Faster Binary Mean Computation Under Dynamic Time Warping

Nathan Schaar
Technische Universität Berlin, Faculty IV, Algorithmics and Computational Complexity, Germany
n.schaar@campus.tu-berlin.de

Vincent Froese
Technische Universität Berlin, Faculty IV, Algorithmics and Computational Complexity, Germany
vincent.froese@tu-berlin.de

Rolf Niedermeier
Technische Universität Berlin, Faculty IV, Algorithmics and Computational Complexity, Germany
rolf.niedermeier@tu-berlin.de

Abstract

Many consensus string problems are based on Hamming distance. We replace Hamming distance by the more flexible (e.g., easily coping with different input string lengths) dynamic time warping distance, best known from applications in time series mining. Doing so, we study the problem of finding a mean string that minimizes the sum of (squared) dynamic time warping distances to a given set of input strings. While this problem is known to be NP-hard (even for strings over a three-element alphabet), we address the binary alphabet case which is known to be polynomial-time solvable. We significantly improve on a previously known algorithm in terms of worst-case running time. Moreover, we also show the practical usefulness of one of our algorithms in experiments with real-world and synthetic data. Finally, we identify special cases solvable in linear time (e.g., finding a mean of only two binary input strings) and report some empirical findings concerning combinatorial properties of optimal means.

2012 ACM Subject Classification Theory of computation → Dynamic programming; Theory of computation → Pattern matching

Keywords and phrases consensus string problems, time series averaging, minimum 1-separated sum, sparse strings

Digital Object Identifier 10.4230/LIPIcs.CPM.2020.28

Supplementary Material Python source code available at `https://www.akt.tu-berlin.de/menue/software/`.

Funding *Nathan Schaar*: Partially supported by DFG NI 369/19.

1 Introduction

Consensus problems are an integral part of stringology. For instance, in the frequently studied Closest String problem one is given k strings of equal length and the task is to find a center string that minimizes the maximum Hamming distance to all k input strings. Closest String is NP-hard even for binary alphabet [11] and has been extensively studied in context of approximation and parameterized algorithmics [6, 9, 7, 8, 13, 15, 17, 20]. Notably, when one wants to minimize the sum of distances instead of the maximum distance, the problem is easily solvable in linear time by taking at each position a letter that appears most frequently in the input strings.

Hamming distance, however, is quite limited in many application contexts; for instance, how to define a center string in case of input strings that do not all have the same length? In context of analyzing time series (basically strings where the alphabet consists of rational numbers), the "more flexible" *dynamic time warping distance* [18] enjoys high popularity and can be computed for two input strings in subquadratic time [12, 14], essentially matching corresponding conditional lower bounds [1, 3]. Roughly speaking (see Section 2 for formal definitions and an example), measuring the dynamic time warping distance (dtw for short) can be seen as a two-step process: First, one aligns one time series with the other (by stretching them via duplication of elements) such that both time series end up with the same length. Second, one then calculates the Euclidean distance of the aligned time series (recall that here the alphabet consists of numbers). Importantly, restricting to the binary case, the dtw distance of two time series can be computed in $O(n^{1.87})$ time [1], where n is the maximum time series length (a result that will also be relevant for our work).

With the dtw distance at hand, the most fundamental consensus problem in this (time series) context is, given k input "strings" (over rational numbers), compute a mean string that minimizes the sum of (squared) dtw distances to all input strings. This problem is known as DTW-Mean in the literature and only recently has been shown to be NP-hard [4, 5]. For the most basic case, namely binary alphabet (that is, input and output are binary), however, the problem is known to be solvable in $O(kn^3)$ time [2]. By way of contrast, if one allows the mean to contain any rational numbers, then the problem is NP-hard even for binary inputs [5]. Moreover, the problem is also NP-hard for ternary input and output [4].

Formally, in this work we study the following problem:

Binary DTW-Mean (BDTW-Mean)

Input: Binary strings $s_1, \dots, s_k$ of length at most n and $c \in \mathbb{Q}$.
Question: Is there a binary string z such that $F(z) := \sum_{i=1}^{k} \operatorname{dtw}(s_i, z)^2 \leq c$?

Herein, the dtw function is formally defined in Section 2. The study of the special case of binary data may occur when one deals with binary states (e.g., switching between the active and the inactive mode of a sensor); binary data were recently studied in the dynamic time warping context [16, 19]. Clearly, binary data can always be generated from more general data by "thresholding".

Our main theoretical result is to show that BDTW-Mean can be solved in $O(kn^{1.87})$ and $O(k(n + m(m - \mu)))$ time, respectively, where m is the maximum and μ is the median condensation length of the input strings (the condensation of a string is obtained by repeatedly removing one of two identical consecutive elements). While the first algorithm, relies on an intricate "blackbox-algorithm" for a certain number problem from the literature (which so far was never implemented), the second algorithm (which we implemented) is more directly based on combinatorial arguments. Anyway, our new bounds improve on the standard $O(kn^3)$-time bound [2]. Moreover, we also experimentally tested our second algorithm and compared it to the standard one, clearly outperforming it (typically by orders of magnitude) on real-world and on synthetic instances. Further theoretical results comprise linear-time algorithms for special cases (two input strings or three input strings with some additional constraints). Further empirical results relate to the typical shape of a mean.

