Dynamic Time Warping in Time Series Analysis

Hongpeng Zhang
Rutgers University
hz227@rutgers.edu

Zihao Ding
Rutgers University
zd75@rutgers.edu

ABSTRACT

Dynamic Time Warping (DTW) is a flexible pattern matching algorithm, which can transform, compress or extend samples appropriately so that samples with similar characteristics can be matched reasonably. It is implemented by nonlinearly normalizing two samples using dynamic programming principle and then matching their similar features to obtain the shortest distance between two samples. The shortest distance is defined as the similarity scale between the samples, and the smaller the similarity scale between the two samples, the better the match between the two samples.

This paper introduces the basic concepts of DTW with runtime tests to show its time complexity.

GitHub Link: https://github.com/JasoDing/DSA_21_group2.git

KEYWORDS

Dynamic Time Warping; Euclidean distance; Time series;

1 Introduction:

The real world is full of time series data, as almost every data collected by real-world sensors is time series data. Many engineering tasks require analysis on such data. For example, human speech produces time series data naturally, and recognition of its content is based on comparisons between multiple series of data; medical sensors collect time series data like electrocardiograms and electroencephalograms, which is very useful in monitoring the health condition of human bodies because several illnesses would produce their unique patterns in such data. The analysis of time series data requires people to develop proper tools, and one of which is Dynamic Time Warping (DTW).

In time series analysis, dynamic time warping (DTW) is one of the algorithms for measuring similarity between two temporal sequences, which may vary in speed. For instance, similarities in walking could be detected using DTW: even if two people are walking at different speeds, DTW can still find out if their gait frequency and distance are similar, in other words, they are walking in a similar pattern. So DTW is very useful in time series analysis as it can normalize the data in time.

Our project is divided into the following parts: Introduce to time series data (what is it, what do they mean, why we need them); the basic concept of Dynamic Time Warping and its opponents (Euclidean distance); Dynamic Time Warping with window; our experiment; result; and some other ways to accelerate the DTW algorithm.

2 Terminology and Definitions:

In this part, we explain the necessary terminologies and the definitions of Dynamic Time Warping.

2.1 Time Series Data

To understand what Time Series Data is, first it is necessary to know what Sequential Data is. Sequential Datta, as its name implies, are data in sequences. In the sequential data, points in the dataset are dependent on the other points in the data, the order of data matters. An example of Sequential Data is Strings, which are sequences of characters. Time Series is another example of Sequential Data, where it is sequences of vectors based on time.

Time Series data is a collection of observations obtained through measurements over time; it is a sequence of observations indexed in time order.

Time Series: A Time Series T is an ordered list:

$$T = \mathrm{t}_1, \mathrm{t}_2, \mathrm{t}_3, \dots, \mathrm{t}_\mathrm{m}$$

Time Series data is ubiquitous because time is an integral part of all observable things. As our world gets more and more instrumented, sensors and systems are constantly transmitting a relentless stream of time series data. Time series data can be used in tracking different things in time manner, some of the examples are: Heart rate, Stock market prices, Speech, and Music. Heart rate is the number of beats over time; Stock price is price over the day; Speech is represented as a sequence of audio measurements at discrete time intervals, and Music is represented as a sequence of pairs of notes and duration.

There are a lot of ways to analyze time series data, one of the most common ways is classification. Time series data classification means assigning time series patterns to specific categories. It has a wide range of applications, take the examples in the previous paragraph as examples, classification in Heart rate can help to determine if a heart is healthy or has disease; classification in stock market data can identify the stocks with similar performance, which

can help to achieve stock performance forecasting; classification in speech and music result in speech and music recognition.

There are different tools for time series classifying tasks, one of the most popular tools nowadays is to use various kinds of deep learning methods. Its popularity is mainly due to the trend of doing everything with deep learning. Deep learning is like a Swiss army knife, but if one wants to do time series classification from scratch using simple but more effective tools, traditional ways such as Euclidean distance (ED) and Dynamic Time Warping (DTW) are better choices.

2.2 Euclidean distance (ED)

In the analysis of the similarity between two time series data, the first thing needs to do is to define a "distance" function which can quantify the difference between them. Among all kinds of distance functions, Euclidean distance is one of the most commonly used methods in Time Series analysis. The Euclidian distance defines the distance between two points with Equation 1.

(1)

$$d(p,q) = \sqrt{(p-q)^2}$$

To calculate the Euclidean distance between two points in the same dimension is simply done by calculating the absolute value of the difference between their coordinates.

