Technische Universität Berlin
Electrical Engineering and Computer Science
Institute of Software Engineering and Theoretical Computer Science
Algorithmics and Computational Complexity (AKT)

Faster DTW-Mean Computation on Binary Data

Bachelorarbeit

von Nathan Schaar

zur Erlangung des Grades „Bachelor of Science“ (B. Sc.)
im Studiengang Computer Science (Informatik)

Erstgutachter:	Prof. Dr. Rolf Niedermeier
Zweitgutachter:	Prof. Dr. Markus Brill
Betreuer:	V. Froese, Prof. Dr. Rolf Niedermeier

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Die selbstständige und eigenständige Anfertigung versichert an Eides statt:

Berlin, den

_____

Unterschrift

Zusammenfassung

Dynamic Time Warping (DTW) ist ein wichtiges Data-Mining Werkzeug, das gerade in den letzten Jahren, da das massenhafte Speichern und Verarbeiten von großen Datenmengen einfacher wurde, deutlich an Relevanz gewann. In dieser Arbeit beschäftigen wir uns mit dem Problem zu einer Menge gegebener binärer Zeitreihen eine neue binäre Zeitreihe zu berechnen, die eine möglichst geringe DTW-Distanz zu allen Inputsequenzen hat. Eine solche „Meansequenz" ist für herkömmliche Clustering-Algorithmen von Vorteil. Wir zeigen eine neue obere Schranke für die asymptotische Laufzeit dieses Problems und diskutieren, ob dieses Problem noch schneller (linear) lösbar ist. Außerdem designen wir einen Algorithmus, der unter praktischen Annahmen sehr schnell eine solche Zeitreihe findet und auch das generelle DTW Problem schnell auf binären Daten löst. Wir gehen auch kurz auf ähnliche Probleme ein, welche die gleiche Struktur haben und zeigen, dass wir für derartige Probleme die selbe asymptotische Laufzeit als obere Schranke annehmen können. Abschließend untersuchen wir die Struktur von Lösungen in Abhängigkeit von bestimmten Merkmalen der Eingaben, um eine gute Linearzeitapproximation für das Problem zu finden.

Abstract

Dynamic Time Warping (DTW) is an important tool for data mining, that has seen a rise in relevance over the last couple of years due to recent improvements in the capabilities of data storage and processing. In this thesis we look into the problem of finding a binary time series that has a low DTW-distance to a number of given input time series. Such a "mean" time series is necessary for common clustering algorithms. We prove a new upper bound for the asymptotic running time of computing such a time series and also discuss whether it may be possible to solve this problem even faster (linear time). Furthermore, we design an algorithm for this problem that runs fast in practice and can also be used to compute the DTW-distance of two binary time series quickly. We also look at similar problems that have the same structure and show that the asymptotic running time of those problems is bounded by the same number. In the end, we analyse the structure of output time series depending on specific features of the input time series to find a good linear time approximation algorithm for this problem.

Contents

1 Introduction

Whenever some sensor measures data over time, a time series is created. The most prominent example is a sensor measuring air pressure about 44 thousand times a second, the microphone. Formally speaking, a time series S is a finite sequence of points in a space, that is, $S = s_1, s_2, ...s_n$ with n being the length of the sequence. If we want to use these time series for pattern recognition or other tasks in data mining or machine learning, we need some sort of similarity measure of two time series, so we need to calculate distances in the space of time series. In Figure 1.1 you can see two such time series created from speech data, they have similar shapes but do not match up very well in the time domain. If we try to measure the distance between theese two series S and T with the traditional Euclidean distance defined by $d_{\text{eucl}}(S,T) := \sqrt{\sum_{i=1}^{n}(s_i - t_i)^2}$, then we run into two problems: First, the two series S and T could have different length so what is n? Second, the Euclidean distance only accounts for the differences in S and T where the indices match. In the bottom left graphic in Figure 1.1 we can see what using the Euclidean distance to compare the two time series would look like and there are lots of big black areas that indicate a large distance between the two series.
To solve these problems, we introduce the dynamic time warping distance—short DTW-distance. The DTW-distance measure can be seen as a two-step procedure where we first try to align one time series with the other such that we can stretch out parts of each time series so that they end up with the same length and hopefully are shaped in a similar way (see Figure 1.1 top right graph), and then calculate the Euclidean distance of those aligned time series (see Figure 1.1 bottom right graph).

Dynamic time warping is widely used in data mining domains such as speech or gesture recognition [CTH18; KKS13; VS15; WL16; ZLL14], and can even be used as a good string comparison algorithm for tasks like genome sequencing or shape matching [MP05; Sku+13]. The DTW-distance can be computed with a simple dynamic program which runs in $O(n^2)$ time with n being the size of the of the input strings. This quadratic complexity has also been shown to be the lower bound for computing the DTW-distance over any alphabet of size at least three[BI15; Kus19]. When it comes to long sequences such as time series with high sample rate over a decently long period of time, the expense of running the standard dynamic programming algorithm is often too high. A vast amount of research tackles this problem by developing heuristics that limit the search space, prune warping paths or reduce data dimensionality while trying to ensure a good approximation of the exact algorithm. Recently, the special case of computing the DTW-distance on binary data has caught more attention and new algorithms [Mue+16; Sha+18] were developed which yield exact results on binary data and good approximations on arbitrary data. Binary time

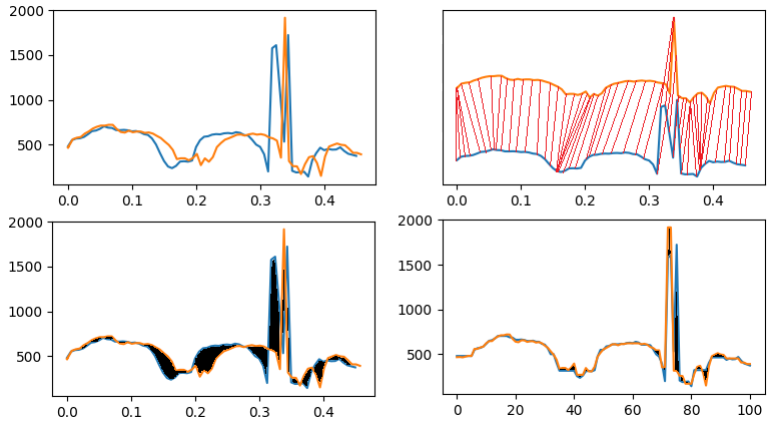


Figure 1.1: The pitch of the first formant of two recordings saying "dynamic" over time. In the top left corner we see the original time series. In the top right corner we see how the two time series would be aligned using dynamic time warping. At the bottom you can see the two time series again with black regions highlighting the distances between them in terms of Euclidean distance. The bottom left graph shows the distances of the original two sequences while the bottom right graph shows the distances of the series after aligning them with dynamic time warping. The distance between the aligned series is about five times lower than the distance between the original series.

series data can not only be found where binary data is measured over time (for example state of on/off switches or open or closed door in a house) but can also be obtained from arbitrary data by applying a boolean function on the original time series such as "Is the temperature higher than 20°C?", or "Did the price go up or down?".

A common task in data mining is clustering inputs and the most prominent algorithm to solve this task is called k-means clustering. This algorithm consists of two basic steps: Starting out with k initial centers, the inputs are grouped to their nearest center, and then new centers are computed from the average of each group. This process is then repeated until the groups, also called clusters, do not change anymore. When we use this algorithm in our time series setting, we run into a problem. How do we calculate the cluster centers? Using the Euclidean mean is not an option because it can produce a mean to two time series S and T which has a larger DTW-distance to both than the actual DTW-distance between them.

Figure 1.2 shows two sine curves with one of them being offset by π. The Euclidean mean of those two curves would actually be $\vec{0}$ which does not represent the sinusoidal shape at all and has a $DTW - distance$ of about 0.25 to both of them. A much better representation of a mean of those two sinusiodal curves would be for example a sine with three peaks and two valleys. This mean would have only distance 0.04 to both input series.
Instead of using an Euclidean mean, we need to calculate new time series that represents the average shape of our input series. The problem of finding the real mean of input series in the dynamic time warping space, called DTW-Mean, is defined by finding a

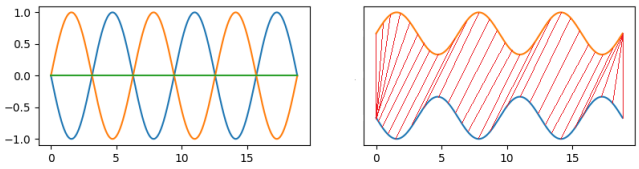


Figure 1.2: Two sine curves over an interval of 6π. The second curve is shifted by π.

new time series μ, that minimizes the summed DTW-distance to all the input series. But this problem has been shown to be NP-hard [BFN18]. Approximating this problem with simple heuristics seems to not work well in a k-means clustering approach [NR07] but some research still aims to refine those techniques [YW17].

In this thesis we will study the binary special case of the DTW-Mean problem. This special case has been shown to be polynomial time solvable [Bri+19] and we will explore how this problem can be solved even faster. As with the DTW-distance, maybe the structure of the binary special case hints to new heuristic approaches for the general DTW-Mean problem.

1.1 Related Work

The general DTW-Mean problem, where input and output strings can have arbitrary real values, has been shown to be NP-hard and W[1]-hard with respect to the number of input strings by Bulteau, Froese, and Niedermeier [BFN18]. In a related paper Brill et al. [Bri+19] also briefly looked at Binary-DTW-Mean, where input and output strings are restricted to be binary, and showed that it is polynomial time solvable in $O(kn^3)$ time, where k is the number of input strings and n is the maximum length. To get this polynomial time they bounded the number of strings that have to be considered as a mean by $O(n)$ and then computed the DTW-distance between each input string and each of those mean candidates with the standard $O(n^2)$ algorithm. We will make use of some of their observations in our proofs as well.

Some recent papers developed heuristic approaches that approximate the dynamic time warping distance to save time [HBG19; Mue+16; Sha+18]. These algorithms (AWarp[Mue+16], BDTW[Sha+18], BSDTW[HBG19]) mainly make use of grouping blocks of the same or similar symbols together. On binary data this approach (when executed correctly) yields exact fast algorithms and we will also use it to develop a fast practical Binary-DTW-Mean algorithm. Mueen et al. [Mue+16] used the CASAS human activity datasets [Coo+19] from the Washington State University which we will also use to test our results.

In 2015, Chan and Lewenstein [CL15] showed that the Bounded Monotone $(\min, +)$ Convolution problem can be solved in $O(n^{1.864})$ time using new techniques from the area of additive combinatorics such as the *Balog-Szemerêdi-Gowers Theorem* and the *FFT Lemma* and also algorithmic techiniques such as randomization. Abboud, Backurs, and Williams [ABW15] showed that Binary-DTW is solvable deterministically in $O(n^{1.87})$ time, breaking the $O(n^2)$ time barrier with the help of this new faster Bounded Mono-

tone $(\min, +)$ Convolution. In their proof for the new upper bound they reduce Binary-DTW to a different problem by also grouping repeating ones and zeros together.

