Check for
updates

Fast Exact Dynamic Time Warping on Run-Length Encoded Time Series

Vincent Froese[1] · **Brijnesh Jain[2]** · **Maciej Rymar[1]** · **Mathias Weller[3]**

Abstract

Dynamic Time Warping (DTW) is a well-known similarity measure for time series. The standard dynamic programming approach to compute the DTW distance of two length-n time series, however, requires $O(n^2)$ time, which is often too slow for real-world applications. Therefore, many heuristics have been proposed to speed up the DTW computation. These are often based on lower bounding techniques, approximating the DTW distance, or considering special input data such as binary or piecewise constant time series. In this paper, we present a first exact algorithm to compute the DTW distance of two run-length encoded time series whose running time only depends on the encoding lengths of the inputs. The worst-case running time is cubic in the encoding length. In experiments we show that our algorithm is indeed fast for time series with short encoding lengths.

Keywords Time series similarity · Sparse data · Block matrix · Line intersections

✉ Vincent Froese
vincent.froese@tu-berlin.de

Brijnesh Jain
brijnesh.jain@oth-regensburg.de

Mathias Weller
mathias.weller@u-pem.fr

[1] Algorithmics and Computational Complexity Institute of Software Engineering and Theoretical Computer Science, Technische Universität Berlin, Berlin, Germany

[2] Department of Computer Science and Mathematics, OTH Regensburg, Regensburg, Germany

[3] CNRS, LIGM, Université Paris Est, Marne-La-Vallée, France

1 Introduction

Time series data is ubiquitous appearing in essentially all scientific domains. Comparing time series requires a measure to determine the similarity of two time series. Dynamic Time Warping (DTW) [24] is an established method which is used in numerous time series mining applications [1, 4, 6, 27].

The quadratic time complexity is often considered to be a major drawback of DTW on very long time series. In general there is not much hope to find strongly subquadratic algorithms since it has been shown (assuming the Strong Exponential Time Hypothesis[1]) that DTW cannot be computed in $O(n^{2-\epsilon})$ time for any $\epsilon > 0$ [2, 7] even on time series over an alphabet of size three [21]. However, there exist sophisticated pruning and lower-bounding techniques which run fast in practice [26]. Long time series of length $n \gg 10,000$ occur, for example, when measuring electrical power of household appliances with a sampling rate of a few seconds collected over several months, twitter activity data sampled in milliseconds, and human activities inferred from a smart home environment [23]. All these time series have in common that they contain long (nearly) constant segments.

Recently, several algorithms have been devised to cope with long time series that contain constant segments (called *runs*) [12, 15–17, 23, 25]. The basic idea of these algorithms is to exploit the repetitions of values within a time series to speed up computation of the DTW distance. We briefly summarize some of these algorithms (see also Table 1).

- AWarp [23]: This algorithm is exact for binary time series (a formal proof is missing) and exploits repetitions of zeros. The running time is $O(m_1 m_2)$, where m_1 and m_2 are the numbers of non-zero entries in the two input time series.
- Sparse DTW (SDTW) [15]: This algorithm yields exact DTW distances for arbitrary time series in $O((m_1 + m_2)n)$ time, where m_1 and m_2 are the numbers of non-zero entries in the two input series (assuming both have length n).
- Binary Sparse DTW (BSDTW) [17]: This algorithm computes exact DTW distances between two binary time series in $O(m_1 m_2)$ time, where m_1 and m_2 are the numbers of non-zero entries in the two input time series. In practice it is often faster than AWarp.
- Blocked DTW (BDTW) [25] (earlier introduced as Coarse-DTW [12]): This algorithm operates on run-length encoded time series. The run-length encoding represents a run of identical values (constant segment) by storing only a single value together with the length of the run. BDTW yields an upper and a lower bound on the DTW distance and is exact on binary time series (a formal proof is missing). The running time is $O(k\ell)$, where k and ℓ are the numbers of runs in the two input time series (note that $k\ell \in O(m_1 m_2)$). BDTW is faster than AWarp in practice.

Clearly, AWarp, BDTW and BSDTW are limited in that they only yield exact DTW distances for binary time series. There are several recent (theoretical) results regarding exact DTW computation. Abboud et al. [2] gave an algorithm which computes exact

---

[1] The SETH asserts that SAT cannot be solved in $(2 - \epsilon)^n \cdot (n + m)^{O(1)}$ time for any $\epsilon > 0$, where n is the number of variables and m is the number of clauses [18].

Table 1 Overview of some DTW algorithms and their characteristics. n: maximum input length, m_1, m_2: number of non-zero entries in inputs, k, ℓ: number of runs in inputs

Algorithm	Running time	Domain	Exactness
AWarp [23]	$O(m_1 m_2)$	arbitrary	binary
SDTW [15]	$O((m_1 + m_2)n)$	arbitrary	arbitrary
BSDTW [17]	$O(m_1 m_2)$	binary	binary
BDTW [12, 25]	$O(k\ell)$	arbitrary	binary

DTW distances on binary length-n time series in $O(n^{1.87})$ time. This was recently improved to linear time by Kuszmaul [22]. Gold and Sharir [14] showed a subquadratic $O(n^2 \log \log \log n / \log \log n)$-time algorithm for arbitrary time series and Kuszmaul [21] developed an $O(n \cdot \text{dtw}(x, y))$-time algorithm assuming that the minimum non-zero local cost is one.

Notably, specialized algorithms for other string problems on run-length encoded strings have also been studied recently, for example, for Longest Common Subsequence [5, 28] and Edit Distance [9, 10], which have applications in sequence alignment in bioinformatics.

Our Contributions. We develop an algorithm that computes exact DTW distances for arbitrary run-length encoded time series. Let x and y be two time series of length m and n, where x contains k runs and y contains ℓ runs. Then, our algorithm (Theorem 1) computes the DTW distance in $O(\kappa)$ time,[2] where κ is a number depending on the individual lengths of the runs in x and y (see Sect. 3 for details). For κ, the following upper bound holds:

$$\kappa \in \begin{cases} O(k^2\ell + k\ell^2) : & \text{if } k \in O(\sqrt{m}) \text{ and } \ell \in O(\sqrt{n}) \\ O(kn + \ell m) : & \text{otherwise} \end{cases}.$$

That is, the running time is at most cubic in $\max(k, \ell)$ and is asymptotically faster than $O(mn)$ if $k \in o(m)$ and $\ell \in o(n)$. To the best of our knowledge, this is the first exact algorithm whose running time only depends on the lengths of the run-length encodings of the inputs.

In addition, we show that if all runs in both time series have the same length, then our algorithm even runs in $O(k\ell)$ time (Corollary 2) and is in fact equivalent to BDTW. That is, we prove that BDTW is exact in this case.

In experiments we compare our algorithm with the previously mentioned alternatives (Table 1) and show that it is indeed the fastest exact algorithm on time series with short run-length encodings.

