Peleg, Eyal

Ben Gurion University - Department of Information System Engineering | Ariel Felner

Dynamic Time Warping using Branch and Bound Search

artificial intelligence course

Contents

[Introduction 2](#_Toc470961200)

[Literature Review 3](#_Toc470961201)

[Dynamic Time Warping - How it works 3](#_Toc470961202)

[The Alignment Path Criteria 3](#_Toc470961203)

[finding the optimal path – Dynamic Programming 4](#_Toc470961204)

[Research Question 5](#_Toc470961205)

[Experiment 6](#_Toc470961206)

[Data Description 6](#_Toc470961207)

[Execution Environment 6](#_Toc470961208)

[Code Implementation - Googles Go Programming Langauge 6](#_Toc470961209)

[Code Description 6](#_Toc470961210)

[Results 10](#_Toc470961211)

[Mean Exeuction Time Analysis 10](#_Toc470961212)

[Execution Time Standard Deviation Analysis 10](#_Toc470961213)

[Analysis 12](#_Toc470961214)

[Levenes Test 12](#_Toc470961215)

[Paired T-Test Analysis 12](#_Toc470961216)

[Wilxcon Sign Test 12](#_Toc470961217)

[Conclusions 13](#_Toc470961218)

[Future work 14](#_Toc470961219)

[Bibliography 15](#_Toc470961220)

# Introduction

This project aims to supply a different approach to the time warping measurement calculation. The time warping measures the distance between two time series in way that is more elastic so that different time points in both series could be associated with each other. This prevents from very similar time series events to be supposedly seen as very far away from each other when they have a different starting point. Take sin(x) and cos(x). when comparing the distance on a regular scale they would be seen as very different time series instances although they have the same trends just with a different starting point. Here is where Time Warping shines, it is able to distinguish this behavior and to signal that these series are actually the same.

Now, the initial approach to perform the calculation was described in Sakoe, Hiroaki, and Seibi Chiba. "Dynamic programming algorithm optimization for spoken word recognition". This approach uses a dynamic programming solution to perform the calculation. Although its execution is very quick ( O(mn) where m, n are the sizes of the time series) it suffers from space complexity issues (has to maintain a matrix of size M\*N in it) and it is also bound to be single threaded.

In this project, I offer a different approach for calculating the time warping distance – Branch and Bound Search. This approach was selected due the ability to multi-thread the search and the low space complexity it requires in comparison to the first approach.

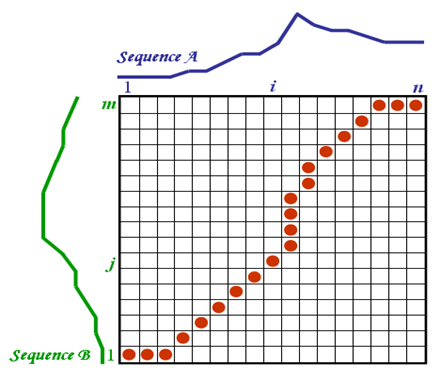
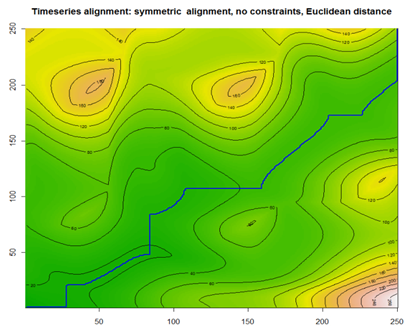
# Literature Review

Dynamic Time Warping algorithm (referenced from now on as DTW) has earned its popularity by being extremely efficient as the **time-series similarity measure** which minimizes the effects of shifting and distortion in time by allowing “elastic" transformation of time series to detect similar shapes with different phases (Pavel Senin).

## Dynamic Time Warping - How it works

Given 2 Time Series - X, Y

1.**building local cost matrix -** the distance matrix representing all pairwise distances between *X* and *Y*

2.**finding the optimal alignment path -** the path for which the sum of all costs is minimized.