1.2 Our Contributions

In this thesis we will prove that (weighted) Binary-DTW-Mean is solvable in at most $O(kn^{1.87})$ time and develop an algorithm which—similarly to Awarp, BDTW etc.—solves this problem fast on sparse data and also yields an algorithm that solves Binary-DTW quickly, improving the BSDTW and AWarp algorithms in most cases in terms of running time.

2 Preliminaries

In this section we will shortly reintroduce the Dynamic Time Warping distance in a more formal way and also formally define other problems that came up in the introduction or will be used in proofs later on. We will also define some notions that help dealing with the binary structure more efficiently and explore some intuitions and basic approaches before we get to the main proofs. Dynamic time warping measures the distance between time series but because we will focus on the binary domain we assume the inputs to be strings over the alphabet $\{0, 1\}$.

2.1 Defining and Computing the DTW-distance

Consider a string s of length n and a string t of length m. A warping path P_t^s that aligns s and t is a list of (i, j) pairs with $1 \leq i \leq n$ and $1 \leq j \leq m$, implying that the i-th symbol in s is aligned with the j-th symbol in t. Any dynamic time warping path P_t^s has to satisfy three conditions:

1. Boundary condition: $(1, 1) \in P_t^s$ and $(n, m) \in P_t^s$
2. Continuity condition: For two following pairs $(i, j), (k, l)$ in P_t^s: $|i - k| \leq 1$ and $|j - l| \leq 1$.
3. Monotonicity condition: For two following pairs $(i, j), (k, l)$ in P_t^s: $k - i \geq 0$ and $l - j \geq 0$.

In Figure 2.1 you can see a visualization of an optimal warping path between two example strings $s =$"000110111011" and $t =$"01111000111". The three conditions also apply visually: The boundary condition states that the path has to go from the top left corner to the bottom right corner, the continuity condition states that the lines can only go from one cell to a neigbouring cell, including corners and the monotonicity condition states that the path can only move right, down or diagonally down-right. The cost $C_{P_t^s}$ of a warping path P_t^s of length l is then the Euclidean distance of the aligned strings, that is, $C_{P_t^s} := \sqrt{\sum_{r=1}^{l}(s_{i_r} - t_{j_r})^2}$ and the DTW-distance is the minimum cost of any of the warping paths, $\mathrm{DTW}(s, t) := \min_{P_t^s}\{C_{P_t^s}\}$. Visually speaking, we want to find a warping path that minimizes the black area it travels through.
Note that since we are looking for optimal warping paths that minimize the cost we can assume that a warping path never contains the same tuple twice in a row. From this observation and the Continuity and Monotonicity conditions we can deduce the following:

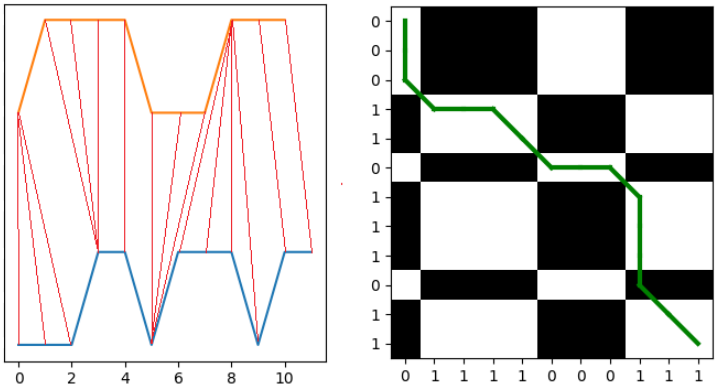


Figure 2.1: Example of an optimal alignment/warping path of two binary strings strings "000110111011" (in blue) and "01111000111" (in orange). The DTW-distance of those two equals one. White cells imply that the aligned symbols match (both ones or both zeros) and black cells imply that the aligned symbols do not match, which increases the DTW-distance.

If $P^s_{t\,r} = (i, j)$, then $P^s_{t\,r+1} \in \{(i+1, j), (i, j+1)(i+1, j+1)\}$.
From this rule, we can easily see how the dynamic program works (see Algorithm 1). Note that if the inputs are binary, then the differences of any two elements can also be only 0 or 1 so we do not need to square the costs. Applying the square root is also optional as it does not change the actual optimal warping path but only the output of the algorithm.

2.2 Problem Definitions

Binary-DTW-Mean
Input: Binary strings $s_1, s_2, \ldots, s_k$ of length at most n.
Output: A binary string μ such that $\sum_{i=1}^{k} \mathrm{DTW}(\mu, s_i)$ is minimzed.

Binary-DTW
Input: Two binary strings s, t of lengths at most n.
Output: The dynamic time warping Distance $\mathrm{DTW}(s, t)$.

Min 1-separated k-Sum
Input: A list of m positive integers $x_1, x_2, \ldots, x_m$ with $m \leq n := \sum_{i=1}^{k} x_i$.
Output: The minimal sum of k distinct elements $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$, where $i_j \neq i_r + 1$ for all j, r.

Min 1-separated Sums
Input: A list of m positive integers $x_1, x_2, \ldots, x_m$ with $m \leq n := \sum_{i=1}^{k} x_i$.
Output: A list A of solutions of Min 1-separated k-Sum for all $k \in \{1, 2, \ldots, \lceil \frac{m}{2} \rceil\}$.

Bounded Monotone $(min, +)$ Convolution
Input: Two lists of m integers $x_1, x_2, ..., x_m$ and $y_1, y_2, ..., y_m$ which are all positive and grow monotonically and $\sum_{i=1}^{m} x_i \leq m$, $\sum_{i=1}^{m} y_i \leq m$.
Output: A list of $2m$ integers s_1, s_2, s_{2m} with $s_i = \min_{j+r=i}(x_j + y_r)$.

Algorithm 1: Standard dynamic program to compute the DTW-distance

Input : Two time series, s of length n and t of length m
Output: The DTW-distance DTW(s,t).
$M \leftarrow n \times m$ Matrix, initialized with infinity
$M[1,1] \leftarrow (s[1]-t[1])^2$
for $i \leftarrow 2$ **to** n **do** // calculating the path costs at the boundary
  $M[i,1] \leftarrow M[i-1,1] + (s[i]-t[1])^2$
for $j \leftarrow 2$ **to** m **do**
  $M[1,j] \leftarrow M[1,j-1] + (s[1]-t[j])^2$
for $i \leftarrow 2$ **to** n **do** // calculating the path costs in the middle
  for $j \leftarrow 2$ **to** m **do**
    $M[i,j] \leftarrow \min(M[i-1,j-1], M[i-1,j], M[i,j]) + (s[i]-t[j])^2$
return $\sqrt{M[n,m]}$

2.3 Other Definitions

Blocks: r repeating zeros (ones respectively) in a binary string will be called a 0-block (1-block) of size r.

Alignemnt: Two blocks are aligned in a Dynamic Time Warping path if all elements of each block are aligned with at least one element of the other.

Match/mismatch: Two aligned blocks are matched if they are both 0-blocks or 1-blocks and mismatched if one of them is a 1-block and the other a 0-block.

Condensation: The condensation of a binary string is obtained by reducing all blocks to size 1.

Cells: A condensed string of length 2 is called a cell. The two types of cells are 01-cell and 10-cell.

2.4 Notation

We use $\tilde{x}$ to denote the condensation of x.
We use $x[i:j]$ to denote the substring of x that starts on index i of x and ends on index j of x. (both i and j are inclusive)
We use $x^\frown y$ to denote a concatenation of strings x and y.
When considering one string s, we use s_i to denote the i-th symbol of s, $s^{(i)}$ to denote the

i-th block of s and s^i to denote the concatenation of i copies of s. We will also use s_{end} to denote the last symbol of s and $s^{(\text{end})}$ to denote the last block of s.
We always assume that strings are binary unless explicitly stated otherwise. Thus $\text{DTW}(s, t)$ always refers to the Binary-DTW problem rather than the general DTW problem.

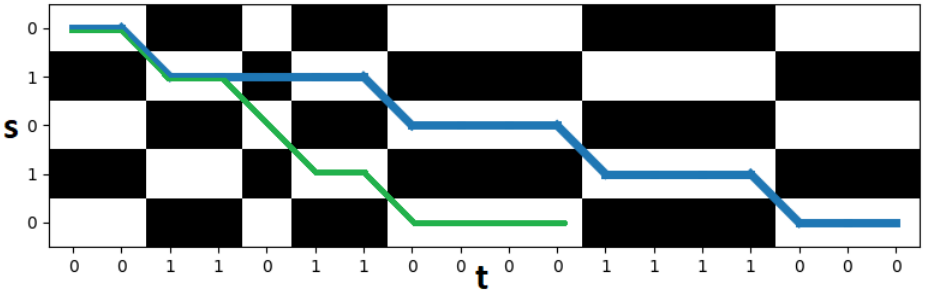
2.5 Important Lemmata and Intuitions

Lemma 1. *There always exists a condensed solution to* Binary-DTW-Mean

Shown by Brill et al. [Bri+19, Proposition 1].
Condensed strings are the best choice for a mean in the binary time warping space because they can be stretched out at any point to be aligned with strings with more blocks but also only have low distance to strings with less blocks because all the redundant blocks that cannot be aligned have only size one.

Lemma 2. *If we consider a condensed string s and a string t with $|\tilde{t}| \geq |s|$ and assume $s_1 = t_1$, $s_{\text{end}} = t_{\text{end}}$, then every symbol in t is aligned exactly once and every block in t is aligned with exactly one symbol in s in an optimal warping path.*

Proof. By Picture:



The only path that has zero cost, is the one aligning $s^{(i)}$ with $t^{(i)}$ for all i (the green diagonal path in the picture). Since t has at least as many blocks as s, it is easy to see that any optimal warping path has to be on the right side of this green diagonal path. If the path aligns a symbol of t with two symbols of s, then that would mean going down and having to go through one additional black block in t to end up on the green path again.
If two following symbols of the same block in t are aligned with different symbols in s then either one of the symbols is aligned with two symbols in s which we just proved cannot happen or each of them is aligned with a different symbol in s. One of those alignments has cost zero and the other alignment has cost 1 so we could align one of the symbols with both symbols of s at the same cost which leads to a contradiction to the first part we proved. □

With this picture and the lemma we can get an intuition for the reduction from Binary-DTW to Min 1-separated k-Sum which is essential in the upcoming sections. We saw that the warping path only goes right and diagonally down right. And we showed that if

a symbol of a block i in t is aligned with a symbol in s then the whole block i is aligned with the symbol in s. Now the DTW-distance is just the amount of black squares we have to go right from the diagonal to reach the diagonal that aligns the last blocks of the two strings. If we go right through such a black block, then the next block will be white again, so only every second block can contribute to the DTW-distance.