2 Preliminaries

We give some preliminary definitions and introduce notation.

---

[2] Throughout this work we neglect running times for arithmetical operations.

Notation. Let $[n] := \{1, \ldots, n\}$ and $[a, b] := \{a, a + 1, \ldots, b\}$. An $m \times n$ table T consists of m rows and n columns, where $T[i, j]$ denotes the entry in the i-th row and j-th column.

Time Series. A time series is a finite sequence $x = (x_1, \ldots, x_n)$ of rationals. The *run-length encoding* of x is the sequence $\tilde{x} = ((\tilde{x}_1, n_1), \ldots, (\tilde{x}_k, n_k))$ of pairs $(\tilde{x}_i, n_i)$ where n_i is a positive integer denoting the number of consecutive repetitions (run length) of the value $\tilde{x}_i$ in x. Note that $\sum_{i=1}^{k} n_i = n$. We call n the *length* of x and we call k the *coding length* of x.

Dynamic Time Warping. The dynamic time warping distance is a distance measure between time series using non-linear alignments defined by warping paths [24].

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 $\ell \in [L - 1]$.

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 time series $x = (x_1, \ldots, x_m)$ and $y = (y_1, \ldots, y_n)$ in the following way: A pair $(i, j) \in p$ *aligns* element x_i with y_j incurring a local cost of $(x_i - y_j)^2$. The *cost* of a warping path p is $C(p) = \sum_{(i,j) \in p} (x_i - y_j)^2$. The DTW distance between x and y is defined as

$$\mathrm{dtw}(x, y) := \min_{p \in \mathcal{P}_{m,n}} \sqrt{C(p)}.$$

It can be computed via dynamic programming in $O(mn)$ time based on an $m \times n$ table [24].

3 The Algorithm

In the following, let $x = (x_1, \ldots, x_m)$ and $y = (y_1, \ldots, y_n)$ be two time series with $\tilde{x} = ((\tilde{x}_1, m_1), \ldots, (\tilde{x}_k, m_k))$ and $\tilde{y} = ((\tilde{y}_1, n_1), \ldots, (\tilde{y}_\ell, n_\ell))$. We define $a_0 := 0$, $a_i := \sum_{j=1}^{i} m_j$ for $i \in [k]$ and $b_0 := 0$, $b_i := \sum_{j=1}^{i} n_j$ for $i \in [\ell]$. Consider the $m \times n$ DTW matrix D, where

$$D[i, j] = \mathrm{dtw}((x_1, \ldots, x_i), (y_1, \ldots, y_j))^2.$$

Note that D can be structured into $k\ell$ blocks $B_{i,j} = [a_{i-1} + 1, a_i] \times [b_{j-1} + 1, b_j]$, $i \in [k]$, $j \in [\ell]$, where each step inside $B_{i,j}$ has local cost $c_{i,j} := (\tilde{x}_i - \tilde{y}_j)^2$. The *right* boundary of $B_{i,j}$ corresponds to column b_j of D and the *top* boundary is formed by row a_i of D (see Fig. 1).

We show that it is sufficient to compute only certain entries on the boundaries of blocks instead of all mn entries in D. To this end, we analyze the structure of optimal warping paths. We begin with the following simple observation.
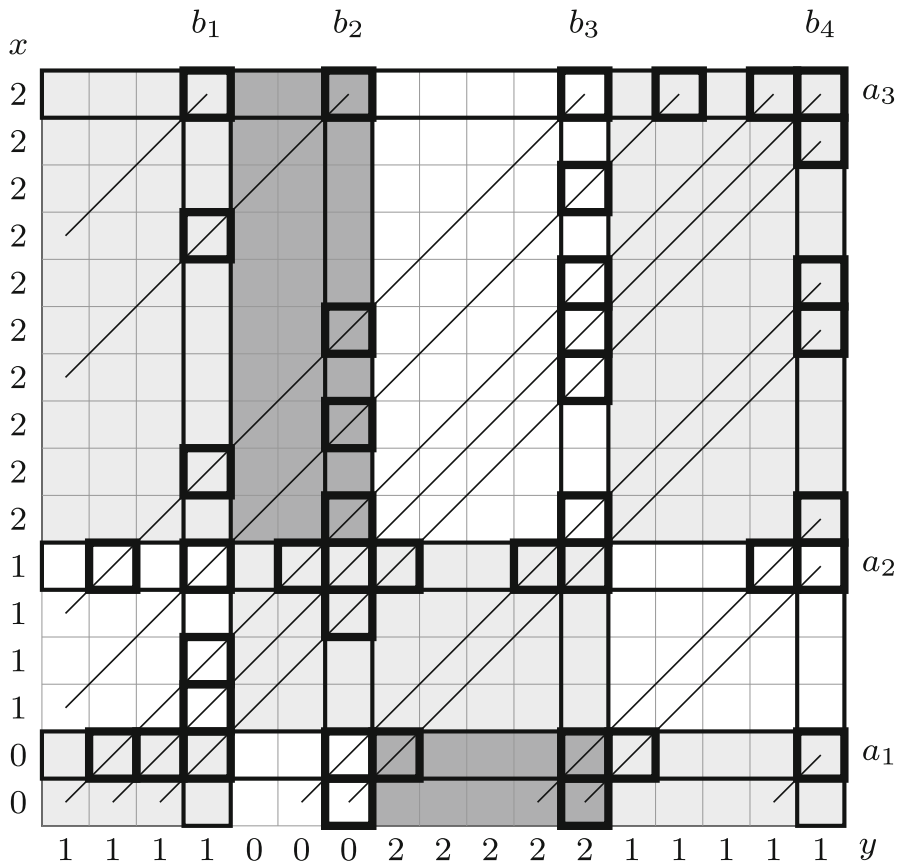
Fig. 1 Example of a DTW matrix for two time series x and y with run-length encodings $\tilde{x} = ((0, 2), (1, 4), (2, 10))$ and $\tilde{y} = ((1, 4), (0, 3), (2, 5), (1, 5))$. Colors indicate the local costs $(x_i - y_j)^2$ of blocks (white = 0, light gray = 1, dark gray = 4). It is sufficient to compute the bold-framed entries in order to determine dtw(x, y) since there exists an optimal warping path moving only along boundaries of blocks (rows a_1, a_2, a_3 and columns b_1, b_2, b_3, b_4) and the indicated block diagonals $\mathcal{L}$

Observation 1 *There exists an optimal warping path p such that the following holds for every block B: If p moves through B, then p first moves diagonally through B until it reaches a boundary of B.*

This is true since every step inside a block costs the same. Hence, it is optimal to maximize the number of diagonal steps (which minimizes the overall number of steps to reach a boundary of a block). Observation 1 implies that there exists an optimal warping path which is an alternation of diagonal and horizontal (or vertical) subpaths where the horizontal (vertical) subpaths are always on top (right) boundaries of blocks. Note that this implies an easy $O(kn + \ell m)$-time algorithm which only computes the entries on the boundaries via dynamic programming.

