Bachelor Thesis

Pattern based sub-sequence search in time-series data

Verfasser

Rastislav Hronsky

angestrebter akademischer Grad

Bachelor of Science (BSc)

Wien, 2018

Studienkennzahl lt. Studienblatt:	A01501140
Fachrichtung:	Informatik - Scientific Computing
Betreuerin / Betreuer:	Univ.-Prof.Torsten Möller, PhD

Contents

This work has been made possible thanks to the greatly passionate scientist Torsten Möller, who supervised this thesis together with Harald Piringer, the head of the visual analytics group at VRVis.

My warmest thank you also goes to Florian Spechtenhauser, Thomas Mühlbacher and the rest of the team, for all their valuable input and cooperation on this work.

Last but not least, I wish to express my gratefulness to my family, for their endless support and motivation throughout both harmonic, and turbulent times of my life.

Abstract

In this work, we introduce a supervised approach for motif discovery in time series, which is mainly inspired by the UCR-Suite. The original implementation of the UCR-suite is a NN-search. The altered version of the code (which has been developed in this work) searches for an unknown amount of pattern occurrences, which are similar enough to the given query-sequence. A need for a new parameter arose, which determines the maximum distance between the search query and candidate sub-sequence, in order for it to be positively classified as a pattern occurrence. The major change in the implementation is, that this parameter (fixed number) replaces the best-so-far (BSF) variable in the code. We found out how to reasonably select the parameter value, assuming the search query (pattern) belongs to a class, and the class is present in the time series. This was done by observing of how the amount of matched sub-sequences grows, while increasing the threshold value. Suppose we plot these results on a bar-chart, having the increasing threshold value on the x-axis, and number of matched sub-sequences on the y-axes, the location where the optimal threshold value can be found is the first significant plateau.

1 Motivation

A lot of things in the world can be measured. There is no need for complex measurements to create an interesting dataset, examples being things like how many times a week someone had a workout, what was the temperature at 8 o'clock, etc. Usually, these things can be measured repeatedly, at different times. This gives to the information an additional, *temporal* context. One could think of other contexts as well. For temperature, it would probably make sense to also note, where was it measured. This would add another context to the information, a *spatial* one. Data with both of these information, are called *spatio-temporal* data.

What is more of our concern, is the temporal information, however. By storing these data, lets say a measured outside temperature every hour, we essentially create *time-series*. Other than giving the possibility of finding out when what happened retrospectively, time-series often have a potential of encoding more knowledge, than the raw data shows, that we see at the first sight. This intuition is probably caused by the very essential substance of our human lives, the observation of *causality* (or cause and effect). By seeing the relationship between 2 events, where one of the events preceded the other one, and therefore might be partly responsible for the way the newer event appeared. Causality is metaphysically prior to notions of time and space [3, 4]. This is also the way that people learn from mistakes - by observing cause and effect, multiple events happening one after another, in different temporal contexts. Essentially, this is how an individual survives in the world - by being able to learn and therefore adapt.

The importance of analyzing and working with time-series is certainly great. The personal motivation to realize this work mainly comes from these facts.

Apart from that, it is also devoted to increase the functionality of the Visplore [5] - the visual data-exploration framework.

1.1 Problem statement

As already stated, time-series are relatively easy to create. By collecting data in time, very long data-sets can be created, because time obviously does not take any breaks. Hence, the algorithms dealing with these data have to take the performance very much into account. The domains where time-series often appear include climate, energy, finance, medicine, speech recognition, and many others.

Often, a common problem that the domain experts face, is looking for a specific behavior of the time-series - a kind of a pattern, motif. In the medical sphere, a good example could be electrocardiogram (ECG). There are several known patterns in an ECG, which have some health consequences. An example is the "Brugada sign" [8]. There are several interesting patterns in financial data, such as the famous "head and shoulders" pattern. There are patterns in thousands of other examples. But perhaps there are also some non-famous patterns, some anomalies, which could trigger an experts intuition completely spontaneously. By all means, the tasks with pattern search in time-series are endless and can be dealt with computationally, rather than manually. Our desire is to solve a situation, when an expert:

1. explores a time-series data-set,
2. the visually available exploration area is limited (the size of the data-set is much bigger than the explored area),
3. spots an interesting pattern, and wants to find its all occurrences throughout the data-set.

The upcoming procedure would be visually selecting the range on the x-axis, where the pattern is. The tool would then find all the pattern occurrences in the given time-series.

1.2 Goals

This work aims to find and develop a way of a pattern search approach to time-series data T1, where the search query is a either a sub-sequence of the data given, or another time series T2, where $|T2| \ll |T1|$. The output has to contain the sub-sequences of T1, which are similar enough to T2. By similar, it is meant, that a human would visually perceive the two patterns as similar - the shape-similarity is important.

Another goal/requirement for the method is to be quick, so that it can also be used in interactive visualization tools, like the Visplore [5]. However, the actual implementation for use in the tool is not within the focus of this thesis. The focus is to develop a method, which could be incorporated by such tools.

1.3 Visplore

Visplore is a powerful software developed at the VRVis research center[1], which can generate various dashboards designed to tackle multiple tasks. These tasks are often time-series related. An example science project implemented into Visplore is [1]. The dashboards are highly interactive, designed to visualize complex datasets of big sizes. One such dashboard is devoted to "Cycle analysis". With this dashboard, experts can analyze time-series with recurring patterns - motifs, and further inspect the features of the extracted subsequences. The subsequence-extraction task is what we are mainly going to focus on. The inspiration (hence also motivation) for this work mainly came from the experience of working with this dashboard, where this kind of functionality was desired.

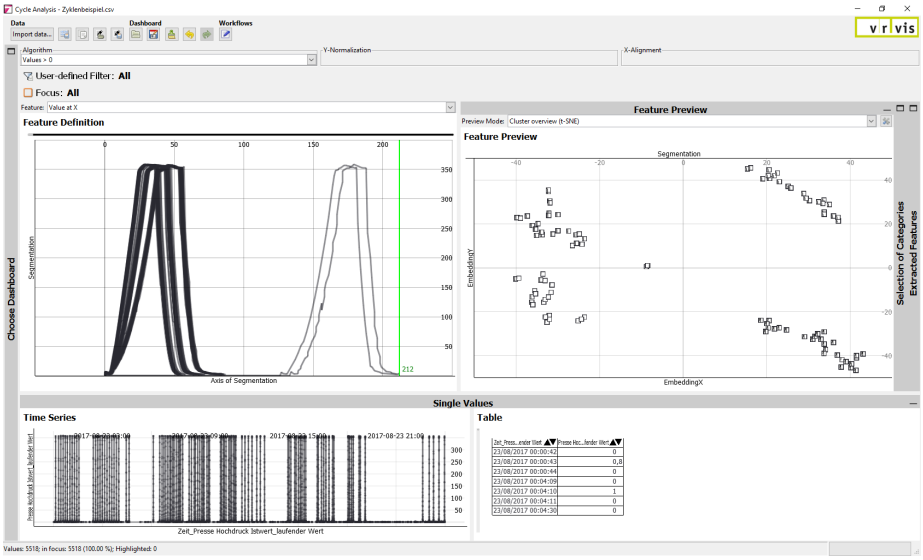

Figure 1: An example of a dashboard in Visplore and its capabilities. On the left side, there are extracted subsequences of time-series data (or simply curves), which are visualized on the right side using a dimensionality reduction algorithm - TSNE [21]. The implementation of this very plugin was part of the experimenting-with-Visplore phase, which preceded this work.