In Lemma 2 we required that s and t start and end with the same symbols. The following lemma will handle all other cases.

Lemma 3. *If* $s_1 \neq t_1$ *then* $\mathrm{DTW}(s,t) = min(x + \mathrm{DTW}(s, t[x+1:|t|]), y + \mathrm{DTW}(s[y+1:|s|], t))$ *where* x *is the length of* $t^{(1)}$*,* y *is the length of* $s^{(1)}$*.*

This result has been proven by Abboud, Backurs, and Williams [ABW15] in the proof of their Claim 6 and we will just scetch their proof idea.: If the first symbol in $t^{(2)}$ is aligned with a symbol in $s^{(1)}$ then all symbols in $t^{(1)}$ also have to be aligned with symbols in $s^{(1)}$ at a cost of at least $|t^{(1)}|$ which we called x. We could now delete $t^{(1)}$ and get a cost of $\mathrm{DTW}(s, t[x+1:|t|])$ for the rest. In total we get a cost of $x + \mathrm{DTW}(s, t[x+1:|t|])$ for this warping path. If the first symbol in $s^{(2)}$ is aligned with a symbol in $t^{(1)}$ we can do an analogous argument to get a cost of $y + \mathrm{DTW}(s[y+1:|s|], t)$. Because of the continuity condition, in all other cases $s^{(1)}$ is only aligned with $t^{(1)}$. This alignment already costs $\max(x,y)$ and, because we could have aligned $s^{(1)}$ with $t^{(2)}$ or $t^{(1)}$ with $s^{(2)}$ for free, the total cost of the alignment is $\max(x,y) + \min(\mathrm{DTW}(s, t[x+1:|t|]), \mathrm{DTW}(s[y+1:|s|], t))$. Since this alignment is worse than the two other cases, we can ignore it.

Note that because of symmetry this lemma can also be applied when s and t have different ending symbols or even when both starting and ending symbols differ. In this case we would then get: $\mathrm{DTW}(s,t) = \min \begin{cases} x_0 + x_1 + \mathrm{DTW}(s & , \ t[x_0+1:|t|-x_1-1]) \\ x_0 + y_1 + \mathrm{DTW}(s[1:|s|-y_1-1] & , \ t[x_0+1:|t|]) \\ y_0 + x_1 + \mathrm{DTW}(s[y_0+1:|s|] & , \ t[1:|t|-x_1-1]) \\ y_0 + y_1 + \mathrm{DTW}(s[y_0+1:|s|-y_1-1] & , \ t) \end{cases}$
with $x_0 = |t^{(1)}|, y_0 = |s^{(1)}|, x_1 = |t^{(\mathrm{end})}|, y_1 = |s^{(\mathrm{end})}|$.

2.6 Naively computing the Binary-DTW-Mean

In the following paragraph we will briefly look at the $O(kn^3)$-time Algorithm that Brill et al. [Bri+19] developed to show that Binary-DTW-Mean is polynomial time solvable and see how it can be improved with a simple trick that is very important for our later proofs as well.

Brill et al. [Bri+19] showed that there is always a condensed solution to Binary-DTW-Mean and also showed that this condensed solution can only have a length of at most $n+1$, with n being the length of the longest input string. So in total only $2(n+1)$ strings remain as candidates for being a mean and we can just compute the distances between each of those $2(n+1)$ strings of length at most $n+1$ and the input strings which have length at most n which yields a running time of $O(kn^3)$ time to compute all those distances. Choosing the mean then only requires summing the k distances from each input string to

the mean candidates to find the best among them.
If we take a closer look at those condensed strings we can observe that half of the strings are prefixes of the longest condensed string that starts with 0 and the other half is are prefixes of the longest condensed string that starts with 1. In the calculation of the DTW-distance (Algorithm 1) we have already computed the distances of all prefixes in the last column of the matrix. So we can compute Binary-DTW-Mean faster by computing the DTW-distance from the k input strings to the two longest condensed strings but using the whole last column of the DTW-Matrix instead of just using the last entry. We will call this approach the naive algorithm from now on.

3 Computing Binary-DTW-Mean in Subquadratic Time in n

We have seen that Binary-DTW-Mean can be solved in $O(kn^2)$ time with a relatively simple algorithm. In this chapter we will prove a better upper bound on the asymptotic running time of this problem. The proof will rely a lot on the reduction from Binary-DTW to Min 1-seperated k-Sum which we gave some intuition for in the previous chapter and the structure of condensed binary strings being prefixes of longer condensed strings.

Theorem 1. Binary-DTW-Mean$(s_1, ..., s_k)$ *is computable in* $O(kn^{1.87})$ *time or* $O(kn + k \cdot T(n))$ *time, where* $T(n)$ *refers to the running time of the* Min 1-separated Sums *problem with inputs that sum up to* n*.*

Proof. We will first reduce the problem space of strings that we have to consider as possible means by a vast amount and end up with only an amount that is linear in n. Then we look at each input string s_i individually and show that we can compute the DTW-distances from all of the possible means to this specific string in $O(n^{1.87})$ time. We split this computation in two parts, first only computing distances to means that are longer than the condensation of the considered string and then computing distances to means that are shorter than the condensation. If we have done this for all k input strings we get $O(kn)$ many distances and can find the mean in that additional time.

As Brill et al. [Bri+19, Proposition 1] derived in their paper, we can assume that a mean is a condensed string of alternating zeros and ones. They have also shown that the mean can have at most length $n + 1$ [Bri+19, Lemma 3]. If it is feasible to compute the DTW-distances of all possible condensed means up to length $n+1$ to one specific string in $O(n^{1.87})$ time, then we can easily compute the mean of k strings in $O(kn^{1.87})$ time.
For the rest of the proof, let us consider one specific given string s of length $m \leq n$. Let $m' = |\tilde{s}|$ denote the length of the condensation of s. We will first compute the distance of s to possible means which are longer than m'. Let μ be such a candidate for a mean with $|\mu| = n' \geq m'$. By [Bri+19, Lemma 2] it holds that for a condensed string x and a string y if $|x| \geq |\tilde{y}|$ then $\mathrm{DTW}(x, y) = \mathrm{DTW}(x, \tilde{y})$. With the choice of our mean candidate it follows that $\mathrm{DTW}(\mu, s) = \mathrm{DTW}(\mu, \tilde{s})$ and by [Bri+19, Lemma 1] the DTW-distance of

any two condensed strings x, y can be simply computed by:

$$\mathrm{DTW}(x,y) = \begin{cases} \lceil |x| - |y| \rceil / 2, & x_1 = y_1 \\ 2, & x_1 \neq y_1 \wedge |x| = |y| \\ 1 + \lfloor (|x| - |y|)/2 \rfloor, & x_1 \neq y_1 \wedge |x| > |y| \end{cases} \tag{3.1}$$

Since this formula is applicable for μ and $\tilde{s}$, it yields an algorithm that lets us compute the distances between s and all mean candidates longer than m' in linear time (constant time for one mean candidate and known m'). It only remains to show that we can compute the distances to means shorter than m' in $O(n^{1.87})$ time.

To compute the cost of means that are shorter than the condensation of s we will first calculate the cost of mean candidates starting and ending with the same symbols as s and then use Lemma 3 to compute the cost of candidates with arbitrary starting and ending symbols.

Abboud, Backurs, and Williams [ABW15, Theorem 8] showed that Binary-DTW can be reduced to Min 1-separated k-Sum and then further presented an algorithm that solves Min 1-separated k-Sum in $O(n^{1.87})$ time with the help of a new faster method to solve the (min, +) - convolution problem by Chan and Lewenstein [CL15]. The main observation used in the reduction to Min 1-separated k-Sum is that given two binary strings s and t, where $|\tilde{t}| \leq |\tilde{s}|$, each block of t is aligned with some blocks of s such that the first and last of those blocks in s contain the same symbol as the block of t (that is a 1-block in t could be aligned with a 1-block, the following 0-block, and 1-block in s but not with just the first two of those blocks because then the next 0-block in t would have to be aligned with that last 1-block but this would correspond to a black diagonal movement as illustrated in Figure 3.2). The cost in that part of the alignment is then the size of the aligned blocks in s containing the other symbol. This has the consequence that two neighbouring blocks in s cannot both contribute to the cost and that the first and the last block in s cannot contribute to the cost either. (Recall that we are only considering the special case where s and t start and end with the same symbol at the moment.)

Figure 3.1 shows the warping paths of two different means μ, μ' and an example string $s = 001101100001111000$ where $m' = 7$. White fields have cost 0 and black fields have cost 1. One can easily see that if two neighbouring blocks were to contribute to the cost (black in the figure) then the warping path needs to go diagonally from the last field of the first black block to the first field of the second block. For any field that is reachable from that position there would have been a cheaper warping path avoiding this diagonal movement (see Figure 3.2).

Let B be a list of the sizes of blocks in s. We can use the algorithm by Abboud, Backurs, and Williams [ABW15, Theorem 10] designed to solve Min 1-separated k-Sum to compute four lists $A^{00}, A^{0*}, A^{*0}, A^{**}$ which are the solutions for Min 1-separated Sums on the sublists $B[3 : m' - 2], B[3 : m' - 1], B[2 : m' - 2], B[2 : m' - 1]$. To obtain a solution for Binary DTW of s and some other string with shorter condensation we

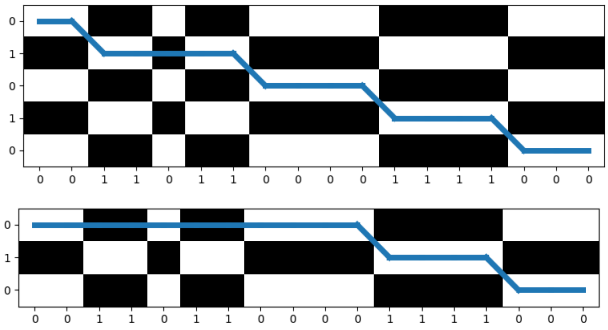


Figure 3.1: Optimal dynamic time warping paths of an example string 001101100001111000 on the x-axis and two different mean candidates 01010 and 010 with shorter condensations on the y-axis. White squares are possible alignments of equal symbols (cost 0). Black squares are alignments of different symbols (cost 1). The warping path has to connect the upper left and lower right corners while minimizing the amount of black squares it passes.

