# Pattern matching under DTW distance

Garance Gourdel, Anne Driemel, Pierre Peterlongo, Tatiana Starikovskaya

# Pattern matching under DTW distance*

Garance Gourdel[1,2], Anne Driemel[3], Pierre Peterlongo[2], and Tatiana Starikovskaya[1]

[1] DIENS, École normale supérieure de Paris, PSL Research University, France
{garance.gourdel,tat.starikovskaya}@gmail.com
[2] IRISA Inria Rennes, France pierre.peterlongo@inria.fr
[3] Hausdorff Center for Mathematics, University of Bonn, Germany
driemel@cs.uni-bonn.de

**Abstract.** In this work, we consider the problem of pattern matching under the dynamic time warping (DTW) distance motivated by potential applications in the analysis of biological data produced by the third generation sequencing. To measure the DTW distance between two strings, one must "warp" them, that is, double some letters in the strings to obtain two equal-lengths strings, and then sum the distances between the letters in the corresponding positions. When the distances between letters are integers, we show that for a pattern $P$ with $m$ runs and a text $T$ with $n$ runs:

1. There is an $\mathcal{O}(m+n)$-time algorithm that computes all locations where the DTW distance from $P$ to $T$ is at most 1;
2. There is an $\mathcal{O}(kmn)$-time algorithm that computes all locations where the DTW distance from $P$ to $T$ is at most $k$.

As a corollary of the second result, we also derive an approximation algorithm for general metrics on the alphabet.

**Keywords:** Dynamic time warping distance · pattern matching · small-distance regime · approximation algorithms

## 1 Introduction

Introduced more than forty years ago [28], the dynamic time warping (DTW) distance has become an essential tool in the time series analysis and its applications due to its ability to preserve the signal despite speed variation in compared sequences. To measure the DTW distance between two discrete temporal sequences, one must "warp" them, that is, replace some data items in the sequences with multiple copies of themselves to obtain two equal-lengths sequences, and then sum the distances between the data items in the corresponding positions.

The DTW distance has been extensively studied for parametrised curves — sequences where the data items are points in a multidimensional space — specifically, in the context of locality sensitive hashing and nearest neighbour search [7,

* This work was partially funded by the grants ANR-20-CE48-0001, ANR-19-CE45-0008 SeqDigger and ANR-19-CE48-0016 from the French National Research Agency.

9]. In this work, we focus on a somewhat simpler, but surprisingly much less studied setting when the data items are elements of a finite set, the alphabet. Following traditions, we call such sequences *strings*.

The classical textbook dynamic programming algorithm computes the DTW distance between two $N$-length strings in $\mathcal{O}(N^2)$ time and space. Unfortunately, unless the Strong Exponential Time Hypothesis is false, there is no algorithm with strongly subquadratical time even for ternary alphabets [1, 5, 18]. On the other hand, very recently Gold and Sharir [13] showed the first weakly subquadratic time algorithm (to be more precise, the time complexity of the algorithm is $\mathcal{O}(N^2 \log\log\log N/\log\log N)$). Kuszmaul [18] gave a $\mathcal{O}(kN)$-time algorithm that computes the value of the distance between the strings if it is bounded by $k$, assuming that the distance between any two distinct letters of the alphabet is at least one, and used it to derive a subquadratic-time approximation algorithm for the general case. Finally, it is known that binary strings admit much faster algorithms: Abboud, Backurs, and Vassilevska Williams [1] showed an $O(N^{1.87})$-time algorithm followed by a linear-time algorithm by Kuszmaul [20].

The problem of computing the DTW distance has also been studied in the sparse and run-length compressed settings, as well as in the low distance regime. In the sparse setting, we assume that most letters of the string are zeros. Hwang and Gelfand [16] gave an $\mathcal{O}((s+t)N)$-time algorithm, where $s$ and $t$ denote the number of non-zero letters in each of the two strings. On sparse binary strings, the distance can be computed in $\mathcal{O}(s+t)$ time [17, 25]. Froese et al. [12] suggested an algorithm with running time $\mathcal{O}(mN+nM)$, where $M, N$ are the length of the strings, and $m, n$ are the sizes of their run length encodings. If $n \in \mathcal{O}(\sqrt{N})$ and $m \in \mathcal{O}(\sqrt{M})$, their algorithm runs in time $\mathcal{O}(nm \cdot (n+m))$. For binary strings, the DTW distance can be computed in $\mathcal{O}(nm)$ time [8].

Nishi et al. [26] considered the question of computing the DTW distance in the dynamic setting when the stings can be edited, and Sakai and Inenaga [27] showed a reduction from the problem of computing the DTW distance to the problem of computing the longest increasing subsequence, which allowed them to give polynomial-time algorithms for a series of DTW-related problems.

In this work, we focus on the pattern matching variant of the problem: Given a pattern $P$ and a text $T$, one must output the smallest DTW distance between $P$ and a suffix of $T[1\,..\,r]$ for every position $r$ of the text.

Our interest to this problem sparks from its potential applications in Third Generation Sequencing (TGS) data comparisons. TGS has changed the genomic landscape as it allows to sequence reads of few dozens of thousand of letters where previous sequencing techniques were limited to few hundred letters [2]. However, TGS suffers from a high error rate (from $\approx 1$ to 10% depending on the used techniques) mainly due to the fact that the DNA sequences are read and thus sequenced at an uneven speed. The uneven sequencing speed has a major impact in the sequencing quality of DNA regions composed of two or more equal consecutive letters. Those regions, called *homopolymers*, are hardly correctly sequenced as, due to the uneven sequencing speed, their size cannot be

precisely determined [15]. In particular, a common post-sequencing task consists in aligning the obtained reads to a reference genome. This enables for instance to predict alternative splicing and gene expression [14] or to detect structural variations [24]. All known aligners use the edit distance, most likely, due to the availability of software tools for the latter (see [23] and references therein). However, we find that the nature of TGS errors is much better described by the DTW distance, which we confirm experimentally in Section 5.

*Our contribution.* As a baseline, we show that the problem of pattern matching under the DTW distance can be solved using dynamic programming in time $\mathcal{O}(MN)$, where $M$ is the length of the pattern and $N$ of the text (Lemma **??**).

We then proceed to show more efficient algorithms for the low-distance regime on run-length compressible data, which is arguably the most interesting setting for the TGS data processing. Formally, in the $k$-DTW *problem* we are given an integer $k > 0$, a pattern $P$ and a text $T$, and must find all positions $r$ of the text such that the smallest DTW distance between the pattern $P$ and a suffix of $T[1\,..\,r]$ does not exceed $k$. One might hope that the DTW distance is close enough to the edit distance and thus is amenable to the techniques developed for the latter, such as [21, 22]. In Appendix A, we show that this is indeed the case for $k = 1$:

**Lemma 1.** *Given run-length encodings of a pattern $P$ and of a text $T$ over an alphabet $\Sigma$ and a distance $d : \Sigma \times \Sigma \to \mathbb{Z}^+$, the* 1-DTW *problem can be solved in $\mathcal{O}(m + n)$ time, where $m$ is the number of runs in $P$ and $n$ is the number of runs in $T$. The output is given in a compressed form, with a possibility to retrieve each position in constant time.*

Unfortunately, extending the approach of [21, 22] to higher values of $k$ seems to be impossible as it is heavily based on the fact that in the edit distance dynamic programming matrix the distances are non-decreasing on every diagonal, which is not the case for the DTW distance (see Fig. 1).

