Efficient search of the best warping window for Dynamic Time Warping

Chang Wei Tan[1] Matthieu Herrmann[1] Germain Forestier[2,1] Geoffrey I. Webb[1] François Petitjean[1]

[1]Faculty of IT, Monash University, Melbourne, Australia – firstname.lastname@monash.edu
[2]MIPS, University of Haute Alsace, Mulhouse, France – firstname.lastname@uha.fr

Abstract

Time series classification maps time series to labels. The nearest neighbor algorithm (NN) using the Dynamic Time Warping (DTW) similarity measure is a leading algorithm for this task and a component of the current best ensemble classifiers for time series. However, NN-DTW is only a winning combination when its meta-parameter – its warping window – is learned from the training data. The warping window (WW) intuitively controls the amount of distortion allowed when comparing a pair of time series. With a training database of N time series of lengths L, a naive approach to learning the WW requires $\Theta(N^2 \cdot L^3)$ operations. This often results in NN-DTW requiring days for training on datasets containing a few thousand time series only. In this paper, we introduce FastWWSearch: an *efficient* and *exact* method to learn WW. We show on 86 datasets that our method is always faster than the state of the art, with at least one order of magnitude and up to 1000x speed-up.

1 Introduction

Since its introduction in the 70s, Dynamic Time Warping (DTW) [16] has played a critical role for the analysis of time series, with hundreds (if not thousands) of papers published every year that make use of it. Many studies [1, 3, 10, 12, 13, 17, 19, 20] have shown that the One Nearest Neighbor Search with DTW (NN-DTW) outperforms most other algorithms when tested on the benchmark datasets [4], in spite of its simplicity.

In addition, the 2017 comprehensive benchmark of all time series classification methods [2] ranked COTE [3] as the most accurate classifier. COTE is an ensemble of classifiers – one of its base learners is NN-DTW with learned warping window (WW). It directly follows that the complexity of NN-DTW is a lower bound on COTE's complexity. This becomes ever more problematic as the size of the training data increases. Figure 1 shows that the state-of-the-art method UCR Suite takes more than a day to learn the best WW from 50,000 or more examples for this satellite image time series data. State-of-the-art methods LB_Keogh [7] and PrunedDTW [17] pass the day threshold with just

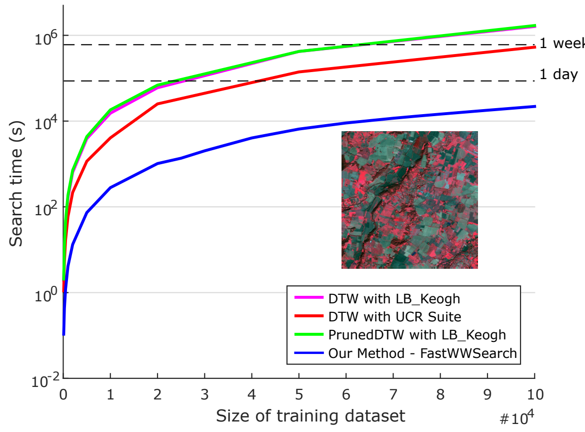

Figure 1: Training time for NN-DTW where the warping window is learned.

20,000 time series. The blue curve shows that our Fast Warping Window Search (FastWWSearch) method learns the best WW in just 2 hours for 50,000 time series and can learn WW for 100,000 time series in about 6 hours, a quantity of data that that is infeasible to process with the state of the art. With a training database of N time series of lengths L, a naive approach to learning the WW requires $\Theta(N^2 \cdot L^3)$ operations.

On the importance of the L^3 term. Figure 1 actually shows quite an optimistic picture, because this large dataset holds only very short series (with the length, $L = 46$), hence limiting the impact of the L^3 factor. For most datasets, L is typically ten times larger, which strongly influences runtime. We will see in Section 4 that speed-up can be up to 1000x for datasets with longer series. Note that this dataset comes from the problem of creating a temporal land-use map from a series of satellite images [11, 18].

On the importance of the N^2 term. It has been shown that for a specific task, the larger the training data available, the smaller the warping window [14]. One could thus wonder if there is a need for a fast technique, because one could limit the scope of the search for the best warping window to a small subset of

WW values. However, for large datasets, the N^2 term takes over and becomes too important to even consider testing a few values of WW.

In this paper, we propose FastWWSearch: a novel approach to speed up the learning process of the warping window for DTW. Our approach builds on the state of the art and introduces new bounds and exact pruning strategies with associated algorithms. FastWWSearch is always at least one order of magnitude faster than state-of-the-art methods, and with speed-ups that can reach 1000x for some datasets. In essence, this algorithm takes a systematic approach to filling a vector representing the nearest neighbor for each WW for each series in the training data. It searches efficiently and systematically to complete this vector, exploiting numerous bounds to avoid most computations. We release our code open-source to ensure reproducibility of our research, and to enable researchers and practitioners to directly use FastWWSearch as a subroutine in further algorithms, including in the state-of-the-art methods for classification: COTE [3] and EE [10].

This paper is organized as follows. In Section 2, we review some background and related work. Section 3 shows the intuition of our work and outlines our approach. Section 4 shows an evaluation of our method with the standard methods. Lastly, Section 5 concludes our work with some future work.

2 Background and Related Work

In this paper, we use $\mathcal{T} = \{T_1, \cdots, T_N\}$ to denote a training dataset of size N where all time series are of length L, the letters S and T to denote two time series, and $T(i)$ to denote the i-th element of T.