Now, we restrict the possible diagonals along which such an alternating optimal warping path might move. To this end, let $L_{i,j}$, $(i, j) \in [k] \times [\ell]$, denote the

diagonal in D going through the upper right corner of block $B_{i,j}$ (that is, through the entry (a_i, b_j)) and let $L_{0,0}$ be the diagonal (corresponding to (a_0, b_0)) going through $(1, 1)$. We denote the set of all these *block diagonals* by $\mathcal{L}$ (see Fig. 1). Now, our key lemma states that there always exists an optimal warping path which only moves along block boundaries and block diagonals (we call such a warping path *diagonal-conform*).

Lemma 1 *There exists an optimal warping path which is diagonal-conform.*

Proof By definition, every warping path initially starts in $(1, 1)$ on the diagonal $L_{0,0} \in \mathcal{L}$. Let p be an optimal warping path which alternates between diagonals and block boundaries as described in Observation 1. Assume that p does not only move along diagonals in $\mathcal{L}$. Then, by assumption, p leaves some diagonal $L \in \mathcal{L}$ on a boundary (wlog horizontally on the top boundary a_i) of a block $B_{i,j}$ and (diagonally) enters the neighboring block $B_{i+1,j}$ before the next intersection of a diagonal $L' \in \mathcal{L}$ with a_i. It then proceeds diagonally in between L and L' until reaching some block boundary where it moves horizontally or vertically again. Note that p has to move horizontally or vertically again at some point since it has to reach a diagonal in $\mathcal{L}$ again (this holds because every warping path eventually ends up on $L_{k,\ell} \in \mathcal{L}$). Assume that p moves diagonally only until reaching the top boundary $a_{i'}$ of a block $B_{i',j'}$, $i' > i$, $j' \geq j$, where p moves horizontally (analogous arguments apply if p moves vertically on a right boundary of a block in between L and L'). See Fig. 2 for an example. Observe that a warping path can only enter blocks from bottom (that is, from the top boundary of the block below) or left (that is, from the right boundary of the block to the left) and exit blocks from top or right boundaries.

Let $h_i \geq 1$ denote the number of horizontal steps of p on a_i and let $h_{i'} \geq 1$ be the number of horizontal steps on $a_{i'}$. Let q denote the diagonal subpath of p from a_i to $a_{i'}$. Now, consider the warping path p' obtained from p by "shifting" q to the right, that is, p' takes $h_i + 1$ horizontal steps on a_i and only $h_{i'} - 1$ horizontal steps on $a_{i'}$. Let q' be the shifted diagonal subpath and note that q' crosses the same blocks as q. This is true since there cannot be an upper right corner of any block anywhere in the region between L and L' (since they are neighboring diagonals from $\mathcal{L}$).

Let us now consider the number of steps taken by p' within each block from $B_{i,j}$ to $B_{i',j'}$. Clearly, p' takes one more step inside $B_{i,j}$ than p. Regarding $B_{i',j'}$, if q enters $B_{i',j'}$ from bottom, then q' takes one step less inside $B_{i',j'}$. Otherwise, if q enters $B_{i',j'}$ from the left, then q' takes the same number of steps inside $B_{i',j'}$ as q. For every block B in between $B_{i,j}$ and $B_{i',j'}$ which is crossed by q, the following holds:

– If q crosses B from left to top, then q' takes one more step.
– If q crosses B from bottom to right, then q' takes one step less.
– If q crosses B from bottom to top (or from left to right), then q' takes the same number of steps.

The above holds since q cannot pass through an upper right corner of a block in between L and L'. Note that the number of steps taken by p and p' through any block differs by at most one.

Now, let $\mathcal{B}$ be the set of blocks where p takes more steps than p' and let $\mathcal{B}'$ be the set of blocks where p' takes more steps than p. Let $C = \sum_{B_{i,j} \in \mathcal{B}} c_{i,j}$ and $C' =$
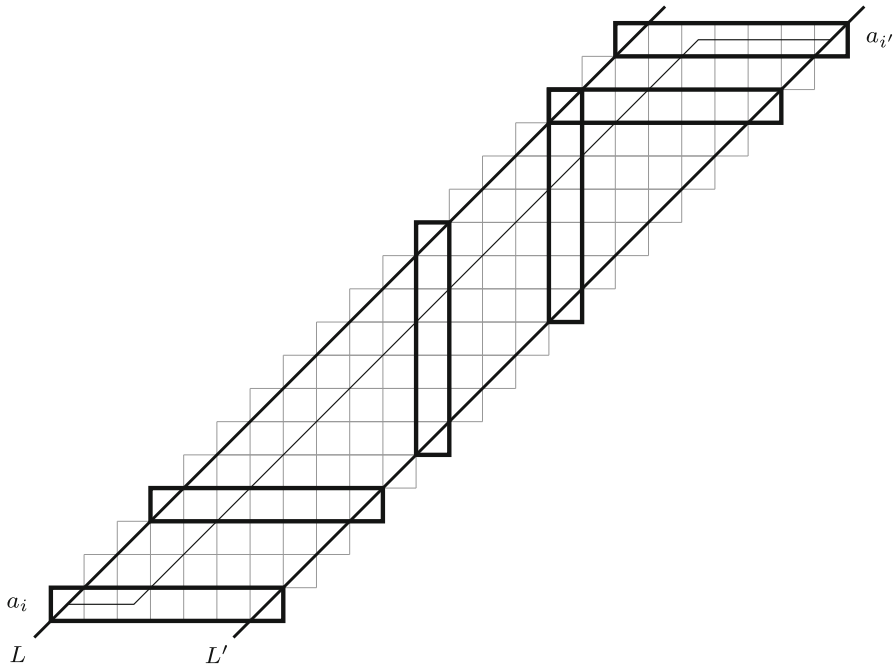
Fig. 2 Example of a warping path moving diagonally in between two neighboring diagonals L and L'. Block boundaries are framed in thick lines. Note that there cannot be an upper right block corner anywhere in between L and L'. Hence, when shifting the warping path to the right from L to L', the cost changes monotonically (linearly)

$\sum_{B_{i,j} \in \mathcal{B}'} c_{i,j}$. Then, the cost difference between p and p' is $C - C'$. By optimality of p, we have $C - C' \leq 0$, that is, $C \leq C'$.

If $C = C'$, then also p' is an optimal warping path. Thus, by analogous arguments, shifting $h_{i'}$ times to the right yields an optimal warping path that does not move horizontally on $a_{i'}$ anymore. If this warping path now already moves diagonally along L' (as it would be the case in Fig. 2 when shifting four times to the right), then this proves the claim. If this is not case, then analogous arguments apply again for the next occurrence of a horizontal (or vertical) subpath in between L and L'. This finally yields an optimal warping path moving along L' (or L) proving the claim.