In Section 3 we develop a different approach. Interestingly, we show that the value of any cell of the bottom row and the right column of a block of the dynamic programming table (i.e. a subtable formed by a run in the pattern and a run in the text) can be computed in constant time given a constant-time oracle access to the left column and the top row. Combining this with a compact representation of the $k$-bounded values, we obtain the following result:

**Theorem 1.** *Given run-length encodings of a pattern $P$ and of a text $T$ over an alphabet $\Sigma$ and a distance $d : \Sigma \times \Sigma \to \mathbb{Z}^+$, the $k$-*DTW *problem can be solved in $\mathcal{O}(kmn)$ time, where $m$ is the number of runs in $P$ and $n$ is the number of runs in $T$. The output is given in a compressed form, with a possibility to retrieve each position in constant time.*

We note that while our algorithm can be significantly faster than the baseline one, its worst-case time complexity is cubic. We leave it as an open question whether there exists an $\mathcal{O}(k \cdot (m + n))$-time algorithm. Finally, in Section 4 we

use Theorem 1 to derive an approximation algorithm for the general variant of pattern matching under the DTW distance.

| | | G | G | T | T | T | T | C | T | T | A | T | T | T | T | G | G | T | G | A | T | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | ∞ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| A | ∞ | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 0 | 1 | 0 |
| T | ∞ | 3 | 3 | 2 | 2 | 2 | 2 | 3 | 2 | 2 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 2 | 3 | 1 | 0 | 1 |
| T | ∞ | 4 | 4 | 2 | 2 | 2 | 2 | 3 | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 1 | 2 | 2 | 3 | 2 | 0 | 1 |
| A | ∞ | 5 | 5 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 1 | 1 | 1 | 1 | 1 | 2 | 3 | 3 | 2 | 1 | 0 |
| T | ∞ | 6 | 6 | 3 | 3 | 3 | 3 | 4 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 1 | 1 |

Fig. 1: Consider $P = AATTAT$ and $T = GGTTTTCTTATTTTGGTGATA$. A cell $(i, j)$ contains the smallest DTW distance between $P[1\,..\,i]$ and $T[1\,..\,j]$, where the distance between two letters equals one if they are distinct and zero otherwise. A non-monotone diagonal of the table is shown in red.

## 2 Preliminaries

We assume a polynomial-size alphabet $\Sigma$ with $\sigma$ *letters*. A *string* $X$ is a sequence of letters. If the sequence has length zero, it is called the *empty string*. Otherwise, we assume that the letters in $X$ are numbered from 1 to $n =: |X|$ and denote the $i$-th letter by $X[i]$. We define $X[i\,..\,j]$ to be equal to $X[i]\ldots X[j]$ which we call a *substring* of $X$ if $i \leq j$ and to the empty string otherwise. If $j = n$, we call a substring $X[i\,..\,j]$ *a suffix* of $X$.

**Definition 1 (Run, Run-length encoding).** *A run of a string $X$ is a maximal substring $X[i\,..\,j]$ such that $X[i] = X[i+1] = \ldots = X[j]$. The run-length encoding of a string $X$,* $\mathrm{RLE}(X)$ *is a sequence obtained from $X$ by replacing each run with a tuple consisting of the letter forming the run and the length of the run. For example,* $\mathrm{RLE}(aabbbc) = (a,2)(b,3)(c,1)$.

Let $d : \Sigma \times \Sigma \to \mathbb{R}^+$ be a distance function such that for any letters $a, b \in \Sigma$, $a \neq b$, we have $d(a,a) = 0$ and $d(a,b) > 0$. The dynamic time warping distance $\mathrm{DTW}_d(X,Y)$ between strings $X, Y \in \Sigma^*$ is defined as follows. If both strings are empty, $\mathrm{DTW}_d(X,Y) = 0$. If one of the strings is empty, and the other is not, then $\mathrm{DTW}_d(X,Y) = \infty$. Otherwise, let $X = X[1]X[2]\ldots X[r]$ and $Y = Y[1]Y[2]\ldots Y[q]$. Consider an $r \times q$ grid graph such that each vertex $(i,j)$ has (at most) three outgoing edges: one going to $(i+1,j)$ (if it exists), one to $(i+1,j+1)$ (if it exists), and one to $(i,j+1)$ (if it exists). A path $\pi$ in the graph starting at $(1,1)$ and ending at $(r,q)$ is called a *warping path*, and its *cost* is defined to be $\sum_{(i,j)\in\pi} d(X[i],Y[j])$. Finally, $\mathrm{DTW}_d(X,Y)$ is defined to be the minimum cost of a warping path for $X, Y$. Below we omit $d$ if it is clear from the context.

Let $M = |P|$, $N = |T|$, and $D$ be an $(M+1) \times (N+1)$ table where the rows are indexed from 0 to $M$, and the columns from 0 to $N$ such that:

1. For all $j \in [0, N]$, $D[0, j] = 0$;
2. For all $i \in [1, M]$, $D[i, 0] = +\infty$;
3. For all $i \in [1, M]$ and $j \in [1, N]$, $D[i, j]$ equals the smallest DTW distance between $P[1\,..\,i]$ and a suffix of $T[1\,..\,j]$.

(See Fig. 1.) To solve the pattern matching problem under the DTW distance, it suffices to compute the last row of the table $D$.

**Lemma 2.** *The table $D$ can be computed in $\mathcal{O}(MN)$ time via a dynamic programming algorithm, using the following recursion for all $1 \le i \le M, 1 \le j \le N$:*

$$D[i, j] = \min\{D[i-1, j-1], D[i-1, j], D[i, j-1]\} + d(P[i], T[j])$$

In the subsequent sections, we develop more efficient solutions for the low-distance regime on run-length compressible data. We will be processing the table $D$ by blocks, defined as follows: A subtable $D[i_p\,..\,j_p, i_t\,..\,j_t]$ is called a *block* if $P[i_p\,..\,j_p]$ is a run in $P$ or $i_p = j_p = 0$, and $T[i_t\,..\,j_t]$ is a run in $T$ or $i_t = j_t = 0$. For $i_p, i_t > 0$, a block $D[i_p\,..\,j_p, i_t\,..\,j_t]$ is called *homogeneous* if $P[i_p] = T[i_t]$. (For example, a block $D[3\,..\,4][3\,..\,6]$ in Fig. 1 is homogeneous.) A block such that all cells in it contain a value $q$, for some fixed integer $q$, is called a *$q$-block*. (For example, a block $D[5\,..\,5][11\,..\,14]$ in Fig. 1 is a 1-block.) The *border* of a block is the set of the cells contained in its top and bottom rows, as well as first and last columns. Consider a cell $(a, b)$ in $B$. We say that a block $B'$ is the *top neighbour* of $B$ if it contains $(a-1, b)$, the *left neighbour* if it contains $(a, b-1)$, and the *diagonal neighbour* if it contains $(a-1, b-1)$.

The following lemma is shown by induction in Appendix B:

**Lemma 3.** *Consider a block $B = D[i_p\,..\,j_p, i_t\,..\,j_t]$ and cell $(a, b)$ in it. If $i_p \le a < j_p$, then $D[a, b] \le D[a+1, b]$ and if $i_t \le b < j_t$, then $D[a, b] \le D[a, b+1]$.*

By Lemma **??**, inside a homogeneous block each value is equal to the minimum of its neighbours. Therefore, the values in a row or in a column cannot increase and we have the following corollary:

**Corollary 1.** *Each homogeneous block is a $q$-block for some value $q$.*

## 3 Main result: $\mathcal{O}(kmn)$-time algorithm

In this section, we show Theorem 1 that for a pattern $P$ with $m$ runs and a text $T$ with $n$ runs gives an $\mathcal{O}(kmn)$-time algorithm. We start with the following lemma which is a keystone to our result:

**Lemma 4.** *For a block $D[i_p\,..\,j_p, i_t\,..\,j_t]$ let $h = j_p - i_p$, $w = j_t - i_t$, and $d = d(P[i_p], T[i_t])$. We have for every $i_p < x \le j_p$:*

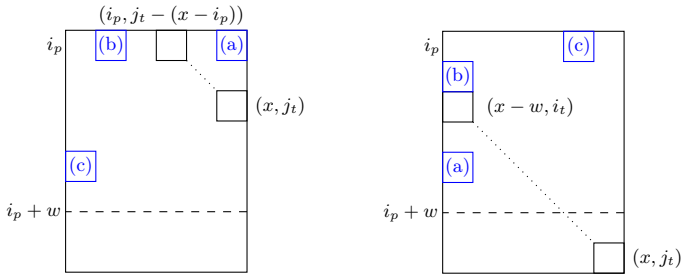
$$D[x, j_t] = \begin{cases} D[i_p, j_t - (x - i_p)] + (x - i_p) \cdot d \text{ if } x - i_p \le w; \\ D[x - w, i_t] + w \cdot d \text{ otherwise.} \end{cases} \tag{1}$$

*For every* $i_t < y \leq j_t$*:*

$$D[j_p, y] = \begin{cases} D[j_p - (y - i_t), i_t] + (y - i_t) \cdot d \text{ if } y - i_t \leq h; \\ D[i_p, y - h] + h \cdot d \text{ otherwise.} \end{cases} \quad (2)$$

*Proof.* For a homogeneous block, we have $d = 0$, and by Corollary 1 all the values in such a block are equal, hence the claim of the lemma is trivially true.