2 Preliminaries

For $n \in \mathbb{N}$, let $[n] := \{1, \dots, n\}$. We consider binary strings $x = x[1]x[2]\dots x[n] \in \{0,1\}^n$. We denote the length of x by $|x|$ and we also denote the last symbol $x[n]$ of x by $x[-1]$. For $1 \leq i \leq j \leq |x|$, we define the substring $x[i, j] := x[i] \dots x[j]$. A maximal substring of consecutive 1's (0's) in x is called a *1-block* (*0-block*). The i-th block of x (from left to right) is denoted $x^{(i)}$. A string x is called *condensed* if no two consecutive elements are equal, that is, every block is of size 1. The *condensation* of x is denoted $\tilde{x}$ and is defined as the

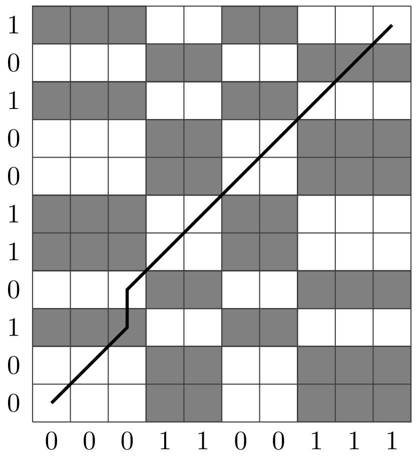

Figure 1 An optimal warping path for the strings $x = 00101100101$ (vertical axis) and $y = 0001100111$ (horizontal axis). Black cells have local cost 1. The string x consists of eight blocks with sizes 2,1,1,2,2,1,1,1 and y consists of four blocks with sizes 3,2,2,3. An optimal warping path has to pass through $(8-4)/2 = 2$ non-neighboring blocks of the six inner blocks of x.

condensed string obtained by removing one of two equal consecutive elements of x until the remaining series is condensed. Note that the condensation length $|\tilde{x}|$ equals the number of blocks in x.

The dynamic time warping distance measures the similarity of two strings using non-linear alignments defined via so-called warping paths.

▶ **Definition 1.** *A* warping path *of order $m \times n$ is a sequence $p = (p_1, \ldots, p_L)$, $L \in \mathbb{N}$, of index pairs $p_\ell = (i_\ell, j_\ell) \in [m] \times [n]$, $1 \le \ell \le L$, such that*

(i) $p_1 = (1,1)$,
(ii) $p_L = (m,n)$, *and*
(iii) $(i_{\ell+1} - i_\ell, j_{\ell+1} - j_\ell) \in \{(1,0),(0,1),(1,1)\}$ *for each* $\ell \in [L-1]$.

A warping path can be visualized within an $m \times n$ "warping matrix" (see Figure 1). The set of all warping paths of order $m \times n$ is denoted by $\mathcal{P}_{m,n}$. A warping path $p \in \mathcal{P}_{m,n}$ defines an *alignment* between two strings $x \in \mathbb{Q}^m$ and $y \in \mathbb{Q}^n$ in the following way: A pair $(i,j) \in p$ *aligns* element x_i with y_j with a local cost of $(x_i - y_j)^2$. The dtw distance between two strings x and y is defined as

$$\mathrm{dtw}(x,y) := \min_{p \in \mathcal{P}_{m,n}} \sqrt{\sum_{(i,j)\in p} (x_i - y_j)^2}.$$

It is computable via standard dynamic programming in $O(mn)$ time[1] [18], with recent theoretical improvements to subquadratic time [12, 14].

3 DTW on Binary Strings

We briefly discuss some known results about the dtw distance between binary strings since these will be crucial for our algorithms for BDTW-MEAN.

[1] Throughout this work, we assume that all arithmetic operations can be carried out in constant time.

Abboud et al. [1, Section 5] showed that the dtw distance of two binary strings of length at most n can be computed in $O(n^{1.87})$ time. They obtained this result by reducing the dtw distance computation to the following integer problem.

Min 1-Separated Sum (MSS)

Input: A sequence $(b_1, \dots, b_m)$ of m positive integers and an integer $r \geq 0$.
Task: Select r integers $b_{i_1}, \dots, b_{i_r}$ with $1 \leq i_1 < i_2 < \cdots < i_r \leq m$ and $i_j < i_{j+1} - 1$ for all $1 \leq j < r$ such that $\sum_{j=1}^{r} b_{i_j}$ is minimized.

The integers of the MSS instance correspond to the block sizes of the input string which contains more blocks.

▶ **Theorem 2** ([1, Theorem 8]). *Let $x \in \{0,1\}^m$ and $y \in \{0,1\}^n$ be two binary strings such that $x[1] = y[1]$, $x[m] = y[n]$, and $|\tilde{x}| \geq |\tilde{y}|$. Then,* $\mathrm{dtw}(x,y)^2$ *equals the sum of a solution for the MSS instance $\big((|x^{(2)}|, \dots, |x^{(|\tilde{x}|-1)}|), (|\tilde{x}| - |\tilde{y}|)/2\big)$.*

The idea behind Theorem 2 is that exactly $(|\tilde{x}| - |\tilde{y}|)/2$ non-neighboring blocks of x are misaligned in any warping path (note that $|\tilde{x}| - |\tilde{y}|$ is even since x and y start and end with the same symbol). An optimal warping path can thus be obtained from minimizing the sum of block sizes of these misaligned blocks. For example, in Figure 1 the dtw distance corresponds to a solution of the MSS instance $((1,1,2,2,1,1),2)$.

Abboud et al. [1, Theorem 10] showed how to solve MSS in $O(n^{1.87})$ time, where $n = \sum_{i=1}^{m} b_i$. They gave a recursive algorithm that, on input $((b_1, \dots, b_m), r)$, outputs four lists C^{00}, C^{0*}, C^{*0}, and C^{**}, where, for $t \in \{0, \dots, r\}$,

- $C^{**}[t]$ is the sum of a solution for the MSS instance $((b_1, \dots, b_m), t)$,
- $C^{0*}[t]$ is the sum of a solution for the MSS instance $((b_2, \dots, b_m), t)$,
- $C^{*0}[t]$ is the sum of a solution for the MSS instance $((b_1, \dots, b_{m-1}), t)$, and
- $C^{00}[t]$ is the sum of a solution for the MSS instance $((b_2, \dots, b_{m-2}), t)$.

Note that $C^{**}[r]$ yields the solution. We will make use of their algorithm when solving BDTW-Mean. We will also use the following simple dynamic programming algorithm for MSS which is faster for large input integers.