As Euclidean distance is a method that is applied between points, not sequences, we need to define a mapping method to applied Euclidean distance in time series analysis, so the distance between sequences can be defined as a function of the point-wise Euclidean distances from each sequence. The simplest way of mapping is to calculate the distance between each corresponding point which is on the same position in each series, and then add all these point-wise distances together. This is also called one-to-one mapping, which means each point will only be used to calculate the distance of one point-pair, as Figure 1 shows. In this way, the distance between two sequences is defined by Equation 2. In general cases, the Euclidean distance between two sequences refers to the distance using one-to-one mapping method.

(2)

$$\boldsymbol{ED(Q,C)} = \sqrt{\sum_{i=1}^{n}(\boldsymbol{q_i - c_i})^2}$$

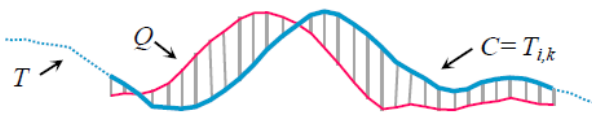


Figure 1. One to one mapping method with Euclidean Distance

2.2.1 Pseudocode

```
Function EuclideanDistance(x1, x2):
    dist = 0
    for i = 1 .. N
        dist = dist +((x1[i] -x2[i]) ^2

    return sqrt(dist)
end
```

2.2.2 Analysis

In the algorithm above, the loop will run N times for two time sequences with length of N. Thus, the time complexity should be O(N).

The one-to-one mapping method is quite straightforward and has relatively low cost, but it has some problems. One of its major problems is it cannot show the similarity between two series that has the same shape but different phases, like sin and cosine wave. Sine and cosine waves have the exact same pattern, but use such method we will get a quite large distance value. Other problems such as variance in speed and offset may also significantly change the final value. In other words, point-wise Euclidean Distance cannot illustrate the similarity between two sequences.

2.3 Dynamic Time Warping (DTW)

The dynamic time warping, also called DTW, is an algorithm that using one-to-many method to apply Euclidean distance, which means for one point, it will not just directly map to the point at the same position in the other series, but find its correspondence in a certain range of points in the other series. This method is to find the optimal legal alignment between the two time-series, by finding the most similar point pairs defined by Euclidean distance, as the picture on the right shows, which will eliminate the influence of different phases or speed between the two series.

During the process of finding such optimal alignments, there are three rules that we must follow: The Boundary conditions, monotonicity, and continuity. The boundary condition means the first index of one series must be matched with the first index of the other series, although they can be matched to multiple points. The same rule should also be followed by the two last indices. The boundary condition ensures that the DTW calculation runs cover the full range of both series. Monotonicity means the mapping of the indices from the first sequence to indices from the other sequence must be monotonically increasing, and vice versa. In other words, the matching lines in the right picture must not be crossed with each other in the middle. Continuity means Every index from the first sequence must be matched with one or more indices from the other sequence and vice versa.

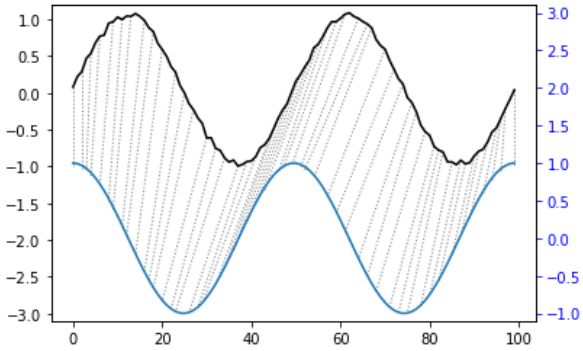


Figure 2: Illustration of DTW

Considering these rules together, finding optimal alignments is just like moving two pointers along the two time-series. Pointers can move forward one index at a time, or stop, but cannot move backward. At the beginning, both pointers are at the first index of each series. For each of the following step, there are three options: pointer A moves forward and pointer B stays, B moves forward and A stays, or both A and B move forward. Such decision is made by comparing the Euclidian distance of the three pairs of indices, and choose the minimum distance to move. The effect of such step is when the patterns of two series are matched, both pointers will move forward, and when the patterns are not matched, one pointer will stay until the other pointer find the next matching index, which is shown by the horizontal and vertical path in the left image and the colored matching line in the image below.