## The Alignment Path Criteria

The Alignment path must satisfy to the following criteria:

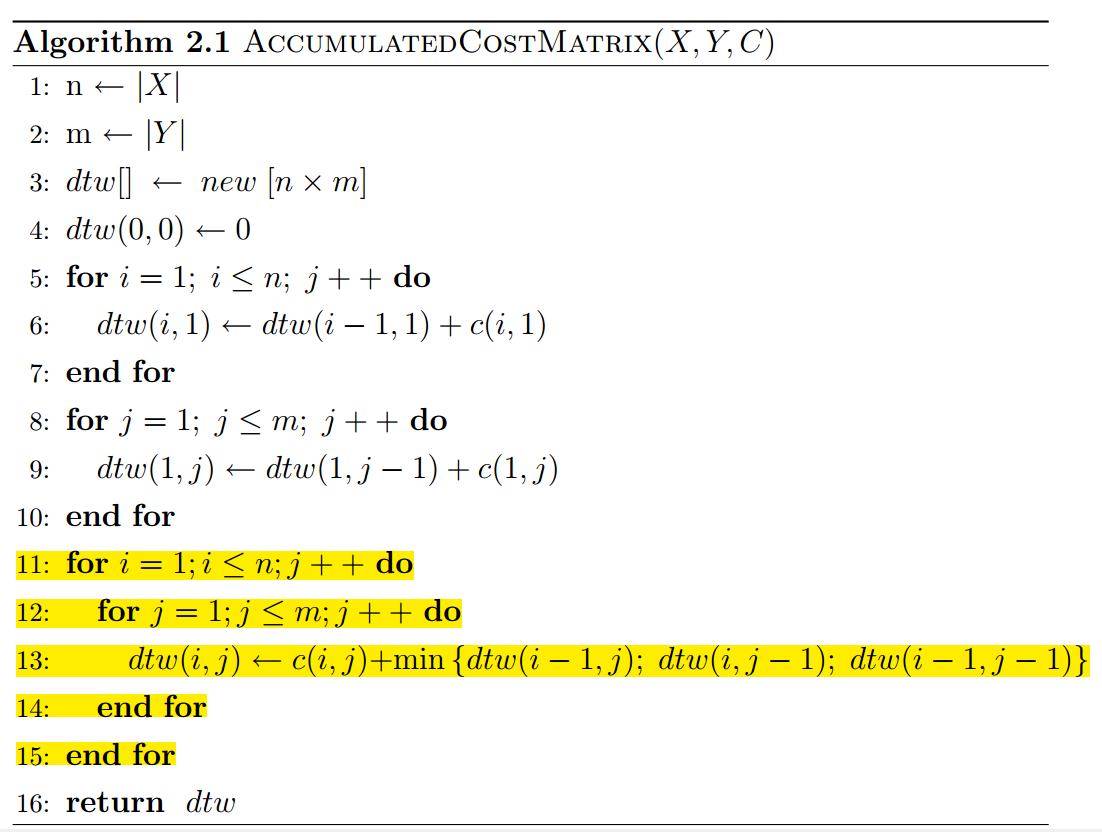
1. **Boundary condition**:  
    *p*1 = (1*;* 1) and *pK* = (*N; M*).   
   The starting and ending points of the warping path must be the first and the last points of aligned sequences.
2. **Monotonicity condition**:   
   *n*1 *≤ n*2 *≤ ::: ≤ nK* and *m*1 *≤ m*2 *≤ ::: ≤mK*.   
   This condition preserves the time-ordering of points.
3. **Step size condition**:   
   this criterion limits the warping path from long jumps (shifts in time) while aligning sequences.

## finding the optimal path – Dynamic Programming

By following the optimal warping path definition to find one, we need to test every possible warping path between X and Y which could be computationally challenging due to the exponential growth of the number of optimal paths as the lengths of X and Y grow linearly. To overcome this challenge, DTW employs the Dynamic Programming - based algorithm with complexity only O(MN). (Sakoe, Hiroaki, and Seibi Chiba)



In the highlighted area we can see the pseudo code for the finding the optimal path using dynamic programming.



# Research Question

I’ve implemented a Branch and bound strategy as an alternative to the dynamic programming approach. There are a few reasons for heading towards this approach.