▶ **Lemma 3.** *Min 1-Separated Sum is solvable in $O(mr)$ time.*

Proof. Let $((b_1, \dots, b_m), r)$ be an MSS instance. We define a dynamic programming table M as follows: For each $i \in [m]$ and each $j \in \{0, \dots, \min(r, \lceil i/2 \rceil)\}$, $M[i,j]$ is the sum of a solution of the subinstance $((b_1, \dots, b_i), j)$. Clearly, it holds $M[i,0] = 0$ and $M[i,1] = \min\{b_1, \dots, b_i\}$ for all i. Further, it holds $M[3,2] = b_1 + b_3$. For all $i \in \{4, \dots, m\}$ and $j \in \{2, \dots, \min(r, \lceil i/2 \rceil)\}$, the following recursion holds

$$M[i,j] = \min(b_i + M[i-2, j-1], M[i-1, j]).$$

Hence, the table M can be computed in $O(mr)$ time. ◀

Note that the above algorithms only compute the dtw distance between binary strings with equal starting and ending symbol. However, it is an easy observation that the dtw distance of arbitrary binary strings can recursively be obtained from this via case distinction on which first and/or which last block to misalign.

▶ **Observation 4** ([1, Claim 6]). *Let $x \in \{0,1\}^m$, $y \in \{0,1\}^n$ with $m' := |\tilde{x}| \geq n' := |\tilde{y}|$. Further, let $a := |x^{(1)}|$, $a' := |x^{(m')}|$, $b := |y^{(1)}|$, and $b' := |y^{(n')}|$. The following holds:*

- *If* $x[1] \neq y[1]$, *then*

$$\operatorname{dtw}(x,y)^2 = \begin{cases} \max(a,b), & m' = n' = 1 \\ a + \operatorname{dtw}(x[a+1,m],y)^2, & m' > n' = 1 \\ \min\left(a + \operatorname{dtw}(x[a+1,m],y)^2, b + \operatorname{dtw}(x,y[b+1,n])^2\right), & n' > 1 \end{cases}.$$

- *If* $x[1] = y[1]$ *and* $x[m] \neq y[n]$, *then*

$$\operatorname{dtw}(x,y)^2 = \begin{cases} a' + \operatorname{dtw}(x[1,m-a'],y)^2, & n' = 1 \\ \min\left(a' + \operatorname{dtw}(x[1,m-a'],y)^2, b' + \operatorname{dtw}(x,y[1,n-b'])^2\right), & n' > 1 \end{cases}.$$

For condensed strings, Brill et al. [2] derived the following useful closed form for the dtw distance (which basically follows from Observation 4 and Theorem 2).

▶ **Lemma 5** ([2, Lemma 1 and 2]). *For a condensed binary string x and a binary string y with $|\tilde{y}| \leq |x|$, it holds that*

$$\operatorname{dtw}(x,y)^2 = \begin{cases} \lceil(|x| - |\tilde{y}|)/2\rceil, & x_1 = y_1 \\ 2, & x_1 \neq y_1 \wedge |x| = |\tilde{y}| \\ 1 + \lfloor(|x| - |\tilde{y}|)/2\rfloor, & x_1 \neq y_1 \wedge |x| > |\tilde{y}| \end{cases}.$$

Note that Lemma 5 implies that one can compute the dtw distance in constant time when the condensation lengths of the inputs are known and the string with longer condensation length is condensed.

Our key lemma now states that the dtw distances between an arbitrary fixed string and all condensed strings of shorter condensation length can also be computed efficiently.

▶ **Lemma 6.** *Let $s \in \{0,1\}^n$ with $\ell := |\tilde{s}|$. Given ℓ and the block sizes $b_1, \ldots, b_\ell$ of s, the dtw distances between s and all condensed strings of lengths $\ell', \ldots, \ell$ for some given $\ell' \leq \ell$ can be computed in*

(i) *$O(n^{1.87})$ time and in*

(ii) *$O(\ell(\ell - \ell'))$ time, respectively.*

Proof. Let x be a condensed string of length $i \in \{\ell', \ldots, \ell\}$. Observation 4 and Theorem 2 imply that we essentially have to solve MSS on four different subsequences of block sizes of s (depending on the first and last symbol of x) in order to compute $\operatorname{dtw}(s,x)$. Namely, the four cases are $(b_2, \ldots, b_{\ell-1})$, $(b_3, \ldots, b_{\ell-1})$, $(b_2, \ldots, b_{\ell-2})$, and $(b_3, \ldots, b_{\ell-2})$. Let $r := \lceil(\ell - \ell')/2\rceil$

(i) We run the algorithm of Abboud et al. [1, Theorem 10] on the instance $((b_2, \ldots, b_{\ell-1}), r)$ to obtain in $O(n^{1.87})$ time the four lists $C^{\alpha\beta}$, for $\alpha, \beta \in \{0,1\}$, where $C^{\alpha\beta}$ contains the solutions of $((b_{2+\alpha}, \ldots, b_{\ell-1-\beta}), r')$ for all $r' \in \{0, \ldots, r\}$. From these four lists, we can compute the requested dtw distances (using Observation 4) in $O(\ell)$ time.

(ii) We compute the solutions of the four above MSS instances using Lemma 3. For each $\alpha, \beta \in \{0,1\}$, let $M^{\alpha\beta}$ be the dynamic programming table computed in $O(\ell(\ell - \ell'))$ time for the instance $((b_{2+\alpha}, \ldots, b_{\ell-1-\beta}), r)$. Again, we can compute the requested dtw distances from these four tables in $O(\ell)$ time (using Observation 4). ◀

4 More Efficient Solution of BDTW-Mean

Brill et al. [2] gave an $O(kn^3)$-time algorithm for BDTW-Mean. The result is based on showing that there always exists a condensed mean of length at most $n+1$. Thus, there are $2(n+1)$ candidate strings to check. For each candidate, one can compute the dtw distance to every input string in $O(kn^2)$ time. It is actually enough to only compute the dtw distance for the two length-$(n+1)$ candidates to all k input strings since the resulting dynamic programming tables also yield all the distances to shorter candidates. That is, the running time can actually be bounded in $O(kn^2)$.