2.1 Dynamic Time Warping (DTW) was introduced in [15, 16]. As it has been presented numerous times in the literature, we simply define the elements that are the most critical for the understanding of this paper and refer the reader to [7] for more information. DTW uses dynamic programming to find an optimal alignment of two time series S and T; it solves Equation 2.1 where a cell (i, j) of the cost matrix $D^{S,T}$ accounts for all elements of S and T, up to i and j respectively, and where $\delta(\cdot,\cdot)$ is an L_p-norm. We then have $\mathrm{DTW}(S, T) = (D^{S,T}(L, L))^{1/p}$. Using dynamic programming, the alignment is solved in $\Theta(L^2)$.

$$D^{S,T}(i,j) = \delta(S(i),T(j)) + \min \begin{cases} D^{S,T}(i-1,j-1) \\ D^{S,T}(i,j-1) \\ D^{S,T}(i-1,j) \end{cases} \tag{2.1}$$

The warping path associated with $\mathrm{DTW}(S, T)$ is the sequence of minimum values taken consecutively by $D^{S,T}(\cdot,\cdot)$. We note such warping path $\vec{p} = \langle \omega_1, \cdots, \omega_K \rangle$ and illustrate it on an example in Figure 2b. A couple $\omega_k = (i, j)$ belonging to the warping path translates into an association $(S(i) - T(j))$ when aligned by DTW (illustrated in Figure 2a). Again here, as this has been covered in numerous papers, we refer the reader to [7] for more details about the warping path and its conditions (boundary, continuity and monotonicity).

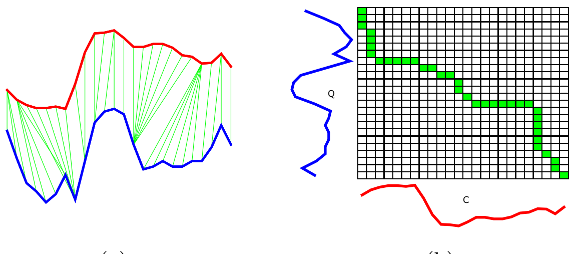

Figure 2: (a) DTW alignment for two time series. (b) Cost matrix D with warping path W (green)

2.2 Warping Window A warping window w is a global constraint on the re-alignment that DTW finds (originally called Sakoe-Chiba band [16]), such that elements of S and T can be mapped only if they are less than w elements apart, and we write $\mathrm{DTW}_w(S, T)$. Formally, this results in a warping path having as constraint $\forall (i, j) \in \vec{p}, |i - j| \leqslant w$. Figure A.1 (in supplementary material) illustrates such a constraint with $w = 3$ – the warping path is found within the gray band. Note that we have $0 \leqslant w \leqslant L - 1$, DTW_0 corresponds to the Euclidean distance, and DTW_{L-1} is equivalent to unconstrained DTW. This added constraint has two main benefits: (1) preventing pathological alignments (and thus increasing the accuracy of the classifier) and (2) reducing the time complexity of DTW from $\Theta(L^2)$ to $\Theta(w \cdot L)$. Note that other types of constraints have been developed in the literature, including the Itakura Parallelogram [6] and the Ratanamahatana-Keogh band [13]. In this paper, we focus on the warping window which, arguably, is by far the most used constraint in the literature [14, 20].

Why should the warping window be learned? The choice of the warping window (WW) has long been known to have a strong influence on accuracy [4, 14, 20]. One of many examples is the `CinC_ECG_Torso` dataset [4], for which using a learned window reduces the error-rate from 35% to 7% [4]. We illustrate in Figure 3 the importance of learning the warping window on some datasets. In an extensive set of experiments Bagnall *et al.* [1] demonstrated that DTW is only competitive when the warping window is set via cross-validation. It is important to note that learning WW can be

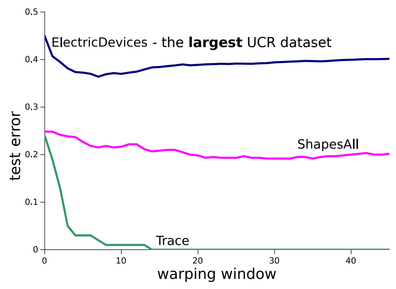

Figure 3: Test error on some datasets

critical even for large datasets. This is illustrated in Figure 3 with the largest dataset in the UCR archive – `ElectricDevices` – for which selecting an appropriate WW reduces error by 7 percentage points relative to Euclidean distance (DTW_0). Finally, it is important to realise that the two state-of-the-art ensembles for time series classification – COTE [3] and EE [10] – include NN-DTW with learned warping window as one of their constituents. The time complexity of learning the WW has become even more significant since Bagnall *et al.* [2] showed that COTE outperforms all existing methods for time series classification.

How do we learn the warping window? Learning of the warping window is not a simple problem; accepted methods in the field are all based on cross-validation: either on leave-one-out (LOO-CV) [1, 2] or on x-fold cross-validation [5]. In this paper, we focus on LOO-CV for the sake of clarity, with all our algorithms directly usable for x-fold cross-validation. In supplementary material, we illustrate the learning/search for LOO-CV as well as the algorithm for it Algorithm A.1.

2.3 Related work As learning the warping window involves leave-one-out cross-validation, the task boils down to being able to find the nearest neighbor of each time series in the training dataset (within the training dataset excluding themselves). We review below the state-of-the-art methods to perform this task efficiently.

Silva *et al.* [17] proposed PRUNEDDTW to speed up DTW computation itself. They first compute an upper bound and skip the cells of the cost matrix D that are larger. The authors were able to learn the optimal window size faster than the naive method [17]. However, as we will show in the experiments, the improvement for warping window search is only minimal.

Rakthanmanon *et al.* [12] proposed the UCR SUITE, which includes 4 optimization techniques: early abandoning, reordering early abandoning, reversing query and data roles in LB_KEOGH (see Section 2.4), and cascading lower bounds. Although they did not directly use their method to learn the warping window, it is natural to repurpose it for this task.

