A Specialized Asymmetric Algorithm for Exact Dynamic Time Warping

Zoe Xi

April 20, 2021

1 Introduction

Dynamic time warping distance (DTW) is a means of measuring a form of similarity between strings, that is, sequences of letters from a given alphabet. In particular, DTW is widely used in applications mining time series data. Computing the DTW between two strings x and y involves expanding them to two strings x' and y' of the same length by stretching letters; that is, replacing one letter with a string of copies of itself. Each letter in x' is then paired with the corresponding letter in y', and the cost between x' and y' is the sum of the all the distances between such pairs of letters. The DTW between x and y is the minumum of such costs.

Given two strings x and y, computing $DTW(x,y)$ can be readily done with dynamic programming, resulting in an algorithm that is of $O(mn)$-time, where m and n are the lengths of x and y, respectively.

Under the Strong Exponential Time Hypothesis (SETH), one cannot expect to have strongly subquadratic algorithms for computing DTW. However, this does not rule out such algorithms for special DTW problems where the involved strings are restricted in some manners. For instance. Kuszmaul gives an algorithm of $(n \cdot DTW(x,y))$-time for computing $DTW(x,y)$ [Kus19], where x and y are assumed to be $O(n)$. There is also an algorithm of $O(n^{1.87})$-time that computes $DTW(x,y)$ for binary strings x and y formed with a two-letter alphabet[ABW15]. And it is reported very recently that $DTW(x,y)$ for such binary strings can even be computed in $O(n)$-time [Kus21]. Moreover, an algorithm for computing the DTW of two run-length-encoded (RLE) strings x and y is reported to be of time-complexity

$O(k^2l + kl^2)$[FJRW20], where k and l and the numbers of runs in x and y, respectively.

In this paper, we also first study a special kind of DTW problem where the DTW distance between a (regular) string and a run-length-encoded (RLE) string is computed. Our primary result is a specialized asymmetric algorithm of $O(m^2l)$-time for computing $DTW(u_0, x_0)$, where m is the length of the string u_0 and l is the number of runs in the RLE-string x_0.

2 Preliminaries

For a metric space Σ, the dynamic time warping distance between two strings x and y in Σ^*, like the well-known edit distance, is a natural measure of similarity between them.

We use a for letters and u for strings. And we use $\delta(a_1, a_2)$ for the distance between two letters in Σ.

We also use x and y for both (regular) strings and RLE-strings. Given a string u, we use $u[i]$ for the letter in u located at position i, where $0 \le i < |u|$ is assumed. Note that $u[0]$ is the first letter in u as the position count starts from 0.

We use $|x|$ for the length of a string x. Also, we use $||x||$ for the number of runs in x if x is a RLE-string.

Definition 2.1 *(String Expansion) A run of a letter in a string x is a substring consisting of only copies of the letter. We define an expansion of x as a string x' that can be obtained from x by replacing some runs in x with longer runs.*

For example, if x = "abbccc", then "bb" is a run in x. And x' = "abbbbccccc" is an expansion of u for which the runs of b and c are extended.

Definition 2.2 *(DTW-Distance) Given two string x and y, a pair of strings (x', y') forms a correspondence between x and y if x' and y' are expansions of x and y of the same length, respectively. The cost of a correspondence (x', y') is the sum of the distances between the corresponding letters in x' and y'. We define the dynamic time warping distance $DTW(x, y)$ to be the minimum-cost of all the correspondences between x and y.*

Given two strings x and y, we only need to consider correspondences (x', y') where $|x'| = |y'| \le |x| + |y|$.

3 A Special DTW Problem

We present an algorithm for computing $DTW(u, x)$, in $O(m^2 l)$-time, where u is a regular string of length m and x is an RLE string consisting of l runs. This algorithm is asymmetric in its treatment of u and x, and its asymptotic advantage is only expected in a case where the length of u is bounded by the average length of a run in x.

Definition 3.1 *Given a string u_0, we say that u_1 and u_2 form a split of u_0 if u_1 and u_2 are a prefix and a suffix of u_0, respectively, and $|u_1| + |u_2| = |u_0|$ holds. This concept of a split can be generalized to a k-split for each natural number k if the requirement of $|u_1| + |u_2| = |u_0|$ is replaced with $|u_1| + |u_2| = |u_0| + k$. Clearly, a 0-split of a string is just a usual split.*

For example, if u_0 = "abcde", then u_1 = "abc" and u_2 = "cde" form a 1-split. In this paper, we are only interested in 0-splits and 1-splits.