Assume now $d > 0$. Consider $x$, $i_p < x \leq j_p$, and let us show Eq. 1, Eq. 2 can be shown analogously. Let $\pi$ be a warping path realizing $D[x, j_t]$. Let $(a, b)$ be the first node of $\pi$ belonging to the block. We have $a \in [i_p, j_p]$ and $b \in [i_t, j_t]$ and either $a = i_p$ or $b = i_t$. The number of edges of $\pi$ in the block from $(a, b)$ to $(x, j_t)$ must be minimal, else there would be a shorter path, thus it is equal to $\max\{x - a, j_t - b\}$ and $D[x, j_t] = D[a, b] + \max\{x - a, j_t - b\} \cdot d$.




Case 1: $x - i_p \leq w$ Case 2: $x - i_p > w$

Fig. 2: Cases of Lemma 4. Possible locations of the cell $(a, b)$ are shown in blue.

Case 1: $x - i_p \leq w$. Consider a cell $(i_p, j_t - (x - i_p))$. There is a path from $(i_p, j_t - (x - i_p))$ to $(x, j_t)$ that takes $x - i_p$ diagonal steps inside the block, and therefore $D[x, j_t] \leq D[i_p, j_t - (x - i_p)] + (x - i_p) \cdot d$. We now show that $D[x, j_t] \geq D[i_p, j_t - (x - i_p)] + (x - i_p) \cdot d$, which implies the claim of the lemma.

(a) If $a = i_p$ and $b \geq j_t - (x - i_p)$, then $\max\{x - i_p, j_t - b\} = x - i_p$. We have $D[x, j_t] = D[i_p, b] + (x - i_p) \cdot d \geq D[i_p, j_t - (x - i_p)] + (x - i_p) \cdot d$ (Lemma **??**).

(b) If $a = i_p$ and $b < j_t - (x - i_p)$, then $\max\{x - i_p, j_t - b\} = j_t - b$. As there is a path from $(a, b) = (i_p, b)$ to $(i_p, j_t - (x - i_p))$ of length $(j_t - (x - i_p) - b)$, we have $D[i_p, j_t - (x - i_p)] \leq D[i_p, b] + (j_t - (x - i_p) - b) \cdot d$. Consequently,

$$\begin{aligned} D[x, j_t] &= D[i_p, b] + (j_t - b) \cdot d \\ &\geq D[i_p, j_t - (x - i_p)] - (j_t - (x - i_p) - b) \cdot d + (j_t - b) \cdot d \text{ (Lem. ??)} \\ &= D[i_p, j_t - (x - i_p)] + (x - i_p) \cdot d \end{aligned}$$

(c) If $b = i_t$, then $i_p \leq a$ and $\max\{x - a, j_t - b\} \leq \max\{x - i_p, w\} = w$. As there is a path from $(i_p, i_t)$ to $(i_p, j_t - (x - i_p))$ of length $(j_t - (x - i_p) - i_t)$, we

have $D[i_p, j_t - (x - i_p)] \leq D[i_p, i_t] + (j_t - (x - i_p) - i_t) \cdot d$. Therefore,

$$\begin{aligned} D[x, j_t] &= D[a, i_t] + w \cdot d \geq D[i_p, i_t] + w \cdot d \text{ (Lemma ??)} \\ &\geq D[i_p, j_t - (x - i_p)] - (j_t - (x - i_p) - i_t) \cdot d + w \cdot d \\ &= D[i_p, j_t - (x - i_p)] + (x - i_p) \cdot d \end{aligned}$$

Case 2: $x - i_p > w$. Consider a cell $(x - w, i_t)$. There is a path from $(x - w, i_t)$ to $(x, j_t)$ that takes $w$ diagonal steps inside the block, and therefore $D[x, j_t] \leq D[x - w, i_t] + w \cdot d$. We now show that $D[x, j_t] \geq D[x - w, i_t] + w \cdot d$, which implies the claim of the lemma.

(a) If $b = i_t$ and $a \geq x - w$, then $\max\{x - a, j_t - b\} = \max\{x - a, w\} = w$ and we have $D[x, j_t] = D[a, i_t] + w \cdot d \geq D[x - w, i_t] + w \cdot d$ (Lemma **??**).
(b) If $b = i_t$ and $a < x - w$, then $\max\{x - a, j_t - b\} = \max\{x - a, w\} = x - a$. As there is a path from $(a, i_t)$ to $(x - w, i_t)$ of length $(x - w - a)$, we have $D[x - w, i_t] \leq D[a, i_t] + (x - w - a) \cdot d$ by definition. Therefore,

$$\begin{aligned} D[x, j_t] &= D[a, i_t] + (x - a) \cdot d \\ &\geq D[x - w, i_t] - (x - w - a) \cdot d + (x - a) \cdot d \\ &= D[x - w, i_t] + w \cdot d \end{aligned}$$

(c) If $a = i_p$, $b \geq i_t$ and thus $\max\{x - a, j_t - b\} \leq \max\{x - i_p, w\} = x - i_p$. Additionally, as there is a path from $(i_p, i_t)$ to $(x - w, i_t)$ of length $(x - w - i_p)$ we have $D[x - w, i_t] \leq D[i_p, i_t] + (x - w - i_p) \cdot d$. Consequently,

$$\begin{aligned} D[x, j_t] &= D[i_p, b] + (x - i_p) \cdot d \geq D[i_p, i_t] + (x - i_p) \cdot d \text{ (Lemma ??)} \\ &\geq D[x - w, i_t] - (x - w - i_p) \cdot d + (x - i_p) \cdot d \\ &= D[x - w, i_t] + w \cdot d \end{aligned}$$

□

We say that a cell in a border of a block is *interesting* if its value is at most $k$. To solve the $k$-DTW problem it suffices to compute the values of all interesting cells in the last row of $D$. Consider a block $B = D[i_p \,..\, j_p, i_t \,..\, j_t]$ and recall that the values in it are non-decreasing top to down and left to right (Lemma **??**). We can consider the following compact representation of its interesting cells. For an integer $\ell$, define $q^\ell_{\text{top}} \in [i_t, j_t]$ to be the last position such that $D[i_p, q^\ell_{\text{top}}] \leq \ell$, and $q^\ell_{\text{bot}} \in [i_t, j_t]$ the last position such that $D[j_p, q^\ell_{\text{bot}}] \leq \ell$. If a value is not defined, we set it equal to $i_t - 1$. Analogously, define $q^\ell_{\text{left}} \in [i_p, j_p]$ to be the last position such that $D[q^\ell_{\text{left}}, i_t] \leq \ell$, and $q^\ell_{\text{right}} \in [i_p, j_p]$ the last position such that $D[q^\ell_{\text{right}}, j_t] \leq \ell$. If a value is not defined, we set it equal to $i_p - 1$. Positions $q^0_{\text{top}}, \ldots, q^k_{\text{top}}$ uniquely describe the interesting border cells in the top row of $B$, $q^0_{\text{bot}}, \ldots, q^k_{\text{bot}}$ in the bottom row, $q^0_{\text{left}}, \ldots, q^k_{\text{left}}$ in the leftmost column, $q^0_{\text{right}}, \ldots, q^k_{\text{right}}$ in the rightmost column.