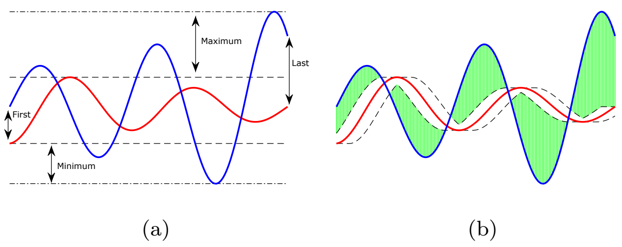

Figure 4: (a) KIM and (b) KEOGH lower bound

However, as shown in Figure 1 (and described later in Section 4), those methods do not scale well for large datasets. This is mostly because they only try to tackle the impact of the length L on the $O(N^2 \cdot L^3)$ complexity. We will see that our FASTWWSEARCH method tackles both the impacts of L and N.

2.4 DTW Lower Bounds Learning the warping window via cross-validation involves being able to efficiently find the neighbor of a time series within the training dataset, i.e. to perform a NN-DTW query for each time series. Lower-bounds to DTW have long been used in this context, they allow us to avoid the (expensive) DTW calculation if it is not needed.

Several bounds have been introduced including LB_KIM [8], LB_KEOGH [7] and LB_IMPROVED [9] (see Figure 4). Lower bounds can also be used in cascade, starting by the looser (and computationally cheap) one and progressing towards tighter lower bounds [12]. Algorithm A.2 in supplementary material shows a simple nearest neighbor search with lower bounds and can easily be modified for cascading lower bounds by changing the LB function in line 3.

3 Fast Warping Window Search for DTW

In this section, we introduce our approach: FASTWWSEARCH. In the first subsection, we start by introducing the mathematical properties that constitute the basis for our algorithm, which we introduce in the second subsection.

It is interesting to start by noting that, to find the best warping window via cross-validation, it is sufficient to know the nearest neighbor T of each time series S ($T \in \mathcal{T} \setminus S$) for each value of the warping window we want to test. The naïve (and almost state-of-the-art) algorithm for finding the best warping window is given in Algorithm A.1 in supplementary material. In this algorithm, we can observe that the loops are independent. The aim of this paper is to make the most of the inter-relation between the iterations of these two loops. As mentioned earlier, we focus our explanation on LOO-CV, but our method is directly extensible for any type of x-fold cross-validation.

3.1 Properties for FastWWSearch

Property #1: Warping path can be valid for several windows

THEOREM 3.1. *Let S, T be two time series, w_1 and w_2 two warping windows, and $\vec{p}_{w_1}$ and $\vec{p}_{w_2}$ their associated warping paths. $\vec{p}_{w_1} = \vec{p}_{w_2} \Rightarrow \text{DTW}_{w_1}(S,T) = \text{DTW}_{w_2}(S,T)$. In other words, $\text{DTW}(S,T)$ can only differ if the warping path differs.*

Proof. *Let $\vec{p}_{w_1} = \langle (i_1^{w_1}, j_1^{w_1}), \cdots, (i_K^{w_1}, k_K^{w_1})\rangle$, $\vec{p}_{w_2} = \langle (i_1^{w_2}, j_1^{w_2}), \cdots, (i_K^{w_2}, k_K^{w_2})\rangle$. We have:*

$$\begin{aligned} \text{DTW}_{w_1}(S,T) &= \textstyle\sum_{k=1}^{K} \delta(S(i_k^{w_1}), T(j_k^{w_1})) && \textit{Eq } 2.1 \\ &= \textstyle\sum_{k=1}^{K} \delta(S(i_k^{w_2}), T(j_k^{w_2})) && \textit{(By hyp.)} \\ &= \text{DTW}_{w_2}(S,T) && \square \end{aligned}$$

THEOREM 3.2. *Let S, T be two time series, w a warping window, and $\vec{p}_w = \langle (i_1, j_1), \cdots, (i_K, k_K)\rangle$, then*

$$(|i_k - j_k| < w)_{\forall k} \Rightarrow \text{DTW}_w(S,T) = \text{DTW}_{w-1}(S,T)$$

In other words, if no point of a warping path 'touches' the extremity of the warping window at w, then the $\text{DTW}_w(S,T) = \text{DTW}_{w-1}(S,T)$.

Proof. *By definition, $\text{DTW}_w(S,T)$ finds the warping path $\vec{p}_w$ such that the $\sum_{k=1}^{K} \delta(S(i_k), T(j_k))$ is minimized respecting constraint $(|i_k - j_k| \leqslant w)$. Our additional requirement $(|i_k - j_k| < w)$ ensures that $(|i_k - j_k| \leqslant w - 1)$. It results that the warping paths are equal, and by Theorem 3.1 so are the distances.* $\square$

In the following, we say that $\text{DTW}_w(S,T)$ has a "window validity" of $[\![v, w]\!]$ if all the warping paths $\vec{p}_w, \vec{p}_{w-1}, \cdots \vec{p}_v$ are the same and thus that the distances are all identical. We will see in Section 3.2 how we can use these theorem to prune many DTW computations, by starting with finding the NNs with larger warping window down to $w = 0$.

Property #2: DTW is monotone with w

THEOREM 3.3. *Let S, T be two time series, and w a warping window, we have $\text{DTW}_w(S,T) \leqslant \text{DTW}_{w-1}(S,T)$.* ***Proof.*** *Reductio ad absurdum: Assume $\text{DTW}_w(S,T) > \text{DTW}_{w-1}(S,T)$, then this means that there exists a warping path $\vec{p}_{w-1}$ such that the associated cost is lower than the one for $\vec{p}_w$. This translates in DTW not having found the optimal solution at window w, which is a contradiction [16].* $\square$

Figure 5 illustrates the combination of Theorem 3.1, 3.2 and 3.3, by showing the value of $\text{DTW}_w(S, T_{\{1,2,3\}})$ as a function of w. It shows that DTW decreases monotonically with w (Theorem 3.3), while the flat sections on the right section of the plot illustrate Theorems 3.1 and 3.2. Figure 5 also shows that the path computed for $w = 24$ remains valid down to $w = 1$ for T_1.