2 Related Work

A lot of work has been done concerning time-series clustering [20], data-mining and classification. The main problem we share with these disciplines is the need for a well suited similarity measure. These papers [6, 7, 10] usually cover the topic of distance measures quite comprehensively, since it is one of the

[1] VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH, https://www.vrvis.at/

major challenges in the domain. In fact, our problem is quite similar to time-series classification. The difference is, that classification problems usually do not need to extract anything, the sub-sequences are given. What we need to do, is extracting the instances of 1 class from time-series. Some interesting classification problems which use the dynamic time warping:

1. Old handwritten musical symbol classification by a dynamic time warping [18]
2. Identification of ancient coins based on fusion of shape and local features [19]

2.1 UCR-Suite

An immense amount of work concerned with time-series has been done by Eamonn Keogh. One of these papers is the UCR Suite [2] - a framework designed for very quick indexing based on the dynamic time warping (DTW) distance measure. The method proposed aims to handle huge data sets, more precisely even a trillion (1,000,000,000,000) data points. To put this into perspective, this is a quote from the paper: "If we have a single time series T of length one trillion, and we assume it takes eight bytes to store each value, it will require 7.2 terabytes to store. If we sample a electrocardiogram at 256Hz, a trillion data points would allow us to record 123 years of data, every single heartbeat of the longest lived human" [2]. The framework uses various lower bounds to quickly prune off the distance computation, before the actual DTW is computed (namely the LB_{Kim} [13], $LB_{Keogh}EC$ and $LB_{Keogh}EQ$ [12]). The idea is to have a "best-so-far" (BSF) sub-sequence, which has the so far least distance to the query. As soon as any of the bounds resolve in a distance greater than the BSF, the currently computed sub-sequence is pruned off and the search jumps on to the next index. In case a sub-sequence is found, which is more similar to the query, the BSF will be updated to this current sub-sequence. The bounds are sorted as follows: starting with the quick ones, but having less "pruning power" (e.g. with constant complexity, but more sub-sequences pass through it), continuing with more complex ones (tighter, with more pruning power - fewer sub-sequences pass through it, but takes more time to compute), all the way to the DTW, which will have to be computed if all the previous tests/bounds "failed".

This paper was a great inspiration for this thesis, and by speeding up the DTW, it solves our main problem - the need for a quick shape-based pattern search in time-series algorithm. Our usage and goal of the approach are a little bit different however. The UCR-Suite is an indexing approach. The main input for it are data and a query (+ some parameters). The program scans the data and finds the best matching sub-sequence (for the given query), which is the output. Our output needs to be multiple sub-sequences, which are reasonably similar to the given query. This obviously raises a question, about how similar a satisfying subsequence should be, which results in a need for another input parameter - in a form of a distance-threshold (which we introduce in our work).

3 Method

3.1 Clarification of terms

Time-series is a vector $T = (t_1, t_2, ..., t_n)$, where n is usually a very high integer value (the length of the time-series).

A *query* is the pattern that we look for in the time-series T. It is a vector Q of length m.

Out of the vector T , it is possible to create $n-m+1$ *sub-sequences* of length m, where $m \ll n$. A sub-sequence of time-series T at position i is defined as $T_i = (t_i, t_{i+1}, t_{i+2}, ..., t_{i+m})$ where $0 < i \leq n - m$. If the set of all possible sub-sequences of length m in T is called Z, it applies that $Q \in Z$.

A *shapelet* is a sub-sequence, which holds some sort of pattern in it, belonging to a class of patterns [16, 17]. A subsequence classified as similar to the query (pattern), could therefore also be called a shapelet. In this work, we sometimes simply refer to the these sub-sequences as to *matches*. Generally, the process of searching multiple occurrences of a pattern in time-series, can be referred to as *motif discovery*.

In this work, referring to the letter n means the *time-series length*, and m means *query length*.

When specifying the value of a distance *distance* in this work (e.g. in visualizations), it is the squared value of it. The abbreviation ED stands for the Euclidean distance, and DTW for the dynamic time-warping distance measure.

3.2 Hands-on experience

In the first phase, we carried out the project by working with the Visplore and one of its dashboards devoted to recurrent pattern analysis. We have been exploring some data-sets with it, programming and incorporating some new components.

That was the initiative that has been made, in order to come up with some reasonable requirements for the method we develop in this work. To get a vision, of how could the eventual result be practically engaged in a widely used tool.

Meanwhile, we have come up with some questions, which originate from the problem of pattern based multiple sub-sequence search. This problem, we consider quite important, because it is often the very first necessary step when analyzing time-series. The main questions to solve are:

1. how to make the initiation of the subsequence-extraction user friendly?
2. how to make it yield meaningful results?

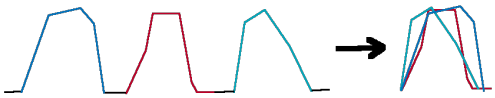

Figure 2: Sketch of how the sub-sequences containing a pattern can be compared to each other when extracted. The Visplore software can extract various features from such curves. To name a few: value or gradient at focal point x, area under curve, etc.

User friendly is a wide term, but for us it mainly determines what the input should look like, and that we should make a lot of effort in making the necessary parameter(s) as easy to use and understand as possible. In the best case, it would be possible to automate the choice of the parameter(s), or completely eliminate them. Also, it would be good for the approach to be fairly quick. Later in this work, we attempt to comprehensively evaluate the parameter-choice, so that this approach at least partly aspires to satisfy the aforementioned requirements.

How the criteria for "yielding meaningful results" have been chosen, will be explained later, in the section "Pattern based".

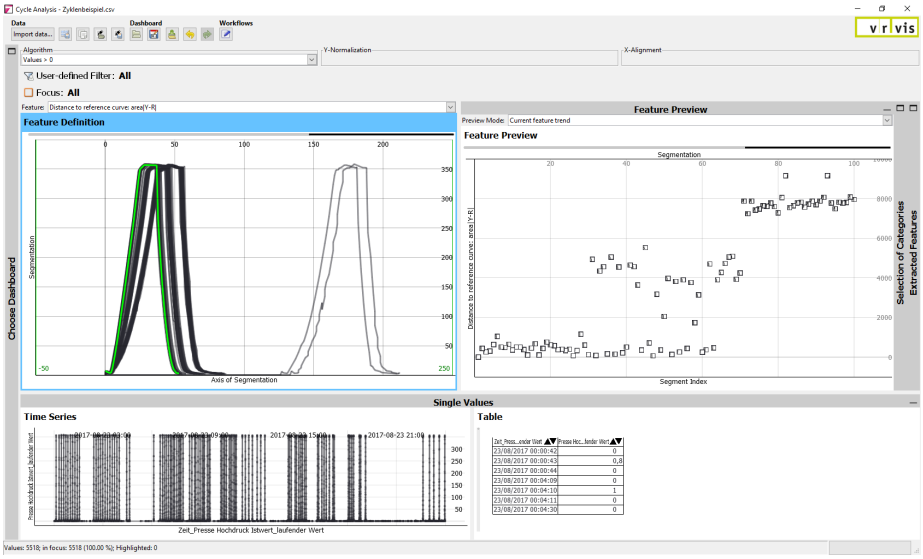

Figure 3: The recurrent pattern analysis dashboard in Visplore. The dashboard contains multiple views, which all are connected together, with the goal of providing a very interactive experience. This is a dashboard, which could possibly make use of our method and incorporate it. In the upper-left frame, there is an ensemble of segments extracted from long time-series, which can be found in the frame underneath it (analogy illustrated in 3.2).