**Lemma 5.** *The compact representations of the interesting border cells in the top row and the leftmost column of a block $B$ can be computed in $\mathcal{O}(k)$ time given the compact representation of the interesting border cells in its neighbours.*

*Proof.* We explain how to compute the representation for the leftmost column of $B$, the representation for the top row is computed analogously. Let $d = d(P[i_p], T[i_t])$. If $d = 0$ (the block is homogeneous), by Corollary 1 the block is a $q$-block for some value $q$ which can be computed in $\mathcal{O}(1)$ time by Lemma **??** if it is interesting (and otherwise we have a certificate that the value is not interesting). We can then derive the values $q^{\ell}_{\text{left}}$, $\ell = 0, 1, \ldots, k$ in $\mathcal{O}(k)$ time.

Assume now $d > 0$. We start by computing $D[i_p, i_t]$ using Lemma **??**. We note that if $D[i_p, i_t] \leq k$, then we know the values of its neighbours realising it and therefore can compute it, otherwise we can certify that $D[i_p, i_t] > k$. Assume $D[i_p, i_t] = v$, which implies that $q^0_{\text{left}}, \ldots, q^{\min\{k,v\}-1}_{\text{left}}$ equal $i_p - 1$. We must now compute $q^{\min\{k,v\}}_{\text{left}}, \ldots, q^k_{\text{left}}$. Consider a cell $(q, i_t)$ of the block with $q > i_p$. The second to the last cell in the warping path that realizes $D[q, i_t] = \ell$ is one of the cells $(q-1, i_t)$, $(q-1, i_t-1)$ or $(q, i_t-1)$, and the value of the path up to there must be $\ell - d$. Note that all the three cells belong either to the leftmost column of $B$, or the rightmost column of its left neighbour. Consequently, for all $\min\{k, v\} < \ell \leq k$, we have $q^{\ell}_{\text{left}} = \min\{\max\{q^{\ell-d}_{\text{left}}, r^{\ell-d}_{\text{right}}\} + 1\}, j_t\}$, and the positions $q^0_{\text{left}}, \ldots, q^k_{\text{left}}$ can be computed in $\mathcal{O}(k)$ time. □

**Lemma 6.** *The compact representations of the interesting border cells in the bottom row and the rightmost column of a block $B$ can be computed in $\mathcal{O}(k)$ time given the compact representation of the interesting border cells in its leftmost column and the top row.*

*Proof.* We explain how to compute the representation for the bottom row, the representation for the rightmost column is computed analogously.

Eq. 2 and the compact representations of the leftmost column and the top row of $B$ partition the bottom row of $B$ into $\mathcal{O}(k)$ intervals (some intervals can be empty), and in each interval the values are described either as a constant or as a linear function. (See Fig. 3.) Formally, let $h = j_p - i_p$. By Eq. 2, for $y \in [i_t, j_p + i_t - q^k_{\text{left}} - 1] \cap [i_t, j_t]$ we have $D[j_p][y] > k$. For $y \in [j_p + i_t - q^{\ell}_{\text{left}}, j_p + i_t - q^{\ell-1}_{\text{left}} - 1] \cap [i_t, j_t]$, $\ell = k, k-1, \ldots, 1$, we have $D[j_p][y] = \ell + (y - i_t) \cdot d$. For $y \in [j_p + i_t - q^0_{\text{left}}, j_p + i_t - i_p] \cap [i_t, j_t]$ we have $D[j_p][y] = (y - i_t) \cdot d$. For $y \in [i_t + h, q^0_{\text{top}} + h - 1] \cap [i_t, j_t]$ we have $D[j_p][y] = h \cdot d$. For $y \in [q^{\ell}_{\text{top}} + h, q^{\ell+1}_{\text{top}} + h - 1] \cap [i_t, j_t]$, $\ell = 0, 1, \ldots, k-1$, we have $D[j_p][y] = \ell + h \cdot d$. Finally, for $y \in [q^k_{\text{top}} + h, j_t]$, there is $D[j_p][y] > k$ again.

By Lemma **??**, the values in the bottom row are non-decreasing. We scan the intervals from left to right to compute the values $q^0_{\text{bot}}, \ldots, q^k_{\text{bot}}$ in $\mathcal{O}(k)$ time. In more detail, let $q^{\ell}_{\text{bot}}$ be the last computed value, and $[i, j]$ be the next interval. We set $q^{\ell+1}_{\text{bot}} = q^{\ell}_{\text{bot}}$. If the values in the interval are constant and larger than $\ell + 1$, we continue to computing $q^{\ell+2}_{\text{bot}}$. If the values are increasing linearly, we find the position of the last value smaller or equal to $\ell + 1$, set $q^{\ell+1}_{\text{bot}}$ equal to this position, and continue to computing $q^{\ell+2}_{\text{bot}}$. Finally, if the values in the interval are constant and equal to $\ell + 1$, we update $q^{\ell+1}_{\text{bot}} = j$ and continue to the next interval. As soon as $q^k_{\text{bot}}$ is computed, we stop the computation. □

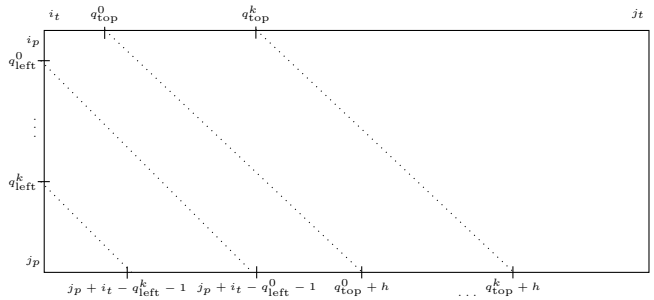



Fig. 3: Compressed representation of interesting border cells.

Since there are $\mathcal{O}(mn)$ blocks in total, Lemmas 5 and 6 immediately imply Theorem 1.

## 4 Approximation algorithm

In this section, we show an approximation algorithm for computing the smallest DTW distance between a pattern $P$ and a substring of a text $T$. We assume that the DTW distance is defined over a metric on the alphabet $\Sigma$. Kuszmaul [18] showed that the problem of computing the smallest DTW distance over an arbitrary metric can be reduced to the problem of computing the smallest distance over a so-called well-separated tree metric:

**Definition 2 (Well-separated tree metric).** *Consider a rooted tree $\tau$ with positive weights on the edges whose leaves form an alphabet $\Sigma$. The tree $\tau$ specifies a metric $\mu_\tau$ on $\Sigma$: The* distance *between two leaves $a, b \in \Sigma$ is defined as the maximum weight of an edge in the shortest path from $a$ to $b$. The metric $\mu_\tau$ is a* well-separated tree metric *if the weights of the edges are not increasing in every root-to-leaf path. The* depth *of $\mu_\tau$ is defined to be the depth of $\tau$.*

Below we show that Theorem 1 implies the following result for well-separated tree metrics:

**Lemma 7.** *Given run-length encodings of a pattern $P$ with $m$ runs and a text $T$ with $n$ runs over an alphabet $\Sigma$. Assume that the* DTW *distance is specified by a well-separated tree metric $\mu_\tau$ on $\Sigma$ with depth $h$, and suppose that the ratio between the largest and the smallest non-zero distances between the letters of $\Sigma$ is at most exponential in $L = \max\{|P|, |T|\}$. For any $0 < \epsilon < 1$, there is an $\mathcal{O}(L^{1-\varepsilon} \cdot hmn \log L)$-time algorithm that computes $\mathcal{O}(L^{\varepsilon})$-approximation of the smallest* DTW *distance between $P$ and a substring of $T$.*

By plugging the lemma into the framework of [18], we obtain:

**Theorem 2.** *Given run-length encodings of a pattern $P$ with $m$ runs and of a text $T$ with $n$ runs over an alphabet $\Sigma$. Assume that the* DTW *distance is*

*specified by a metric $\mu$ on $\Sigma$, and suppose that the ratio between the largest and the smallest non-zero distances between the letters of $\Sigma$ is at most exponential in $L = \max\{|P|, |T|\}$. For any $0 < \epsilon < 1$, there is a $\mathcal{O}(L^{1-\varepsilon} \cdot mn \log^3 L)$-time algorithm that computes $\mathcal{O}(L^{\varepsilon})$-approximation of the smallest* DTW *distance between $P$ and a substring of $T$ correctly with high probability*[4].