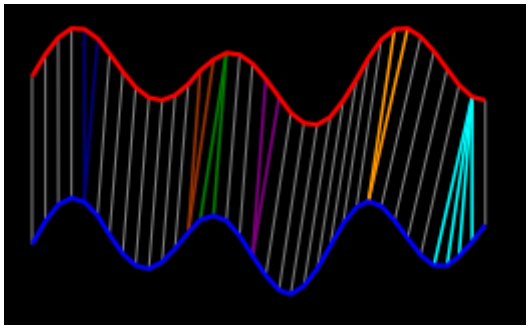
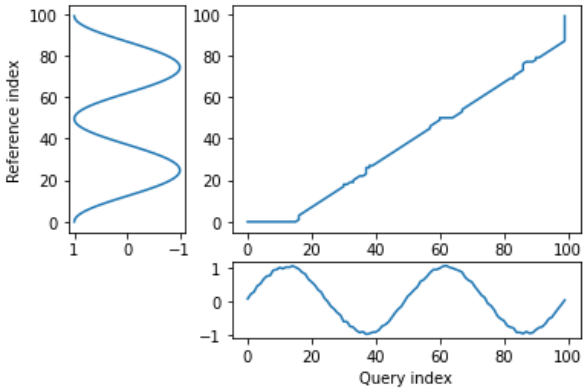


Figure 3: matching point pairs in DTW

Using mathematical equations to express the process mentioned above. First, we need to create an M by N matrix A, where M and N are the length of each series. Then we define a cost function as the picture shows to fill in matrix A. Each cost value is equal to the Euclidean distance between two indices at that entry plus the local minimum of its possible predecessors. So we can trace a path from A(0,0) to A(m,n), with all of its values are local minimums. The cost at A(m,n) is defined as the DTW distance between the two series**.**



$$D(i,j) = Dist(i,j) + \min[D(i-1,j), D(i,j-1), D(i-1,j-1)]$$

Figure 4: Calculation of distance matrix

2.3.1 Pseudocode

```
def dtw(x, y):
    # Initialization
    for i = 1..n
        for j = 1..m
            C[i, j] = inf

    C[0, 0] = 0.

  # Main loop
  for i = 1..n        # For Each Row
        for j = 1..m      # For Each Column
            dist = d(x_i, y_j) ** 2      # ED distance
            C[i, j] = dist + min(C[i-1, j], C[i, j-1],
                                 C[i-1, j-1])

  return sqrt(C[n, m])
end
```

2.3.2 Analysis

Notice that in the step marked as '# ED distance' only does the square of x[i] - y[i], and the final square root in the ED distance method is placed at the end of the DTW function. This is due to the fact that the computer does square root much slower than other operations. By placing the time-consuming square root operation at the end of the function outside of any loop can save a significant amount of time.

For the core part of the DTW algorithm, the line :

C[i, j] = dist + min(C[i-1, j], C[i, j-1], C[i-1, j-1])

Can be visualized as Fig.5 Where C[i-1, j] is represented as the blue arrow pointing upwards; C[i, j-1] is represented by the green arrow pointing to the left; and the C[i-1, j-1] is represented in red pointing at the starting point. The step visualization in Fig further explains how the path is selected in the DTW matrix.

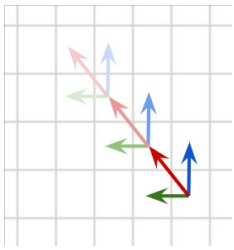


Figure 5: DTW's Step Visualization

The time complexity of Basic DTW is obvious since the pseudocode only has two loops. One is initialization, which is set all elements in the matrix to the maximum number allowed. The other is the core of DTW, which is to go through all elements in the matrix, calculate the ED distance, and then calculate the DTW distance for the current position. The second loop clearly takes longer than the first one, which needs to run (m* n) times, where n, m are the number of rows and columns in the DTW matrix (or the size of input array). Thus the overall time complexity is O(m* n).

There is no best or worst case since for the Basic DTW algorithm, it always needs to compute all the values in the n by m DTW matrix.