If $C < C'$, then we can analogously shift q to the left to obtain a warping path p''. Clearly, the blocks where p'' takes one more step than p are exactly the blocks $\mathcal{B}$, and the blocks where p takes one more step than p'' are exactly the blocks $\mathcal{B}'$. Hence, the cost difference between p'' and p is also $C - C' < 0$, which contradicts the optimality of p.   □

Clearly, an optimal diagonal-conform warping path can be computed from only those entries in D which are an intersection of a block boundary and a block diagonal in $\mathcal{L}$ (in Fig. 1 these intersections are framed in bold). In the following, we denote the number of these intersections by κ. Note that

$$kℓ \leq \kappa \leq (k + ℓ)|\mathcal{L}| \leq (k + ℓ)(kℓ + 1),$$

that is, $\kappa \in O(k^2\ell + k\ell^2)$. We need to compute optimal diagonal-conform warping paths to these intersections. From the proof of Lemma 1, we can actually infer the following corollary about optimal diagonal-conform warping paths to any intersection.

Corollary 1 *Let $B_{i,j}$ be a block and consider an intersection z of its top or right boundary with a diagonal $L \in \mathcal{L}$. There is an optimal diagonal-conform warping path to z whose diagonal subpaths are only on diagonals from $\{L\} \cup \{L_{i',j'} \mid i' \leq i, j' \leq j\}$.*

Corollary 1 essentially follows from the same shifting argument as in the proof of Lemma 1. Consider an optimal diagonal-conform warping path to z that contains a diagonal subpath q on a block diagonal $L_{i',j'} \neq L$, where $i' > i$ or $j' > j$. Note that we can actually shift the diagonal subpath q (without increasing the cost) until it lies on L or goes through an upper right corner of some block, that is, the shifted subpath is on the diagonal of this block. Clearly, this is a block B_{i^*,j^*} with $i^* \leq i$ and $j^* \leq j$.

We are now ready to prove our main result.

Theorem 1 *The DTW distance between time series x and y can be computed from $\tilde{x}$ and $\tilde{y}$ in $O(\kappa)$ time, where κ is the number of intersections between block boundaries and block diagonals in the DTW matrix.*

Proof The algorithm builds an optimal diagonal-conform warping path "block-by-block" via dynamic programming (iterating over blocks $B_{i,j}$ for $i = 1, \ldots, k$ and $j = 1, \ldots, \ell$) using optimal diagonal-conform warping paths to intersections of block boundaries with block diagonals (see algorithm 1 for the pseudocode). Whenever a block $B_{i,j}$ is added, the corresponding block diagonal $L_{i,j}$ is inserted (if it does not already exist) in a sorted doubly-linked list (diagonals) of previously encountered block diagonals. Then, the costs of optimal diagonal-conform warping paths to all intersections of previously encountered diagonals with the boundaries of $B_{i,j}$ are computed (using appendentry) as well as the costs for the intersections of $L_{i,j}$ with the boundaries of blocks $B_{i',j'}$, $i' \leq i$, $j' \leq j$ (trace). Before we prove correctness, we introduce some preliminary definitions.

In our algorithm, a diagonal $L_{i,j} \in \mathcal{L}$ (going through the upper right corner of block $B_{i,j}$) is a sorted list of its intersections with block boundaries. The *offset* of $L_{i,j}$ is $b_j - a_i$. We define a linear order on diagonals as follows: $L_{i,j}$ is "to the left of" $L_{i',j'}$ (denoted $L_{i,j} < L_{i',j'}$) if and only if $b_j - a_i < b_{j'} - a_{i'}$, that is, its offset is smaller.

For the correctness, we show that after a block $B_{i,j}$ is handled, all intersections between block boundaries and block diagonals of blocks $B_{i',j'}$ with $i' \leq i$ and $j' \leq j$ are correctly determined and stored on the corresponding diagonals (sorted with increasing row and column indices) together with the cost of an optimal diagonal-conform warping path.

To this end, consider block $B_{i,j}$ and assume that for all previous blocks $B_{i',j'}$ with $i' < i$ or $j' < j$ the above claim holds (this is trivially true before the first block $B_{1,1}$ is handled). Moreover, we assume that diagonals is sorted with increasing offset (which initially holds before line 11, where it only contains the diagonals $-\infty$, $L_{0,0}$, and ∞ in that order). Note that, by Corollary 1, we only need to consider new

---

Algorithm 1: Exact DTW for run-length encoded time series.

Input: Run-length encodings $((\tilde{x}_1, m_1), \ldots, (\tilde{x}_k, m_k))$ and $((\tilde{y}_1, n_1), \ldots, (\tilde{y}_\ell, n_\ell))$ of time series x and y.

Output: DTW distance between x and y.

1 **foreach** $(i, j) \in [k] \times [\ell]$ **do** $c_{i,j} := (\tilde{x}_i - \tilde{y}_j)^2$;   // compute local block costs
2 $a_0 := 0$
3 **foreach** $i \in [k]$ **do** $a_i := a_{i-1} + m_i$;   // compute indices of top boundaries
4 $b_0 := 0$
5 **foreach** $j \in [\ell]$ **do** $b_j := b_{j-1} + n_j$;   // compute indices of right boundaries
6 diagonals $\leftarrow$ doubly-linked list of diagonals
7 add dummy diagonal $-\infty$ with offset $-\infty$ containing entry $(-\infty, -\infty)$ with cost ∞
8 add dummy diagonal ∞ with offset ∞ containing entry (∞, ∞) with cost ∞
9 insert an empty diagonal $L_{0,0}$ with offset 0 between $-\infty$ and ∞
10 add entry $(0, 0)$ with cost 0 to $L_{0,0}$
11 **foreach** $i \in [k]$ **do**
12 　$L \leftarrow$ first diagonal in diagonals   // $L = -\infty$ with offset $-\infty$
13 　**foreach** $j \in [\ell]$ **do**
14 　　**if** $L \leq L_{i,j-1}$ **then** $L \leftarrow$ diagonals.next(L)
15 　　**while** $L < L_{i,j}$ **do**   // diagonals intersecting top boundary of $B_{i,j}$
16 　　　appendentry(L, i, j)
17 　　　$L \leftarrow$ diagonals.next(L)
18 　　$L' \leftarrow L$
19 　　**while** $L' < L_{i-1,j}$ **do** $L' \leftarrow$ diagonals.next(L')
20 　　$L' \leftarrow$ diagonals.previous(L')
21 　　**while** $L' > L_{i,j}$ **do** // diagonals intersecting right boundary of $B_{i,j}$
22 　　　appendentry(L', i, j)
23 　　　$L' \leftarrow$ diagonals.previous(L')
24 　　**if** $L > L_{i,j}$ **then**   // insert new diagonal $L_{i,j}$
25 　　　insert empty diagonal $L_{i,j}$ with offset $b_j - a_i$ into diagonals before L
26 　　　trace$(L_{i,j}, i, j,$ last entry on diagonals.previous$(L_{i,j}),$ last entry on $L)$
27 　　**else** appendentry(L, i, j);   // diagonal $L_{i,j}$ exists already
28 **return** cost of last computed entry

---

intersections, that is, intersections of previous block diagonals with the boundaries of $B_{i,j}$ and intersections of $L_{i,j}$ with previously handled block boundaries (if $L_{i,j}$ does not yet exist). For all other previously computed intersections, there exists an optimal diagonal-conform warping path which does not use $L_{i,j}$, hence, we do not need to update them.