At first, we have been trying to perform the sub-sequence extraction with some simple strategies.

For example, the "$Values > 0$" recognizes a start of a subsequence when the time-series value exceeds 0, and ends it when the value drops back to 0. This way, the subsequences do not even have to end up having the same length. In the figure 3.2, this approach was used. The spikes in the bottom "Time series" view, are the same data as in the upper "Feature definition" view, only in the upper view they are extracted (by the rule just described). The other three options however cut the time-series into subsequences of equal length (the time-series will be divided by fixed intervals).

Such simple approaches can deliver surprisingly satisfying results, when tailored for the data well. A lot of *cyclic* data, often resulting from natural processes, can be simply divided by fixed time intervals. E.g., when measuring solar radiation, the interval of 1 day delivers the daily patterns, which are fairly regular. The resulting subsequences would be perfectly sufficient without the need for fancy algorithms.

3.3 Pattern based

However, the dashboard still missed a more sophisticated approach, which would be able to tackle the challenge of pattern based sub-sequence search. When speaking about "pattern based", one needs to measure similarity between 2 time-series somehow, in order to recognize the presence of a pattern (a pattern is present, if the distance between the query (containing the pattern) and candidate subsequence is low enough).

Similarity function works very similar to a distance function. A similarity is indirectly proportional to a distance, therefore, we can work with a distance measure just as good as with a similarity measure. There are several distance/similarity measures. For time-series, some relevant distance measures are the Euclidean distance, Pearson's correlation coefficient, the Kullback-Leibler divergence, or the dynamic time warping (DTW). In [9, 6], a lot of research has been done into comparing these (and many others) under various time-series clustering tasks.

Usually, the time-series similarity measures target one of three main objectives, with respect to different approaches [10]:

1. similarity in **time** - similarity occurs, when same events happen at the same time; can be typically achieved with correlation coefficient, or Euclidean distance

2. similarity in **shape** - similarity occurs, when the curves share same shape-features/trends, which can be even occuring with a phase shift, etc.; typically achieved with DTW

3. similarity in **change** - "...the similarity in how they [time-series] vary from time step to time step. For example, a stock analyst may wish to cluster

together shares that tend to follow a rise in share price with a fall the next day." - a quote from [10].

Our focus lies on the *shape* objective, which is why the dynamic time warping should work well for us and we choose to use it.

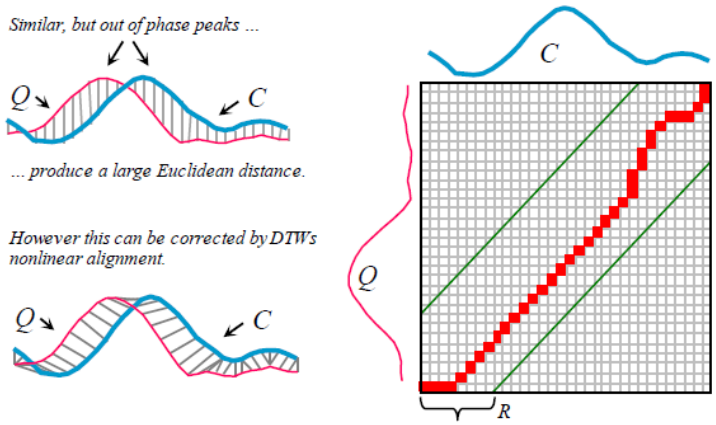

Figure 4: A figure from [2], nicely illustrating the difference between the Euclidean distance and DTW. The DTW is robust against phase-shifts on a local level, whereas the ED strictly compares the given time-point. On the right side, there is a visualization of how the warping path (within a warping window - green) is computed from a matrix. The need for a matrix is basically responsible for the quadratic complexity of the algorithm. With a warping window of size R, the complexity is $O(n * R)$.

The DTW also has its downsides, the main one being the complexity. It takes $O(n^2)$ for computing a whole warping window, and $O(n*R)$, for a warping window of size R. This is too high for most use-cases [2]. However, there are various dynamic programming approaches, which are capable of speeding up this method miraculously. One such method is the famous UCR-Suite.

4 Implementation

The UCR-Suite implementation is written in C++. The main part of the code consists of a loop, which iterates over the time-series. The time-series have to be equidistant (regular sampling). Before the loop, a BSF (best-so-far) variable is initialized to an infinity (numerically very high value). Throughout the iterations, this variable holds the least distance computed so far. Every time a new smaller distance is found, the BSF distance will be overwritten to this new distance.

In each iteration, a hierarchy of lower bounds will (or will not) be computed. If all the bounds are lower than the BSF, the DTW has to be computed. The value then may or may not be lower than the BSF-distance. In case it is lower,

as already mentioned, the BSF value will be updated to this new value. Additionally, the location (index) of this value will be saved for future referencing. The computation then continues with the next iteration.

Algorithm 1
Original version

```
BSF = INF // set best-so-far to infinity
loc = -1
T = input data, n = length(T)
Q = query, m = length(Q)
while i < n − m + 1 do
  if bound1(T[i : i + m], Q) < BSF then
    if bound2(T[i : i + m], Q) < BSF then
      if bound3(T[i : i + m], Q) < BSF then
        dist = DTW(T[i:i+m], Q)
        if dist < BSF then
          BSF = dist
          loc = i
        end if
      end if
    end if
  end if
  i = i + 1
end while
```

From the algorithm 1, it is apparent, that the computation performs the best, when it reaches iteration, when the BSF is already low. In this situation, it is more likely, that the iteration will be abandoned quickly, even by checking the first lower bound. The DTW-distance will only be rarely computed (usually in less than 1% of the iterations).

The algorithm 2 outlines our major change to the implementation. We do not have a BSF, we have a threshold, which stays constant throughout the whole process. All the sub-sequences, with a smaller distance than the threshold are stored in an array.

This adjustment is responsible for the change of input/output. The input now requires to specify the threshold, and the output is not a single subsequence, but multiple subsequences.

Apart from this change, there are 2 more details. The first question is, when to start considering an index being matching. Stumbling upon an index, where the distance is already less than the threshold does not mean, that this is the best match possible in this area, because right at the next index, there could be an even lower distance, than here. In our approach, we look for the first local minimum. As soon as the distance is not lower anymore, meaning $dist(T[i+1], Q) > dist(T[i], Q)$, the subsequence will be considered *similar* (pattern found).

Algorithm 2
Modified (and simplified) version

```
subsequences = empty array
i = 0
T = input data, n = length(T)
Q = query, m = length(Q)
th = threshold
while i < n − m + 1 do
  if bound1(T[i : i + m], Q) < th then
    if bound2(T[i : i + m], Q) < th then
      if bound3(T[i : i + m], Q) < th then
        dist = DTW(T[i:i+m], Q)
        if dist < th then
          subsequences.push(i)
        end if
      end if
    end if
  end if
  i = i + 1
end while
```

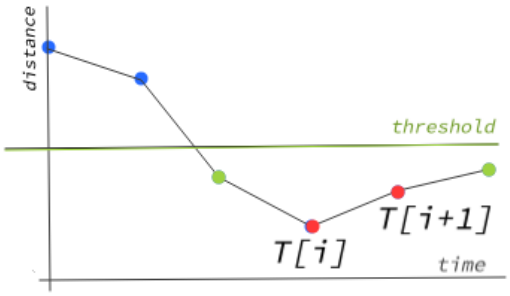

Figure 5: Sketch indicates when a similar subsequence is detected. Blue points are above threshold, green and red points are below the threshold, at $T[i+1]$ a similar subsequence has been detected.