The proof follows the lines of the full version [19] of [18], we provide it in Appendix C for completeness. We now show Lemma 7. Compared to [18], the main technical challenge is that our $k$-DTW algorithm (Theorem 1) assumes an integer-valued distance function on the alphabet. We overcome this by developing an intermediary 2-approximation algorithm for real-valued distances (see the two claims below).

*Proof of Lemma 7.* For brevity, let $\delta$ be the smallest $\mathrm{DTW}_{\mu_\tau}$ distance between $P$ and a substring of $T$.

*Claim.* Let $0 < \varepsilon < 1$. Assume that for all $a, b \in \Sigma$, $a \neq b$, there is $\mu_\tau(a, b) \geq \gamma$ and that the value of $\mu_\tau(a, b)$ can be evaluated in $\mathcal{O}(t)$ time. There is an $\mathcal{O}(L^{1-\varepsilon} tmn)$-time algorithm which either computes a 2-approximation of $\delta$ or concludes that it is larger than $\gamma \cdot L^{1-\varepsilon}$.

*Proof.* Define a new distance function $\mu'_\tau(a, b) = \lceil \mu_\tau(a, b)/\gamma \rceil$. For all $a, b \in \Sigma$, $a \neq b$, we have $\mu_\tau(a, b) \leq \gamma \cdot \mu'_\tau(a, b) \leq \mu_\tau(a, b) + \gamma \leq 2\mu_\tau(a, b)$. Consequently, for all strings $X, Y$ we have $\mathrm{DTW}_{\mu_\tau}(X, Y) \leq \gamma \cdot \mathrm{DTW}_{\mu'_\tau}(X, Y) \leq 2\mathrm{DTW}_{\mu_\tau}(X, Y)$. Let $\delta' = \min_{S- \text{ substring of } T} \min\{2k + 1, \mathrm{DTW}_{\mu'_\tau}(P, S)\}$ for $k = L^{1-\varepsilon}$. By Theorem 1, it can be computed in $\mathcal{O}(L^{1-\varepsilon} tmn)$ time. If $\delta' = 2L^{1-\varepsilon} + 1$, we conclude that $\delta \geq \gamma \cdot L^{1-\varepsilon}$, and otherwise, output $\gamma\delta'$. □

W.l.o.g., the minimum non-zero distance between two distinct letters of $\Sigma$ is 1 and the largest distance is some value $M$, which is at most exponential in $L$. We run the algorithm above for $\gamma = 1$, which either computes a 2-approximation of $\delta$ which we can output immediately, or concludes that $\delta \geq L^{1-\varepsilon}$. Below we assume that $\delta \geq L^{1-\varepsilon}$.

**Definition 3 ($r$-simplification).** *For a string $X \in \Sigma^*$ and $r \geq 1$, the* $r$-simplification *$s_r(X)$ is constructed by replacing each letter $a$ of $X$ with its highest ancestor $a'$ in $\tau$ that can be reached from $a$ using only edges of weight $\leq r/4$.*

**Fact 3 (Corollary of [18, Lemma 4.6], see also [4])** *For all $X, Y \in \Sigma^{\leq L}$, the following properties hold:*

1. $\mathrm{DTW}_{\mu_\tau}(s_r(X), s_r(Y)) \leq \mathrm{DTW}_{\mu_\tau}(X, Y)$.
2. *If* $\mathrm{DTW}_{\mu_\tau}(X, Y) > Lr$, *then* $\mathrm{DTW}_{\mu_\tau}(s_r(X), s_r(Y)) > Lr/2$.

[4] The preprocessing time $\mathcal{O}(|\Sigma|^2 \log L)$ that is required to embed $\mu$ into a well-separated metric is not accounted for in the runtime of the algorithm.

Fix $r \geq 1$ and $0 < \varepsilon < 1$. In the $(L^\varepsilon, r)$-DTW *gap pattern matching problem*, we must output 0 if the smallest DTW distance between $P$ and a substring of $T$ is at most $L^{1-\varepsilon}r/4$ and 1 if it is at least $Lr$, otherwise we can output either 0 or 1.

*Claim.* The $(L^\varepsilon, r)$-DTW gap pattern matching problem can be solved in $\mathcal{O}(L^{1-\varepsilon} \cdot hmn)$ time.

*Proof.* Let $\delta_r$ be the smallest $\mathrm{DTW}_{\mu_\tau}$ distance between $s_r(P)$ and a substring of $s_r(T)$. If $L^{1-\varepsilon} > L/2$, then $L = \mathcal{O}(1)$ and we can compute $\delta$ exactly in $\mathcal{O}(1)$ time by Lemma **??**. Otherwise, we run the 2-approximation algorithm for $\gamma = r/4$, which takes $O(L^{1-\varepsilon} \cdot hmn)$ time (we can evaluate the distance between two letters in $\mathcal{O}(h)$ time). If the algorithm concludes that $\delta_r > L^{1-\varepsilon}r/4$, then $\delta > L^{1-\varepsilon}r/4$ by Fact 3, and we can output 1. Otherwise, the algorithm outputs a 2-approximation $\delta'_r$ of $\delta_r$, i.e. $\delta_r \leq \delta'_r \leq 2\delta_r$. If $\delta'_r \leq L^{1-\varepsilon}r \leq Lr/2$, then we have $\delta_r \leq Lr/2$. Therefore, $\delta \leq Lr$ by Fact 3 and we can output 0. Otherwise, $\delta \geq \delta_r \geq \delta'_r/2 > L^{1-\varepsilon}r/2 > L^{1-\varepsilon}r/4$, and we can output 1. □

Consider the $(L^\varepsilon/2, 2^i)$-DTW gap pattern matching problem for $0 \leq i \leq \lceil \log ML \rceil$. If the $(L^\varepsilon/2, 2^0)$-DTW gap pattern matching problem returns 0, then we know that $\delta \leq L$, and can return $L^{1-\varepsilon}$ as a $L^\varepsilon$-approximation for $\delta$. Therefore, it suffices to consider the case where the $(L^\varepsilon/2, 2^0)$-DTW gap pattern matching problem returns 1. We can assume, without computing it, that the $(L^\varepsilon/2, 2^{\lceil \log ML \rceil})$-DTW gap pattern matching returns 0 as $\delta \leq ML$. Consequently, there must exist $i^*$ such that $(L^\varepsilon/2, 2^{i^*-1})$-DTW gap pattern matching returns 1 and $(L^\varepsilon/2, 2^{i^*-1})$-DTW returns 0. We can find $i^*$ by a binary search which takes $\mathcal{O}(L^{1-\varepsilon}hmn \log\log ML) = \mathcal{O}(L^{1-\varepsilon}hmn \log L)$ time. We have $\delta \geq 2^{i^*-1}L^{1-\varepsilon}/4$ and $\delta \leq 2^{i^*}L$, and therefore can return $2^{i^*-1}L^{1-\varepsilon}/4$ as a $\mathcal{O}(L^\varepsilon)$-approximation of $\delta$. □

## 5 Experiments

This section provides evidence of the advantage of the DTW distance over the edit distance when processing the third generation sequencing (TGS) data. Our experiment compares how the two distances are affected by biological mutation as opposed to sequencing errors, including homopolymer length errors.

We first simulate two genomes, $G$ and $G'$, which can be considered as strings on the alphabet $\Sigma = \{A, C, G, T\}$. The genome $G$ is a substring of the E.coli genome (strain SQ110, NCBI Reference Sequence: NZ_CP011322.1) of length 10000 (positions 100000 to 110000, excluded). The genome $G'$ is obtained from $G$ by simulating biological mutations, where the probabilities are chosen according to [6]. The algorithm initializes $G'$ as the empty string, and $\texttt{pos} = 1$. While $\texttt{pos} \leq |G|$ it executes the following:

1. With probability 0.01, simulate a substitution: chose uniformly at random $a \in \Sigma$, $a \neq G[pos]$. Set $G' = G'a$ and $\texttt{pos} = \texttt{pos} + 1$.