As regards the intersections on the boundaries of $B_{i,j}$, observe that a diagonal L intersects the top boundary a_i if $L_{i,j-1} < L \leq L_{i,j}$. If this is the case, then clearly the intersection is $(a_i, a_i + \sigma)$, where σ is the offset of L. Now, by definition, there are two options for a diagonal-conform warping path to reach this intersection: either diagonally on L (from the last intersection stored on L) or from the left on the boundary a_i. For the latter option, a diagonal-conform warping path has to go over the intersection of the diagonal that is directly to the left of L (that is, the predecessor of L in diagonals) with a_i. By assumption, this intersection is the last one stored on the predecessor of L in diagonals. The optimum of these two cases can easily be determined (see minimum computation in appendentry which is called in line 16 of algorithm 1). The intersections on the right boundary of $B_{i,j}$ are handled analogously

---

Function appendentry(L, i, j)

Input: A diagonal L and block indices i, j such that L intersects a boundary of $B_{i,j}$.
Output: Compute intersection of L and the boundary of $B_{i,j}$ and add this entry to L with the cost of an optimal diagonal-conform warping path.

$z_L \leftarrow$ last entry on L
$(a, b) \leftarrow (a_i, b_j)$
$c \leftarrow \infty$
if $L \leq L_{i,j}$ **then**   // L intersects top boundary of $B_{i,j}$
  $\quad b \leftarrow a_i + \mathsf{offset}(L)$ // column of intersection
  $\quad z' \leftarrow$ last entry on $\mathsf{diagonals.previous}(L)$
  $\quad c \leftarrow \min\{z'.\mathsf{cost} + c_{i,j} \cdot (b - z'.\mathsf{col}), z_L.\mathsf{cost} + c_{i,j} \cdot (b - z_L.\mathsf{col})\}$
if $L \geq L_{i,j}$ **then**   // L intersects right boundary of $B_{i,j}$
  $\quad a \leftarrow b_j - \mathsf{offset}(L)$ // row of intersection
  $\quad z' \leftarrow$ last entry on $\mathsf{diagonals.next}(L)$
  $\quad c \leftarrow \min\{c, z'.\mathsf{cost} + c_{i,j} \cdot (a - z'.\mathsf{row}), z_L.\mathsf{cost} + c_{i,j} \cdot (a - z_L.\mathsf{row})\}$
add (a, b) with cost c to the end of L

Function trace(L, i, j, z_p, z_n)

Input: A diagonal L, block indices i, j such that L intersects the boundary of $B_{i,j}$, and entries z_p on the previous and z_n on the next diagonal of L.
Output: Cost of an optimal diagonal-conform warping path to the intersection of L with a boundary of $B_{i,j}$. Recursively fills the diagonal L with all intersections of L with previous block boundaries.

$L_p \leftarrow \mathsf{diagonals.previous}(L)$
$L_n \leftarrow \mathsf{diagonals.next}(L)$
while $z_p \neq \bot$ **and** $z_p.\mathsf{row} > a_i$ **do** $z_p \leftarrow L_p.\mathsf{previous}(z_p)$
while $z_n \neq \bot$ **and** $z_n.\mathsf{col} > b_j$ **do** $z_n \leftarrow L_n.\mathsf{previous}(z_n)$
if $z_p \neq \bot$ **and** $z_n \neq \bot$ **and** $i, j \geq 1$ **then**
  $\quad (a, b) \leftarrow (a_i, b_j)$
  $\quad c \leftarrow \infty$
  $\quad$**if** $L \leq L_{i,j}$ **then**   // L intersects top boundary a_i
  $\quad\quad b \leftarrow a_i + \mathsf{offset}(L)$
  $\quad\quad c \leftarrow \min(c, z_p.\mathsf{cost} + c_{i,j} \cdot (b - z_p.\mathsf{col}))$
  $\quad$**if** $L \geq L_{i,j}$ **then**   // L intersects right boundary b_j
  $\quad\quad a \leftarrow b_j - \mathsf{offset}(L)$
  $\quad\quad c \leftarrow \min(c, z_n.\mathsf{cost} + c_{i,j} \cdot (a - z_n.\mathsf{row}))$
  $\quad$**if** $L \geq L_{i-1,j-1}$ **then**   // L intersects top boundary a_{i-1} of $B_{i-1,j}$
  $\quad\quad c \leftarrow \min(c, \mathsf{trace}(L, i-1, j, z_p, z_n) + c_{i,j} \cdot (a - a_{i-1}))$
  $\quad$**else**   // L intersects right boundary b_{j-1} of $B_{i,j-1}$
  $\quad\quad c \leftarrow \min(c, \mathsf{trace}(L, i, j-1, z_p, z_n) + c_{i,j} \cdot (b - b_{j-1}))$
  $\quad$add (a, b) with cost c to the end of L
  $\quad$**return** c
else return ∞

---

in line 22 (using the successor of L in $\mathsf{diagonals}$). Note that if there already exists a diagonal with the same offset as $L_{i,j}$, then its intersection with the boundary of $B_{i,j}$ (which is the upper right corner of $B_{i,j}$) is added in line 27.

If $L_{i,j}$ does not yet exist, then it is newly inserted into $\mathsf{diagonals}$ in line 25 before the first diagonal in $\mathsf{diagonals}$ with a larger offset. Hence, $\mathsf{diagonals}$ is correctly sorted. Then, all intersections of $L_{i,j}$ with block boundaries are recursively

added via trace in line 26. This is done as follows: Consider an intersection of $L_{i,j}$ with a boundary of a block $B_{i',j'}$, $i' \leq i$, $j' \leq j$. Again, by definition, an optimal diagonal-conform warping path only has the options to reach this intersection via $L_{i,j}$ or via the boundary. For the boundary option, we can again use the previously computed intersections on the neighboring diagonals of $L_{i,j}$ in diagonals. For the diagonal option, we need to compute the preceding intersection of $L_{i,j}$ with a previous block boundary first. This is done recursively. Note that the previous intersection of $L_{i,j}$ is on the top boundary of $B_{i'-1,j'}$ if $L_{i,j} > L_{i'-1,j'-1}$, and it is on the right boundary of $B_{i',j'-1}$ if $L < L_{i'-1,j'-1}$ (note that $L_{i,j} = L_{i'-1,j'-1}$ is not possible since $L_{i,j}$ is a new diagonal). Moreover, this intersection can easily be determined (as described above) and an optimal diagonal-conform warping path to this intersection can again be determined using only the neighboring diagonals of $L_{i,j}$ in diagonals. The recursion terminates when there exists no intersection of $L_{i,j}$ with a previous block boundary (that is, the border of the DTW matrix D is reached). In this case, a diagonal-conform warping path to the current intersection can only come from the corresponding boundary. If there is no intersection on this boundary with one of the neighboring diagonals of $L_{i,j}$, then this intersection cannot be reached by any diagonal-conform warping path. Hence, its cost can be set to ∞. This completes the correctness of algorithm 1.