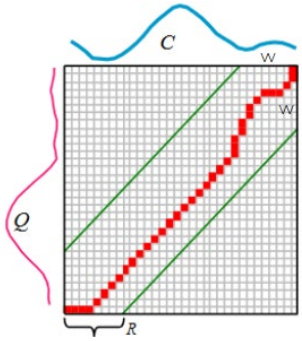


Figure 5: Example of DTW with warping window (w)Figure 6: Example of DTW with warping window (w)

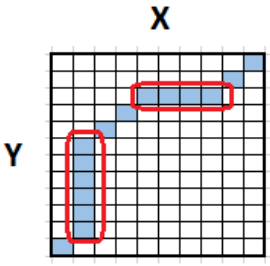


Figure 6: Example of pathological warping. Figure 7: Example of pathological warping. The parts in red circle are the relatively small section of one sequence maps onto a large section of the other

The space complexity of Basic DTW is simply the size of the final DTW matrix. Since the size of DTW matrix is (m *n), the space complexity of Basic DTW algorithm is O(m*n).

To sum up, the time complexity of computing the Basic DTW is O(m * n) where m and n represent the length of each sequence, the space complexity is the same as the time complexity which is also O(m * n). Since O(m *n) can be written as O(n^2), the time and space complexity for the Basic DTW algorithm can be considered as O(n^2).

2.4 Dynamic Time Warping with window constraint

It is obvious that the basic Dynamic Time Warping will take a significant amount of time to compute when the input data is too large. The O(n^2) complexity is not ideal at all. There is one simple but common way to speed DTW up without sacrifice much of the accuracy. The idea is to add a boundary, which limits how far it may stray from the diagonal of the matrix. The part of the matrix that the warping path is allowed to visit is called a warping window or a band. There are different ways to implement the warping window, in this paper, we are going to use Sakoe-Chiba Band [3] due to its simplicity. The warping window not only speeds up the DTW calculation but also prevents a pathological warping, where a relatively small section of one sequence maps onto a large section of another.

2.4.1 Pseudocode

```
def dtw_window(x, y, w):
    # Initialization
    w := max(w, abs(n-m)) // adapt window size (*)
    for i = 1..n
        for j = 1..m
            C[i, j] = inf

    C[0, 0] = 0

    for i := 1 to n
        for j := max(1, i - w) to min(m, i +w)
            DTW[i, j] := 0

  # Main loop
  for i = 1..n        # For Each Row
        for j in range (max(1, i - w), min(m, i + w))
                                # For Each Column
            dist = d(x_i, y_j) ** 2     # ED distance
            C[i, j] = dist + min(C[i-1, j], C[i, j-1],
                                        C[i-1, j-1])

  return sqrt(C[n, m])
end
```

2.4.2 Analysis

There is a limitation to the window size when applying, where the window parameter w must be adapted so that | n – m | is less or equal to w.

The time and space complexity of DTW with warping window is similar to the Basic DTW algorithm. The only difference is that with the window defined, it only needs to calculate a much smaller area of the matrix instead of the whole DTW matrix. The time and space complexity can be determined by calculating the area within the window. Where both can be summarized as O(w(m+n-w)), where n, m are the number of rows and columns in the DTW matrix (or the size of input array), and w is the window size. However, the analysis above only covers the time complexity of the DTW calculation part, which is shown by the "main loop" in the pseudocode above. For the initialization of the algorithm, it still needs O(n^2) time to set the distance matrix. As a result, the time complexity of the warping-window-DTW is still O(n^2). It should be noticed that even the time complexity is unchanged, the implementation of a warping window can still significantly reduce the time cost. The time complexity analysis is only meaningful for the same algorithm with different input size, not for different algorithms.

3 Experiment

The main focus of our experiment is to code up the algorithm, execute it at relevant scales to extract the performance behaviors. This part includes the dataset we used, the machine specification of our experiment, and how we chose the input data size.

We choose the input data size based on the power of 2. In this way, the final result will be more meaningful, and it would be easier for performance review.

The experiment itself is to calculate the runtime of the DTW algorithm. Thus, we mainly tested the performance of basic DTW and the DTW with warping window. Due to the input size of data, we did select a relatively small window size where w = 20. The experiment is simple, we calculate the average runtime of different DTW algorithms based on different input size ranging from 60 to 32000. We also did 2 large scale test which is compares all 56 files under the accelerometer folder (1540 total cross-file comparisons) the result is presented in the presentation.

3.1 Dataset

The dataset we used for this paper is from DataSet - RealWorld (HAR) [4], created by the University of Mannheim - Research Group Data and Web Science. It includes acceleration, GPS, gyroscope, light, magnetic field, and sound level data of fifteen people (age 31.9±12.4, height 173.1±6.9, weight 74.1±13.8, eight males and seven females) during their activities climbing stairs down and up, jumping, lying, standing, sitting, running/jogging, and walking. Each person performed each activity for about 10 minutes except for jumping due to the physical exertion (~1.7 minutes). For each activity, they recorded simultaneously the acceleration of the body positions chest, forearm, head, shin, thigh, upper arm, and waist.