An alternative approach could be creating a whole distance profile of the data first, and finding the minima afterwards, in a second run over the data (when distances for all data points are known). Like this, it may be possible to select a more reasonable local minima, since the distance is computed for every single point, without skipping. We did not implement it this way, because the greedy approach that we used

1. yields satisfying results as well (as shown in the visualizations later, com-

paring to a distance profile computed by pure DTW),

2. is faster (since a lot of iterations can be skipped, and there is no need for a second optimization-run over the data)

The second change is, that at this point, when a pattern has just been detected, we do not have to compute anything for the next m iterations (indexes). Therefore, from now on, the next m iterations will be skipped. The longer the query is, the better the effects on runtime this should have (more iterations skipped).

The majority of our code is identical with the original implementation. It is written in plain C++, without using any kind of framework. The compilation is very simple, all the code resides in one file. The application can be simply run from the command line. The original application needs 4 parameters to work:

1. the data file
2. the query file
3. query length
4. warping window (float value between 0 and 1; 1 means computing the full DTW, 0 is essentially the Euclidean distance; a good default value is 0.05), visualized in 3.3 by the green lines in the matrix (R)

It gives a brief output containing the time it took to finish, the portions of the bounds used, the least distance found, the index where it was found, etc.

Our modified code works similarly, with a few additional things. A fifth parameter has to be specified - the distance threshold. It can be any float value. For debugging purposes, additionally it is possible to specify:

1. whether to use pruning at all, or just compute the DTW for every subsequence,
2. and whether to output the brief information set as in the original version (instead of the least distance we output the amount of matches), or a data output.

The data output is in a form of a CSV file, and is useful for visualizing the results afterwards.

The program reads the data in *double* precision. The query as well, and so are performed all the operations. Both the data and the query are z-normalized. A lot of reasoning about why z-normalizing makes sense can be found in [2, 11]. All the logic, that we are concerned with, happens in the *main* method. Apart from that, the implemented functions are either "housekeeping", or the implementation of the lower bounds.

5 Evaluation & Discussion

The main method we used for evaluation was visualizing the outcome of the algorithm. We did it by using various data sets, and observing the behavior across different parameter settings. Three datasets will be used for a demonstration in this work (later described).

5.1 Distance profile

The first experiment we did, was creating distance profiles. A distance profile is generated by computing the distance for every sub-sequence (with the length of the query) in the data. We did this to get a better insight into the DTW-distance behavior, thus to improve our ability of justifying the results better. We got pictures such as the following:

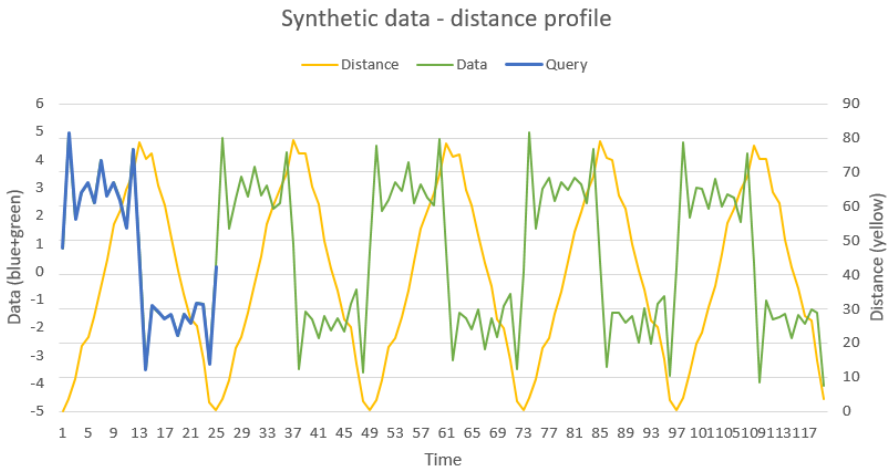

Figure 6: The actual distance profile is the yellow line, which corresponds to the Y-Axis on the right. The green-blue line are the data (green), and the query (blue). The query is an actual sub-sequence of the data, which is why it lies over it at points 1 to 23.

The data on this plot is raw. However, the computation of the distance is performed with z-normalized data. For this plot, no pruning was used, these are all actual squared DTW-distances. This example does not provide too many surprises though. The distance profile oscillates with the same frequency as the original time-series, the minima having at the start of the period, and maxima in the middle.

5.2 The additional parameter

A good algorithm should have as few parameters as possible. If there is no way of eliminating some of them, they should be at least well understood, so that

they can be properly and easily set. Many parameters in data-mining algorithms bring a lot of pitfalls and dangers [14], such as:

1. failing to find existing patterns
2. finding patterns which do not exist
3. over/underestimating the significance of a pattern.

More on this topic can be found in [14, 15].

A good way of communicating a similarity-threshold like ours, is having an "envelop" around the query. This is not doable with our approach, since our distances are computed on normalized data, and DTW is by its nature a *warping* approach, which means there are too many ways of how the sequences can be misaligned, misshapen, etc. What we did to help to understand the impact of our new parameter, was iteratively analyzing the amount of matching sub-sequences throughout all possible threshold-values. All the way to a state, where no more new sub-sequences can be gained.

We found out, that the parameter setting is very dependent on 2 factors. The first being the *query length*. Since the parameter we use is a distance threshold, it is expressed in "distance units" (actually, in our case they are *squared* distance units). When computing a Euclidean distance between two points (tuples), the resulting absolute value is generally higher for high-dimensional points, than for low-dimensional points. Therefore, with a longer query, we also have to expect bigger distances.

This problem could be eliminated by changing up the parameter a little bit. We could specify it *relative* to the query length - as a *multiple* (e.g. $2 * m$). We did not do this however, because of 2 reasons.

1. keeping the parameter simple - explanation as absolute distance value is easier
2. we would otherwise end up mostly specifying very low multiples, such as 0.005, although the parameter-range would not be 0-1 anyway, which then does not bring too much advantage

The second factor influencing the correct choice of distance-threshold is relatively obvious. The *character* of the data. The semantics of the data is a strong factor, which needs to be taken into account. Some important questions are:

1. Are there clearly separable classes?
2. Is there randomness present? How much?

At this point, a little confusion about setting this parameter properly is absolutely on point. Picking up a correct one is not easy. Perhaps the exact opposite, at this point. However, we thought of a method, which could make this easier for us, and maybe could even be used to automatically set this threshold.

5.3 Selecting the threshold correctly

Because it is a rather challenging task to get this parameter right, we decided to evaluate it by trying its all possible values, and visualizing the results. The assumption was as follows. Lets assume that the query is a legit pattern (an instance of a pattern class), which is actually present in the data. Suppose, that there are multiple occurrences of that pattern in the data. There might be also other patterns in the data, however they are *distinguishable* from this query.

On this assumption, if we start with the threshold at a low value (lets say 1), and we will increase it by small steps. We can expect, that suddenly more and more matching patterns are getting picked up from the data quite soon. Then, suddenly the newly picked up amount of matches should fall to nearly 0 (if not exactly 0), for the next few threshold increases. After a while, when still increasing the threshold, we should start picking up some new matches again (which could be the other patterns in the data, or simply noise).