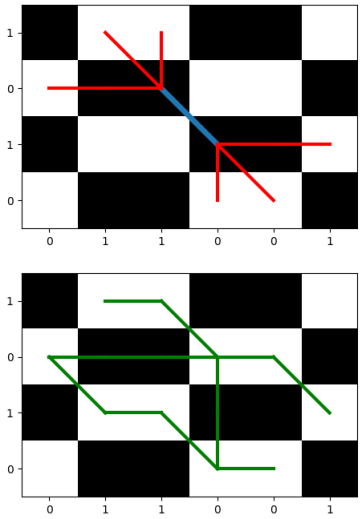


Figure 3.2: Segments of warping paths (not necessarily optimal) of two strings or segments of strings. The upper graphic shows a warping path including a diagonal movement between black squares and all possible directions the path could have taken before and after that movement. The lower graphic shows paths cheaper or of equal cost to the above ones avoiding the black diagonal movement.

would now only use one entry from the list A^{**} but for our purposes we will use almost all the entries from all four lists. The r-th entry of the list A^{**} is the minimal cost of r non neighbouring blocks (already excluding the first and the last block) in s and thus relates to the cost of the mean of size $m' - 2r$ with same starting and ending symbols as s. The other lists will be used to compute the DTW-distances occurring in Lemma 3. Algorithm 2 computes the costs of all means that are shorter than the condensation of s.

The List C_1 contains distances to all mean candidates starting and ending with the same symbols as s and the other three lists contain the other different variants of starting and ending symbols. For example, c_2 is the cost of a mean candidate of length $m' - 2r$ with an added one (or zero respectively) to the start. The two resulting possiblities are then either mismatching the new one (zero) with $s^{(1)}$ at an addional cost of 1 compared to c_1 or mismatching $s^{(1)}$ and $s^{(2)}$ with the new one (zero) and thus having to mismatch one block less in the rest of s. The other cases c_3, c_4 are analoguous uses of Lemma 3. Figure 3.3 shows an example for $s = 001101100001111000$ and $r = 2$.

Algorithm 2: Computes $\mathrm{DTW}(\mu, s)$ for all μ with $|\mu| < m'$

Input : Binary string s.
Output: Four lists C_1, C_2, C_3, C_4 of distances from s to condensed strings shorter than m' each list corresponds to a specific combination of starting and ending symbol.

$B \leftarrow$ list of sizes of blocks in s
$x \leftarrow B_1, y \leftarrow B_{m'}$
compute $A^{00}, A^{0*}, A^{*0}, A^{**}$
// with input $B[2 : m' - 1]$ for the algorithm [ABW15, Theorem 10] in $O(n^{1.87})$*time*
// in $4O(T(n))$ time, where $T(n)$ is the running time of Min 1-separated Sums
initialize four lists $C_1, ..., C_4$
for $r \leftarrow 1$ **to** $\lfloor \frac{m'-1}{2} \rfloor$ // r is the number of mismatched blocks in s
do
  $c_1 \leftarrow A^{**}_r$
  $c_2 \leftarrow min(1 + A^{**}_r, x + A^{0*}_{r-1})$
  $c_3 \leftarrow min(1 + A^{**}_r, y + A^{*0}_{r-1}))$
  $c_4 \leftarrow min(2 + A^{**}_r, 1 + x + A^{0*}_{r-1}, 1 + y + A^{*0}_{r-1}, x + y + A^{00}_{r-2})$
  // using Lemma 3 and its symmetric statement, see Figure 3.3 for visual explanation
  add $c_1, ..., c_4$ to the corresponding lists

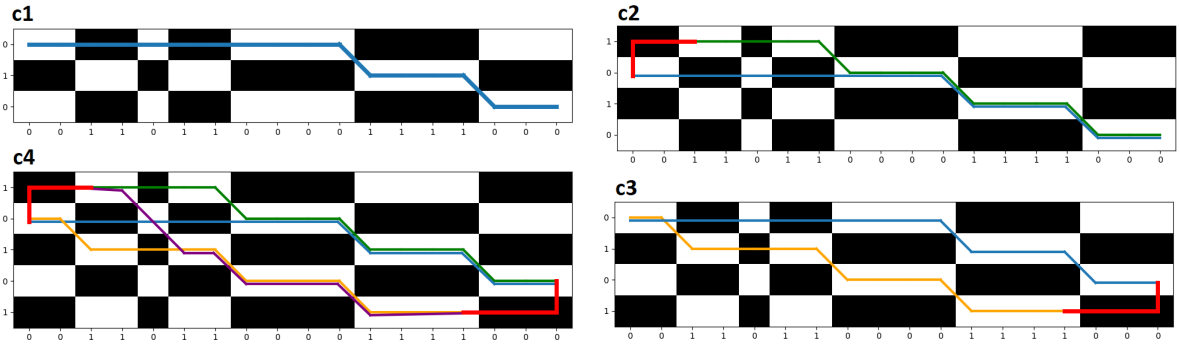


Figure 3.3: Example of the case analysis of Algorithm 2 with $s = 001101100001111000$ and $r = 2$. The red paths differentiate the possibilities of warping paths where μ and s have different starting or ending symbols. The colored paths are the optimal solutions of Min 1-separated k-Sum on sublists given by the red paths. Each color corresponds to the entry of one specific list: A^{**}_2, A^{0*}_1, A^{*0}_1, A^{00}_0. Note that the purple subpath corresponding to A^{00}_0 has cost 0 and would not actually be calculated by the algorithm.

Note that Algorithm 2 does not compute the cost of a single one or zero as a mean if s starts and ends with different symbols. But that is obviously no problem since the cost can be computed in linear time.

With Algorithm 2 and Equation (3.1) given by Brill et al. [Bri+19] we can compute the distance $\mathrm{DTW}(\mu, s)$ between all possible means μ up to length $n+1$ and the k given strings in $O(kn^{1.87})$ time or $O(k \cdot T(n))$ time. We can then find the mean with the lowest cost in an additional $O(kn)$ time. □

3.1 Mean in linear time for $k = 2$

We have seen that Binary-DTW-Mean can be computed in strongly subquadratic time (in n). Now we will look at some special cases to see if we can get an even faster asymptotic time. First of all we look at the mean of two binary strings in the time warping space. Approximation alogirithms for the general DTW-Mean problem often sequentially compute weighted means of two strings so maybe having a linear algorithm for Binary-DTW-Mean of two strings could yield a good linear approximation algorithm for general k as well.

Theorem 2. Binary-DTW-Mean *of two input strings is comptable in $O(n)$ time.*

Proof. Let s and t be two binary strings of length at most n. As before we will consider the strings having the same starting and ending symbol and then derive the means for other cases from there.
Let $x_1 := s_1 = t_1$ be the first symbol and $x_{\text{end}} := s_{\text{end}} = t_{\text{end}}$ be the last symbol of the two strings.

Claim 1. *Any mean of s and t has to start with x_1 (and end with x_{end}).*

Proof. Assume μ is a mean and does not start with x_1. Note that $\mu^{(1)}$ has to be aligned with x_1 in s and t. If $\mu^{(1)}$ is not aligned with any more blocks other than $s^{(1)}$ and $t^{(1)}$ then we can delete it to obtain μ' starting with x_1 with smaller cost which contradicts μ being a mean. If $\mu^{(1)}$ is aligned with more than the first symbol in either s or t then we can let $\mu' := x_1 \frown \mu$. Aligning just μ'_1 (x_1) with the first symbol of s or t and $\mu'^{(2)}$ with all the symbols as $\mu^{(1)}$ except the first symbol will yield a smaller cost which contradicts μ being a mean. □

Let without loss of generality $|\tilde{s}| \leq |\tilde{t}|$.

Claim 2. *$\mu := \tilde{t}$ is a mean of s and t.*

Proof. We know that there is a condensed mean starting with x_1 and ending with x_{end}. The only variability left is the length of the mean. Assume without loss of generality that $x_{\text{end}} =$ "0". Let $c_s = \mathrm{DTW}(\mu, s)$ and $c_t = \mathrm{DTW}(\mu, t)$ and let $c = c_s + c_t$ be the cost of μ. If we add a 10-cell to the end of μ, then the costs c_s and c_t will increase by 1 [Bri+19, Lemma 1+2] so the cost c will increase too. If we delete a cell from μ, then the cost c_s will decrement (one less mismatched block in μ), but c_t will increase by at least 1 (one more mismatched block in t), so c will not decrease. □

Now that we know a mean for any two strings s and t with same starting symbol x_1 and ending symbol x_{end}, let us consider the slightly modified strings s' and t' which do not necessarily have same starting and ending symbols and let $c_{s'}, c_{t'}$ be the modified costs. Let $y_1 = 1 - x_1$ and $y_{\text{end}} = 1 - x_{\text{end}}$ be the opposites of the starting/ending symbol. If we obtain s' and t' by adding a y_1-block of arbitrary size to the start of either of them (not to both since then they start with the same symbol again) or a y_{end} block to the end then we just need to consider the two following possibilities:

1. $|\tilde{s}'| < |\tilde{t}'|$: We can still choose $\mu = \tilde{t}'$. As before let $c = c_{s'} + c_{t'}$ be the cost of that mean. We know $c_{t'} = 0$ and $c_{s'} = c_s + 1$ if we added exactly one y_1-block to the start s or t (or y_{end}-block to the end). And $c_{s'} = 2$ if we added a y_1-block to the start of either s or t and a y_{end}-block to the end of either of the two. Any mean μ' with smaller cost would have to be shorter to lower $c_{s'}$ which would result in equally many blocks being mismatched in t' so the total cost would not decrease.

2. $|\tilde{s}'| = |\tilde{t}'|$: In this case s' and t' both start and end on opposite symbols, for example, s' starts and ends with 0 and t' starts and ends with 1. We can choose $\mu = s'_1 \frown \tilde{t}'$ (or $t'_1 \frown \tilde{s}'$) as a mean with distance 1 to both strings. The condensations of s' and t' are both substrings of μ and thus only the first or last symbol in μ will be misaligned in the warping paths. If there was a cheaper mean, then it would have to have distance 0 to either s' or t' and thus start and end with the same symbol as one of the two which would yield a distance of at least two to the other which contradicts it having less cost.

Note that $|\tilde{s}'| > |\tilde{t}'|$ could happen after adding a y-block to s but those cases are symmetric to $|\tilde{s}'| < |\tilde{t}'|$. □

3.2 Binary-DTW-Mean in $f(k)O(n)$ time?

In the previous section we have seen that Binary-DTW-Mean is solvable in linear time for $k = 2$. Now the question naturally arises as to whether it is not possible to solve this problem in time depending only linearly on n for all k.
We will prove that for the relaxation of Binary-DTW-Mean where all input strings start with x_0 and end with x_1 we can solve this problem in linear time for $k = 3$ as well and show why this approach does not work for $k \geq 4$.

Relaxed Binary-DTW-Mean
Input: Binary strings $s_1, s_2, \ldots, s_k$ of length at most n and $x_0, x_1 \in \{0, 1\}$ such that all input strings start with x_0 and end with x_1
Output: A binary string μ such that $\sum_{i=1}^{k} \text{DTW}(\mu, s_i)$ is minimzed.