2. Else, with probability 0.0005 simulate an insertion or a deletion of a substring of length $x$, where $x$ is chosen uniformly at random from an interval $[1, \texttt{max\_len\_ID}]$, where $\texttt{max\_len\_ID}$ is fixed to 10 in the experiments:
   (a) With probability 0.5, set $\texttt{pos} = \texttt{pos} + x + 1$ (deletion);
   (b) With probability 0.5, choose a string $X \in \Sigma^x$ uniformly at random, set $G' = G'X$ and $\texttt{pos} = \texttt{pos} + 1$ (insertion).
3. Else, set $G' = G'G[\texttt{pos}]$ and $\texttt{pos} = \texttt{pos} + 1$.

To simulate reads, we extract substrings of $G'$ and add sequencing errors:

1. For each read, extract a substring $R$ of length 500 at a random position of $G'$. As $G'$ originates from $G$, we know the theoretical distance from $R$ to $G$, which we call the "*biological diversity*". The biological diversity is computed as the sum of the number of letter substitutions, letter insertions, and letter deletions that were applied to the original substring from $G$ to obtain $R$.
2. Add sequencing errors by executing the following for each position $i$ of $R$:
   (a) With probability 0.001, substitute $R[i]$ with a letter $a \in \Sigma$, $a \neq R[i]$. The letter $a$ is chosen uniformly at random.
   (b) If $R[i] = R[i-1]$, insert with a probability $p_{hom}$ a third occurrence of the same letter to simulate a homopolymer error.

Fig. 4 shows the difference between the biological diversity and the smallest edit and DTW distances between a generated read and a substring of $G$ depending on $p_{hom}$. It can be seen that the DTW distance gives a good estimation of the biological diversity, whereas, as expected, the edit distance is heavily affected by homopolymer errors. To ensure reproducibility of our results, our complete experimental setup is available at https://github.com/fnareoh/DTW.

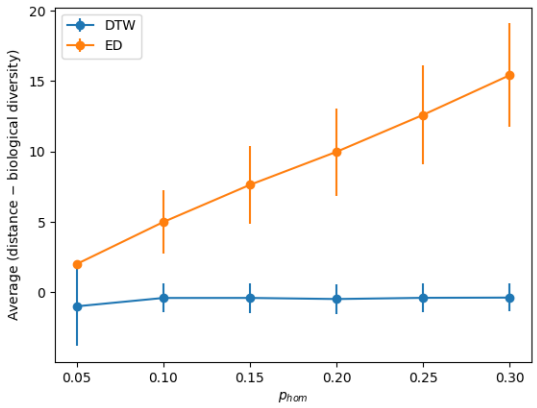



Fig. 4: Edit and DTW distances offset by the biological diversity as a function of $p_{hom}$. Each point is averaged over 600 reads (×30 coverage).

## References


1. Abboud, A., Backurs, A., Williams, V.V.: Tight hardness results for LCS and other sequence similarity measures. In: FOCS'15. pp. 59–78. IEEE Computer Society (2015). https://doi.org/10.1109/FOCS.2015.14
2. Amarasinghe, S.L., Su, S., Dong, X., Zappia, L., Ritchie, M.E., Gouil, Q.: Opportunities and challenges in long-read sequencing data analysis. Genome biology **21**(1), 1–16 (2020)
3. Bansal, N., Buchbinder, N., Madry, A., Naor, J.: A polylogarithmic-competitive algorithm for the k-server problem. In: FOCS'11. pp. 267–276 (2011). https://doi.org/10.1109/FOCS.2011.63
4. Braverman, V., Charikar, M., Kuszmaul, W., Woodruff, D.P., Yang, L.F.: The one-way communication complexity of dynamic time warping distance. In: SoCG'19. LIPIcs, vol. 129, pp. 16:1–16:15 (2019). https://doi.org/10.4230/LIPIcs.SoCG.2019.16
5. Bringmann, K., Künnemann, M.: Quadratic conditional lower bounds for string problems and dynamic time warping. In: FOCS'15. pp. 79–97 (2015). https://doi.org/10.1109/FOCS.2015.15
6. Chen, J.Q., Wu, Y., Yang, H., Bergelson, J., Kreitman, M., Tian, D.: Variation in the Ratio of Nucleotide Substitution and Indel Rates across Genomes in Mammals and Bacteria. Molecular Biology and Evolution **26**(7), 1523–1531 (03 2009). https://doi.org/10.1093/molbev/msp063
7. Driemel, A., Silvestri, F.: Locality-Sensitive Hashing of Curves. In: SoCG'17. LIPIcs, vol. 77, pp. 37:1–37:16 (2017). https://doi.org/10.4230/LIPIcs.SoCG.2017.37
8. Dupont, M., Marteau, P.: Coarse-DTW for sparse time series alignment. In: AALTD'15. LNCS, vol. 9785, pp. 157–172 (2015). https://doi.org/10.1007/978-3-319-44412-3_11
9. Emiris, I.Z., Psarros, I.: Products of Euclidean Metrics and Applications to Proximity Questions among Curves. In: SoCG'18. LIPIcs, vol. 99, pp. 37:1–37:13 (2018). https://doi.org/10.4230/LIPIcs.SoCG.2018.37
10. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. In: STOC'03. pp. 448–455 (2003). https://doi.org/10.1145/780542.780608
11. Fischer, J., Heun, V.: Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In: CPM'06. pp. 36–48 (2006)
12. Froese, V., Jain, B.J., Rymar, M., Weller, M.: Fast exact dynamic time warping on run-length encoded time series. CoRR **abs/1903.03003** (2019)
13. Gold, O., Sharir, M.: Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. ACM Trans. Algorithms **14**(4), 50:1–50:17 (2018). https://doi.org/10.1145/3230734
14. Gonzalez-Garay, M.L.: Introduction to isoform sequencing using pacific biosciences technology (iso-seq). In: Transcriptomics and gene regulation, pp. 141–160. Springer (2016)
15. Huang, Y.T., Liu, P.Y., Shih, P.W.: Homopolish: a method for the removal of systematic errors in nanopore sequencing by homologous polishing. Genome Biology **22**(1), 95 (2021). https://doi.org/10.1186/s13059-021-02282-6
16. Hwang, Y., Gelfand, S.B.: Sparse dynamic time warping. In: MLDM'17. LNCS, vol. 10358, pp. 163–175 (2017)

17. Hwang, Y., Gelfand, S.B.: Binary sparse dynamic time warping. In: MLDM'19. pp. 748–759. ibai Publishing (2019)
18. Kuszmaul, W.: Dynamic time warping in strongly subquadratic time: Algorithms for the low-distance regime and approximate evaluation. In: ICALP'19. LIPIcs, vol. 132, pp. 80:1–80:15 (2019). https://doi.org/10.4230/LIPIcs.ICALP.2019.80
19. Kuszmaul, W.: Dynamic time warping in strongly subquadratic time: Algorithms for the low-distance regime and approximate evaluation. CoRR **abs/1904.09690** (2019). https://doi.org/10.48550/ARXIV.1904.09690
20. Kuszmaul, W.: Binary dynamic time warping in linear time. CoRR **abs/2101.01108** (2021)
21. Landau, G.M., Myers, E.W., Schmidt, J.P.: Incremental string comparison. SIAM J. Comput. **27**(2), 557–582 (1998). https://doi.org/10.1137/S0097539794264810
22. Landau, G.M., Vishkin, U.: Fast string matching with k differences. Journal of Computer and System Sciences **37**(1), 63–78 (1988). https://doi.org/10.1016/0022-0000(88)90045-1
23. Li, H.: Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics **34**(18), 3094–3100 (05 2018). https://doi.org/10.1093/bioinformatics/bty191
24. Mahmoud, M., Gobet, N., Cruz-Dávalos, D.I., Mounier, N., Dessimoz, C., Sedlazeck, F.J.: Structural variant calling: the long and the short of it. Genome biology **20**(1), 1–14 (2019)
25. Mueen, A., Chavoshi, N., Abu-El-Rub, N., Hamooni, H., Minnich, A.: Awarp: fast warping distance for sparse time series. In: ICDM'16. pp. 350–359. IEEE (2016)
26. Nishi, A., Nakashima, Y., Inenaga, S., Bannai, H., Takeda, M.: Towards efficient interactive computation of dynamic time warping distance. In: SPIRE'20. LNCS, vol. 12303, pp. 27–41 (2020). https://doi.org/10.1007/978-3-030-59212-7_3
27. Sakai, Y., Inenaga, S.: A reduction of the dynamic time warping distance to the longest increasing subsequence length. In: ISAAC'20. LIPIcs, vol. 181, pp. 6:1–6:16 (2020). https://doi.org/10.4230/LIPIcs.ISAAC.2020.6
28. Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. IEEE transactions on acoustics, speech, and signal processing **26**(1), 43–49 (1978)

## Appendix A

In this section, we show Lemma **??** that for a pattern $P$ with $m$ runs and and text $T$ with $n$ runs gives an $\mathcal{O}(m+n)$-time algorithm.