With this scenario, the ideal threshold would be exactly at the point when the amount of newly picked up matches stopped growing for the first time (the first plateau). An example of such a plot is figure 8.

5.4 Performance & bounds usage

The UCR-Suite is only as fast, as it is able to use its lower bounds. Obviously, when setting the best-so-far (BSF) to a fixed number, we cannot benefit from the bounds as much as if we were doing the original, nearest-neighbour search. The more benevolent (higher) the threshold is, the less the usage of the bounds will be. The method thus favors having the BSF as low as possible.

The performance gain from the original UCR-suite is known. It is very high, because only in less than 1% of the cases, the DTW has to be actually computed. The method attacks the linear time-complexity, sometimes even being faster than the Euclidean distance (as claimed by [2]). LB_{Kim} has a constant time complexity, LB_{Keogh} has the complexity of around O(n) [2]. Our performance will suffer from having a "fixed BSF", but benefit from skipping the iterations, where a pattern is present. We evaluate our performance by observing, how much use do we make from the lower bounds.

5.5 Case studies

Three case studies are presented below, summarized in the following table.

Data	*Clusters*	*Query length*	*Data length*
Synthetic	Clearly separable	23	2402
Random walk	No clusters	200	1000000
Arrow tip shapelets	Moderately separable	444	19152

5.5.1 Synthetic data

The first case study has been conducted with a synthetic data-set, which contains 2 kinds (classes) of patterns, and some noise. The query is one of the occurrences of pattern 1. Altogether, there are 50 occurrences of pattern 1, and 50 occurrences of pattern 2. The patterns look as follows:

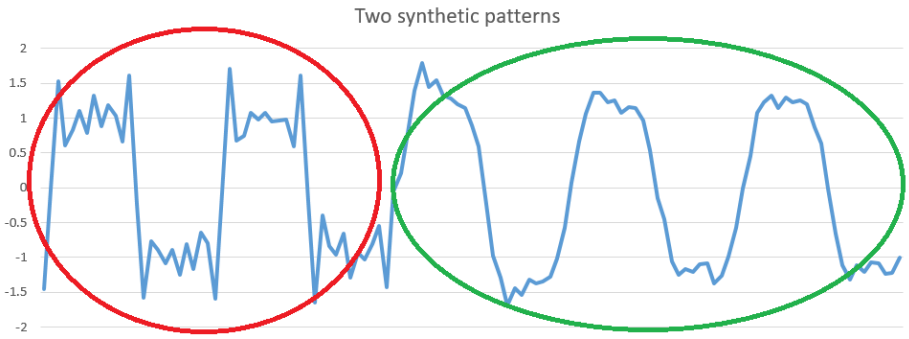

Figure 7: The 2 synthetic kinds of patterns

Two instances of pattern 1 are annotated with the red circle, and three instances of the pattern 2 are annotated with the green circle. The patterns are quite similar, but clearly distinguishable with a human eye.

The distance profile with the query are visualized in the figure 6.

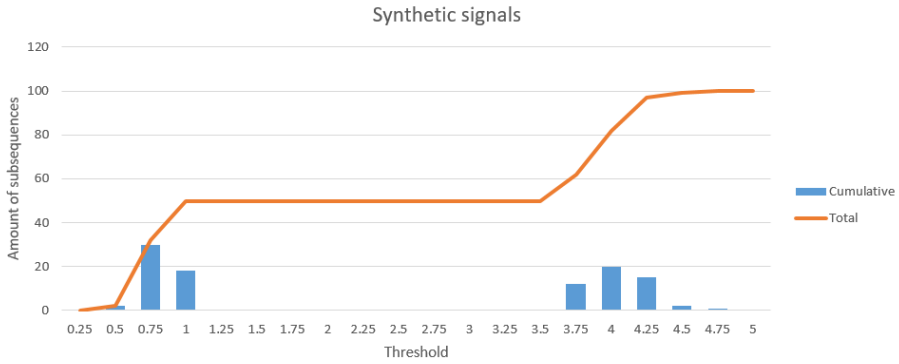

Figure 8: The development of the amount of matches with increasing threshold in the synthetic dataset. the current threshold is on the x-axis. On the y-axis, the unit is *number of matches*.

In figure 8, The threshold has been increased by 0.25 in each run (that corresponds to the size of the steps on x-axis). The orange line represents the total amount of matches at a certain threshold. The blue bars represent the newly found subsequences since the last threshold increase. According to the method we proposed, the optimal threshold with this query should be approximately

1.25. At that point, all the instances of the pattern 1 should be already classified positively, while picking no other, undesired non-matching sub-sequences. Around the threshold of 3.75, the patterns of the second class start being classified as a match. At around 4.75, all of the patterns (both 1 and 2) are already classified positively.

This is a sample illustration, of how the results look like with the threshold set to 1:

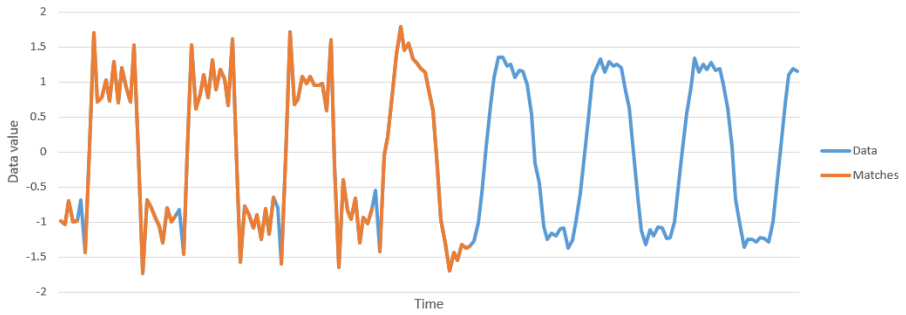

Figure 9: The orange parts are the found pattern occurences. Between them, there is a short blue bit, where the neighbouring subsequence has not been detected yet. Blue is plain data. The pattern-query that was used is the same one, as visualized in 6

It is visible, that the last subsequence classified as similar looks a bit like an instance of the second pattern, but has those two spikes, which make it also look a bit like the pattern 1. In this picture, the first 3 patterns had a distance of approximately 0.3 with not too much variation, and the last one was 0.72 - slightly more.

Since the lower-bounds utilization gets worse with increasing distance-threshold, we created a visualization throughout increasing threshold settings. The visualization is created based on the same data-set, as the previous one - the synthetic signal with 50 patterns of one kind, 50 patterns of other kind, while all of them lie right next to each other (without areas, where there is no pattern).

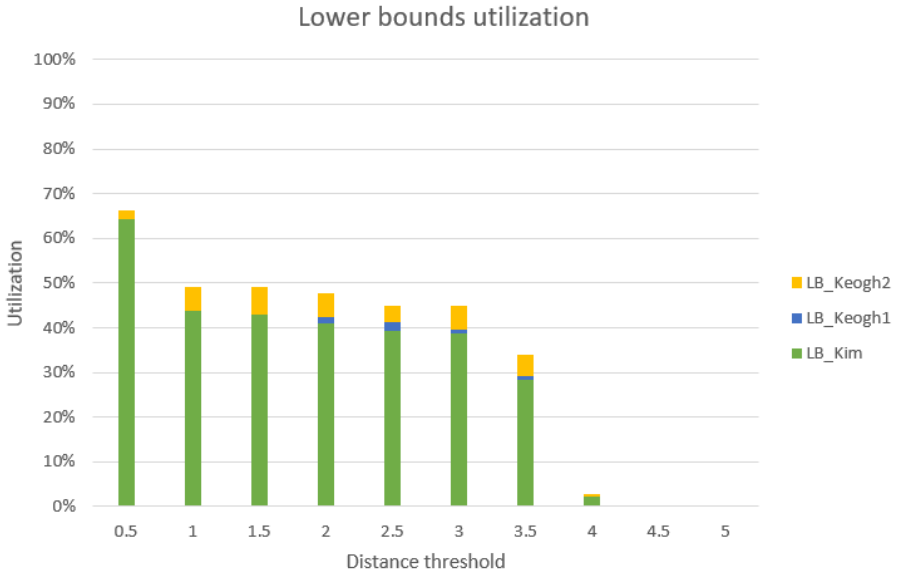

Figure 10: On the x-axis, the threshold increases by steps of 0.5. On the y-axis, the usage of the 3 lower bounds is displayed as a percentage.