Theorem 3. Relaxed Binary-DTW-Mean *is computable in $O(n)$ for three input strings.*

Proof. Let s_1, s_2, s_3 be ordered by sizes of their condensations. We can easily see that $|\mu| \geq |\tilde{s_2}|$. Recall that increasing the length of the mean by two (appending a cell) will increase the distance to s_1 and s_2 by one and the distance to s_3 decreases with one block less contributing to the cost, so it is only worth to increase the size of the mean by a cell if we can save at least a block of size at least two in s_3. Let r be the number of 1-separated blocks of size one in s_3. We can compute r in linear time by greedily counting every one we see and then ignoring the next element in the list of block sizes of s_3. If $r = 0$ then there is no block of size one so making the mean longer by a cell will always save a block of size at least two in s_3 and thus $\mu = \tilde{s_3}$. If $r \geq \frac{|\tilde{s_3}|-|\tilde{s_2}|}{2}$ then there is no block that can be saved by making μ longer and so $\mu = \tilde{s_2}$. Otherwise we can choose $|\mu| = |\tilde{s_3}| - 2r$ and by a more general version of Claim 1 the mean has to start with x_0 and end with x_1 which makes it unambiguous. □

When we chose the mean length somewhere between $|\tilde{s_2}|$ and $|\tilde{s_3}|$ depending on r there is one thing we assumed implicitly, that is, as long as $\frac{|\tilde{s_3}|-|\mu|}{2} < r$ then appending one cell to the mean actually decreases the distance to s_3 by at least two. For that we need the following lemma to hold.

Lemma 4. *Let B be the list of block sizes with $|B| = b$, let A be the solution of* Min 1-separated Sums(B) *and let r be the maximum number of 1-separated blocks of size one in B. Then for $i > r$ it holds that $A_i - A_{i-1} \geq 2$.*

Proof. For any $i > r$ let $j_1, j_2, ..., j_i$ be the indices of 1-separated blocks in B such that $\sum_{l=1}^{i} B_{j_l} = A_i$. Since $i > r$ there has to be some block of size at least two beyond those. Let $B[j_x]$ be that block then choose $j'_1, ..., j'_{i-1} := j_1, j_2, ..., j_{x-1}, j_{x+1}, ..., j_i$. The new $i - 1$ indices are 1-separated and $A_{i-1} \leq \sum_{l=1}^{i-1} B_{j'_l} \leq A_i - 2$. □

To finish the proof of the linear solvability for $k = 3$ for the non-relaxed version one would have to make a case distinction regarding which string starts (ends respectively) with a block containing different symbol than the other two and then slightly modify the mean depending on which string it is and what are the lengths of the starting (ending) blocks. When considering four input strings s_1, s_2, s_3, s_4 we can assume that the mean is at least as long as $\tilde{s_3}$. For that consider a mean that is a bit shorter, appending one cell would increase the distance to s_1 and s_2 by one and decrease the distance to s_3 and s_4 by at least one. Now to get a better result by appending cells to the mean we would need to save at least a cost of three in s_4. So one might find some r that is the maximum number of 1-separated blocks of length at most two in s_4 which is possible in linear time again, and then choose $|\mu| = \max(|\tilde{s_4}| - 2r, |\tilde{s_3}|)$. But now we run into the following problem: The found mean could still be too short! This can happen because the difference between r and $r - 1$ 1-separated Blocks of size at most two can be indeed bigger than two. In fact it can be arbitrarily big. Consider the following example: $s_1 = s_2 = s_3 =$ "0" and $s_4 =$ "000011110010010011110000". The list B_4 of block sizes of s_4 would look like this: $B_4 = [4, 4, 2, 1, 2, 1, 2, 4, 4]$ and $r = 3$. The computed mean would have length $9 - 6 = 3$ so $\mu =$ "010". This mean would have cost $3 + 6 = 9$ but by appending one 01-cell to

the start of the mean we can get $\mu' :=$ "01010" with cost $6 + 2 = 8$ and having such a 00100100-structure multiple times within s_4 would make it necessary to increase the mean size multiple times. So for $k \geq 4$ we cannot find a mean with this approach without actually solving the Min 1-separated Sums problem at some point.
Although we discussed why our approach for solving Binary-DTW-Mean for $k = 3$ cannot be used to solve the problem for larger k, we did not prove that there is no way to compute Binary-DTW-Mean in $f(k)O(n)$ and it remains open whether there is such an algorithm or not.

4 Binary-DTW-Mean in Time Quadratic in Condensed Lengths

We have shown that the mean of binary time warping data is computable in $O(kn^{1.87})$ time where k is the number of strings and n is the maximum length but some parts of the underlying algorithm by Chan and Lewenstein [CL15] use highly complex theoretical results and randomization which makes it hard to implement efficiently in practice. In this section we will develop another algorithm which is easy to implement and runs in much better time than a naive solutions like computing the whole DTW-Array for each string which would take about $O(kn^2)$ time because it uses the facts that blocks of the same symbol can be grouped together and that we only have to consider warping paths that pass through a smaller number of black blocks. We will again use the reduction to Min 1-separated k-Sum but this time use a dynamic program which runs in $O(m^2)$ time, where m only depends on condensed sizes. This algorithm is especially efficient on sparse data.

Claim 3. *If the binary input strings $s_1, s_2, ..., s_k$ are ordered by number of blocks (length of condensations), then there exists a condensed mean μ with $|\mu| \geq |\tilde{s}_{\lfloor \frac{k}{2} \rfloor + 1}|$.*

Proof. Suppose there is a condensed mean μ' with shorter length. Now we know there are at least $\lceil \frac{k}{2} \rceil$ strings with a longer condensation than μ'. The distance to those strings is at least the added size of mismatched blocks in $\tilde{s}_{\lfloor \frac{k}{2} \rfloor + 1}, ..., s_k$ and some constant cost for the start and end. If we now create μ'' by adding one cell to μ' then the distance to $s_1, ..., \tilde{s}_{\lfloor \frac{k}{2} \rfloor}$ will increase by exactly 1 while the distance to $\tilde{s}_{\lfloor \frac{k}{2} \rfloor + 1}, ..., s_k$ decreases by at least 1. The new μ'' has at most the cost of μ' which and is also a mean. □

Let $m' = |\tilde{s}_{\lfloor \frac{k}{2} \rfloor + 1}|$ be the new lower bound on the length of the mean. Now we know for any string with longer condensation $s_i, i > \lceil \frac{k}{2} \rceil$ that we can match at least m' blocks correctly with blocks in the mean and only half of the remaining blocks will contribute to the cost. So we only have to solve the 1-separated k-Sum problem for 1 up to $k' \leq \lfloor \frac{|\tilde{s}_i| - m'}{2} + 1 \rfloor$. The $+1$ emerges from possible differences of starting or ending symbols. Algorithm 3 solves this problem.

The lines 17 to 23 of Algorithm 3 are the core of the dynamic programming approach. The entry $M[i, r]$ is the minimal sum of r 1-separated elements in the sublist $s_{enc}[1 : i]$. In the base case of only having one element in the sum ($r = 1$) the minimum is always either the same as before or the new element of the substring. If $r > 1$ we can either take the new element and then use the minimal $r - 1$ element sum of the substring not containing

Algorithm 3: Computes DTW(μ, s) for all μ with $|\mu| < m'$ and specific starting and ending symbols.

Input : Binary string s, μ_1 and μ_{end}
Output : An array M with distances between s and all μ up to length m'

1 $s_{\text{enc}} \leftarrow$ list of block sizes of s
2 $r \leftarrow \text{length}(s_{\text{enc}}) - m'$
3 $b_1 \leftarrow 0, b_{\text{end}} \leftarrow 0$ // indicates whether there is a black block in the top left or bottom right corner
4 **if** $\mu_1 \neq s_1$ **then**
5 $\quad b_1 \leftarrow 1$ // if the top left block is black then we already count the first step through that block that any warping path has to make, see line 24
6 $\quad r \leftarrow r - 1$
7 $\quad s_{\text{enc}}[1] \leftarrow s_{\text{enc}}[1] - 1$ // if the warping path passes through the top left black block horizontically then we do not count the first step again
8 **else**
9 $\quad$ delete $s_{\text{enc}}[1]$
10 **if** $\mu_{\text{end}} \neq s_{\text{end}}$ **then**
11 $\quad b_{\text{end}} \leftarrow 1$
12 $\quad r \leftarrow r - 1$
13 $\quad s_{\text{enc}}[last] \leftarrow s_{\text{enc}}[last] - 1$
14 **else**
15 $\quad$ delete $s_{\text{enc}}[last]$
16 $k' \leftarrow \lfloor \frac{r}{2} \rfloor + b_1 + b_{\text{end}}$ // k' is the number of black blocks the smallest mean starting with μ_1 and ending with μ_{end} has to pass through
17 $M \leftarrow |\tilde{s}_i| \times k'$ Array, initialized with infinity
18 **for** $r \leftarrow 1$ **to** k' **do** // r is the number of 1-separated blocks
19 $\quad$ **for** $i \leftarrow 2(r-1)+1$ **to** $\text{length}(s_{\text{enc}})$ **do** // there cannot be r 1-separated blocks in less than $2(r-1)+1$ blocks
20 $\quad\quad$ **if** $r = 1$ **then** // for $r = 1$ we just look for the smallest block
21 $\quad\quad\quad M[i, r] \leftarrow min(s_{\text{enc}}[i], M[i-1, r])$
22 $\quad\quad$ **else**
23 $\quad\quad\quad M[i, r] \leftarrow min(M[i-1, r], s_{\text{enc}}[i] + M[i-2, r-1])$ // in the general case either take the best i 1-separated blocks from before or the new $s_{\text{enc}}[i]$ and the $i-1$ 1-separated blocks so that $s_{\text{enc}}[i-1]$ is not among them
24 $\quad M[\text{length}(s_{\text{enc}}), r] \leftarrow M[\text{length}(s_{\text{enc}}), r] + b_1 + b_{\text{end}}$

the new element and the one before (to ensure the 1-separation) $s_{enc}[i] + M[i-2, r-1]$, or we do not take the new element in which case the minimal sum is the same as before $M[i-1, r]$.
The rest of the algorithm handles the differences of starting and ending symbols (in constant time). To show this is correct assume that the mean μ and s start with different symbols.