We now give an improved algorithm. To this end, we first show the following improved bounds on the (condensation) length of a mean.

▶ **Lemma 7.** *Let $s_1, \dots, s_k$ be binary strings with $|\tilde{s}_1| \leq \dots \leq |\tilde{s}_k|$ and let z be a mean of these k strings, that is,*

$$z \in \operatorname*{arg\,min}_{x \in \{0,1\}^*} \sum_{i=1}^{k} \mathrm{dtw}(s_i, x)^2.$$

Then, it holds $\mu - 2 \leq |\tilde{z}| \leq m + 1$, where $\mu := |\tilde{s}_{\lceil k/2 \rceil}|$ is the median condensation length and $m := |\tilde{s}_k|$ is the maximum condensation length.

Proof. It suffices to show the claimed bounds for condensed means. Since $\mathrm{dtw}(\tilde{x}, y) \leq \mathrm{dtw}(x, y)$ holds for all strings x, y [2, Proposition 1], the bounds also hold for arbitrary means.

The upper bound $m+1$ can be derived from Lemma 5. Let x be a condensed string of length $|x| \geq m+2$ and let $x' := x[1, m]$. If $|x| > m+2$, then $\mathrm{dtw}(x', s_i)^2 < \mathrm{dtw}(x, s_i)^2$ holds for every $i \in [k]$, which implies $F(x') = \sum_{i=1}^{k} \mathrm{dtw}(s_i, x')^2 < \sum_{i=1}^{k} \mathrm{dtw}(s_i, x)^2 = F(x)$. Hence, x is not a mean. If $|x| = m+2$, then $\mathrm{dtw}(x', s_i)^2 \leq \mathrm{dtw}(x, s_i)^2$ holds for every $i \in [k]$, that is, $F(x') \leq F(x)$. If $F(x') < F(x)$, then x is clearly not a mean. If $F(x') = F(x)$, then $\mathrm{dtw}(x', s_i)^2 = \mathrm{dtw}(x, s_i)^2$ holds for all $i \in [k]$. In fact, $\mathrm{dtw}(x', s_i)^2 = \mathrm{dtw}(x, s_i)^2$ only holds if $|\tilde{s}_i| = m$ and $s_i[1] \neq x[1]$, in which case $\mathrm{dtw}(x, s_i)^2 = 2$. Thus, we have $F(x) = 2k$ and $\tilde{s}_1 = \tilde{s}_2 = \dots = \tilde{s}_k$. But then $\tilde{s}_1$ is clearly the unique mean (with $F(\tilde{s}_1) = 0$).

For the lower bound, let x be a condensed string of length $\ell < \mu - 2$ and let $x' := x[1] \dots x[\ell] x[\ell-1] x[\ell]$. Then, for every s_i with $|\tilde{s}_i| \leq \ell$ (of which there are less than $\lceil k/2 \rceil$ since $\ell < \mu$), it holds $\mathrm{dtw}(x', s_i)^2 \leq \mathrm{dtw}(x, s_i)^2 + 1$ (by Lemma 5).

Now, for every s_i with $|\tilde{s}_i| > \ell + 2$ (of which there are at least $\lceil k/2 \rceil$ since $\ell + 2 < \mu$), it holds $\mathrm{dtw}(x', s_i)^2 \leq \mathrm{dtw}(x, s_i)^2 - 1$. This is easy to see from Theorem 2 for the case that $s_i[1] = x'[1]$ and $s_i[-1] = x'[-1]$ holds since the number of misaligned blocks of s_i decreases by at least one. From this, Observation 4 yields the other three possible cases of starting and ending symbols since the sizes of the first and last block of x and of x' are clearly all the same (one).

It remains to consider input strings s_i with $\ell < |\tilde{s}_i| \leq \ell + 2$. We show that in this case $\mathrm{dtw}(x', s_i)^2 \leq \mathrm{dtw}(x, s_i)^2$ holds. Let $|\tilde{s}_i| = \ell + 2$. Note that then either $x'[1] = s_i[1]$ and $x'[-1] = s_i[-1]$ holds or $x'[1] \neq s_i[1]$ and $x'[-1] \neq s_i[-1]$ holds. In the former case, it clearly holds $\mathrm{dtw}(x', s_i)^2 = 0$ by Lemma 5. In the latter case, we clearly have $\mathrm{dtw}(x, s_i)^2 \geq 2$, and, by Lemma 5, we have $\mathrm{dtw}(x', s_i)^2 = 2$. Finally, let $|\tilde{s_i}| = \ell + 1$ and note that then either $x'[1] = s_i[1]$ and $x'[-1] \neq s_i[-1]$ holds or $x'[1] \neq s_i[1]$ and $x'[-1] = s_i[-1]$ holds. Thus, we clearly have $\mathrm{dtw}(x, s_i)^2 \geq 1$. By Lemma 5, we have $\mathrm{dtw}(x', s_i)^2 = 1$.

Summing up, we obtain $F(x') \leq F(x) + a - b$, where $a = |\{i \in [k] \mid |\tilde{s}_i| < \ell\}| < \lceil k/2 \rceil$ and $b = |\{i \in [k] \mid |\tilde{s}_i| > \ell + 2\}| \geq \lceil k/2 \rceil$. That is, $F(x') < F(x)$ and x is not a mean. ◀

Note that the length bounds in Lemma 7 are tight. For the upper bound, consider the two strings 000 and 111 having the two means 01 and 10. For the lower bound, consider the seven strings 0, 0, 0, 101, 101, 010, 010 with the unique mean 0.