In the bars at thresholds 1 and 1.5, the non-pruned part (from the top of the bar all the way to 100%) approximately corresponds to the portion of sub-sequences, which have been classified as a pattern (50%). If a sub-sequence is classified as a pattern, the actual DTW had to be computed once, and for the next $m - 1$ iterations, nothing was done. Hence, the space between a bar and 100% contains some iterations, where DTW was computed, but the major part are skipped iterations. However, this portion can be also bigger than the amount of matches (50% in this case). It happens, when all the lower bounds have been passed, then the DTW has been computed, but in the end, the distance was still bigger than the threshold. These were the cases between thresholds 2 and 3.

5.5.2 Random walk dataset

In the next example, we will use a random walk data-set of length 10000 data-points, and a query length of 200 data-points. Again, the query has been picked as a sub-sequence from the data. Therefore, at 1 point, we should get a distance of 0 (which is also the case, around point 206 - see figure 12).

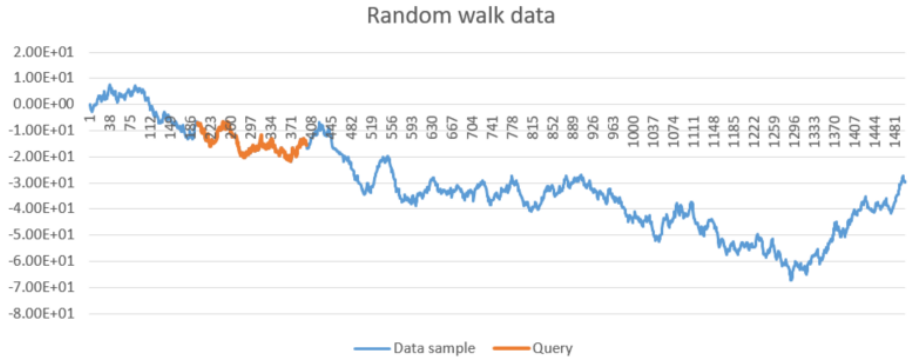

Figure 11: A plot of a 1500 points long sample of the random walk data, including the query (orange).

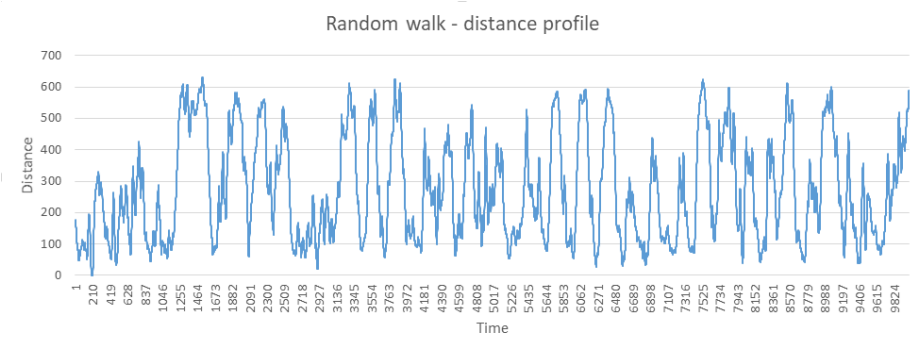

Figure 12: Distance profile of the random walk data (10000 points long sample).

As we can see, the distance oscillates in a more of an irregular pattern, and reaches relatively high values (> 600). Around 600 would be a threshold, where pretty much every sub-sequence should be picked up, and by increasing the threshold we would get no more new subsequences.

This is what we get, if we prune off at a threshold of 40 during the distance computations:

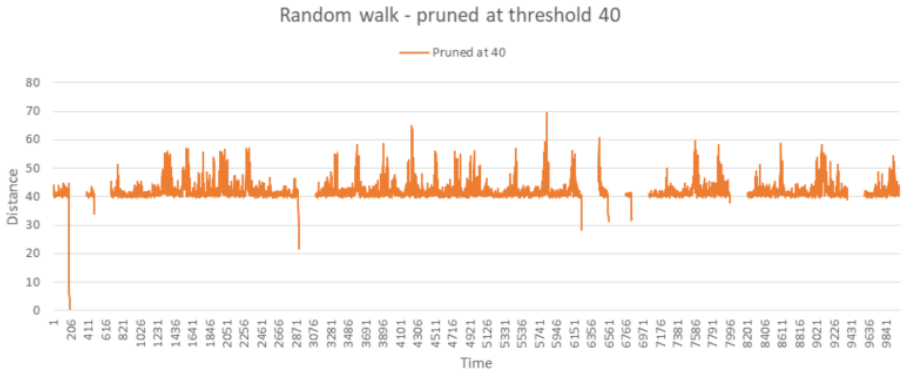

Figure 13: Distance profile of the random walk data, with the threshold $t = 40$ (10000 points long sample). Every significant "spike-down" means a similar subsequence has been found, hence stop computing for the next *query-length* points.

First apparent difference is, that there are "gaps" present. These spaces happen to be the discovered subsequences, where no distances have been computed. Also the values (distances) are significantly smaller, than those computed without early pruning. This is also reasonable, because it is only needed for the distance-computation to reach the threshold, to be early abandoned. The actual distance, which would most probably be higher (because the computation would keep summing it up) is irrelevant, because we already know, it will not pass as a similar subsequence. In the next figure, there are both lines - the pruned and unpruned one, but with the y-axis scale restricted to the maximum distance of 80. The gaps in the blue line (unpruned) are not caused by the presence of a pattern, but simply because of the restricted y-axis.

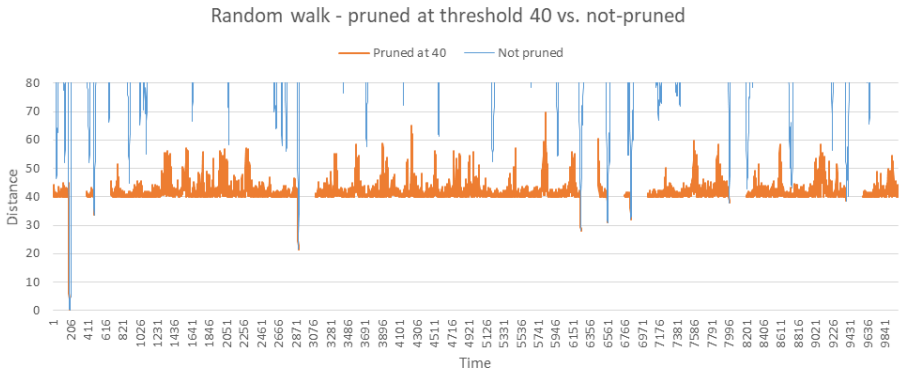

Figure 14: Distance profile: pruned with the threshold $t = 40$ vs. not-pruned. The y-axis is limited to 0-80, the actual values of the distance reach values up to 600.