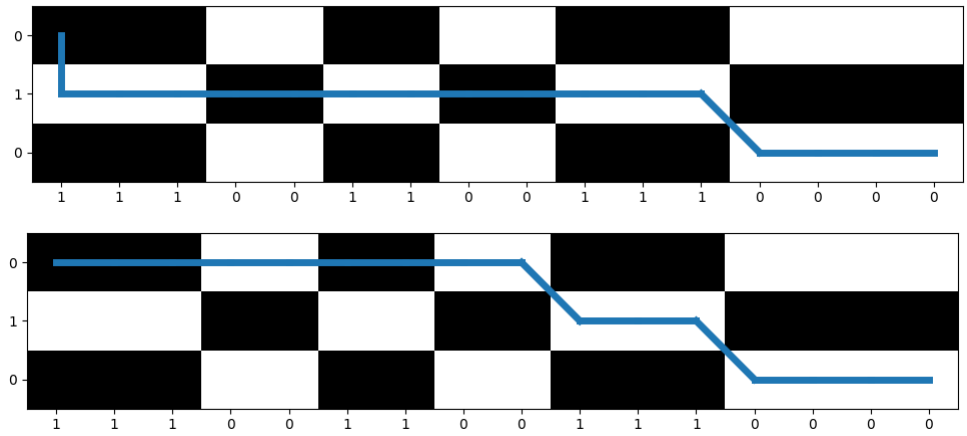


Figure 4.1: Two optimal warping paths of $\mu = 010$ and $s = 1110011001110000$ with $\mathrm{DTW}(\mu, s) = 5$. The path in the upper picture aligns μ_1 with only S_1 at a cost of one but then has to align μ_2 with $s^{(1)}, s^{(2)}, s^{(3)}, s^{(4)}, s^{(5)}$ of which $s^{(2)}$ and $s^{(4)}$ contribute to the cost. The path in the lower image aligns μ_1 with $s^{(1)}, s^{(2)}, s^{(3)}, s^{(4)}$ of which $s^{(1)}$ and $s^{(3)}$ contribute to the cost.

In this case the optimal alignment path has two options: Either align μ_1 and μ_2 with the first symbol of s at a cost of one but then having to mismatch one additional block in s. Or align s^1 and s^2 with μ_1 at cost of the size of s^1 (see Figure 4.1). The algorithm handles this by decreasing the cost of s^1 by one and adding this cost of one to both cases at the end. It also changes the number of mismatched blocks accordingly. If μ and s start with the same symbol, then μ_1 and s^1 are always aligned in an optimal warping path, so s^1 cannot contribute to the cost and is thus erased from the list by the algorithm. Handling the last symbols is symmetric.

If $m = \max\{|\tilde{s}_i| \ : \ 1 \leq i \leq k\}$ and m' is the median condensed length as before, then Algorithm 3 runs in $\Theta(\frac{(m-m')^2}{4})$ to compute distances for one specific case of starting and ending symbol. If we run this algorithm four times—for every possible combination of starting and ending symbols—then we get a running time in $\Theta((m-m')^2)$.
Now we can create a $\frac{k}{2}$ by $4 \cdot (m - m')$ list of distances between all possible means and given binary strings. Notice that we only need to keep track of all the strings with longer condensation than m'. For the shorter strings we can sum up the distances. For each combination of starting and ending symbol then find the mean with the shortest length in $\frac{k}{2} \cdot (m - m')$ time and choose the best mean out of those four to be the final result.

4.1 Solving Binary-DTW with our Algorithm

In the previous section we have developed an alternative algorithm that solves the Min 1-separated Sums problem in a dynamic programming approach. Now we will use this

algorithm again to solve the original problem Binary-DTW where the reduction came from and compare it to other fast algorithms that we introduced before (BDTW[Sha+18], Awarp[Mue+16], BSDTW[HBG19]) in terms of running time.

Let s, t be two binary strings, let $n := \max(|s|, |t|)$ and without loss of generality assume $|\tilde{s}| \geq |\tilde{t}|$. As Abboud, Backurs, and Williams [ABW15] showed in their reduction from Binary-DTW to Min 1-separated k-Sum we need to solve Min 1-separated k-Sum for $k = \frac{|\tilde{s}|-|\tilde{t}|}{2}$ on the list of block sizes of s. If s and t differ in starting and ending symbols we additionally need to make use of Lemma 3 and in the worst case (if s and t start and end with different symbols) we have to run Algorithm 3 four times.
For a given k, Algorithm 3 runs in $\Theta(\sum_{i=1}^{k} |\tilde{s}| - 2i) = \Theta(k(|\tilde{s}| - k - 1))$ time. Let $p_1 := \frac{|\tilde{s}|}{n}$ and $p_2 := \frac{|\tilde{t}|}{n}$ be the sparsities of s and t. Note that this notion of sparsity differs slightly from sparsities as defined by Hwang and B. Gelfand [HBG19]. They defined sparsity to be the number of ones devided by n, thus only making use of big 0-blocks, while we take both 1-blocks and 0-blocks into account. In the worst case we gain a factor of two compared to their definition, but if 1-blocks are on average at least of size two, then our definition of sparsity will yield lower values and thus better running time.
We can express the running time in terms of n and p_1, p_2: $k = \frac{n(p_1-p_2)}{2}$, $|\tilde{s}| = np_1$ so in the worst case we get a running time of $4\frac{n(p_1-p_2)}{2}(np_1 - \frac{n(p_1-p_2)}{2} - 1) \leq 2n^2({p_1}^2 - p_1p_2)$.
If all 1-blocks have size one, then we can express the running time of BSDTW as $p_1p_2\frac{n^2}{2}$, and the running time of AWarp as $p_1p_2n^2$. If the difference of condensed sizes of s and t is sufficiently small, that is $p_1 - p_2 \leq \frac{p_2}{4}$, then we get a better running time than BSDTW and AWarp even for the worst case. If s and t start and end with same symbols, then $p_1 - p_2 \leq p_2$ suffices and if there are larger 1-blocks, then we get an even faster algorithm. So in conclusion this algorithm would be best used when binary data is generated with high enough sample rate that there are some large 1-blocks or when binary data is already known to be pretty similar so the condensed sizes are almost equally big. Comparing our algorithm to BDTW is easier because BDTW also uses repitions of both, zero and one-blocks and achieves a running time of $\Theta(n^2p_1p_2)$. We can achieve a better running time with our algorithm if $p_1 \leq \frac{3}{2}p_2$ in the worst case and even better running times if starting and ending symbols match.

5 Related Problems

In this section we will look at some problems that are similar to Binary-DTW-Mean and see how we can use our algorithms and structural information of binary dynamic time warping to solve those.

5.1 Weighted Binary-DTW-Mean

Input:	Binary strings $s_1, s_2, ..., s_k$ and weights $w_1, w_2, ..., w_k$ with , $\lvert s_i\rvert \leq n$, $w_i \geq 0$ for all i.
Output:	A binary string μ such that $\sum_{i=1}^{k} w_i \operatorname{DTW}(\mu, s_i)$ is minimzed.

Corollary 1. Weighted Binary-DTW-Mean *is computable in* $O(kn^{1.87})$ *time.*

Proof. First, we can still assume that any mean is condensed by the same argument as before. Consider a non-condensed mean. Then, it is easy to see that the condensation of it has at most the same DTW-distance to each string. It is easy to see that the length of the weighted mean is still bounded by the condensed sizes of the input string with shortest condensation and the input string with the largest condensation since making it shorter (or longer respectively) would increase the distance to all strings.
As before we can compute the costs for all possible means to one specific string in $O(n^{1.87})$ time and end up with at most $2(n+1)\cdot k$ values in total which we can then weight according to the given parameters and compute the mean in additional $O(nk)$ time. □

5.2 Binary-DTW-Center

Input:	Binary strings $s_1, s_2, \ldots, s_k$ of lengths at most n.
Output:	A binary string c such that $\max_{1\leq i\leq k} \operatorname{DTW}(c, s_i)$ is minimzed.

Corollary 2. Binary-DTW-Center *is computable in* $O(kn^{1.87})$ *time.*

Proof. As above, the center can also be assumed to be condensed with length bounded by shortest and longest condensation of input strings. We can compute the same $2(n+1)\cdot k$ values in $O(kn^{1.87})$ time which contain the distances from all possible condensed centers to all input strings. Now we can find the center similarly to finding the mean by considering only the maximum distance to one of the k input strings for each possible mean instead of summing the distances of all input strings. So finding the center only takes additional $O(nk)$ time. □

6 Testing and Experiments

In this section we will answer some of the arising questions by testing the algorithm developed in chapter 4 and also compare it to the trivial algorithm that calculates the whole $n \times n+1$ matrix for each string. To do this we will mostly use the human activity datasets provided by the Washington State University CASAS [Coo+19] and also use some randomly generated data. The use of those human activity datasets was inspired by Mueen et al. [Mue+16] who also used those datasets to test their DTW algorithm.
To derive meaningful information from the tests we will use our notion of sparsity as defined in Section 4.1, that is, for a binary string s with length n with m blocks the sparsity is defined as $p_s := \frac{m}{n} = \frac{|\bar{s}|}{|s|}$. This definition of sparsity yields values between zero and one with values close to zero implying long blocks in s and values close to one implying many short blocks in s. We will call strings with sparsity greater than 0.5 dense and strings with sparsity smaller than 0.2 sparse.
The data in the CASAS datasets consists of different sensors which signal changes in the environment (for example door got opened/closed). We use this data in two different ways (further using the door example):

1. To create mostly sparse data we will create lots of evenly spaced time points, for example 10 seconds or less apart, and check on each time if the most recent event was opening or closing the door. See Figure 6.2. Notice that we may lose track of the door being opened and closed within less than 10 seconds this way.

2. To create less sparse data where the data points are maybe 10 minutes apart we require each event of the door being opened or closed to be represented by at least 1 of the data points. See Figure 6.3. Of cause this does not work if we have less data points than events.

6.1 Comparing our Algorithm with the naive Solution

Regarding the speed of Algorithm 3 we want to answer the following questions:

Question 1. *Is Algorithm 3 always at least as fast as the naive approach?*

Question 2. *How much does the running time of Algorithm 3 improve with lower sparsity?*

To answer the first two questions regarding the speedup of Algorithm 3 we compare the naive dynamic $O(kn^2)$ algorithm and the fast dynamic program on sparse and very sparse

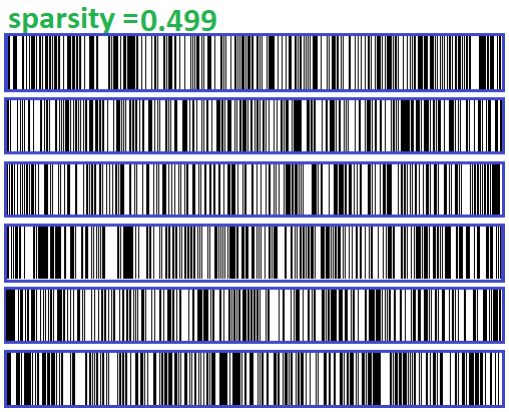


Figure 6.1: Randomly generated data with $n = 20000$, $k = 6$

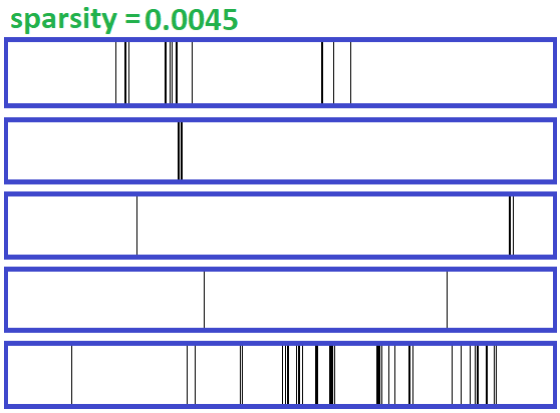


Figure 6.2: Motion sensor data from 7pm to 0am in one hour splits. Data points are 0.5 seconds apart. For DTW-Mean here $n = 7200$, $k = 5$.