First, the branch and bound search could be parallelized whereas the dynamic programming approach requires a single thread which starts at the beginning of the cost matrix and spreads towards its end;

Second, considering space complexity, the dynamic programming approach demands that all the matrix frontier would be kept in memory for the entire execution of the program whereas in Branch and Bound only the existing paths which were not trimmed should be kept in memory. This way the more paths we disqualify the less memory would be consumed;

A third argument in favor of branch and bound is that it’s an “anytime” algorithm, that is, for any given moment we can produce the best result so far, whereas the dynamic programming algorithm must finish its execution to produce a solution;

Nonetheless there are many arguments against the Branch and Bound approach.

First and foremost, the execution time of the algorithm depends on the structure of the matrix (the search tree) and in some cases, might produce very poor performance whereas the dynamic programming algorithm guarantees exactly O(NM) time complexity (given that the time series length are N and M);

Second, B&B depends on how good your initial solutions are and how fast you trim non-optimal paths. In our case, since we have an aggregated cost as our boundary, the trimming would be done late in the search tree which would cause for a lot of non-optimal paths to be searched through.

Given these arguments described above, my research question is whether a branch and bound parallelized algorithm would outperform the dynamic programming approach for Dynamic Time Warping problems, and for which cases.

# Experiment

## Data Description

For each round

* 20 different lengths – (4-24 values each).
* For each length - 10 iterations
* For each iteration – 2 randomly generated time series values.

The execution time was measured for each iteration and compared later on.

## Execution Environment

* Intel i5 Quad Core 2.6 GHz)
* 10 GB free memory

## Code Implementation - Googles Go Programming Language

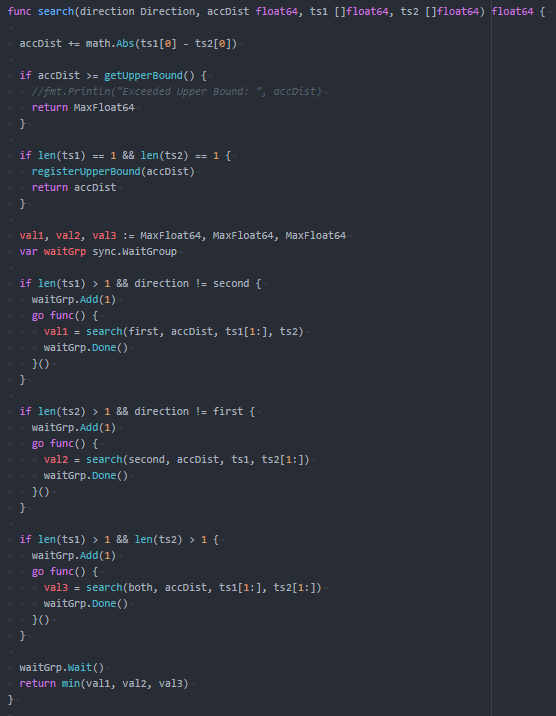
The experiment was conducted using Googles Go language. Go was chosen thanks to its very efficient and lightweight thread implementation. In go, creating a new thread takes only 2KB of memory with minimal framework effort (https://golang.org/)

Thanks to the above, the Branch and Bound algorithm was implemented using parallel threads running in the optional paths. An upper bound was shared across these threads and before discovering a new path, each thread would query the upper bound to make sure that in hasn’t exceeded it. If so, the thread would terminate itself by returning infinite sum result. Otherwise the thread would recursively open new threads to continue the search.

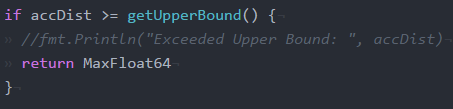
## Code Description

The following code snippets are the essence of my branch and bound implementation. Each snippet comes to illustrate a different aspect of the algorithm.

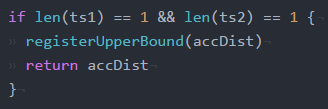
First here is the complete search method



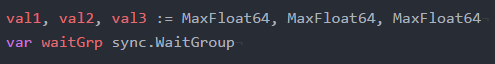
This is the termination condition for a given path. If we exceed the upper bound return infinite sum result.