Figure 8. Two volunteers of the HAR dataset. The red circle indicates the sensors' locations.

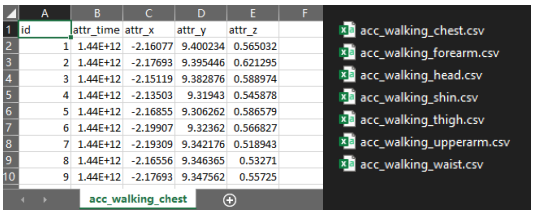


Figure 9. Examples of components of HAR dataset.

In this dataset, the motion data (which is in the form of time sequence) of different activities collected from different body parts are stored in separate CSV files. Each file contains about 32000+ data points. For our experiment, we chose two files that are from the same activity of different person and applied the algorism in the first N data points, where N varies from 10 to 16000. The similarity analysis between such time series can be used in gait identification in the real-world scenario.

3.2 Device Specification

All algorithms were coded in Python, in Python 3.6 environment. All experiments were performed on a desktop computer using a 3.2-GHz Intel Core i7-8700 CPU with 16 GB DDR4 RAM running Windows 10 Pro operating system.

4 Results

Our experiment can be mainly separated into two parts, the first part is the result of the Basic DTW algorithm. The second part is the result from the DTW algorithm with warping window. For each part, we run the algorithm with different size of input from 10 to 16000 and compares the performance with runtimes. The program is run 4 times for smaller input size and 2 times for larger input size due to the computational limitation of our device. We take the average for these runtimes and analyze the relations between the average runtime as the input size increases.

4.1 Basic DTW

For the first part, the runtimes of basic DTW algorithms in different size of input is shown below:

Input Size	Run1 (s)	Run2 (s)	Run3 (s)	Run4 (s)	Average (s)	Meanful log
10	0.03095	0.03092	0.02992	0.03192	0.03093	
60	0.05388164	0.05385828	0.0598414	0.055851	0.05586	0.832536657
100	0.09574	0.09574	0.09571	0.11070	0.099473	2.307164557
250	0.515125	0.451762	0.4797146	0.5226	0.49230	1.853863366
500	1.721416	1.7802438	1.881963	1.7343883	1.77950	2.014064784
1000	7.34934	7.23165	7.09291	7.07708	7.187744	2.002892786
2000	30.67257	28.70869	27.71690	28.13657	28.80868	2.009324252
3000	62.82902	63.09929	64.19027	70.34281	65.11535	
4000	113.64424	119.22028	114.30231	116.76081	115.9819	1.980398971
5000	180.55012	181.41136			180.9807	
8000	455.33249	460.00183			457.6672	2.075117479
10000	713.38994	726.41550			719.9027	
16000	1935.16011	1921.86401			1928.512	