Lemma 7 upper-bounds the number of mean candidates we have to consider in terms of the condensation lengths of the inputs. In order to compute the dtw distances between mean candidates and input strings, we can now use Lemma 6. We arrive at the following result.

▶ **Theorem 8.** *Let $s_1, \dots, s_k$ be binary strings with $|\tilde{s}_1| \le \dots \le |\tilde{s}_k|$ and $n := \max_{j=1,\dots,k} |s_j|$, $\mu := |\tilde{s}_{\lceil k/2 \rceil}|$, and $m := |\tilde{s}_k|$. The condensed means of $s_1, \dots, s_k$ can be computed in*
- **(i)** *$O(kn^{1.87})$ time and in*
- **(ii)** *$O(k(n + m(m - \mu)))$ time.*

Proof. From Lemma 7, we know that there are $O(m - \mu)$ many candidate strings to check. First, in linear time, we determine the block lengths for each s_j. Now, let x be a candidate string, that is, x is a condensed binary string with $\mu - 2 \le |x| \le m + 1$. We need to compute $\operatorname{dtw}(x, s_j)^2$ for each $j = 1, \dots, k$. Consider a fixed string s_j. For all candidates x with$|x| \ge |\tilde{s}_j|$, we can simply compute $\operatorname{dtw}(x, s_j)^2$ in constant time using Lemma 5. For all x with $|x| < |\tilde{s}_j|$, we can use Lemma 6. Thus, overall, we can compute the dtw distances between all candidates and all input strings in $O(kn^{1.87})$ time, or alternatively in $O(km(m - \mu))$ time. Finally, we determine the candidates with the minimum sum of dtw distances in $O(k(m - \mu))$ time. ◀

We remark that similar results also hold for the related problems Weighted Binary DTW-Mean, where the objective is to minimize $F(z) := \sum_{i=1}^{k} w_i \operatorname{dtw}(s_i, z)^2$ for some $w_i \ge 0$, and Binary DTW-Center with $F(z) := \max_{i=1,\dots,k} \operatorname{dtw}(s_i, z)^2$ (that is, the dtw version of Closest String). It is easy to see that also in these cases there exists a condensed solution. Moreover, the length is clearly bounded between the minimum and the maximum condensation length of the inputs. Hence, analogously to Theorem 8, we obtain the following.

▶ **Corollary 9.** *Weighted Binary DTW-Mean and Binary DTW-Center can be solved in $O(kn^{1.87})$ time and in $O(k(n + m(m - \nu)))$ time, where m is the maximum condensation length and ν is the minimum condensation length.*

5 Linear-Time Solvable Special Cases

Notably, Theorem 8 (ii) yields a linear-time algorithm when $m - \mu$ is constant and also when all input strings have the same length n and $m(m - \mu) \in O(n)$. Now, we show two more linear-time solvable cases.

▶ **Theorem 10.** *A condensed mean of two binary strings can be computed in linear time.*

Proof. Let $s_1, s_2 \in \{0, 1\}^*$ be two input strings. We first determine the condensations and block sizes of s_1 and s_2 in linear time. Let $\ell_i := |\tilde{s}_i|$, for $i \in [2]$, and assume that $\ell_1 \le \ell_2$. In the following, all claimed relations between dtw distances can easily be verified using Observation 4 (together with Theorem 2) and Lemma 5.

If $\ell_1 = \ell_2$, then, by Theorem 8 (with $\mu = m = \ell_1$), all condensed means can be computed in $O(\ell_1)$ time.

If $\ell_1 < \ell_2$, then $\tilde{s}_2$ is a mean. To see this, note first that $F(\tilde{s}_2) = \operatorname{dtw}(s_1, \tilde{s}_2)^2$. Let x be a condensed string. If $|x| < \ell_1$, then $\operatorname{dtw}(s_1, x)^2 > 0$ and $\operatorname{dtw}(s_2, x)^2 \ge \operatorname{dtw}(s_2, \tilde{s}_1)^2 \ge \operatorname{dtw}(\tilde{s}_2, \tilde{s}_1)^2 = \operatorname{dtw}(\tilde{s}_2, s_1)^2$. Thus, $F(x) > F(\tilde{s}_2)$. Similarly, if $|x| > \ell_2$, then $\operatorname{dtw}(s_1, x)^2 \ge \operatorname{dtw}(s_1, \tilde{s}_2)^2$, $\operatorname{dtw}(s_2, x)^2 > 0$, and $F(x) > F(\tilde{s}_2)$. If $\ell_1 \le |x| < \ell_2$, then $\operatorname{dtw}(s_1, x)^2 + \operatorname{dtw}(s_2, x)^2 \ge \operatorname{dtw}(s_1, \tilde{s}_2)^2$, and thus $F(x) \ge F(\tilde{s}_2)$. ◀

For three input strings, we show linear-time solvability if all strings begin with the same symbol and end with the same symbol.

▶ **Theorem 11.** *Let s_1, s_2, s_3 be binary strings with $s_1[1] = s_2[1] = s_3[1]$ and $s_1[-1] = s_2[-1] = s_3[-1]$. A condensed mean of s_1, s_2, s_3 can be computed in linear time.*

Proof. We first determine the condensations and block sizes of s_1, s_2, and s_3 in linear time. Let $\ell_i := |\tilde{s}_i|$, for $i \in [3]$, and assume $\ell_1 \leq \ell_2 \leq \ell_3$. Note that every mean starts with $s_1[1]$ and ends with $s_1[-1]$. To see this, consider any string x with $x[1] \neq s_1[1]$ (or $x[-1] \neq s_1[-1]$) and observe that either removing the first (or last) symbol or adding $s_1[1]$ to the front (or $s_1[-1]$ to the end) yields a better F-value. Moreover, it is easy to see that every condensed mean has length at least ℓ_2 since increasing the length of any shorter condensed string by two increases the dtw distance to s_1 by at most one (Lemma 5) and decreases the dtw distances to s_2 and s_3 by at least one (Theorem 2).