As a reminder, in this picture we can also see, that this approach shall not be mistaken for an approximation. All the subsequences which would be picked up by the actual DTW-distance, get discovered by the early-pruning approach as well.

In this example, we happen to get quite a reasonable number of subsequences - not too little and not too many. This of course only depends on the fact, that we chose a reasonable threshold value.

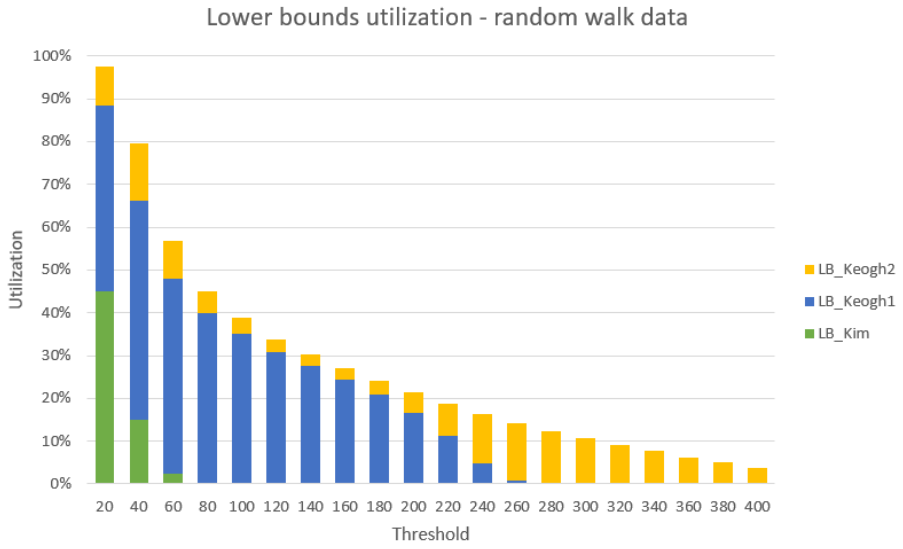

Figure 15: Lower bounds utilization analysis performed on a random walk data set, with no meaningful patterns in it.

In a random walk data-set, suppose it would be infinitely long, the amount of newly discovered patterns would increase with the threshold in a linear fashion. It is not the case with a limited data-set though, because with every iteration, there are less and less possible patterns to pick. Therefore, the bars in the figure decrease slower and slower.

The random walk dataset is 1 million data-points long. On this data-set, we measured the development of runtimes with different threshold settings. The results look as follows.

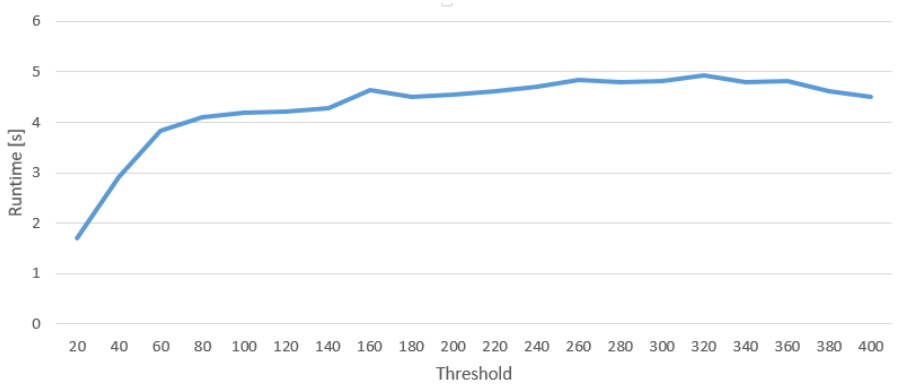

Figure 16: The runtime development across threshold values 20-400

Until the threshold of approximately 300, the runtimes grow. However, the trend is slower and slower, and towards the values of 400, they even begin to fall. The reason for this is, that from some point, the advantage of skipping iterations (because of a found pattern) outperforms the disadvantage of utilizing less lower bounds (because of higher threshold).

5.5.3 Arrow classification problem

Among other experiments, we tried our approach with a dataset, which contains shapelets of various arrow tips. The arrow tips have been transformed into time-series, and concatenated together. It is mainly a classification problem, where the subsequences are not very hard to extract, which is a downside to this experiment. However, the classes themselves are not so easy to detect. Some features gradually transition between the classes, some features just appear. This is what the arrow-tips in the dataset look like:

Figure 17: The arrow tip images used in this example. Only their shape is relevant for the experiment (hence shapelets).

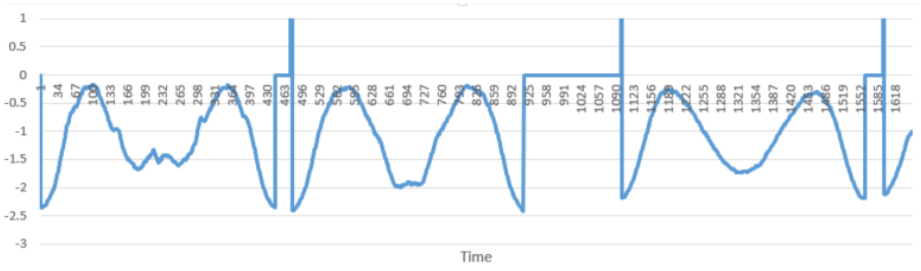

Figure 18: An example from the transformed time-series, which includes 3 arrow tips.

The query (the first arrow tip you can see in the time-series) is 444 data points long, the whole dataset is nearly 20000 points long. These are the amounts of matches we got for various thresholds:

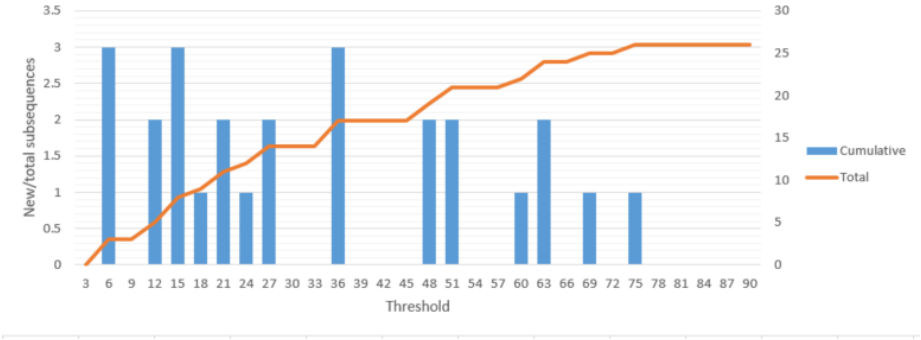

Figure 19: Thresholds ranging from 3 to 90, with a step-size of 3.

The most reasonable threshold in this case would be 6. This is a threshold for very similar arrows. Eventually, for merging with a cluster of slightly less similar arrows (but arguably still in the class), one could opt for the threshold of 27.