**Definition 4 (RLE-diagonals).** *We say that a sequence of blocks forms an* RLE-diagonal *if the blocks are formed by runs $i, i+1, \ldots, j$ of $P$ and $i+\delta, i+1+\delta, \ldots, j+\delta$ of $T$, for some integers $i, j, \delta$.*

**Definition 5 (Streak).** *A q-streak is a maximal subsequence of an RLE-diagonal containing sequential homogeneous q-blocks.*

**Observation 4** *If $D[i,j] = 0$, then it belongs to a 0-streak. Furthermore, each 0-streak necessarily starts in the first row of $D$.*

*Proof.* By definition, there must be a path from the first row of $D$ to $D[i,j]$ containing 0-values only. For every 0-value $D[i',j']$ we must have $P[i'] = T[j']$,

and therefore every such value must belong to a homogeneous 0-block. Furthermore, two homogeneous blocks can only be neighbours diagonally, else it would contradict the maximality of the runs. The claim follows. □

**Observation 5** *If $D[i,j] = 1$, then $D[i,j]$ belongs to a 1-streak or neighbours a block in a 0-streak.*

*Proof.* If $P[i] = T[j]$, we are in a homogeneous block and $D[i,j]$ belongs to a 1-streak, and we are done. Otherwise, we have $P[i] \neq T[j]$ and there is a path $(i_1, j_1), (i_2, j_2), \ldots, (i_q, j_q)$ such that $i_1 = 1$, $(i_q, j_q) = (i, j)$, and $D[i_q, j_q] = \sum_{q'=1}^{q} d(P[i_{q'}], T[j_{q'}])$. As $d(P[i_q], T[i_q]) \geq 1$, it follows that $d(P[i_q], T[i_q]) = 1$ for all $1 \leq q' \leq q-1$, $d(P[i_{q'}], T[j_{q'}]) = 0$, and therefore $D[i_{q'}, j_{q'}]$ must belong to a 0-streak by Observation 4. □

**Lemma 1.** *Given run-length encodings of a pattern $P$ and of a text $T$ over an alphabet $\Sigma$ and a distance $d : \Sigma \times \Sigma \to \mathbb{Z}^+$, the* 1-DTW *problem can be solved in $\mathcal{O}(m+n)$ time, where $m$ is the number of runs in $P$ and $n$ is the number of runs in $T$. The output is given in a compressed form, with a possibility to retrieve each position in constant time.*

*Proof.* For a string $S$, define $\overline{RLE}(S)$ to be a string such that $\overline{RLE}(S)[i]$ contains the letter forming the $i$-th run of $S$. For example, $\overline{RLE}(aabbbc) = abc$. We preprocess $P' = \overline{\mathrm{RLE}}(P)$ and $T' = \overline{\mathrm{RLE}}(T)$ in $\mathcal{O}(m+n)$ time and space to maintain longest common suffix queries in constant time [11]. The input of a longest common suffix query are two positions $i, j$ of $P'$ and $T'$ respectively, and the output is the largest $\ell$ such that $P'[i-\ell\,..\,i] = T'[j-\ell\,..\,j]$.

Let $B_i$, $1 \leq i \leq n$, be the block of $D$ formed by the $m$-th run in $P$ and the $i$-th run in $T$. Using one longest common suffix query for each block $B_i$, we find the maximal streak containing it. If this streak reaches the first row of $D$, it is a 0-streak and all the values in the bottom row of $B_i$ are zeros.

We must now decide which entries in the $M$-th row of $D$ must be filled with one. Consider an entry $D[M, \ell] \neq 0$ that belongs to a block $B_i$.

If $B_i$ is contained in a streak of length at least one, then for $D[M, \ell]$ to be equal to one, it must be a 1-streak. Consider the first block in the maximal streak containing $B_i$, and let $c$ be the cell in its top left corner. Because $c$ can not be equal to zero, it suffices to check whether the value in $c$ equals one. Consider a path realizing the value of $c$. It goes either through the left neighbour $\ell$ of $c$, the top neighbour $t$ of $c$, or the diagonal neighbour $d$ of $c$. Furthermore, the value in $c$ equals the minimum of the values in $\ell, d, t$. Therefore, the value in $c$ equals one iff one of the values in $\ell, d, t$ equals one. Note that neither of $\ell, d, t$ belongs to a streak. By Observation 5, for the value in a cell $\ell$, $d$, or $t$ to be equal to one, the cell must neighbour a block in a zero-streak. For each block neighbouring the cells $\ell, d, t$, we use one longest common suffix query to decide whether they are contained in a 0-streak. If they are, then we can compute the value in that cell and if it equals one, the value in $c$ and all the cells in the bottom row of $B_i$ equal one as well.

Suppose now that $B_i$ does not belong to a streak. For $D[M, \ell]$ to be equal to one, it must neighbour a block in a 0-streak. Therefore, there can be only one such cell in $B_i$, the one in the left bottom corner, and we can decide whether the value in it equals to one in constant time similar to above. □

## Appendix B

**Lemma 2.** *The table $D$ can be computed in $\mathcal{O}(MN)$ time via a dynamic programming algorithm, using the following recursion for all $1 \le i \le M, 1 \le j \le N$:*

$$D[i,j] = \min\{D[i-1,j-1], D[i-1,j], D[i,j-1]\} + d(P[i], T[j])$$

*Proof.* If $i = 0$, then for all $j$, $D[i,j]$ equals the minimum distance between the empty prefix of $P$ and a suffix of $T[1\,..\,j]$, which is zero by the definition. If $i > 1$ and $j = 0$, then $D[i,j]$ equals the minimum distance between a non-empty prefix of $P$ and the empty string, which is $\infty$ by the definition.

Assume $i, j \ge 1$. Let us show that $D[i,j] \ge \min\{D[i-1,j-1], D[i-1,j], D[i,j-1]\} + d(P[i], T[j])$ and $D[i,j] \le \min\{D[i-1,j-1], D[i-1,j], D[i,j-1]\} + d(P[i], T[j])$, which implies equality. We start by showing the first inequality. Recall that $D[i,j]$ is the smallest DTW distance between $P[1\,..\,i]$ and a suffix of $T[1\,..\,j]$. Let this minimum be realised by a suffix $T[j'\,..\,j]$, where $1 \le j' \le j$ (by definition, $T[j'\,..\,j]$ is not empty: the distance from $P[1\,..\,i]$ to a non-empty suffix is finite, while that to the empty suffix equals $\infty$). Let $\pi$ be a warping path such that its cost equals $\mathrm{DTW}(P[1\,..\,i], T[j'\,..\,j])$. Consider the last edge in $\pi$. If it is from $(i-a, j-b)$ to $(i,j)$, where $a, b \in \{0,1\}$ and $a + b > 0$, then

$$\begin{aligned}
&\mathrm{DTW}(P[1\,..\,i], T[j'\,..\,j]) \\
&\qquad \ge d(P[i], T[j]) + \mathrm{DTW}(P[1\,..\,i-a], T[j'\,..\,j-b]) \\
&\qquad \ge d(P[i], T[j]) + D[i-a, j-b] \\
&\qquad \ge d(P[i], T[j]) + \min\{D[i-1,j-1], D[i-1,j], D[i,j-1]\}
\end{aligned}$$