Note that a mean could be even longer than ℓ_2 since further increasing the length by two increases the dtw distance to s_1 and s_2 by at most one and could possibly decrease the dtw distance to s_3 by at least two (if a misaligned block of size at least two can be saved). In fact, we can determine an optimal mean length in $O(\ell_3)$ time by greedily computing the maximum number ρ of 1-separated (that is, non-neighboring) blocks of size one among $s_3^{(2)}, \ldots, s_3^{\ell_3 - 1}$. Then there is a mean of length $\ell_3 - 2\rho$ (that is, exactly ρ size-1 blocks of s_3 are misaligned). Clearly, any longer condensed string has a larger F-value and every shorter condensed string has at least the same F-value. ◀

We strongly conjecture that similar but more technical arguments can be used to obtain a linear-time algorithm for three arbitrary input strings. For more than three strings, however, it is not clear how to achieve linear time, since the mean length cannot be greedily determined.

6 Empirical Evaluation

We conducted some experiments to empirically evaluate our algorithms and to observe structural characteristics of binary means. In Section 6.1 we compare the running times of our $O(k(n + m(m - \mu)))$-time algorithm (Theorem 8 (ii)) with the standard $O(kn^2)$-time dynamic programming approach [2] described in the beginning of Section 4. We implemented both algorithms in Python.[2] Note that we did not test the $O(kn^{1.87})$-time algorithm since it uses another blackbox algorithm (which has not been implemented so far) in order to solve MSS. However, we believe that it is anyway slower in practice. In Section 6.2, we empirically investigate structural properties of binary condensed means such as the length and the starting symbol (note that these two characteristics completely define the mean). All computations have been done on an Intel i7 QuadCore (4.0 GHz).

For our experiments we used the CASAS human activity datasets[3] [10] as well as some randomly generated data. The data in the CASAS datasets are generated from sensors which detect (timestamped) changes in the environment (for example, a door being opened/closed) and have previously been used in the context of binary dtw computation [16]. We used the datasets HH101–HH130 and sampled from them to obtain input strings of different lengths and sparsities (for a binary string s, we define the *sparsity* as $|\tilde{s}|/|s|$). For the random data, the sparsity value was used as the probability that the next symbol in the string will be different from the last one (hence, the actual sparsities are not necessarily exactly the sparsities given but very close to those).

[2] Source code available at `https://www.akt.tu-berlin.de/menue/software/`.
[3] Available at `http://casas.wsu.edu/datasets/`.

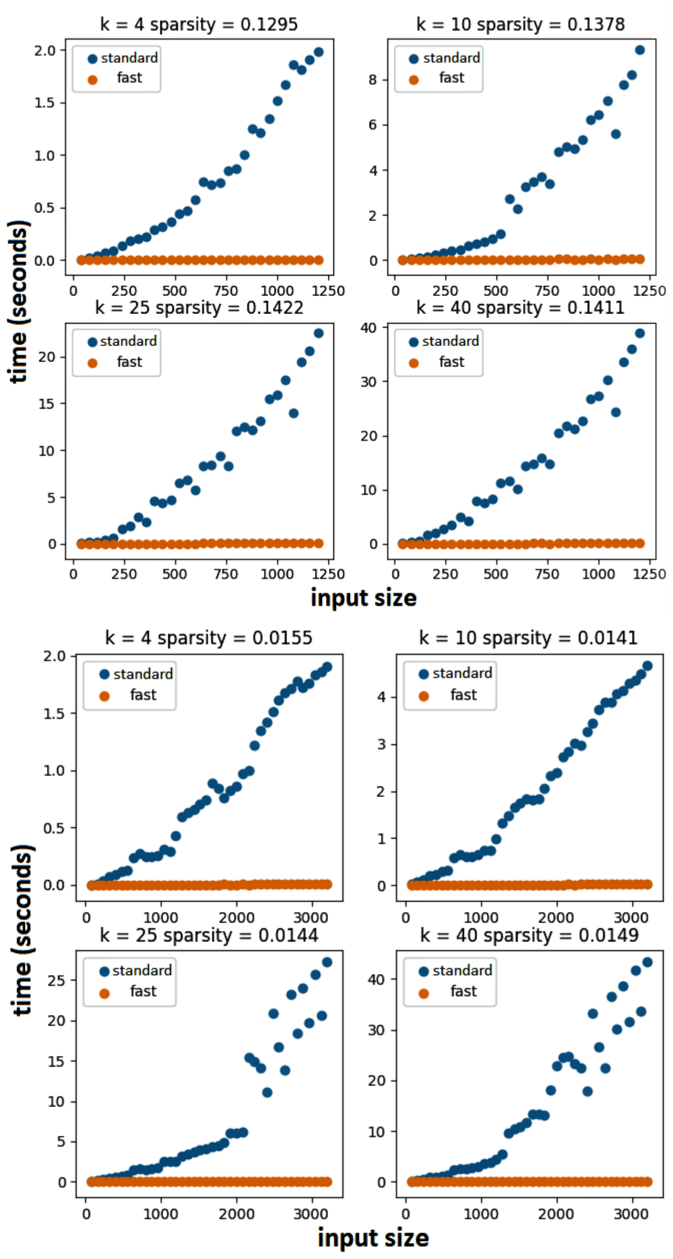

Figure 2 Running times of the standard and the fast algorithm on sparse data (sensor D002 in dataset HH101) in 10-minute intervals (top) and 1-minute intervals (bottom).