Table 1. The performance of basic DTW algorithm

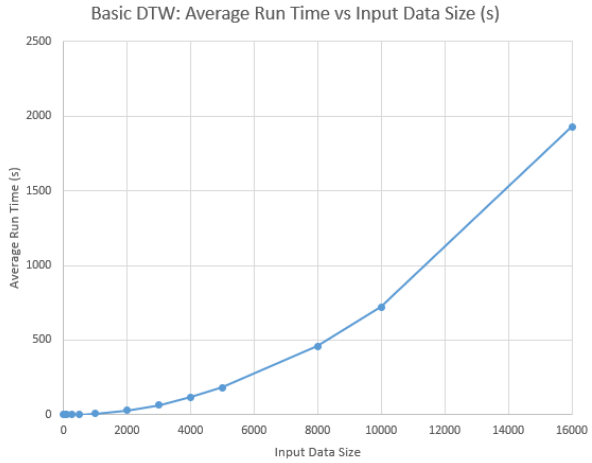


Figure 10. The runtimes of Basic DTW algorism, in linear coordinates

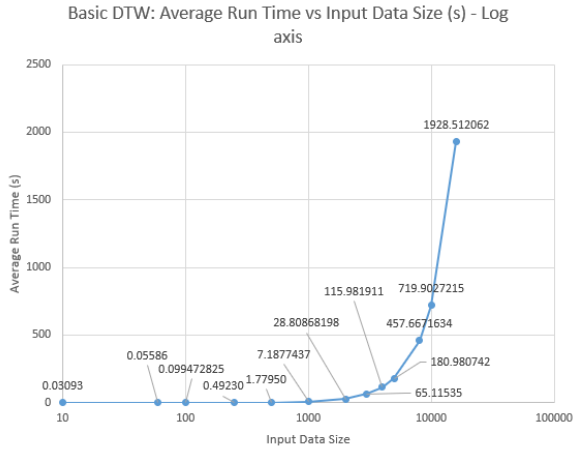


Figure 11. The runtimes of Basic DTW algorism, in log coordinates

As table 1 and figures show, the runtime increased by about 4 times when the input size doubles. The show this more clearly, we divide each runtime value by the run time whose input size is half of the current input, then take log function on it. The result should be around 2, as the time complexity of the basic DTW algorithm is O(n^2). This means basic DTW is a time-expensive algorithm. Please note, the blanks at the end of Run3 and Run4 are due to the significant amount of run time, we choose to only run the algorithm only twice to get the average run time.

Input Size	Average Run Time (s)	log - log
10	0.030925324	
60	0.055858073	0.832536657
100	0.099472825	2.307164557
250	0.4923004	1.853863366
500	1.779502775	2.014064784
1000	7.1877437	2.002892786
2000	28.80868198	2.006091066
4000	115.722278	1.983632157
8000	457.6671634	2.080082249
16000	1935.160115	

Table 2. Taking log(x) on the increase factor between runtimes

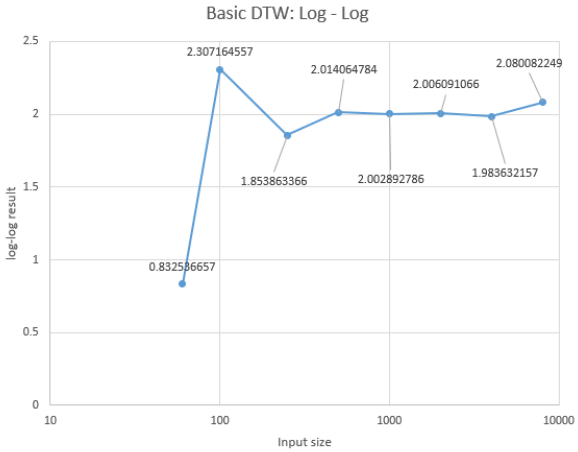


Figure 12. Taking log(x) on the increase factor between runtimes

The log-log in Table 2 is calculated by $\log 2\left(\frac{next\ average\ run\ time}{current\ average\ run\ time}\right)$. Figure 12 shows its result in most cases the result is around 2, with an exception that happened when the input size is 60. In our opinion, such exception is caused by some time cost that exist in real-world scenario but excludes by the time complexity analysis from the algorithm level. For example, the time cost of data input/output is considered as 0 in the time complexity analysis. However, the time cost of I/O is a significant

portion of the program's runtime, especially when the input size is relatively small. In this case, when the input size is small, the I/O process will take more time than DTW calculation. Thus, the real runtime is not strictly followed the relation of O(n^2) when the input size is relatively small.

4.2 DTW with Warping Window

For the first part, the runtimes of DTW algorithms with warping window implementation in different size of input is shown below. In our experiment, the window size is 20 and fixed.

DTW with Warping Window, W = 20				
Input Size	Run1 (s)	Run2 (s)	Average (s)	log(prev/current)
60	0.044851	0.04488	0.04487	0.354493
100	0.05588	0.05884	0.057362	0.888796
250	0.106709	0.105719	0.10621	1.01353
500	0.216428	0.212431	0.21443	1.251826
1000	0.52061	0.50069	0.510648	1.451172
2000	1.46209	1.33044	1.396266	1.713539
3000	2.61601	2.60802	2.61202	
4000	4.54781	4.61070	4.579252	1.801758
5000	6.44377	6.35625	6.400013	
8000	16.22863	15.70203	15.96533	1.810873
10000	22.89181	23.08523	22.98852	
16000	55.97111	56.05907	56.01509	1.851176
32000	202.8613	201.3369	202.0991	