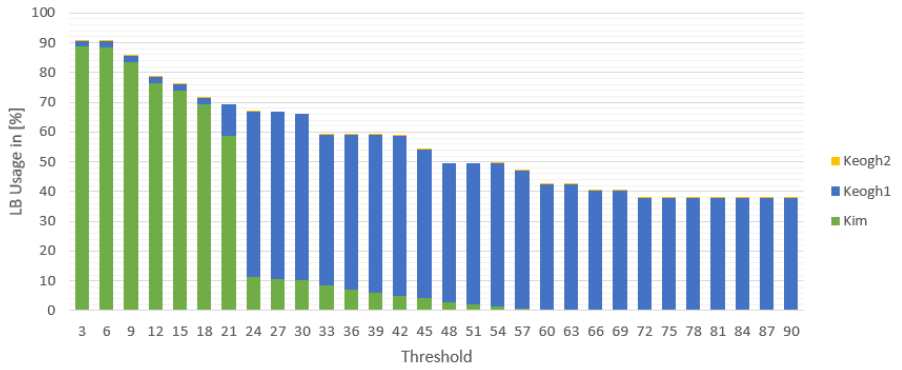

Figure 20: The lower-bounds usage in arrow-tip dataset.

In 5.5.3, we can see how well the quickest LB_{Kim} was used, until the threshold of 24. This indicates, that until that threshold, the pattern was relatively easy to spot, since the non-patterns were pruned off straight with the first LB. Since this threshold also approximately corresponds to the point we previously evaluated as suitable for sub-sequence search, this implies, that the usage of lower bounds is the most effective, when the threshold is set precisely to the "edge of the class's envelop".

6 Conclusions

The proposed approach has accomplished the input/output requirements which have been set. Also, a shape-based, expensive distance measure was used, which measures similarity of time-series in a similar way to the human eye. Those should be 2 checks.

The quick implementation of the distance measure - UCR-DTW - has been successfully modified for our needs, while partly sacrificing the full benefits it offers, when used of nearest-neighbour indexing. However the sacrifice is fairly small, when keeping the threshold low. Therefore, this method performs well, when the patterns are rarely expected to be found, hence the threshold can be kept low.

The approach/idea we developed for choosing an optimal thresholds is a good starting point. But also, it is still not very trivial. This will probably remain a downside of this whole approach. The most intuitive way of communicating distance thresholds are probably envelops, and for DTW under normalized data, such envelop is hardly doable. The general disadvantage of working with DTW is, that the distances are not very easy to imagine, as there are endless possibilities of twists and warps on the curve. The visualization of the threshold that we developed, we consider the main contribution of our work.

7 Future Work

A reasonable future work would definitely be bringing the method to an actual life, by incorporating it into the aforementioned *Visplore* framework. Testing it in a production environment with lots of real-world time-series, will definitely bring up some more insights, problems and ideas. As the work is devoted to the domain experts, it would also make sense to make a study with their cooperation, of what the actual usefulness of the approach is.

The most challenging part of this method, the right choice of the threshold, is probably a chapter itself. The question of how to incorporate this parameter in a most user-friendly way is very tricky. Some ideas and alternatives, which could be taken into account could be:

1. showing the whole bar-chart, which visualizes the amount of newly discovered patterns after increasing the threshold by certain amount, and the user would pick the threshold location according to her visual intuition (our suggestion being the end of the first significant hill),
2. another approach could be trying to automate this completely, by picking the end of the first (or n-th) significant hill automatically,
3. having a slider, which would increase the threshold value exponentially on a scale from 0 to the highest found sub-sequence distance in the data,
4. or maybe, a combination of those approaches.

Again, this would again require some kind of user-study to evaluate, what works the best.

8 Supplemental Material

The code and data can be found at [22], together with a guide for using the application.

References

[1] Piringer, H., Pajer, S., Berger, W., Teichmann, H. (2012, June). Comparative visual analysis of 2d function ensembles. In Computer Graphics Forum (Vol. 31, No. 3pt3, pp. 1195-1204). Oxford, UK: Blackwell Publishing Ltd.

[2] Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., ... & Keogh, E. (2012, August). 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). ACM.

[3] Robb, A. A. (1911). Optical geometry of motion: A new view of the theory of relativity. W. Heffer.

[4] Whitehead, A. N., Sherburne, D. W. (1957). Process and reality (pp. 349-350). New York, NY: Macmillan.

[5] Arbesser, C. and Mhlbacher, T., Komornyik, S. and Piringer, H. (2017). Visual Analytics for Domain Experts: Challenges and Lessons Learned. Proceedings of the second international symposium on Virtual Reality & Visual Computing (1-6).

[6] Aghabozorgi, Saeed, Ali Seyed Shirkhorshidi, and Teh Ying Wah. "Time-series clustering - A decade review." Information Systems 53 (2015): 16-38.

[7] Lin, J., Williamson, S., Borne, K., & DeBarr, D. (2012). Pattern recognition in time series. Advances in Machine Learning and Data Mining for Astronomy, 1, 617-645.

[8] Junttila, M. J., Raatikainen, M. J. P., Karjalainen, J., Kauma, H., Kesniemi, Y. A., & Huikuri, H. V. (2004). Prevalence and prognosis of subjects with Brugada-type ECG pattern in a young and middle-aged Finnish population. European heart journal, 25(10), 874-878.

[9] Chen, Y., Nascimento, M. A., Ooi, B. C., & Tung, A. K. (2007, April). Spade: On shape-based pattern detection in streaming time series. In Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on (pp. 786-795). IEEE.

[10] Bagnall, A., & Janacek, G. (2005). Clustering time series with clipped data. Machine Learning, 58(2-3), 151-178.

[11] Keogh, E., & Kasetty, S. (2003). On the need for time series data mining benchmarks: a survey and empirical demonstration. Data Mining and knowledge discovery, 7(4), 349-371.

[12] Fu, A. W. C., Keogh, E., Lau, L. Y., Ratanamahatana, C. A., & Wong, R. C. W. (2008). Scaling and time warping in time series querying. The VLDB JournalThe International Journal on Very Large Data Bases, 17(4), 899-921.

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

[14] Keogh, E., Lonardi, S., & Ratanamahatana, C. A. (2004, August). Towards parameter-free data mining. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 206-215). ACM.

[15] Keogh, E., Lin, J., & Truppel, W. Clustering of Time Series Subsequences is Meaningless: Implications for Past and Future Research. In proc. of the 3rd IEEE ICDM, 2003. Melbourne, FL. Nov 19-22, 2003. pp 115-122.

[16] Ye, L., Keogh, E. (2009, June). Time series shapelets: a new primitive for data mining. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 947-956). ACM.

[17] Ulanova, L., Begum, N., & Keogh, E. (2015, June). Scalable clustering of time series with u-shapelets. In Proceedings of the 2015 SIAM International Conference on Data Mining (pp. 900-908). Society for Industrial and Applied Mathematics.

[18] Forns, A., Llads, J., & Snchez, G. (2007, September). Old handwritten musical symbol classification by a dynamic time warping based method. In International Workshop on Graphics Recognition (pp. 51-60). Springer, Berlin, Heidelberg.

[19] Huber-Mrk, R., Zambanini, S., Zaharieva, M., & Kampel, M. (2011). Identification of ancient coins based on fusion of shape and local features. Machine vision and applications, 22(6), 983-994.

[20] Liao, T. W. (2005). Clustering of time series dataa survey. Pattern recognition, 38(11), 1857-1874.

[21] Maaten, L. V. D., & Hinton, G. (2008). Visualizing data using t-SNE. Journal of machine learning research, 9(Nov), 2579-2605.

[22] https://homepage.univie.ac.at/hronskyr97/BACC.html

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine Bachelorarbeit in diesem Studiengebiet erstmalig einzureichen.

Bratislava, December 23, 2023 Rastislav Hronsky