Also, LB_{MS} has the best pruning power, LB_{Keogh} comes second, and LB_{KimFL} has the least pruning power.

Compressing Q. Here we show the bandwidth consumption of both our framework and the baseline approach with or without compressing Q (or each level of Q) before sending it. We try only the lossless compression to ensure to get the exact kNN, but it shall be clear that similar range of saving can be derived with lossy compression. In the Unix environment we use the *zip* command to compress Q for the baseline approach or each level of Q for our framework. The local sites may unzip what they receive and continue the same original process. Table III shows the bandwidth ratio of two approaches on three datasets, where our framework significantly outperforms or works closely in saving bandwidth compared to baseline. The results show that our model is still very effective under compression.

Table II
AVERAGE DISTRIBUTION OF CANDIDATE TIME SERIES HANDLED BY THE FRAMEWORK ON THE UCR DATASETS, $k = 10, S = 500$

	$M = 50$	$M = 100$	$M = 150$
Used for initialization	18.43%	9.64%	6.65%
Pruned by the level-1 query in the server	18.59%	30.03%	37.63%
Pruned by LB_{KimFL}	2.79%	2.45%	2.03%
Pruned by LB_{Keogh}	24.99%	20.86%	17.08%
Pruned by LB_{MS}	34.44%	36.13%	35.65%
Unable to be pruned	0.76%	0.90%	0.96%
Sum	100%	100%	100%

Table III
COMPARISON OF BANDWIDTH CONSUMPTION BETWEEN COMPRESSING Q AND NOT, $k = 10$, $S = 500$, $M = 50$

B: baseline, F: our framework, +c: with compressing Q

Bandwidth ratio	CinC_ECG_torso	Yoga	ChlorineConcentration
F / B	0.2834	0.5203	0.8447
F+c / B+c	**0.1768**	**0.4377**	**0.9696**

V. RELATED WORK

Papadopoulos and Manolopoulos [17] analyzed four schemes to tackle the k-nearest neighbor queries in the distributed environment. They mainly considered how to reduce the bandwidth usage for returning candidate objects from the local sites while ignoring the huge bandwidth usage for the server to send many queries to many local sites. Yeh et al. [2] consider the same problem on the time series objects. They proposed a level-wise approach by leveraging the multi-resolution property of Haar wavelets on decomposing the query reference. In such a way, only rough information about the query reference need to be sent to local sites for pruning candidates. Wang et al. [1] further extended the same problem to multiple query references. Both studies provided distance bounds for candidate pruning without causing any false dismissals and showed the proposed approaches were very bandwidth-efficient. However, their distance bound designs based on the Haar wavelet decomposition work only for the Euclidean distance. Although Chan et al. [11] showed the approximate DTW distance computation based on Haar wavelets, they failed to provide a theoretical bound guarantee.

To accelerate the task of finding similar time series using the DTW distance, of which the computation complexity is quadratic to the time series length, a common way is to use lower bounds that require less computation to early prune unqualified candidates [9], [7], [10], [5]. Rakthanmanon et al. [5] further proposed the cascade of lower bounds so that the early abandoning of unpromising candidates can be achieved. However, to our knowledge, all these studies focus on saving the computation cost in a centralized environment. They did not consider the bandwidth consumption required to exchange data between the server and local sites in a distributed environment.

VI. Conclusion

With the coming of the big-data era, there will be more and more data gathered and stored in a distributed manner, thus demanding different types of analysis techniques that take bandwidth cost into account. Different from several existing works on approximated DTW matching, here we do not want to sacrifice the matching quality for efficiency, and thus concentrate our efforts on kNN matching for DTW that finds the exact solution with theoretical guarantees. Furthermore, our framework can seamlessly integrate the existing research on lower-bounding DTW for better performance, as well as combine data compression techniques to enjoy further saving.

Acknowledgement

This work was supported in part by the Ministry of Science and Technology, National Taiwan University, and Intel Corporation under Grants MOST 103-2911-I-002-001, NTU-ICRP-104R7501, and NTU-ICRP-104R7501-1

The work was done while the 5-th author was with Intel Labs.

References

[1] J.-P. Wang, Y.-C. Lu, M.-Y. Yeh, S.-D. Lin, and P. B. Gibbons, “Communication-efficient distributed multiple reference pattern matching for m2m systems,” in *Proc. of IEEE ICDM*, 2013.

[2] M.-Y. Yeh, K.-L. Wu, P. S. Yu, and M.-S. Chen, “Leewave: Level-wise distribution of wavelet coefficients for processing knn queries over distributed streams,” *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 586–597, Aug. 2008.

[3] M. Müller, “Dynamic time warping,” in *Information Retrieval for Music and Motion*. Springer Berlin Heidelberg, 2007, pp. 69–84.

[4] L. Rabiner and B.-H. Juang, *Fundamentals of Speech Recognition*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[5] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, “Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping,” *ACM Trans. Knowl. Discov. Data*, vol. 7, no. 3, pp. 10:1–10:31, Sep. 2013.

[6] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, “Querying and mining of time series data: Experimental comparison of representations and distance measures,” *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1542–1552, Aug. 2008.

[7] E. Keogh and C. A. Ratanamahatana, “Exact indexing of dynamic time warping,” *Knowledge and Information Systems*, vol. 7, no. 3, pp. 358–386, 2005.

[8] E. Keogh, L. Wei, X. Xi, M. Vlachos, S.-H. Lee, and P. Protopapas, “Supporting exact indexing of arbitrarily rotated shapes and periodic time series under euclidean and warping distance measures,” *The VLDB Journal*, vol. 18, no. 3, pp. 611–630, Jun. 2009.

[9] S.-W. Kim, S. Park, and W. Chu, “An index-based approach for similarity search supporting time warping in large sequence databases,” in *Proc. of IEEE ICDE*, 2001, pp. 607–614.

[10] Y. Sakurai, M. Yoshikawa, and C. Faloutsos, “Ftw: Fast similarity search under the time warping distance,” *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 326–337, 2005.

[11] F. Kin-Pong Chan, A. Wai-chee Fu, and C. Yu, “Haar wavelets for efficient similarity search of time-series: With and without time warping,” *IEEE Trans. on Knowl. and Data Eng.*, vol. 15, no. 3, pp. 686–705, Mar. 2003.

[12] https://www.dropbox.com/sh/u6xl96jnd9azt3m/AAAp7lF5pASOEqobyNuXYLOFa.

[13] E. Keogh, S. Chu, D. Hart, and M. Pazzani, “Segmenting time series: A survey and novel approach,” in *In an Edited Volume, Data mining in Time Series Databases. Published by World Scientific*. Publishing Company, 1993, pp. 1–22.

[14] E. Keogh, Q. Zhu, B. Hu, H. Y., X. Xi, L. Wei, and Ratanamahatana, “The ucr time series classification/clustering homepage,” 2011, http://www.cs.ucr.edu/~eamonn/time_series_data/.

[15] K. M. et al., “Multivariate analysis,” in *Academic Press*, 1979.

[16] E. Keogh and S. Kasetty, “On the need for time series data mining benchmarks: A survey and empirical demonstration,” *Data Min. Knowl. Discov.*, vol. 7, no. 4, pp. 349–371, Oct. 2003.

[17] A. N. Papadopoulos and Y. Manolopoulos, “Distributed processing of similarity queries,” *Distrib. Parallel Databases*, vol. 9, no. 1, pp. 67–92, Jan. 2001.