We now show the second inequality. Let $D[i-a, i-b] = \min\{D[i-1,j-1], D[i-1,j], D[i,j-1]\}$, where $a, b \in \{0,1\}$ and $a + b > 0$. Assume that $D[i-a, j-b]$ is realised on $P[1\,..\,i-a]$ and $T[j'\,..\,j-b]$ and a warping path $\pi$. We can then consider a warping path $\pi' = \pi \cup e$, where $e$ is an edge from $(i-a, j-b)$ to $(i,j)$ for $P[1\,..\,i]$ and $T[j'\,..\,j]$. We have

$$\begin{aligned}
D[i,j] \le \mathrm{DTW}(P[1\,..\,i], T[j'\,..\,j]) &\le \sum_{(x,y)\in\pi'} d(P[x], T[y]) \\
&= d(P[i], T[j]) + \sum_{(x,y)\in\pi} d(P[x], T[y]) \\
&= d(P[i], T[j]) + \mathrm{DTW}(P[1\,..\,i-a], T[j'\,..\,j-b]) \\
&= d(P[i], T[j]) + D[i-a, j-b] \\
&= \min\{D[i-1,j-1], D[i-1,j], D[i,j-1]\} + d(P[i], T[j])
\end{aligned}$$

□

**Lemma 3.** *Consider a block $B = D[i_p \,..\, j_p, i_t \,..\, j_t]$ and cell $(a, b)$ in it. If $i_p \leq a < j_p$, then $D[a, b] \leq D[a + 1, b]$ and if $i_t \leq b < j_t$, then $D[a, b] \leq D[a, b + 1]$.*

*Proof.* Let us first give an equivalent statement of the lemma: if $(a, b)$ and $(a + 1, b)$ are in the same block, then $D[a, b] \leq D[a + 1, b]$, and if $(a, b)$ and $(a, b + 1)$ are in the same block, then $D[a, b] \leq D[a, b + 1]$.

We show the lemma by induction on $a + b$. The base of the induction are the cells such that $a = 0$ or $b = 0$, and for them the statement holds by the definition of $D$. Consider now a cell $(a, b)$, where $a, b \geq 1$. Assume that the induction assumption holds for all cells $(x, y)$ such that $x + y < a + b$. By Lemma **??**, we have:

$$\begin{aligned}
D[a, b] &= \min\{D[a - 1, b - 1], D[a - 1, b], D[a, b - 1]\} + d \\
D[a + 1, b] &= \min\{D[a, b - 1], D[a, b], D[a + 1, b - 1]\} + d \\
D[a, b + 1] &= \min\{D[a - 1, b], D[a - 1, b + 1], D[a, b]\} + d
\end{aligned}$$

Assume that $(a, b)$ and $(a + 1, b)$ are in the same block. We have $D[a, b] \leq D[a, b - 1] + d$ and trivially $D[a, b] \leq D[a, b] + d$. By the induction assumption, $D[a, b - 1] \leq D[a + 1, b - 1]$ (the cells $(a, b - 1)$ and $(a + 1, b - 1)$ must belong to the same block). Therefore,

$$\begin{aligned}
D[a + 1, b] &= \min\{D[a, b - 1], D[a, b], D[a + 1, b - 1]\} + d \\
&= \min\{D[a, b - 1] + d, D[a, b] + d, D[a + 1, b - 1] + d\} \\
&\geq \min\{D[a, b], D[a, b], D[a, b - 1] + d\} \\
&\geq \min\{D[a, b], D[a, b], D[a, b]\} = D[a, b].
\end{aligned}$$

Assume now that $(a, b)$ and $(a, b + 1)$ are in the same block. We have $D[a, b] \leq D[a - 1, b] + d$. Furthermore, as $(a - 1, b)$ and $(a - 1, b + 1)$ are in the same block, we have $D[a - 1, b] \leq D[a - 1, b + 1]$ by the induction assumption. Therefore,

$$\begin{aligned}
D[a, b + 1] &= \min\{D[a - 1, b], D[a - 1, b + 1], D[a, b]\} + d \\
&= \min\{D[a - 1, b] + d, D[a - 1, b + 1] + d, D[a, b] + d\} \\
&\geq \min\{D[a - 1, b] + d, D[a - 1, b] + d, D[a, b]\} \\
&\geq \min\{D[a, b], D[a, b], D[a, b]\} = D[a, b].
\end{aligned}$$

This concludes the proof of the lemma. □

## Appendix C

**Theorem 2.** *Given run-length encodings of a pattern $P$ with $m$ runs and of a text $T$ with $n$ runs over an alphabet $\Sigma$. Assume that the* DTW *distance is specified by a metric $\mu$ on $\Sigma$, and suppose that the ratio between the largest and*

*the smallest non-zero distances between the letters of $\Sigma$ is at most exponential in $L = \max\{|P|, |T|\}$. For any $0 < \epsilon < 1$, there is a $\mathcal{O}(L^{1-\varepsilon} \cdot mn \log^3 L)$-time algorithm that computes $\mathcal{O}(L^{\varepsilon})$-approximation of the smallest* DTW *distance between $P$ and a substring of $T$ correctly with high probability*[5].

*Proof.* Any metric $\mu$ can be embedded in $\mathcal{O}(\sigma^2)$ time into a well-separated tree metric $\mu_\tau$ of depth $\mathcal{O}(\log \sigma)$ with expected distortion $\mathcal{O}(\log \sigma)$ (see [10] and [3, Theorem 2.4]). Furthermore, the ratio between the smallest distance and the largest distance grows at most polynomially. Formally, for any two letters $a, b$ we have $\mu(a, b) \leq \mu_\tau(a, b)$ and $\mathbb{E}(\mu_\tau(a, b)) \leq \mathcal{O}(\log \sigma) \cdot d(a, b)$. Therefore, we have:

$$\mathrm{DTW}_\mu(X, Y) \leq \mathrm{DTW}_{\mu_\tau}(X, Y) \tag{3}$$

$$\mathbb{E}(\mathrm{DTW}_{\mu_\tau}(X, Y)) \leq \mathcal{O}(\log \sigma) \cdot \mathrm{DTW}_\mu(X, Y) \tag{4}$$

Let $\delta = \min_{S- \text{ substr. of } T} \mathrm{DTW}_\mu(P, S)$ and $\delta_\tau = \min_{S- \text{ substr. of } T} \mathrm{DTW}_{\mu_\tau}(P, S)$. Assume that $\delta$ is realised on a substring $X$, and $\delta_\tau$ on a substring $X_\tau$. By Eq. 3, we then obtain:

$$\delta = \mathrm{DTW}_\mu(P, X) \leq \mathrm{DTW}_\mu(P, X_\tau) \leq \delta_\tau$$

And Eq. 4 gives the following:

$$\mathbb{E}(\delta_\tau) \leq \mathbb{E}(\mathrm{DTW}_{\mu_\tau}(P, X)) \leq \mathcal{O}(\log \sigma) \cdot \mathrm{DTW}_\mu(P, X) = \mathcal{O}(\log \sigma) \cdot \delta$$

We apply the embedding $\log L$ times independently to obtain well-separated tree metrics $\mu_\tau^i$, $i = 1, 2, \ldots, \log L$. From above and by Chernoff bounds,

$$\min_i \min_{S- \text{ substring of } T} \mathrm{DTW}^i_{\mu_\tau}(P, S)$$

gives an $\mathcal{O}(\log \sigma) = \mathcal{O}(\log L)$ approximation of $\delta$ with high probability and can be computed in time $\mathcal{O}(L^{1-\varepsilon} \cdot mn \log^3 L)$ by Lemma 7, concluding the proof of the theorem. □

[5] The preprocessing time $\mathcal{O}(|\Sigma|^2 \log L)$ that is required to embed $\mu$ into a well-separated metric is not accounted for in the runtime of the algorithm.