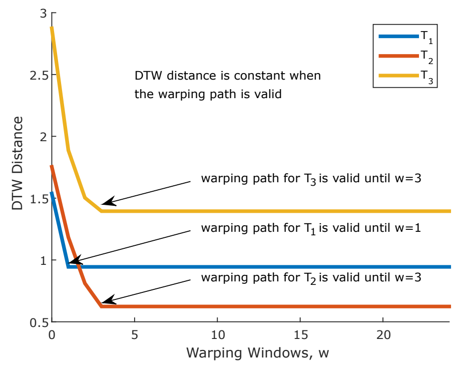

Figure 5: DTW distance at different w

Property #3: LB_Keogh is monotone with w

THEOREM 3.4. *Let S, T two time series, and w a warping window, we have $\text{LB\_KEOGH}_w(S,T) \leqslant \text{LB\_KEOGH}_{w-1}(S,T)$.*

Proof. *Let us define U^T and L^T as the upper and lower envelopes for any time series T, such that the i^{th} element of the envelopes are defined as $U_w^T(i) = \max(T(i-w) : T(i+w))$ and $L_w^T(i) = \min(T(i-w) : T(i+w))$ [7]. Reductio ad absurdum using [7, Eq. 9]:*

$$\begin{aligned} &\text{LB\_KEOGH}_w(S,T) > \text{LB\_KEOGH}_{w-1} \\ &\Rightarrow \sum_{i=1}^{L} \begin{cases} U_w^T(i) & \textit{if } S(i) > U_w^T(i) \\ L_w^T(i) & \textit{if } S(i) < L_w^T(i) \end{cases} < \\ &\quad\ \sum_{i=1}^{L} \begin{cases} U_{w-1}^T(i) & \textit{if } S(i) > U_{w-1}^T(i) \\ L_{w-1}^T(i) & \textit{if } S(i) < L_{w-1}^T(i) \end{cases} \end{aligned}$$

which contradicts the definition of lower and upper envelopes that gives $U_w^T(i) \leqslant U_{w-1}^T(i)$ and $L_w^T(i) \geqslant L_{w-1}^T(i), \forall i$. $\square$

Intuitively, the smaller the window, the closer the envelopes are to the reference time series. This then results to a larger lower bound (the green part in Figure 4b). We will see in Section 3.2 that we use LB_KEOGH_{w+1} as themselves lower bounds to LB_KEOGH_w.

3.2 The FastWWSearch algorithms

We have presented the theoretical basis for our work; we now proceed with our algorithm.

Intuition behind FastWWSearch Learning the warping window can be seen as creating a $(N \times L)$-table, illustrated in Table 1, giving the nearest neighbor (NN) of every time series for all windows. Once that table is filled, the best value of the window can be learned in one pass over it. We can reframe the aim of FASTWWSEARCH as the construction of such an $(N \times L)$-table. Filling it naively, i.e. by computing

	Nearest neighbor at warping windows				
	0	1	$\cdots$	$L-2$	$L-1$
T_1	$T_{24}(2.57)$	$T_{55}(0.98)$	$\cdots$	$T_{55}(0.98)$	$T_{55}(0.98)$
$\vdots$			$\vdots$		
T_N	$T_{60}(4.04)$	$T_{47}(1.61)$	$\cdots$	$T_{47}(1.61)$	$T_{47}(1.61)$

Table 1: Table of NNs for each WW. A cell $(i, w) = T_k(dist)$ means T_i has T_k as its NN for window w with distance $dist$.

DTW for each nearest neighbor $\text{NN}(i, w)$ using DTW only, requires $\Theta(N^2 \cdot L^3)$ operations. Note that, an exhaustive search of all L warping windows for large L and N can be extremely computationally demanding. Thus, most practitioners settle with a subset of WW [10]. Our method applies to either all or a subset of the possible warping windows, simply starting from the largest and scanning through the windows down to the smallest.

Until recently, most of the research had focused on finding bounds for a fixed value of a warping window. Our method will explore bounds across columns of this table. Section 3.1 gives us two additional lower bounds that we use to prune potential nearest neighbors before we have to compute DTW_w:

Lower Bound name	Complexity
LB_KIM	$\Theta(1)$
LB_KEOGH$_{w+k}$	$\Theta(L)$
LB_KEOGH$_w$	$\Theta(L)$
DTW$_{w+k}$	$\Theta(w \cdot L)$

Note that we also use the fact that LB_KEOGH is not symmetrical in FASTWWSEARCH below. To take advantage of Theorems 3.3 and 3.4, we order our computations by *decreasing window size*. Our algorithm iterates from larger values of windows down to smaller ones. This has the consequence of obtaining either LB_KEOGH$_{w+k}$ or DTW$_{w+k}$ 'for free' when assessing window w (if pruning at step $w + k$ wasn't possible with LB_KIM), because those will have been calculated in a previous step with a higher w. Of course, we will see that these bounds should only be used if they have already been computed; there is indeed no point in computing DTW$_{w+k}$ to potentially prune DTW$_w$ of which the value is less expensive to compute.

Moreover, ordering our computation by decreasing window size also allows us to make the most of Theorem 3.1 and 3.2. Referring to Figure 5, the long flat tails correspond to large validity windows for DTW$_{L-1}$. It means that, in that flat section (and any subsequent other), no bounds are at all necessary, because we already know that the value of DTW (previously computed) has not changed. This element actually has two important consequences.

First imagine that we did find the nearest neighbor for a time series at window $w = L$; this implies that we had to calculate DTW$_L$ for those 2 series. If the warping path is valid down to $w = 0$, then a consequence of our Theorem 3.1, 3.2 and 3.3 is that we *know* that this will also be its nearest neighbor down to $w = 0$, and this without any additional calculations.