For the running time, note that each intersection is computed exactly once (either by appendentry or by trace). Moreover, the computation required to handle a single intersection takes constant time. Thus, the overall running time is linear in the total number κ of intersections.   □

As regards the value of κ, note that $\kappa \leq kn + \ell m - k\ell$ clearly holds since this is the overall number of entries on all block boundaries. Hence, a (tight) worst-case upper bound is

$$\kappa \in O(\min(k^2\ell + k\ell^2, kn + \ell m)).$$

In practice, κ might be smaller since not every block diagonal will intersect every boundary (depending on the specific block sizes) and some block diagonals might even be identical (for example, if square blocks appear). Such beneficial block sizes can be achieved, for example, when using piecewise aggregate approximation [20, 29] as preprocessing where the time series are approximated by piecewise constant series with a fixed run length. For the case that all blocks have equal sizes, the following improved upper bound on κ holds.

Lemma 2 *Let x and y be two time series such that x consists of k runs of length m' and y consists of ℓ runs of length n', where $n' \leq m'$. Then, the number κ of intersections between block diagonals and block boundaries is in $O(k\ell \cdot M/n')$, where M is the least common multiple of m' and n'.*

Proof Let $m = km'$ be the length of x and $n = \ell n'$ be the length of y. Let M be the least common multiple of m' and n' and let $\alpha = M/m'$ and $\beta = M/n'$. Clearly, for every $\alpha < i \leq k$ and $\beta < j \leq \ell$, the block diagonal $L_{i,j}$ is the same diagonal as $L_{i-\alpha,j-\beta}$. Thus, the set $\mathcal{L}$ of block diagonals can be written as

$$\mathcal{L} = \mathcal{A} \cup \mathcal{B} \cup \{L_{0,0}\},$$

where $\mathcal{A} = \{L_{i,j} \mid i \in [\alpha], j \in [\ell]\}$ and $\mathcal{B} = \{L_{i,j} \mid i \in [k], j \in [\beta]\}$.

Let us consider the intersections of boundary a_i with a diagonal $L_{i',j} \in \mathcal{L}$. There are two cases: For $i < i'$, there exists an intersection if $b_j - (i' - i)m' \geq 1$. For $i \geq i'$, there exists an intersection if $b_j + (i - i')m' \leq n$. Since $m' \geq n'$, boundary a_i can thus only have intersections with diagonals $L_{i',j}$ where $i - \ell \leq i' \leq i + \ell$. Hence, there are at most $2\ell \cdot \beta$ intersections with diagonals in $\mathcal{B}$ and at most $\alpha \cdot \ell$ intersections with diagonals in $\mathcal{A}$ on a_i. Overall, there are at most $k\ell(2\beta + \alpha) \leq k\ell \cdot 3\beta$ intersections on all top boundaries.

Analogously, for boundary b_j, there exists an intersection with $L_{i,j'} \in \mathcal{L}$ if $a_i - (j' - j)n' \geq 1$ (for $j' > j$) or if $a_i + (j - j')n' \leq m$ (for $j \geq j'$). Thus, there are at most $\beta \cdot k$ intersections with diagonals in $\mathcal{B}$ and at most $\alpha \cdot m/n'$ intersections with diagonals in $\mathcal{A}$ on b_j. This yields at most $k\ell(\beta + \alpha \cdot m'/n') = k\ell \cdot 2\beta$ intersections on all right boundaries. Thus, altogether there are at most $O(k\ell \cdot M/n')$ many intersections. $\square$

Note that if $M \in O(n')$ holds in Lemma 2 (for example, if $m' = \alpha n'$ for a constant integer $\alpha \geq 1$), then this implies $\kappa \in O(k\ell)$. Hence, we obtain the following.

Corollary 2 *Let x and y be two time series such that x consists of k runs of length m' and y consists of ℓ runs of length $n' \leq m'$. If the least common multiple of m' and n' is in $O(n')$, then the DTW distance between x and y can be computed from $\tilde{x}$ and $\tilde{y}$ in $O(k\ell)$ time.*

If $m' = n'$ (that is, all blocks are squares), then there are $\kappa = k\ell$ intersections which are exactly the upper right block corners. In this special case the following holds: If an optimal warping path moves through a block $B_{i,j}$, then it takes exactly m' steps through $B_{i,j}$ without loss of generality. The algorithm Blocked_DTW_UB [25, Algorithm 1] (and accordingly also Coarse-DTW [12, Algorithm 2] with $\phi_{\max}$) uses the value $\max(m', n') \cdot c_{i,j}$ (which clearly equals $m' \cdot c_{i,j}$) for the cost of crossing block $B_{i,j}$. Hence, these algorithms are equivalent to our algorithm in this case. That is, we proved the following.

Corollary 3 Blocked DTW *[25]* and Coarse-DTW *[12]* are exact if all blocks are squares.

4 Experiments

We conducted experiments to empirically evaluate our algorithm comparing it to alternatives.

Data. We considered all seven datasets from the UCR repository [11] whose time series have a length of at least $n \geq 1000$ (time series within the same dataset have identical length). Table 2 lists the selected datasets and their characteristics.

Setup. We compared our run-length encoded DTW algorithm (RLEDTW) with the following alternatives[3] (see Table 1 for descriptions):

---

[3] C++ implementations are available at www.akt.tu-berlin.de/menue/software/.