6.1 Running Time Comparison

To examine the speedup provided by our algorithm, we compare it with the standard $O(kn^2)$-time dynamic programming algorithm on (very) sparse real-world data (sparsity ≈ 0.1 and ≈ 0.01) and on sparse (sparsity ≈ 0.1) and dense (sparsity ≈ 0.5) random data, both for various values of k. Figure 2 shows the running times on real-world data. For sparsity ≈ 0.1, our algorithm is around 250 times faster than the standard algorithm and for sparsity ≈ 0.01 it is around 350 times faster. Figure 3 shows the running times of the algorithms on larger

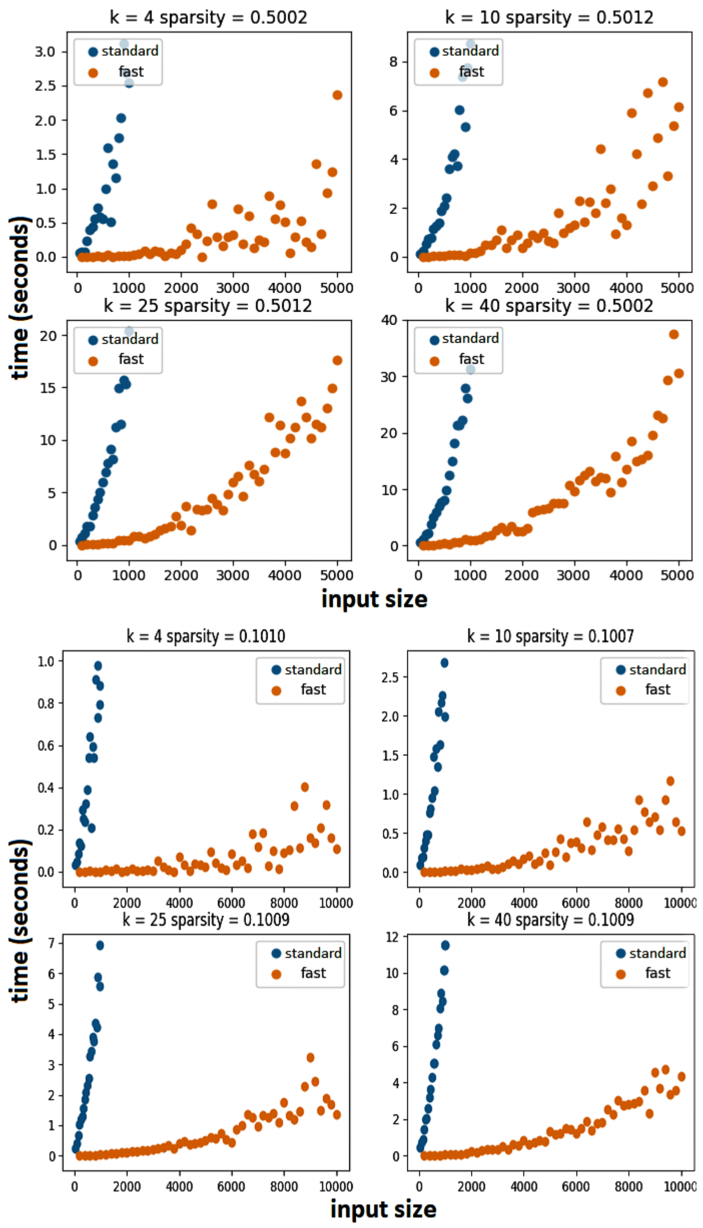

Figure 3 Running times of the standard algorithm for $n \leq 1000$ and of the fast algorithm for $n \leq 5000$ on dense random data (top) and for $n \leq 10,000$ on sparse random data (bottom).

random data. For sparsity ≈ 0.1, our algorithm is still twice as fast for $n = 10,000$ as the standard algorithm for $n = 1000$. These results clearly show that our algorithm is valuable in practice.

6.2 Structural Observations

We also studied the typical shape of binary condensed means. The questions of interest are "What is the typical length of a condensed mean?" and "What is the first symbol of a condensed mean?". Since the answers to these two questions completely determine a condensed mean, we investigated whether they can be easily determined from the inputs.

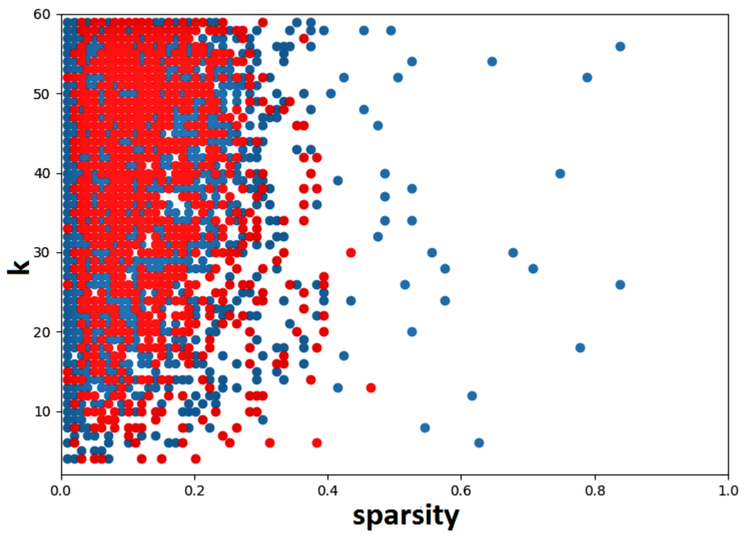

Figure 4 Difference between median condensed length and calculated mean length depending on sparsity and number of input strings. For every pair $(\sigma, k) \in \{0.01, \dots, 1.0\} \times [60]$, we calculated one mean for k strings with sparsity σ. No dot means that the median condensed length and the mean length did not differ by more than one. A blue (dark gray) dot means they differed slightly (difference between two and four) and a red (light gray) dot means they differed by at least five.