Second, when calculating an actual DTW$_w(S, T)$, even if the candidate T does not become the nearest neighbor of S at w, we know nonetheless that we do not need to recompute DTW$_{w'}(S, T)$ for all windows w' such that the warping path is valid (see Theorem 3.1 and 3.2).

Lazy Nearest Neighbor Assessment We now have all the elements necessary to the presentation of our FASTWWSEARCH algorithm. We start by presenting LAZYASSESSNN in Algorithm 1. LAZYASSESSNN is a function that, given a pair of time series (S, T), establishes if they can be less than a given distance d apart for a warping window w. It functions in a lazy fashion, by making the most of all possible bounds that we presented in Section 3.2. The algorithm tries lower bounds of increasing complexity until one of two things happen: (1) either a lower bound or DTW$_w(S, T)$ itself is greater than d, in which case the procedure aborts; or (2) we have DTW$_w(S, T) < d$, and we have actually calculated DTW$_w(S, T)$.

LAZYASSESSNN is lazy in that it postpones calculations for as long as it is possible to do so. As we will see next, one has to imagine here that LAZYASSESSNN will be called several times for the same pair of time series (S, T) for decreasing values of w. When w decreases, any value that was previously calculated for a larger window, becomes a lower bound for the current w. We use a cache $\mathcal{C}_{(S,T)}$ to store the previous results of LAZYASSESSNN obtained for larger w.

We first start by checking if the cache has been initialized; if not we compute LB_KIM, which is valid regardless of the warping window (line 1). On line 2, we then test where the cache last stopped, i.e. was it computing a lower bound for that target window w, was it computing a lower bound for another window $w' > w$, or was it computing an actual *DTW* distance (that might be still valid). We then assess if it last stopped having had calculated DTW for a larger window (lines 3–5); if that DTW$_{w'}$ has a path that is still valid and its value is smaller than d, then we return that value of the distance. We then assess if it had calculated DTW$_{w'}$ that is still valid and smaller than d. It is important to observe here that the code terminates when a distance is larger than d or DTW$_w$ is computed. If we cannot prune with DTW$_{w'}$, we proceed and check if we are able to prune using previously computed bounds (lines

Algorithm 1: LazyAssessNN($\mathcal{C}_{(S,T)}, w, d, S, T$)

```
Input: $\mathcal{C}_{(S,T)}$: cache storing the previous
       measure between $S$ and $T$
Input: $w$: the warping window
Input: $d$: the distance to beat
Input: $S, T$: the time series to measure
Result: $\mathrm{DTW}_w(S,T)$ if $\geqslant d$, else pruned
1  if $\mathcal{C}_{(S,T)} = \varnothing$ then $\mathcal{C}_{(S,T)} \leftarrow$ LB_Kim$(S,T)$
2  switch $\mathcal{C}_{(S,T)}$.stoppedAt do
       // LB calculated with larger window $w'$
3      case $\mathrm{DTW}_{w'}$ do
4          if $w \in \mathcal{C}_{(S,T)}$.valid $\wedge\, \mathcal{C}_{(S,T)}$.value $< d$ then
5              return $\mathcal{C}_{(S,T)}$.value
6      case LB_Kim or LB_Keogh$_{w'}$ do
7          if $\mathcal{C}_{(S,T)}$.value $\geqslant d$ then return pruned
       // Cascading LB_Keogh and DTW
8      otherwise do
9          $\mathcal{C}_{(S,T)} \leftarrow$ LB_Keogh$_w(S,T)$
10         if $\mathcal{C}_{(S,T)}$.value $\geqslant d$ then return pruned
11         $\mathcal{C}_{(S,T)} \leftarrow$ LB_Keogh$_w(T,S)$
12         if $\mathcal{C}_{(S,T)}$.value $\geqslant d$ then return pruned
13         $\mathcal{C}_{(S,T)} \leftarrow \mathrm{DTW}_w(S,T)$
14         if $\mathcal{C}_{(S,T)}$.value $\geqslant d$ then return pruned
15         return $\mathcal{C}_{(S,T)}$.value
```

6–7). Otherwise from lines 8 to 12, we use cascading lower bounds, testing if we can prune after each of them. Finally, if all the bounds fail to prune T, we compute DTW_w, store the results and if $\mathrm{DTW}_w < d$, T is the new NN for S (line 13). We will see in our main algorithm that the next call to LazyAssessNN for the pair (S,T) will be with a smaller w.

Although our scheme *does* make it possible to use Early Abandon on LB_Keogh [12] and to use LB_Improved [9], we disregarded them after observing that they both increased computation time: early abandoning because it significantly increases the number of times the function has to be restarted; LB_Improved because it requires to compute a projection of a series onto the other, which has an additional cost not justified by the added pruning power in our case.

We take the bounds ordered by computational complexity, which in practice usually correlates with tightness [12]. $\mathrm{DTW}_{w'}$ will also generally be tighter than LB_Keogh$_w$, especially when w' is close to w.

Our core FastWWSearch algorithm

We now turn to our core algorithm: FastWWSearch. Let us recall that the central aim of our algorithm is to build an $(N \times L)$-table that contains the nearest neighbor of each time series (out of N) and for each value of the warping window (out of L) – illustrated in Table 1. Once this table has been calculated, one pass over it is sufficient to determine the best value of the warping window. That pass is the entry point to FastWWSearch and is presented in Algorithm 2. It assumes there exists a method to fill the table and returns the warping window with the lowest leave-one-out cross-validation error on $\mathcal{T}$. The result of Algorithm 2 is identical to the state-of-the-art presented in supplementary material – Algorithm A.1 – obviously assuming that the $(N \times L)$-table is calculated correctly, which is illustrated with our fail-safe experiments in Section B of the supplementary material.