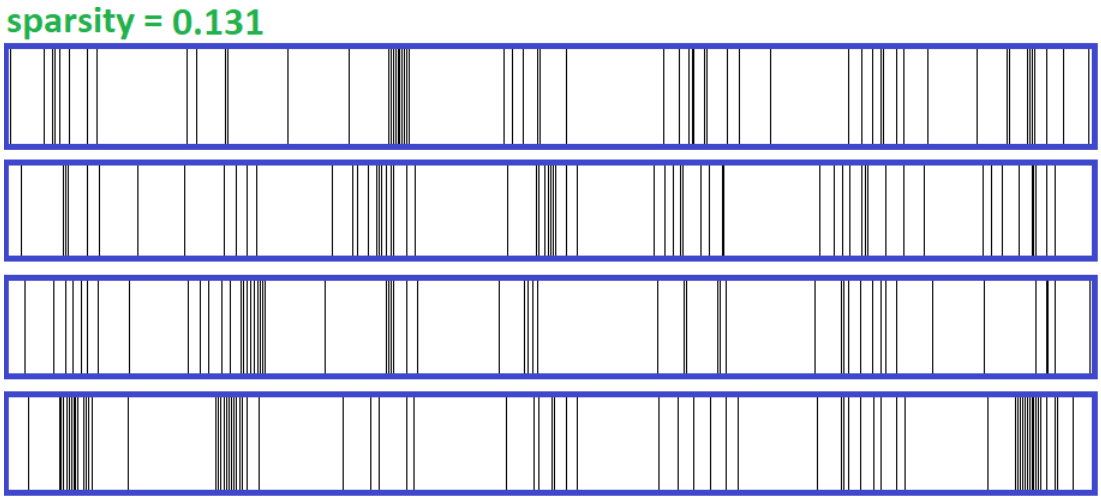


Figure 6.3: State of a door (open or closed) in a house over four weeks. Here the interval between two data points is ten minutes and if the door was opened and closed within those ten minutes then it will count as open for one data point. For DTW-Mean here $n = 1008$, $k = 4$.

real data obtained from the door sensor D002 in the dataset H001 and on sparse and dense randomly generated data, both for various values of k. As you can see in Figure 6.4 and Figure 6.5 our algorithm performs so much better that it seems to have zero running time if we only look at input sizes where the naive algorithm still runs in somewhat feasable time. In fact, in terms of total running time in each of the diagrams with sparsity ≈ 0.1 in Figure 6.4, the new algorithm performs about 300 times better than the naive algorithm and in Figure 6.5 it performs about 800 times better. Notice that for this sparsity of ≈ 0.002, most of the running time is probably just overhead of the function because the

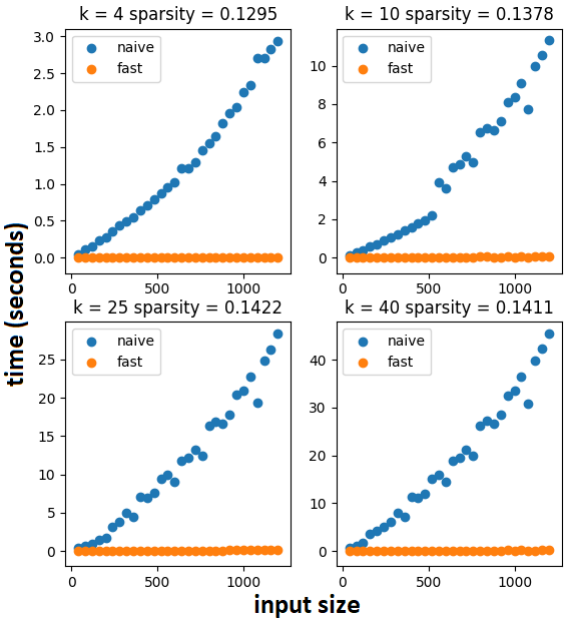


Figure 6.4: Running time of the naive $O(kn^2)$ and the new algorithm on relatively sparse data obtained from the sensor D002 in 10 minute intervals.

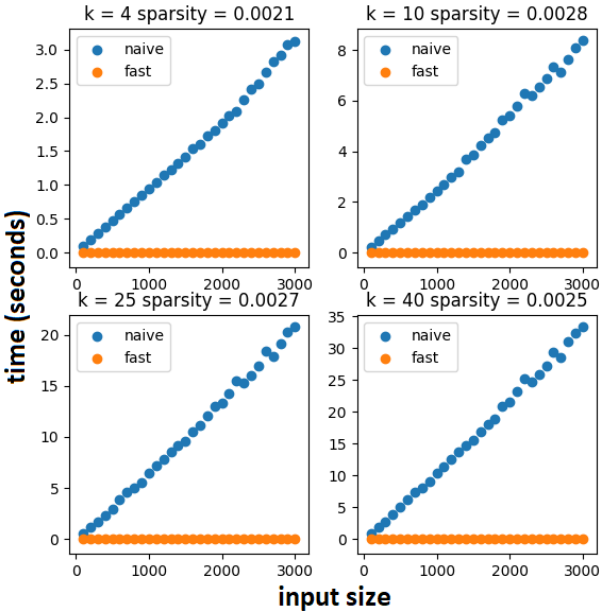


Figure 6.5: Running time of the naive $O(kn^2)$ and the new algorithm on relatively sparse data obtained from the sensor D002 in 5 second intervals.

data is not long enough yet. Which makes it seem like an almost constant running time. In Figure 6.6 and Figure 6.7 you can see the running time of our algorithm on larger input sizes. For sparsities around 0.1 you can see that our algorithm still performs about twice as well at input size 10000 as the naive algorithm does on input size 1000.
It is certain that our algorithm runs a lot faster than the naive algorithm, even for low k and small input sizes.

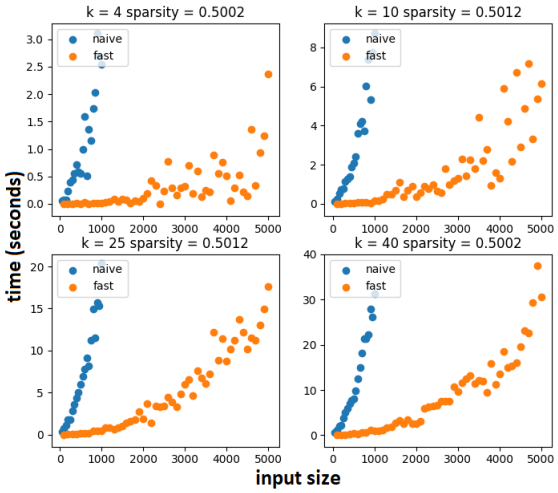


Figure 6.6: Running time of the naive $O(kn^2)$ till input size of 1000 and running time of the new algorithm till input size 5000 on dense randomly generated data.

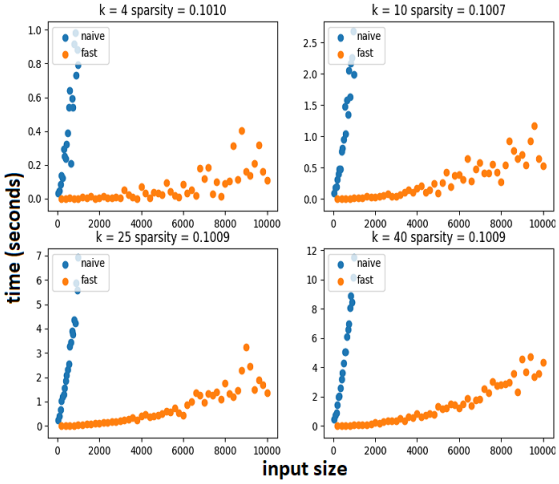


Figure 6.7: Running time of the naive $O(kn^2)$ till input size of 1000 and running time of the new algorithm till input size 10000 on sparse randomly generated data.

6.2 Testing stuctural qualities of the mean

Question 3. *How closely related are the length of the mean and the median condensed length?*

Question 4. *Is it more likely that the mean length equals the median condensed length if we consider more input strings?*

Question 5. *Is the mean length always the same as the median condensed length if we consider dense input strings?*

Question 6. *How does the starting (and ending symbol) of the mean relate to the starting symbols or starting blocks of the input strings?*

To answer Questions 3 to 5 regarding the structure of solutions for Binary-DTW-Mean, we will look at how the mean length differs from the median condensed length. Recall that by Claim 3 we know that if we assume that the input strings $s_1, s_2, ..., s_k$ are sorted by the lengths of their condensations, then there is always a (condensed) mean μ with $|\mu| \geq |\tilde{s}_{\lfloor \frac{k}{2} \rfloor + 1}|$ and we call this lower bound the median condensed length. The intuition being that for every cell we append before reaching this length, we always save at least as many blocks in the strings with longer condensations as we have to mismatch blocks in the mean for strings with shorter condensations.
In Figure 6.8 you can see how the median condensed length differs from the mean length for different sparsities. The data has been created with $n = 1000$ and $k = 10$ and the sparsity was used as the chance that the next symbol in the string will be different from

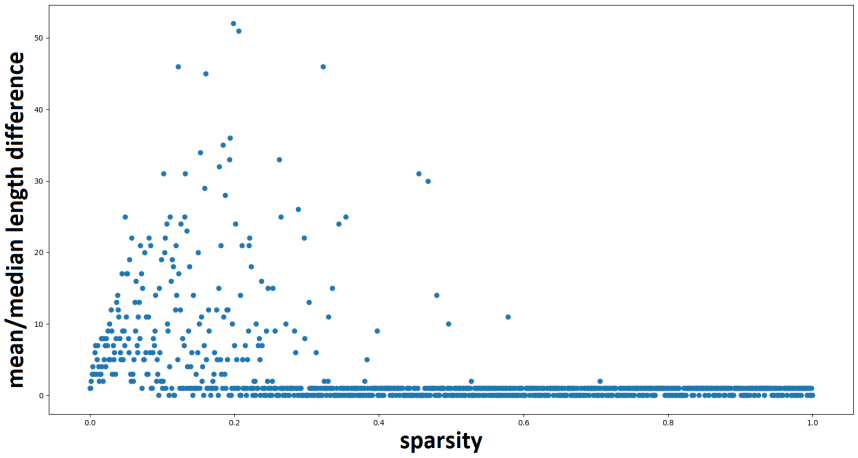


Figure 6.8: Differnce between median condensed length and calculated mean length with changing sparsity.

the last one so the actual sparsities are not necessarily exactly the sparsities given by the x-axis but very close to those. You can clearly see that for sparsities greater than 0.5, which we called dense there is only three points with difference greater than one from the median condensed length and only one of those points with difference greater than two. The common difference of one is likely caused by the differences in starting or ending symbols. Our fifth hypothesis is pretty much answered with this, that is, for dense data there almost always is a mean with median condensed length give or take one.
To answer the other two questions we have a look at Figure 6.9. Here the data was created with $n = 400$ and for each sparsity the strings were created symbol by symbol as explained before. Interestingly we can see that for sparse strings the mean very often differs quite a lot from the median condensed length and for a higher number of input strings it seems to differ even more often than for a low number of input strings. This could be the case because the more input strings we consider the more likely it is that there is one input string with high condensed length and long block sizes. We can also clearly see that for dense strings there is not a single red dot for any k.