Table 3. The performance of DTW with warping window

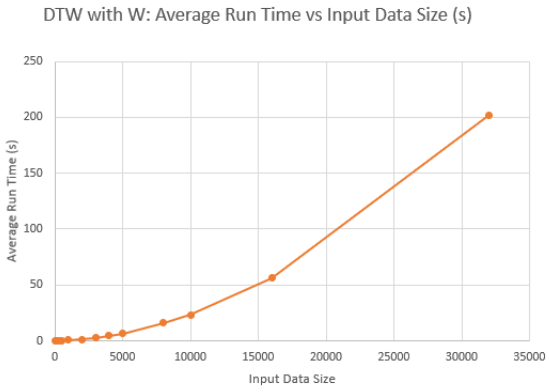


Figure 13. The runtimes of DTW with warping window, in linear coordinates

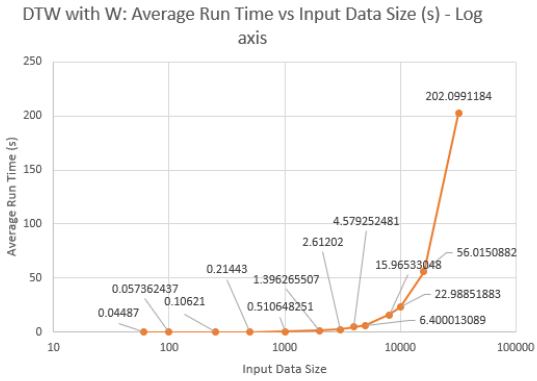


Figure 14. The runtimes of DTW algorism with warping window, in log coordinates

As table 2 and figures show, the runtime increased by about 4 times when the input size doubles. The show this more clearly, we divide each runtime value by the run time whose input size is half of the current input, then take the log function on it. The result should be around 2. This is because the time complexity of the DTW calculation with warping window size w is O(w(m+n-w)) = O(n), while the initialization part still has a time complexity of O(n^2), so the total time complexity of the DTW algorithm with warping window is O(n^2+n) = O(n^2), but this O(n^2) would be smaller than the basic DTW's O(n^2) if under the same circumstance.

Input Size	Average Run Time (s)	log - log
60	0.0448657	0.354492521
100	0.0573624	0.888796342
250	0.106214	1.013529613
500	0.2144296	1.251825679
1000	0.5106483	1.451171535
2000	1.3962655	1.713538808
4000	4.5792525	1.8017584
8000	15.96533	1.810873066
16000	56.015088	1.85117564
32000	202.09912	

Table 4. Taking log(x) on the increase factor between runtimes

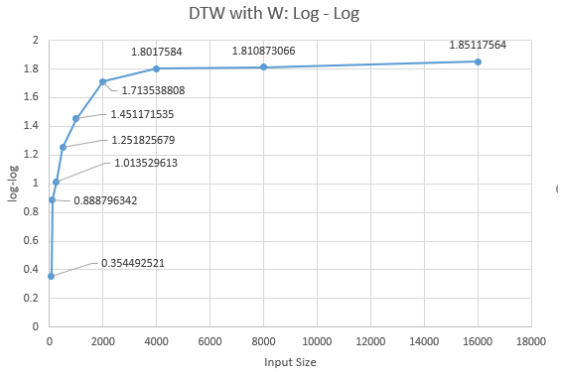


Figure 15. Taking log(x) on the increase factor between runtimes

Figure 15 shows in most cases the result is around 2, with more exceptions happened when the input size is relatively small. In our opinion, the increase of exceptions is caused by the same reason as what happened in the Basic DTW experiment. However, the runtime of DTW calculation is significantly reduced because of the warping window. As a result, the proportion of DTW calculation is much less than that in the basic DTW algorithm, which makes such exceptions happened in a larger range of input size.

4.3 Comparison between Basic DTW and DTW with Warping Window

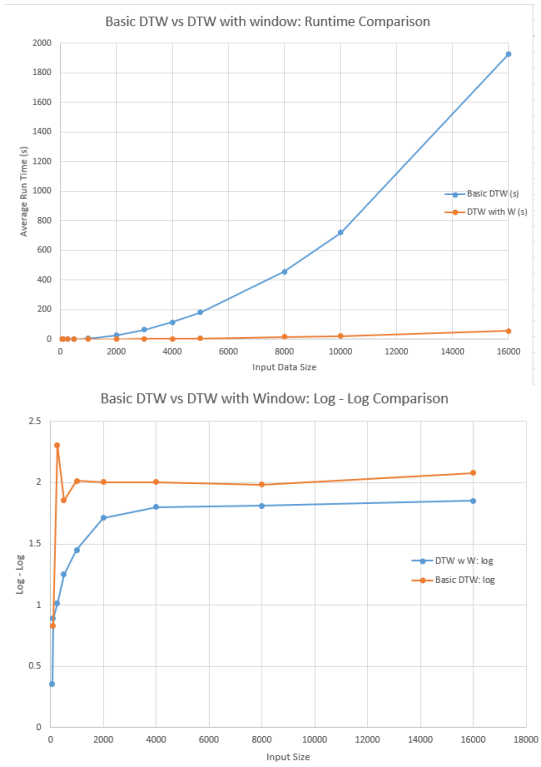


Figure 16: Side by side runtime and log-log comparison for Basic DTW and DTW with warping window