Algorithm 2: FastWWSearch

```
Data: $\mathcal{T}$: training data
Result: $w^\star$: best warping window
1  NNs $\leftarrow$ FastFillNNTable$(\mathcal{T})$
2  bestNErrors $\leftarrow |\mathcal{T}| + 1$
3  for $w \leftarrow 0$ to $L-1$ do
4      nErrors $\leftarrow 0$
5      foreach $T_t \in \mathcal{T}$ do
6          if NNs$[t][w]$.class $\neq T$.class then nErrors++
7      if nErrors $<$ bestNErrors then
8          bestNErrors $\leftarrow$ nErrors
9          $w^\star \leftarrow w$
```

Obviously, the core of our approach resides in how we calculate this table, which we present in Algorithm 3. At the highest level, our algorithm works by building this table for a subset $\mathcal{T}' \subseteq \mathcal{T}$ of increasing size, until $\mathcal{T}' = \mathcal{T}$. We start by building the table for $TSet'$ comprising only 2 first time series T_1 and T_2, and fill this $(2 \times L)$-table as if $TSet'$ was the entire dataset. At this stage it is trivial that T_2 is the nearest neighbor of T_1 and vice versa. We then add a third time series T_3 from $\mathcal{T} \setminus \mathcal{T}'$ to our growing set $\mathcal{T}'$. At this point, we have to do two things: (a) find the nearest neighbor of T_3 within $\mathcal{T}' \setminus \{T_3\} = \{T_1, T_2\}$ and (b) check if T_3 has become the nearest neighbor of T_1 and/or T_2. We can then add a fourth time series T_4 and so on until $\mathcal{T}' = \mathcal{T}$.

We now describe Algorithm 3 line by line. We start by initializing the NNs $(N \times L)$-table to $(\_, +\infty)$, which means that the table is empty and the distances are thus $+\infty$ (line 2). We then initialize $\mathcal{T}'$ in line 3. Line 4: we start the iteration at 2 as the definition of NN only makes sense if there is at least 2 time series. We then proceed with some initializations (lines 5–7), including for the cache associated with S (line 7). We are then ready to find (a) the NN of S within $\mathcal{T}'$, and (b) update the NN for all $T \in \mathcal{T}'$ now that S has been added. We will do this operation for all w in descending order, starting with the largest value $L-1$ (line 8). Note that to only assess a subset of all possible L values for w, one only need to modify this line; our only requirement is for the values to be taken in descending order.

Line 9: we start by checking if we already have a NN for S from previous (i.e. larger) windows. Note that NNs$[s][w]$ here comes compulsorily from $DTW_{w',w'>w}(S,\_)$ for which its path is still valid. We then already have the NN of S and only need to check whether S has also become a NN for other time series in $\mathcal{T}'$ (lines 10–14). To this end, we get the previously calculated NN for such $T \in \mathcal{T}'$ (line 11), which we will use as the threshold of distance that we have to 'beat' (i.e. be smaller than) for S to become the NN of T. On line 12 we then call our LazyAssessNN function to assess if S has actually become the NN of T. If LazyAssessNN exits with `pruned`, it then means that S is not the NN of T, and thus that the previous NN of T is still valid, hence nothing has to be done. LazyAssessNN only exits with something else than `pruned` if $\text{DTW}_w(S,T) <$ toBeat, which means that S has indeed become the NN of T; we update this information on line 14.

The `else` case starting on line 15 is if we didn't already have the NN of S from a previous window. We will then analyze all couple $(S,T)_{T\in\mathcal{T}'}$ and perform the NN update for S and T simultaneously. At this stage, it is possible that we will already have – from previous windows – some information about which $T \in \mathcal{T}'$ might be a better candidate to be the NN of S. This information is stored in the cache $\mathcal{C}$, which may contains different types of lower bounds. The number of calculations will be minimized if the very first T is actually the NN of S, because it will give the tightest possible pruning threshold first. This is why we should first examine the time series that have highest potential to become the NN of S and order the time series.[1]

At line 17, we obtain the distance threshold from NNs$[s][w]$; the first time, we will have NNs$[s][w] = \varnothing$ which is associated with a value of $+\infty$, there will thus be computation of $\text{DTW}_w(S,T)$, which will later on be stored in NNs$[s][w]$ (line 20). From the second T, NNs$[s][w]$`.distance` stores the distance to the so-far NN of S; we use LazyAssessNN to prune candidates or replace the current one if it is better (lines 19–20). We then proceed by checking if S is the NN of T on lines 21–24 (same as lines 11–14). Finally on line 25,

[1]Ranking LB_Keogh lower bounds has been previously studied in [18]. Here, we however have different types of lower bounds to interlace. LB_Kim often has a smaller value than LB_Keogh simply because it only looks at 4 elements of series (vs L for LB_Keogh). In addition, because $\text{DTW}_{w',w'>w}$ has tried to align S and T, it will probably represent a better estimate of DTW_w than LB_Keogh. To reflect this, we rank the time series in $\mathcal{T}'$ using LB_Kim$/4$, LB_Keogh$/L$, $0.8 \cdot$ DTW$/L$ (the 0.8 factor is used to push DTWs forward when close to the LB_Keogh values).