Lemma 3.2 *Let u_0 be a regular string and x_1 be a run of a single letter. Then we can build a table for $DTW(u_1, x_1)$ in $O(m)$-time, where $m = |u_0|$ and u_1 ranges over the prefixes of u_0.*

For example, let u_0 = "abc" and x_1 = "aaaaa". Assume that $\delta(\mathtt{a}, \mathtt{b}) = 1$ and $\delta(\mathtt{a}, \mathtt{c}) = 2$. Then we have $DTW(\text{"a"}, x_1) = 0$, $DTW(\text{"ab"}, x_1) = 1$, and $DTW(\text{"abc"}, x_1) = 3$.

Proof Given a prefix u_1 of u_0, we have two possibilities.

- Assume that $|u_1| > |x_1|$ holds. Then $DTW(u_1, x_1)$ equals the sum of $\delta(u_1[i], a)$ for i ranging from 0 to $|u_1| - 1$, where a is the letter appearing in x_1.
- Assume that $|u_1| \leq |x_1|$ holds. Then $DTW(u_1, x_1)$ equals the sum of $\delta(u_1[i], a)$ plus $d \cdot (|x_1| - |u_1|)$ for i ranging from 0 to $|u_1| - 1$, where d is the distance between a and any letter in u_1 that is the closest to a among those in u_1.

It is clear that we can build a table in $O(m)$-time for $DTW(u_1, x_1)$ (where u_1 ranges over all the prefixes of u_0) since it takes only $O(1)$-time to compute the next entry of the table given the current entry plus some accumulated information. **Q.E.D.**

Lemma 3.3 *Let u_0 and x_0 be two strings, and let x_1 and x_2 form a split of x_0. Then there are strings u_1 and u_2 which form either a 0-split or 1-split of u_0 such that $DTW(u_0, x_0) = DTW(u_1, x_1) + DTW(u_2, x_2)$.*

For example, let u_0 = "abc" and x_0 = "aabbcc". Then $DTW(u_0, x_0) = 0$. Suppose x_1 = "aa" and x_2 = "bbcc". Then u_1 = "a" and u_2 = "bc" forms a 0-split of u_0 such that $DTW(u_1, x_1) = DTW(u_2, x_2) = 0$. Now suppose x_1 = "aab" and x_2 = "bcc". Then u_1 = "ab" and u_2 = "bc" form a 1-split of u_0 such that $DTW(u_1, x_1) = DTW(u_2, x_2) = 0$.

Proof By defintion, we have a correspondence (u'_0, x'_0) such that u'_0 and x'_0 are expansions of u_0 and x_0 of the same length, respectively, and $DTW(u_0, x_0)$ is the cost between u'_0 and x'_0. Note that x'_0 is the concatenation of some strings x'_1 and x'_2, which themselves are some expansions of x_1 and x_2 respectively. Let u'_1 and u'_2 be the prefix and suffix of u'_0 that correspond to x'_1 and x'_2 in x_0, repectively. In other words, we have $|u'_1| = |x'_1|$ and $|u'_2| = |x'_2|$. Clearly u'_1 is an expansion of some prefix in u_0. Let u_1 of the maximal prefix of such. Similarly, let u_2 be the maximal suffix of u_0 such that u'_2 is an expansion of u_2. We have two cases:

- The last letter of u_1 is different from the first letter of u_2. Then u_1 and u_2 form a 0-split of u_0.
- The last letter of u_1 is the same as the first letter of u_2. Then u_1 and u_2 forms a k-split of u_0 for some $k \geq 0$. If $k \geq 2$ holds, we can always trim some letters from the end of u_1 to ensure that u_1 and u_2 to form a 1-split.

Q.E.D

We first outline as follows an algorithm for computing $DTW(u_0, x_0)$, where u_0 is a regular string of length m and x_0 is a RLE-string consisting of n runs.

Assume that x_0 is not empty. We split x_0 into x_1 (a prefix of x_0 consisting of just one run) and x_2 (a suffix of x_0). By Lemma 3.3, for some prefix u_1 and suffix u_2 that form either a 0-split or a 1-split of u_0, we have:

$$DTW(u_0, x_0) = DTW(u_1, x_1) + DTW(u_2, x_2)$$

We create a memoization table M of size m where each $M[i]$ stores the value of $DTW(u_1, x_1)$ for the prefix u_1 containing the first i letters of u_0. By Lemma 3.2, we can build M in $O(m)$-time.

For each u_1, there are two possibilities for u_2 as u_1 and u_2 form either a 0-split or a 1-split of u_0. Let m and l be $|u_0|$ and $||x_0||$, respectively. Clearly, There are ml subproblems of the form $DTW(u_2, x_2)$ as there are m possibilities for u_2 and l possibilities for x_2. And each subproblem can be solved in $O(m)$-time (with dynamic programming) since the essential work is just to find the minimum of $2m$ sums of the form $DTW(u_1, x_1) + DTW(u_2, x_2)$. Therefore, our algorithm runs in $O(m^2 l)$-time.