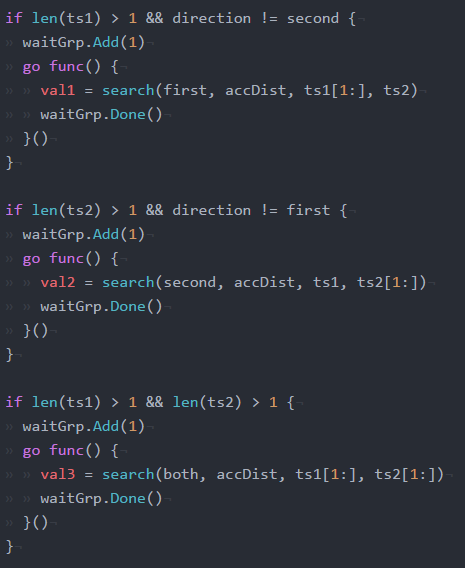
Here we find a solution and we register it with the upper bound manager. We cannot set the upper bound itself here since it is a shares resource and we have to do using a locking mechanism.



As described in the literature review, each path could have up to three sub paths and we initialize their values to infinite, until a lower value would be discovered.



Here we start the 2 or 3 threads (depending on the previous step). You can see that each one takes +1 step towards the end of one of the time series. Moreover, we register the thread in a countdown latch, when all threads terminate, the latch will open and the main thread would be able to proceed.



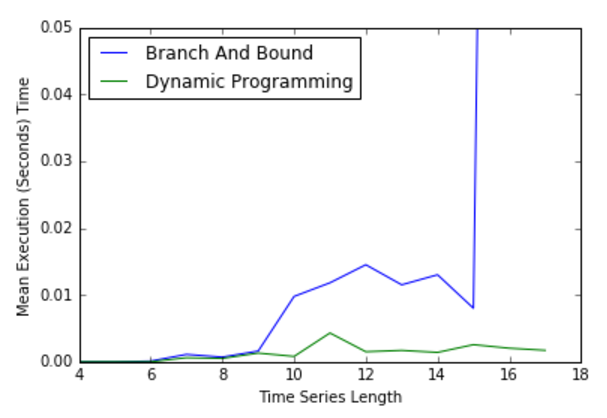
Here we can see the main thread waiting for all the sub threads to finish their course and return the sum path they have found. Than the function returns the minimal sum-distance discovered between the 3 threads. The main search function would therefore return the optimal path using one of these values.



# Results

## Mean Execution Time Analysis

Graph 1 shows us the Mean Execution Time in seconds per Time Series Length. We can see that from the length of 10, the difference between the performances of the algorithms really starts to come out. At the size of 15 and above there is an increase in the execution time of the Branch and bound algorithm to such an extent that the two algorithms could not even be measured together on the same scale.

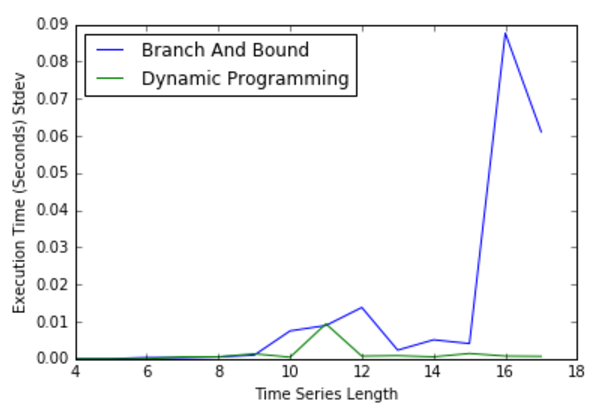


Graph 1- Mean Execution Time

I assume that the main reason for why the Branch and Bound algorithm is much more susceptible to the increase in dimensionality is that in incurs much more computation resources in the thread-managing respect. That is, for each increase in the time series length, more thread would run concurrently, whereas for the dynamic programming algorithm there is no effect other than a few more aggregation computation to be done.

## Execution Time Standard Deviation Analysis

Graph 2 shows the variation in execution time. It is important to note that the variation could result from two aspects: first, the randomization of the data; second, the state of the machine on which the computations were held. As for the latter aspect, although I made sure that the minimum amount of processes would run on the machine at the time of the experiment, these were not optimal conditions and therefore we might expect to see it manifest in our results. As for the first aspect - data variation, some algorithms are more susceptible to noisy data and some are less. The Dynamic Programming algorithm does not depend on the data what so ever, and only on the size the time series. On the other hand, Branch and Bound depends heavily on the data values, as many search algorithms do, and its performance varies depending on the dataset.