Algorithm 3: FastFillNNTable($\mathcal{T}$)

```
   Input: $\mathcal{T}$ the set of time series
   Result: NNs$[N][L]$ the nearest neighbors table
1  Define LANN as LazyAssessNN
2  NNs.fillAll$(\_,+\infty)$
3  $\mathcal{T}' \leftarrow \emptyset$
4  for $s \leftarrow 2$ to $N$ do
       // We want to update NNs wrt adding $S$
5      $S \leftarrow \mathcal{T}_s$
6      $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{\mathcal{T}_{s-1}\}$
7      foreach $T \in \mathcal{T}'$ do $\mathcal{C}_{S,T} \leftarrow \varnothing$
8      for $w \leftarrow L-1$ down to $0$ do
           // If we already have NN of $S$ for $w$
9          if NNs$[s][w] \neq \varnothing$ then
               // Update table NNs$[t][w]_{1\leqslant t\leqslant s-1}$
10             for $t \leftarrow 1$ to $s-1$ do
11                 toBeat $\leftarrow$ NNs$[t][w]$.distance
12                 res $\leftarrow$ LANN$(\mathcal{C}_{(S,T_t)}, w, \text{toBeat}, S, T_t)$
13                 if $res \neq$ pruned then
14                     NNs$[t][w] \leftarrow (S, \text{res})$
15         else
               // Check $S$ against previous $T \in \mathcal{T}'$
16             foreach $T \in \mathcal{T}'$ in asc. order using $\mathcal{C}$ do
17                 toBeat $\leftarrow$ NNs$[s][w]$.distance
18                 res $\leftarrow$ LANN$(\mathcal{C}_{(S,T)}, w, \text{toBeat}, S, T)$
19                 if $res \neq$ pruned then
20                     NNs$[s][w] \leftarrow (T, \text{res})$
                   // Update NNs$[t][w]$ if needed
21                 toBeatT $\leftarrow$ NNs$[t][w]$.distance
22                 resT $\leftarrow$ LANN$(\mathcal{C}_{(S,T)}, w, \text{toBeatT}, S, T)$
23                 if $resT \neq$ pruned then
24                     NNs$[t][w] \leftarrow (S, \text{resT})$
               // Propagate NN for path validity
25             for $w' \in$ NNs$[s][w]$.valid do
                   NNs$[s][w'] \leftarrow$ NNs$[s][w]$
```

having processed all $T \in \mathcal{T}'$, NNs$[s][w]$ contains the actual NN of S at w, and we propagate this information for all $w', w' < w$ for which the warping path is also valid. Note that the cache $\mathcal{C}$ is never reused once the row NNs$[s]$ is computed, which makes it possible for our algorithm to have $\Theta(N)$ memory complexity.

4 Empirical Evaluation

This section describes the experiments that evaluate our FastWWSearch method. To facilitate others to build on our work, as well as to ensure reproducibility, we have made our code available open-source at https://github.com/ChangWeiTan/FastWWSearch and the full raw results at http://bit.ly/SDM18.

We compare FastWWSearch's ability to learn the warping window compared to the state of the art:

- **LB_Keogh** [7]: It searches for the best warping window as described in Algorithm A.1 (sup. material) using LB_Keogh as LB.
- **UCR Suite** [12]: It is the state of the art for fast NN-DTW and uses cascading lower bounds to replace LB in Algorithm A.1.
- **LB_Keogh–PrunedDTW**: The PrunedDTW algorithm [17] was introduced to speed up the calculation of DTW using upper bounds (instead of lower bounds). It assesses warping windows in ascending order and uses the results for a smaller w as the upper bound for the larger w. Note that we actually improve here on the original paper [17] by adding LB_Keogh to the search mechanism.
- **UCR Suite–PrunedDTW:** To make the comparison as fair as possible, we propose to combine the power of UCR Suite's lower bounding and of PrunedDTW'upper bounding. Again, this method is an improvement on both methods.

We performed our experiments using all of the 85 freely available benchmark UCR time series datasets [4] and use the original train/test split from [4]. Note that these datasets are provided z-normalized but our method is directly applicable to unnormalized series. We perform an exhaustive search for all methods; as mentioned in Section 3.2, all methods are directly applicable to any subset of $[\![0, L-1]\!]$ that one might want to use instead of the full set. All methods have linear memory complexity, so we were able to conduct all experiments on a small machine (64-bit Linux with AMD Opteron 63xx Class CPU @1.80GHz and 6GB RAM). As the ordering of the time series in $\mathcal{T}$ affects all compared methods. the time to search for the best warping window, we report the average results over 10 runs for different reshuffles of $\mathcal{T}$. We report the full leave-one-out cross-validation running time.

All methods are exact: they all learn the same value of the warping window, and thus all return the same accuracy, which is why we focus on running time. A fail-safe check is presented in Section B (supplementary material) and shows that all methods indeed learn the same warping window.

4.1 Speed-up Figure 6 shows the scatter plot of learning time for our FastWWSearch method (x-axis) vs all competitors (y-axis). This is a clear-cut and significant result: our method is faster than the state of the art for all datasets (above the line means that our method is faster). There is also a slight upwards trend: as the task becomes more and more complicated, it seems that so is the improvement of

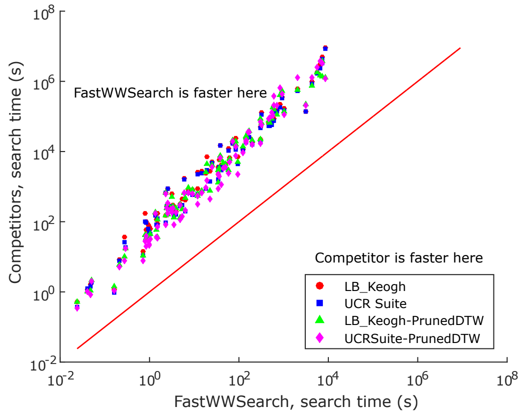

Figure 6: Average 10 runs results on the benchmark datasets (better seen in color)

our FastWWSearch method over the state of the art. For easy task requiring less than 10 seconds with the state of the art, FastWWSearch gains one order of magnitude and performs in less than 1 seconds. This makes sense and this is not where the gain is the most interesting. However, as the task becomes harder, so is the advantage of our method over others getting more important. This even reaches up to 3 orders of magnitude speed-up for very demanding tasks. For dataset `HandOutlines` for instance, the fastest state-of-the-art method requires 100 days ($9 \cdot 10^6$ s), while FastWWSearch only needs 2.5 hours ($9 \cdot 10^3$s).