Theorem 3.4 *Given a string u_0 and a RLE-string x_0, $DTW(u_0, x_0)$ can be computed in $O(m^2 l)$-time, where $m = |u_0|$ and $l = ||x_0||$.*

The standard algorithm for computing $DTW(u, x)$ (based on dynamic programming) is $O(ml)$-time, where $m = |u|$ and $l = ||x||$. In order for our specialized DTW algorithm to have the time-complexity $O(mn)$, we need $m^2 l$ to be $O(mn)$, which means that m is $O(n/l)$. Note that n/l is the average length of a run in x.

For our specialized DTW algorithm to be competive when compared with the DTW algorithm on RLE-strings [FJRW20], we need $m^2 l$ to be $O(k^2 l + kl^2)$. Since $k \leq m$, this requirement implies that $k \leq l$. Therefore, $m^2 l$ needs to be $O(kl^2)$, which in turn implies that m^2 is $O(kl)$. In the case where $m = k$, our specialized DTW algorithm is of a strictly lower time bound if m is $o(l)$. In particular, we have the following corollary:

Corollary 3.5 *Assume that u_0 is a string of length m and x_0 is a RLE-string such that $|x_0| = n$ and $||x_0|| = l$. If l is $O(n^{1/2})$ and m is $O(n^{1/2-\alpha})$ for some $0 < \alpha < 1/2$, then our specialized DTW algorithm for computing $DTW(u_0, x_0)$ is $O(n^{3/2-2\alpha})$.*

Note that the standard algorithm (based on dynamic programming) for computing $DTW(u_0, x_0)$ is $O(n^{3/2-\alpha})$. And the algorithm given in [FJRW20] for computing $DTW(u_0, x_0)$ is also $O(n^{3/2-\alpha})$ in this case.

4 Two Special Variants

4.1 DTW on Strings of Long Runs

When establising Theorem 3.4 for computing $DTW(u_0, x_0)$, we need to exhaustively try all the prefixes u_1 in u_0. Natually, we attempt to identify scenarios where only some of the prefixes of u_0 need to be tried. We report one scenario of such in the following result:

Theorem 4.1 *Let u_0 and x_0 be two RLE-strings such that $|u_0|$ is bounded by the length of each run in x_0. Then $DTW(u_0, x_0)$ can be computed in $O(k^2 l)$-time, where $k = ||u_0||$ and $l = ||x_0||$.*

4.2 DTW on Strings of String-Letters

Definition 4.2 *Let us fix a set W of strings and refer to each $w \in W$ a string-letter. We use W^* for strings that are formed by concatenating string-letters. We may refer to each string in W^* as a string-string.*

For example, suppose that "`ab`", "`abc`", "`cde`", and "`de`" are string-letters. Then "`abcde`" is a concatenation of the string-letters "`ab`" and "`cde`" and also a concatenation of the string-letters "`abc`" and "`de`".

A string-letter-encoded (SLE) string is just a sequence consisting of string-letters. And concatenating the string-letters in a SLE-string returns a string-string. We use x for a SLE-string as well and $||x||$ for the number of string-letters in x.

Theorem 4.3 *Let u_0 be a string and x_0 be a SLE-string. For any string-letter w_1 in x_0, assume that a table for $DTW(u_1, w_1)$ can be built in $O(m)$-time, where m is the length of u_0 and u_1 ranges over the substrings (not just prefixes) of u_0. Then $DTW(u_0, x_0)$ can be computed in $O(m^2 l)$-time, where l is the number of string-letters in x_0.*

One possible scenario is that there are only a fixed number of string-letters. For each string-letter w_1, the table for $DTW(u_1, w_1)$ can be built first, where u_1 ranges of the substrings (not just prefixes) of u_0. Then $DTW(u_0, x_0)$ for any SLE-strings x_0 can be computed in $O(m^2 l)$-time.

Corollary 4.4 *Let u_0 be a string of length m and x_0 be a SLE-string containing only string-letters from a set of $O(1)$-size. Also assume that the length of each string-letter in x_0 is $O(n^{1/2})$ and the number of string-letters in x_0 is $O(n^{1/2})$. Then $DTW(u_0, x_0)$ can be comupted in $O(m^2 n^{1/2})$-time.*

5 Conclusion

We give the presentation of a specialized asymmetric DTW algorithm.

References

[ABW15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for lcs and other sequence similarity measures. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 59–78, 2015.

[FJRW20] Vincent Froese, Brijnesh Jain, Maciej Rymar, and Mathias Weller. Fast exact dynamic time warping on run-length encoded time series, 2020.

[Kus19] William Kuszmaul. Dynamic time warping in strongly subquadratic time: Algorithms for the low-distance regime and approximate evaluation, 2019.

[Kus21] William Kuszmaul. Binary dynamic time warping in linear time, 2021.