Table 2 Characteristics of the datasets we used in our experiments. Type refers to the problem domain, size to the overall number of time series in the dataset, and length to the number of elements of a time series

dataset	type	size	length
HandOutlines	IMAGE	1370	2709
InlineSkate	MOTION	650	1882
CinCECGTorso	ECG	1420	1639
Haptics	MOTION	463	1092
Mallat	SIMULATED	2400	1024
StarLightCurves	SENSOR	9236	1024
Phoneme	SOUND	2110	1024
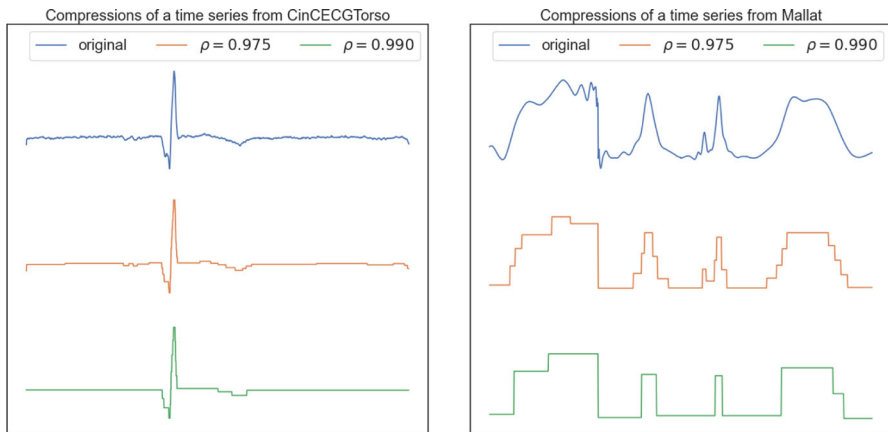
Fig. 3 Examples of compressed time series from CinCECGTorso (left) and Mallat (right)

- DTW (standard $O(n^2)$-time dynamic program) [24],
- AWarp [23],
- SDTW [15],
- BDTW [12, 25].

To compare the algorithms, we applied the following procedure: From each of the seven UCR datasets, we randomly sampled a subset $\mathcal{D}$ of 100 time series (of length n). Then, for a specified encoding length $k < n$, we transformed the subset $\mathcal{D}$ into a subset $\mathcal{D}^k$ by compressing the time series to consist of k runs. The compression is achieved by computing a best piecewise constant approximation with k constant segments minimizing the squared error (also called adaptive piecewise constant approximation). This can be done using dynamic programming [8, 13, 19]. The encoding length k was controlled by the space-saving ratio $\rho = 1 - k/n$. We used the space-saving ratios $\rho \in \{0.1, 0.5, 0.75, 0.9, 0.925, 0.95, 0.975, 0.99\}$. Thus, we generated eight compressed versions of each subset $\mathcal{D}$ in run-length encoded form (see Fig. 3 for examples of compressed time series). For every compressed dataset, we computed all pairwise DTW distances using the five different algorithms.

Results. Figure 4 shows the average speedup factors of the algorithms compared to the DTW baseline as a log-function of the space-saving ratio ρ. The speedup of an
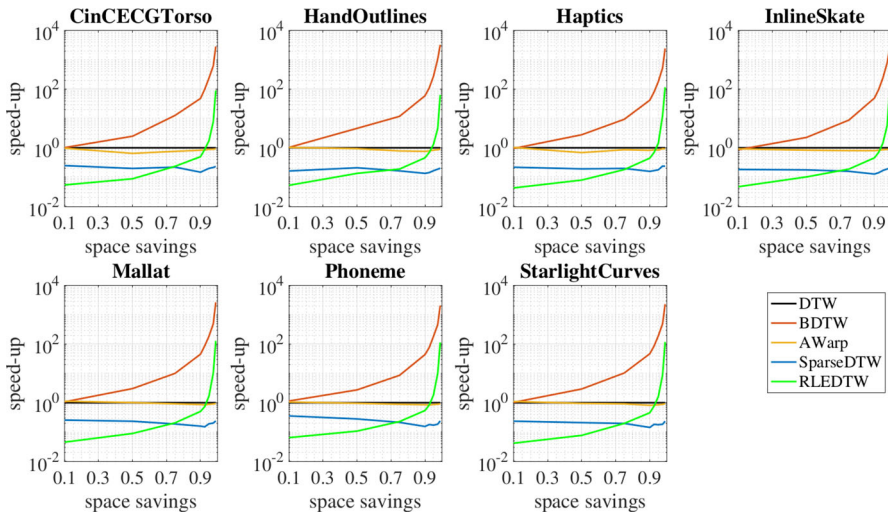
Fig. 4 Average speedup factor as a function of the space-saving ratio

algorithm A for computing a DTW distance between two time series is defined by $\sigma_A = t_{DTW}/t_A$, where t_A is the computation time of A and t_{DTW} is the computation time of the standard dynamic program. That is, for $\sigma_A > 1$ ($\sigma_A < 1$), algorithm A is faster (slower) than the baseline method.

The results show that the speedup factors of AWarp and SDTW are independent of the space-saving ratio and less than one. Hence, both algorithms are actually slower than standard dynamic programming. This is due to the fact that both algorithms have been designed for time series with runs of zeros. The results indicate that AWarp and SDTW are of limited use for the general case of time series having only few runs of zeros. In contrast, the speedup factors of the BDTW heuristic and our exact RLEDTW grow superexponentially with increasing space-saving ratio. For all but the smallest space-saving ratios, BDTW is faster than all other algorithms. In the best case, BDTW is up to more than 1000 times faster than DTW. Our algorithm is the slowest for all but the highest space-saving ratios. At the lowest space-saving ratios, RLEDTW is nearly 100 times slower than DTW. This is caused by the overhead of computing the intersections. In fact, the number κ of intersections always attained the upper bound of $2kn - k^2$ for $k \geq 0.1n$ (that is, $\rho \leq 0.9$). Hence, the simple $O(kn)$-time dynamic program (mentioned in Sect. 3) might be faster here. For $k < 0.075n$ ($\rho > 0.925$), RLEDTW is the fastest exact algorithm and up to 100 times faster than DTW.

While all other algorithms returned exact solutions (AWarp yields exact solutions if there are no runs of zeros), the speedup of BDTW is at the expense of solution quality. Figure 5 shows the average absolute error percentage of the lower and upper bound of BDTW as a log-function of the space-saving ratio. The absolute error percentage of an approximated DTW distance $d(x, y)$ between two time series x and y is defined by
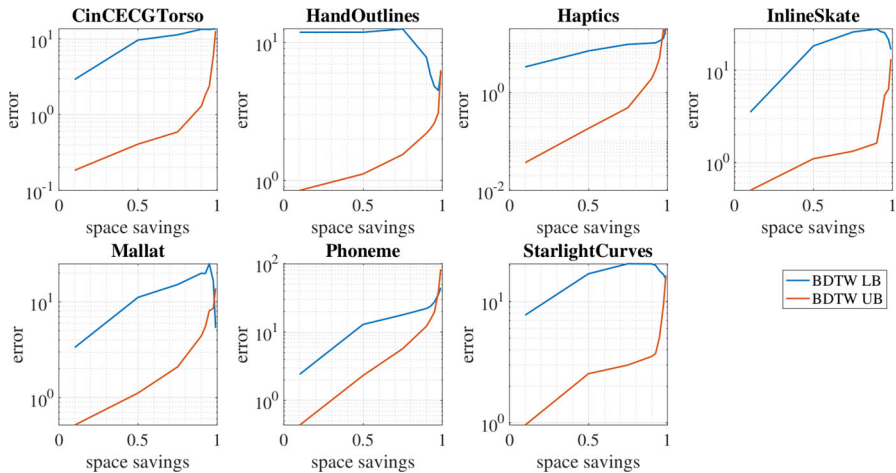
Fig. 5 Average error percentage of the BDTW bounds as a function of the space-saving ratio

$$E = 100 \cdot \frac{|\operatorname{dtw}(x, y) - d(x, y)|}{\operatorname{dtw}(x, y)}.$$

The general trend is that BDTW becomes increasingly inaccurate with increasing space-saving ratio with error percentages by more than 10% on average. In addition, the upper bound better approximates the DTW distance than the lower bound for all but the highest space-saving ratios.