It is also interesting to summarize the results depending on size of the data and length of the series. We calculated the average speed-up for datasets with smaller training size ($N \leqslant 200$) and larger training size ($N > 200$). The average speed-up for smaller datasets was of 106x, while it was of 184x for larger ones. Regarding length, we calculated the average speed-up for datasets with shorter series ($L \leqslant 300$) and longer ones ($L > 300$). The average speed-up for datasets with shorter series was of 67x, while it was of 250x for longer ones. These results confirm that our algorithm is tackling both the N and L terms.

4.2 Scalability to 100,000 time series It is now interesting to comment again on Figure 1 that was presented in the introduction. This dataset corresponds to 100,000 time series, associated to 100,000 'pixels' observed over time (over a series of satellite images) (more details in [11]). In this task, the aim is to establish the land-use of a geographic area based on its radiometric evolution over a series of satellite images. Time series are required because, for instance, one needs the temporal dynamic to distinguish between types of crops (when they grow and are harvested, how fast they grow, etc). Note that we used a computer with 16GB memory for this experiment to be able to store the data.

This dataset is also particularly interesting because the time series are short with $L = 46$, which tends to isolate the influence of N on the scalability. We can see in Figure 1 that when $N = 100$, FastWWSearch makes it possible to gain 1.5 orders of magnitude in running time over UCR Suite and 2 orders over the other state-of-the-art methods; and those gains seem very stable when N grows. For $N = 100,000$, FastWWSearch only requires 6 hours to complete, while the fastest state-of-the-art method – in this case UCR Suite – requires 7 days (and more than 18 days for the others). Note finally that, in Section C of the supplementary material, we study the question of whether incorporating PrunedDTW would make it faster and show that it has little influence on the results.

5 Conclusion

In this paper, we proposed FastWWSearch: a novel algorithm and underlying theory to efficiently learn the warping window for Dynamic Time Warping. Our experiments show that it is one to two orders of magnitude faster than the state of the art. This result is important both for the use of NN-DTW and also for incorporation into the state-of-the-art classifiers EE and Cote.

Acknowledgements

This work was supported by the Australian Research Council under grant DE170100037. This material is based upon work supported by the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AOARD) under award number FA2386-16-1-4023. The authors would like to thank Professor Eamonn Keogh, Hoang Anh Dau and anonymous reviewers for their comments on this paper.

References

[1] A Bagnall and J Lines, *An experimental evaluation of nearest neighbour time series classification. technical report #CMP-C14-01*, Department of Computing Sciences, University of East Anglia, Tech. Rep, (2014).

[2] A Bagnall, J Lines, A Bostrom, J Large, and E Keogh, *The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances*, Data Mining and Knowledge Discovery, 31 (2017), pp. 606–660.

[3] A Bagnall, J Lines, J Hills, and A Bostrom, *Time-series classification with COTE: the collective of transformation-based ensembles*, IEEE Transactions on Knowledge and Data Engineering, 27 (2015).

[4] Y Chen, E Keogh, B Hu, N Begum, A Bagnall, A Mueen, and G Batista, *The UCR Time Series Classification Archive*, 7 2015. www.cs.ucr.edu/~eamonn/time_series_data/.

[5] HA Dau, DF Silva, F Petitjean, A Bagnall, and E Keogh, *Judicious setting of DTW's warping window width allows more accurate classification of time series*, in IEEE Big Data Conference, 2017, pp. 1–4.

[6] F Itakura, *Minimum prediction residual principle applied to speech recognition*, IEEE Transactions on Acoustics, Speech, and Signal Processing, 23 (1975), pp. 67–72.

[7] E Keogh and CA Ratanamahatana, *Exact indexing of dynamic time warping*, Knowledge and information systems, 7 (2005), pp. 358–386.

[8] SW Kim, S Park, and WW Chu, *An index-based approach for similarity search supporting time warping in large sequence databases*, in IEEE ICDE, 2001, pp. 607–614.

[9] D Lemire, *Faster retrieval with a two-pass dynamic-time-warping lower bound*, Pattern recognition, 42 (2009), pp. 2169–2180.

[10] J Lines and A Bagnall, *Time series classification with ensembles of elastic distance measures*, Data Mining and Knowledge Discovery, 29 (2015), pp. 565–592.

[11] F Petitjean, J Inglada, and P Gançarski, *Satellite image time series analysis under time warping*, IEEE Transactions on Geoscience and Remote Sensing, 50 (2012), pp. 3081–3095.

[12] T Rakthanmanon, B Campana, A Mueen, G Batista, B Westover, Q Zhu, J Zakaria, and E Keogh, *Searching and mining trillions of time series subsequences under DTW*, in ACM SIGKDD, 2012, pp. 262–270.

[13] CA Ratanamahatana and E Keogh, *Making time-series classification more accurate using learned constraints*, in SIAM SDM, 2004.

[14] ———, *Three myths about DTW data mining*, in SIAM SDM, 2005, pp. 506–510.

[15] H Sakoe and S Chiba, *A dynamic programming approach to continuous speech recognition*, in International Congress on Acoustics, vol. 3, 1971, pp. 65–69.

[16] ———, *Dynamic programming algorithm optimization for spoken word recognition*, IEEE Transactions on Acoustics, Speech, and Signal Processing, 26 (1978).

[17] DF Silva and GE Batista, *Speeding up all-pairwise dynamic time warping matrix calculation*, in SIAM SDM, 2016, pp. 837–845.

[18] CW Tan, GI Webb, and F Petitjean, *Indexing and classifying gigabytes of time series under time warping*, in SIAM SDM, 2017, pp. 282–290.

[19] X Wang, A Mueen, H Ding, G Trajcevski, P Scheuermann, and E Keogh, *Experimental comparison of representation methods and distance measures for time series data*, Data Mining and Knowledge Discovery, 26 (2013), pp. 275–309.

[20] X Xi, E Keogh, C Shelton, L Wei, and CA Ratanamahatana, *Fast time series classification using numerosity reduction*, in ICML, 2006, pp. 1033–1040.