Graph 2 - Execution Time Stdev

Therefore, when analyzing Graph 2 we could easily see that as predicted the Dynamic Programming algorithm produces very low variations. One exception, might be length of 11 which most probably could be attributed to the state of the machine rather than the data variation. As for Branch and bound, we can see that the longer the time series, the higher the variation values. Again, this could be attributed to both the machine variation as well as the data variation.

# Analysis

## Levenes Test

In statistics, Levene's test is an inferential statistic used to assess the equality of variances for a variable calculated for two or more groups. Some common statistical procedures assume that variances of the populations from which different samples are drawn are equal such as ANOVA analysis of variance and t-tests.

Levene's test tests the null hypothesis that the population variances are equal (called homogeneity of variance or homoscedasticity). If the resulting p-value of Levene's test is less than some significance level (typically 0.05), the obtained differences in sample variances are unlikely to have occurred based on random sampling from a population with equal variances. Thus, the null hypothesis of equal variances is rejected and it is concluded that there is a difference between the variances in the population.

When performing levenes test on our samples we get the following results:

**statistic=1.0981427347146711, pvalue=0.30252486681634133**

The Pvalue>0.05 than we fail to reject the null hypothesis, and for alpha of 0.05 the variance of the two distribution is equal.

Since we can assume equal variance on the distribution we can proceed on to a parametric Paired T-Test.

## Paired T-Test Analysis

When performing Paired T-Test test on our samples we get the following results:

**statistic=3.2187453343448613, pvalue=0.0081772999362577214**

The Pvalue<0.05 therefore we reject the null Hypothesis, and the Dynamic Programming produces better results with significance of 0.008.

## Wilxcon Sign Test

Another approach would be to conduct a non-parametric wilxcon sign test. When performing the test we get the following results:

**statistic=0.0, pvalue=0.00065495834338569539**

Here as well we can see that the Pvalue<0.05 therefore we reject the null Hypothesis, and the Dynamic Programming produces better results with significance of 0.0006.

# Conclusions

The strength in this implementation of the Branch and Bound algorithm for the Dynamic Time Warping use case is intensive use of concurrency together with the selecting the Go language as one that stands out in its performance in multi-threaded tasks.

The execution time results show that the original solution to the problem– dynamic programming, out performs this solution – Concurrent Branch and Bound, with significance of 0.05.

Nonetheless I see several aspects in which the Branch and Bound might produce superior results, or have advantages over the dynamic programming approach.

First, when running a concurrent application, one depends heavily on the processing environment, the more CPU, RAM, and more importantly processors you have, the faster the application would run. Therefore, I believe that tests with longer time series lengths should be conducted in a more advanced machine where the Branch and Bound could utilize more resources.

Second, the dynamic programming approach requires M\*N memory, whereas the branch and bound requires only the current possible optimal paths. If we limited the number of threads that were running concurrently, we could take advantage of less space for the algorithm execution. Now, some applications require better time complexity and others might require better space complexity. In the latter case, we could take advantage of the Branch and Bound approach.

Third, the Branch and Bound approach produces anytime results, which again, depending on the application, might be a better fit for some use cases.

In conclusion, if you run on commodity machines, I would opt for using the dynamic programming solution, otherwise, I would consider opting for the Branch and Bound solution.

# Future work

I see two aspects in which we can take this work further.

First, I believe that this should be tested in much more capable machine – more CPUs, more cores, RAM. As mentioned earlier, this solution relies heavily on concurrency, and it would have much more advantage in environments that are more fit to run multi-threaded applications.

Second, this implementation uses an initial upper bound which is the sum of the cost matrix diagonal. If we could think of a better initial upper bound, we could filter out inferior solution more quickly and shorten time execution as well as memory consumption dramatically.

# Bibliography

Senin, Pavel. "Dynamic time warping algorithm review." Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA (2008): 1-23.

Sakoe, Hiroaki, and Seibi Chiba. "Dynamic programming algorithm optimization for spoken word recognition." *IEEE transactions on acoustics, speech, and signal processing* 26.1 (1978): 43-49.

https://golang.org/