To answer the question regarding the mean length, we tested how much the actual mean length differs from the median condensed length. Recall that by Lemma 7 we know that every condensed mean has length at least $\mu - 2$, where μ is the median input condensation length. We call this lower bound the median condensed length. We used our algorithm (Theorem 8 (ii)) to compute condensed means on random data with sparsities $0.01, \dots, 1.0$, $k = 1, \dots, 60$ and $n \leq 400$. Figure 4 clearly shows that on dense data (sparsity > 0.5), the difference between the mean length and the median condensed length is rarely more than one. This can be explained by the fact that for dense strings all blocks are usually small such that there is no gain in making the mean longer than the median condensed length. We remark that a difference of one appears quite often which might be caused by different starting or ending symbols of the inputs. In general, for dense data the mean length almost always is at most the median condensed length plus one, whereas for sparser data the mean can become longer. As regards the dependency of the mean length on the number k of inputs, it can be observed that, for sparse data (sparsity < 0.5), the mean length differs even more for larger k. This may be possible because more input strings increase the probability that there is one input string with long condensation length and large block sizes. For dense inputs, there seems to be no real dependence on k.

To answer the question regarding the first symbol of a mean, we tested on random data with different k values and different sparsities ($n \leq 500$), how the starting symbol of the mean depends on the starting symbols or blocks of the input strings. First, we tested how often the starting symbol of the mean equals the majority of starting symbols of the input strings (see Table 1). Then, we also summed up the lengths of all starting 1-blocks and the lengths of all starting 0-blocks and checked how often the mean starts with the symbol corresponding to the larger of those two sums (see Table 2). Overall, the starting symbol of the mean matches the majority of starting symbols or blocks of the input strings in most

Table 1 Frequency (over 1000 runs) 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 2 Frequency (over 1000 runs) of the first symbol of the mean also being 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%

cases ($\approx$ 70–90%, increasing with higher sparsity). For low sparsities, however, taking the length of starting blocks into account seems to yield less matches. This might be due to large outlier starting blocks (note that this effect is even worse for larger k).

To sum up the above empirical observations, we conclude that a condensed binary mean typically has a length close to the median condensed length and starts with the majority symbol among the starting symbols in the inputs.

7 Conclusion

In this work we made progress in understanding and efficiently computing binary means of binary strings with respect to the dtw distance. First, we proved tight lower and upper bounds on the length of a binary (condensed) mean which we then used to obtain fast polynomial-time algorithms to compute binary means by solving a certain number problem efficiently. We also obtained linear-time algorithms for $k \leq 3$ input strings. Moreover, we empirically showed that the actual mean length is often very close to the proven lower bound.

As regards future research challenges, it would be interesting to further improve the running time with respect to the maximum input string length n. This could be achieved by finding faster algorithms for our "helper problem" MIN 1-SEPARATED SUM (MSS). Can one solve BDTW-MEAN in linear time for every constant k (that is, $f(k) \cdot n$ time for some function f)? Also, finding improved algorithms for the weighted version or the center version (see Section 4) might be of interest.

References

1 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 (FOCS '15)*, pages 59–78, 2015.

2 Markus Brill, Till Fluschnik, Vincent Froese, Brijnesh J. Jain, Rolf Niedermeier, and David Schultz. Exact mean computation in dynamic time warping spaces. *Data Mining and Knowledge Discovery*, 33(1):252–291, 2019.

3 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS '15)*, pages 79–97, 2015.

4 Kevin Buchin, Anne Driemel, and Martijn Struijs. On the hardness of computing an average curve. *CoRR*, abs/1902.08053, 2019. Preprint appeared at the *35th European Workshop on Computational Geometry (EuroCG '19)*.

5 Laurent Bulteau, Vincent Froese, and Rolf Niedermeier. Tight hardness results for consensus problems on circular strings and time series. *CoRR*, abs/1804.02854, 2018.

6 Laurent Bulteau, Falk Hüffner, Christian Komusiewicz, and Rolf Niedermeier. Multivariate algorithmics for NP-hard string problems. *Bulletin of the EATCS*, 114, 2014.

7 Zhi-Zhong Chen, Bin Ma, and Lusheng Wang. A three-string approach to the closest string problem. *Journal of Computer and System Sciences*, 78(1):164–178, 2012.

8 Zhi-Zhong Chen, Bin Ma, and Lusheng Wang. Randomized fixed-parameter algorithms for the closest string problem. *Algorithmica*, 74(1):466–484, 2016.

9 Zhi-Zhong Chen and Lusheng Wang. Fast exact algorithms for the closest string and substring problems with application to the planted (ℓ, d)-motif model. *IEEE/ACM Transactions Computational Biology and Bioinformatics*, 8(5):1400–1410, 2011.

10 Diane Cook, Aaron Crandall, Brian Thomas, and Narayanan Krishnan. Casas: A smart home in a box. *Computer*, 46, 2013.

11 Moti Frances and Ami Litman. On covering problems of codes. *Theory of Computing Systems*, 30(2):113–119, 1997.

12 Omer Gold and Micha Sharir. Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. *ACM Transactions on Algorithms*, 14(4):50:1–50:17, 2018.

13 Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for closest string and related problems. *Algorithmica*, 37(1):25–42, 2003.

14 William Kuszmaul. Dynamic time warping in strongly subquadratic time: Algorithms for the low-distance regime and approximate evaluation. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP '19)*, pages 80:1–80:15, 2019.

15 Ming Li, Bin Ma, and Lusheng Wang. On the closest string and substring problems. *Journal of the ACM*, 49(2):157–171, 2002.

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 '16)*, pages 350–359, 2016.

17 Naomi Nishimura and Narges Simjour. Enumerating neighbour and closest strings. In *7th International Symposium on Parameterized and Exact Computation, (IPEC '12)*, pages 252–263. Springer, 2012.

18 H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978.

19 Anooshiravan Sharabiani, Houshang Darabi, Samuel Harford, Elnaz Douzali, Fazle Karim, Hereford Johnson, and Shun Chen. Asymptotic dynamic time warping calculation with utilizing value repetition. *Knowledge and Information Systems*, 57(2):359–388, 2018.

20 Shota Yuasa, Zhi-Zhong Chen, Bin Ma, and Lusheng Wang. Designing and implementing algorithms for the closest string problem. *Theoretical Computer Science*, 786:32–43, 2019.