Basic DTW algorithm and DTW with warping window share some common points: both of them have time complexity of O(n^2). And in real-world cases, both of their runtimes are more depends on the I/O time cost when the input size is small.

However, the implementation warping window significantly reduced the number of computations needed by DTW, as a large number of indices becomes excluded from the possible shortest path. Figure 16 shows that the runtime after implementing the warping window has significantly reduced. For the case of input = 16000, the runtime has been reduced by over 30 times. The warping window makes DTW more capable of dealing with large amount of data, which makes it more applicable.

5 Conclusion

In this paper, we introduced the definition of Dynamic Time Warping in time series analysis. We introduced the two forms of DTW algorithm: the basic DTW and DTW with warping window. For each algorithm, we analyzed the run time in different input sizes. The purpose of DTW is to define and quantify the similarity between time series data, which is useful in speech recognition, human activity recognition, and many other areas that are related to time series analysis classification. However, the high time cost of the basic DTW algorithm limits its application in the real-world. One way to reduce the time cost is by implementing a warping window. According to our experiment, such method can significantly reduce the time cost of DTW and make it more applicable.

Besides of wrapping window, there are many other techniques to optimize the performance of DTW, includes:[2]

a) Using the Squared Distance (both have square root calculation, ignore the root directly compare the result of squares)

b) Lower Bounding (use a cheap-to-compute lower bound to eliminate unpromising candidates)

c) Early Abandoning of ED and LB_Keogh (if the current sum of the squared differences between each pair of corresponding data points exceeds the best-so-far, then stop the calculation)

d) Early Abandoning of DTW (while computing DTW, if the sum of DTW+ LB_Keogh exceeds the best-so-far distance, then current C can be pruned)

e) Exploiting Multicores (speedup calculations by using multicores)

These optimization methods were mentioned in Rakthanmanon's paper [2], however due to the time limitation of this project, we are not able to hand on and take them into test. We may use these alternative DTW methods in the future to analyze their performance.

Contributions:

Zihao and Hongpeng have almost equal contributions during the whole stage of this project. During the initial researching stage, Zihao and Hongpeng both purpose several different algorithms for this project, the final decision of DTW is due to its importance in processing nowadays' data. Both of the members read reviews and surveys to understand the current trend and find out what paper to read further. For the coding part, Zihao coded up the basic DTW algorithm with the hand computable example, and Hongpeng coded the Windowed DTW and the validation part for our DTW algorithm. For the consistency of the runtime result, the experiment is performed only on Zihao's Desktop. Both team member contributes to the final presentation, Zihao took over the general introduction, the details of Time Series, DTW Example and pseudocodes, runtime breakdown, and result analysis. Hongpeng covered the detailed explanations of the ED and DTW algorithm, the experiment setup, and the analysis of runtime result. For the writing of this final paper, both members also contribute equally.

REFERENCES

[1] Salvador, Stan and Chan, Philip. 'Toward Accurate Dynamic Time Warping in Linear Time and Space'. 1 Jan. 2007 : 561 – 580.

[2] Rakthanmanon, Thanawin, Bilson, Campana, Abdullah, Mueen, Gustavo, Batista, Brandon, Westover, Qiang, Zhu, Jesin, Zakaria, and Eamonn, Keogh. "Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping." . In Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 262–270). Association for Computing Machinery, 2012.

[3] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," in IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 26, no. 1, pp. 43-49, February 1978, doi: 10.1109/TASSP.1978.1163055

[4] T. Sztyler and H. Stuckenschmidt, "On-body localization of wearable devices: An investigation of position-aware activity recognition," 2016 IEEE International Conference on Pervasive Computing and Communications (PerCom), 2016, pp. 1-9, doi: 10.1109/PERCOM.2016.7456521.

[5] Giorgino, Toni. "Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package." Journal of Statistical Software [Online], 31.7 (2009): 1 - 24.