5 Conclusion

We developed an asymptotically fast algorithm to compute exact DTW distances between run-length encoded time series. The running time is cubic in the maximum coding lengths of the inputs. This is actually the first exact algorithm whose running time only depends on the input coding lengths. Experiments indicate that our method yields improved performance for time series with short coding lengths (which could be achieved, for example, when using preprocessings such as piecewise aggregate approximation [8, 13, 20, 29]).

An immediate question is whether there exists an $O(\max(k, \ell)^{3-\epsilon})$-time algorithm for any $\epsilon > 0$ or whether we can exclude such an algorithm assuming the SETH. Finally, studying the complexity of DTW with respect to other compressions (as has been done for other string problems [3]) might lead to interesting results.

References

1. Abanda, A., Mori, U., Lozano, J.A.: A review on distance based time series classification. Data Min. Knowl. Disc. **33**, 1–35 (2018)
2. Abboud, A., Backurs, A., Williams, V.V.: Tight hardness results for LCS and other sequence similarity measures. In: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS '15), pp 59–78 (2015)
3. Abboud, A., Backurs, A., Bringmann, K., Künnemann, M.: Fine-grained complexity of analyzing compressed data: Quantifying improvements over decompress-and-solve. In: Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS '17), IEEE, pp 192–203 (2017)
4. Aghabozorgi, S., Shirkhorshidi, A.S., Wah, T.Y.: Time-series clustering-a decade review. Inf. Syst. **53**, 16–38 (2015)
5. Ahsan, S.B., Aziz, S.P., Rahman, M.S.: Longest common subsequence problem for run-length-encoded strings. J. Comput. **9**(8), 1769–1775 (2014)
6. Bagnall, A., Lines, J., Bostrom, A., Large, J., Keogh, E.: The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min. Knowl. Disc. **31**(3), 606–660 (2017)
7. Bringmann, K., Künnemann, M.: Quadratic conditional lower bounds for string problems and dynamic time warping. In: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS '15), pp 79–97 (2015)
8. Chakrabarti, K., Keogh, E., Mehrotra, S., Pazzani, M.: Locally adaptive dimensionality reduction for indexing large time series databases. ACM Trans. Database Syst. **27**(2), 188–228 (2002)
9. Chen, K., Chao, K.: A fully compressed algorithm for computing the edit distance of run-length encoded strings. Algorithmica **65**(2), 354–370 (2013)
10. Clifford, R., Gawrychowski, P., Kociumaka, T., Martin, D.P., Uznanski, P.: RLE edit distance in near optimal time. In: Proceedings of the 44th International Symposium on Mathematical Foundations of Computer Science (MFCS '19), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, LIPIcs, vol. 138, pp 66:1–66:13 (2019)
11. Dau, H.A., Keogh, E., Kamgar, K., Yeh, C.C.M., Zhu, Y., Gharghabi, S., Ratanamahatana, C.A., Yanping, Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G., Hexagon-M.L.: The UCR time series classification archive. (2018) https://www.cs.ucr.edu/~eamonn/time_series_data_2018/
12. Dupont, M., Marteau, P.F.: Coarse-DTW for sparse time series alignment. In: First ECML PKDD Workshop on Advanced Analysis and Learning on Temporal Data (AALTD '15), pp 157–172 (2016)
13. Faloutsos, C., Jagadish, H., Mendelzon, A., Milo, T.: A signature technique for similarity-based queries. In: Proceedings of the Compression and Complexity of Sequences 1997 (SEQUENCES '97), IEEE, pp 11–13 (1997)
14. Gold, O., Sharir, M.: Dynamic time warping and geometric edit distance: breaking the quadratic barrier. ACM Trans. Algorithm. **14**(4), 50:1-50:17 (2018)
15. Hwang, Y., Gelfand, S.B.: Sparse dynamic time warping. In: Proceedings of the 13th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM '17), pp 163–175 (2017)
16. Hwang, Y., Gelfand, S.B.: Constrained sparse dynamic time warping. In: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA '18), pp 216–222 (2018)
17. Hwang, Y., Gelfand, S.B.: Binary sparse dynamic time warping. In: Proceedings of the 15th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM '19) (2019)
18. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**(4), 512–530 (2001)
19. Jain, B.J., Froese, V., Schultz, D.: An average-compress algorithm for the sample mean problem under dynamic time warping. CoRR (2019) arXiv:1909.13541

20. Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Dimensionality reduction for fast similarity search in large time series databases. Knowl. Inf. Syst. **3**(3), 263–286 (2001)
21. Kuszmaul, W.: 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), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, LIPIcs, vol 132, pp 80:1–80:15 (2019)
22. Kuszmaul, W.: Binary dynamic time warping in linear time. CoRR (2021) arXiv:2101.01108
23. Mueen, A., Chavoshi, N., Abu-El-Rub, N., Hamooni, H., Minnich, A.: AWarp: Fast warping distance for sparse time series. In: 2016 IEEE 16th International Conference on Data Mining (ICDM '16), pp 350–359 (2016)
24. Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans. Acoust. Speech Signal Process. **26**(1), 43–49 (1978)
25. Sharabiani, A., Darabi, H., Harford, S., Douzali, E., Karim, F., Johnson, H., Chen, S.: Asymptotic dynamic time warping calculation with utilizing value repetition. Knowl. Inf. Syst. **57**(2), 359–388 (2018)
26. Silva, D.F., Giusti, R., Keogh, E., Batista, G.: Speeding up similarity search under dynamic time warping by pruning unpromising alignments. Data Min. Knowl. Disc. **32**(4), 988–1016 (2018)
27. Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., Keogh, E.: Experimental comparison of representation methods and distance measures for time series data. Data Min. Knowl. Disc. **26**(2), 275–309 (2013)
28. Yamada, K., Nakashima, Y., Inenaga, S., Bannai, H., Takeda, M.: Faster STR-EC-LCS computation. In: Proceedings of the 46th International Conference on Current Trends in Theory and Practice of Informatics, (SOFSEM '20), Springer, LNCS, vol. 12011, pp 125–135 (2020)
29. Yi, B.K., Faloutsos, C.: Fast time sequence indexing for arbitrary $\mathcal{L}_p$ norms. In: Proceedings of the 26th VLDB Conference, pp 385–394 (2000)