To answer the last question we tested for different k and different sparsities how the starting symbol of the mean depends on the starting symbols or blocks of the input strings. To do that, we first tested how often the starting symbol of the mean is the same as the majority of starting symbols of input strings (see Table 6.1) and then also summed up the lengths of all starting 1-blocks and the lengths of starting 0-blocks and checked how often the mean corresponds to the bigger of those two sums (see Table 6.2). Overall the starting symbol of the mean matches the majority of starting symbols or blocks of the input strings pretty often but interestingly, for lower sparsities taking the length of starting blocks into account seems to show worse results than taking only the first symbol into account.

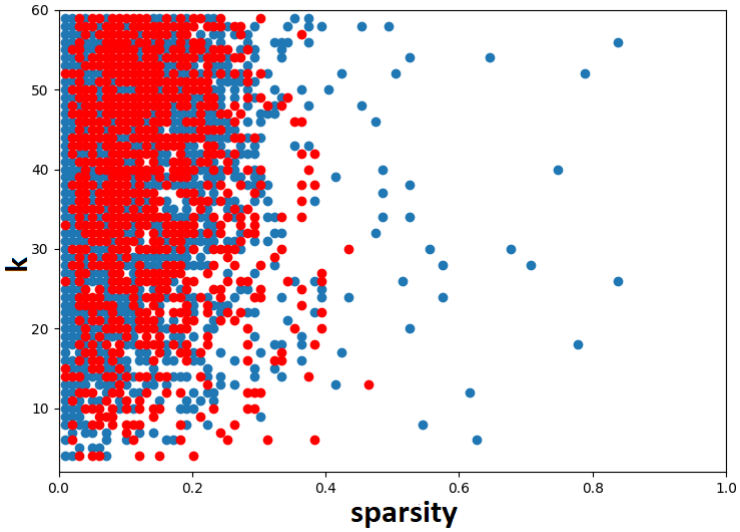


Figure 6.9: Difference of mean and median condensed length depending on sparsity and number of input strings. For every point in the 100×60 grid we calculated one mean for the respective k strings and given sparsity. No dot means that the median condensed length and the mean length did not differ by more than one. A blue dot means they differed slightly ($2-4$ difference) and a red dot means they differed by at least five.

Table 6.1: Frequency of the first symbol of the mean also being the first symbol in the majority of input strings

k/sparsity	0.05	0.1	0.2	0.5	0.8	1
5	76%	79%	82%	82%	82%	80%
15	75%	81%	82%	83%	85%	85%
40	82%	84%	88%	87%	91%	97%

Table 6.2: Frequency of the first symbol of the mean also being the in the majority of symbols throughout the first blocks of input strings

k/sparsity	0.05	0.1	0.2	0.5	0.8	1
5	69%	73%	75%	83%	85%	80%
15	67%	73%	75%	82%	88%	85%
40	66%	70%	74%	81%	91%	97%

7 Conclusion

We have seen that the problem of finding a mean in a dynamic time warping space can be solved very efficiently for binary data. It can also be reduced to a more well researched problem for which some efficient algorithms already exist. However, it still remains unclear whether Binary-DTW-Mean could be solved in $f(k)O(n)$ time but we think of this as unlikely.
For relatively dense data we have seen that we can choose a condensed string with starting and ending symbols corresponding with the majorities of starting and ending symbols in the input strings and length of the median condensed input string and it is very likely that this string will also be a mean of all input strings. For k given input strings we could do this in $O(nk)$ time and for run length encoded strings this would even be possible in $O(k)$ time. Of course this computed string is not certain to be an exact solution so computing it would only be useful if a minor error is allowed. One could try to generalize this notion to arbitrary data, where finding a mean is a very hard problem.
We have also seen that related problems like Binary-DTW-Center or more general problem formulations like Weighted Binary-DTW-Mean can be solved efficiently with the same reductions used when solving Binary-DTW-Mean.

Literature

[ABW15] A. Abboud, A. Backurs, and V. V. Williams. "Tight Hardness Results for LCS and Other Sequence Similarity Measures". In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS). Berkeley, CA, USA: IEEE, Oct. 2015, pp. 59–78. URL: http://ieeexplore.ieee.org/document/7354388/ (visited on 02/11/2019) (cit. on pp. 7, 13, 16, 18, 26).

[BFN18] L. Bulteau, V. Froese, and R. Niedermeier. "Hardness of Consensus Problems for Circular Strings and Time Series Averaging". In: *CoRR* abs/1804.02854 (2018). arXiv: 1804.02854. URL: http://arxiv.org/abs/1804.02854 (cit. on p. 7).

[BI15] A. Backurs and P. Indyk. "Edit Distance Cannot Be Computed in Strongly Subquadratic Time (Unless SETH is False)". In: *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*. STOC '15. Portland, Oregon, USA: ACM, 2015, pp. 51–58. URL: http://doi.acm.org/10.1145/2746539.2746612 (cit. on p. 5).

[Bri+19] M. Brill, T. Fluschnik, V. Froese, B. Jain, R. Niedermeier, and D. Schultz. "Exact mean computation in dynamic time warping spaces". In: *Data Mining and Knowledge Discovery* 33.1 (2019), pp. 252–291. URL: https://doi.org/10.1007/s10618-018-0604-8 (cit. on pp. 7, 12, 13, 15, 19).

[CL15] T. M. Chan and M. Lewenstein. "Clustered Integer 3SUM via Additive Combinatorics". In: *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*. STOC '15. Portland, Oregon, USA: ACM, 2015, pp. 31–40. URL: http://doi.acm.org/10.1145/2746539.2746568 (cit. on pp. 7, 16, 23).

[Coo+19] D. Cook, A. Crandall, J. Doppa, H. Ghasemzadeh, L. Holder, B. Shirazi, M. Schmitter-Edgecombe, and M. Tayler. *WSU CASAS Datasets*. [Accessed: 21.06.2019]. June 21, 2019. URL: http://casas.wsu.edu/datasets/ (visited on 06/27/2019) (cit. on pp. 7, 28).

[CTH18] C.-F. Chiu, C.-L. Tsai, and Y.-C. Hsu. "The Design of Music Conducting System Using Kinect and Dynamic Time Warping". In: *Frontier Computing*. Ed. by N. Y. Yen and J. C. Hung. Lecture Notes in Electrical Engineering. Springer Singapore, 2018, pp. 227–237 (cit. on p. 5).

[HBG19] Y. Hwang and S. B. Gelfand. In: MLDM 2019. To appear. July 2019 (cit. on pp. 7, 26).

[KKS13] S. K. Kumar, L. K. Kant, and S. Shachi. "HMM Based Enhanced Dynamic Time Warping Model for Efficient Hindi Language Speech Recognition System". In: *Mobile Communication and Power Engineering*. Ed. by V. V. Das and Y. Chaba. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2013, pp. 200–206 (cit. on p. 5).

[Kus19] W. Kuszmaul. "Dynamic Time Warping in Strongly Subquadratic Time: Algorithms for the Low-Distance Regime and Approximate Evaluation". In: *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Ed. by C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi. Vol. 132. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 80:1–80:15. URL: `http://drops.dagstuhl.de/opus/volltexte/2019/10656` (cit. on p. 5).

[MP05] A. Marzal and V. Palazón. "Dynamic Time Warping of Cyclic Strings for Shape Matching". In: *Pattern Recognition and Image Analysis*. Ed. by S. Singh, M. Singh, C. Apte, and P. Perner. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 644–652 (cit. on p. 5).

[Mue+16] A. Mueen, N. Chavoshi, N. Abu-El-Rub, H. Hamooni, and A. Minnich. "AWarp: Fast Warping Distance for Sparse Time Series". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016 IEEE 16th International Conference on Data Mining (ICDM). Barcelona, Spain: IEEE, Dec. 2016, pp. 350–359. URL: `http://ieeexplore.ieee.org/document/7837859/` (visited on 02/11/2019) (cit. on pp. 5, 7, 26, 28).

[NR07] V. Niennattrakul and C. A. Ratanamahatana. "On Clustering Multimedia Time Series Data Using K-Means and Dynamic Time Warping". In: *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*. 2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07). Apr. 2007, pp. 733–738 (cit. on p. 7).

[Sha+18] A. Sharabiani, H. Darabi, S. Harford, E. Douzali, F. Karim, H. Johnson, and S. Chen. "Asymptotic Dynamic Time Warping calculation with utilizing value repetition". In: *Knowledge and Information Systems* 57.2 (2018), pp. 359–388. URL: `https://doi.org/10.1007/s10115-018-1163-4` (cit. on pp. 5, 7, 26).

[Sku+13] H. Skutkova, M. Vitek, P. Babula, R. Kizek, and I. Provaznik. "Classification of genomic signals using dynamic time warping". In: *BMC Bioinformatics* 14.10 (Aug. 12, 2013), S1. URL: `https://doi.org/10.1186/1471-2105-14-S10-S1` (visited on 07/19/2019) (cit. on p. 5).

[VS15] Vikas and R. Sharma. "Analysis of Voice for Parkinson's Disease Persons Using Dynamic Time Warping Technique". In: *Intelligent Computing, Communication and Devices*. Ed. by L. C. Jain, S. Patnaik, and N. Ichalkaranje. Advances in Intelligent Systems and Computing. Springer India, 2015, pp. 753–759 (cit. on p. 5).

[WL16] H. Wang and Z. Li. “Accelerometer-based gesture recognition using dynamic time warping and sparse representation”. In: *Multimedia Tools and Applications* 75.14 (July 1, 2016), pp. 8637–8655. URL: https://doi.org/10.1007/s11042-015-2775-2 (visited on 07/19/2019) (cit. on p. 5).

[YW17] X. Yao and H. Wei. “Improving K-means clustering performance using a new global time-series averaging method”. In: *2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. 2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI). June 2017, pp. 1–6 (cit. on p. 7).

[ZLL14] X.-L. Zhang, Z.-G. Luo, and M. Li. “Merge-Weighted Dynamic Time Warping for Speech Recognition”. In: *Journal of Computer Science and Technology* 29.6 (Nov. 1, 2014), pp. 1072–1082. URL: https://doi.org/10.1007/s11390-014-1491-0 (visited on 07/19/2019) (cit. on p. 5).