Ultra fast warping window optimization for Dynamic Time Warping

Chang Wei Tan
Department of Data Science and AI
Monash University
Melbourne, Australia
chang.tan@monash.edu

Matthieu Herrmann
Department of Data Science and AI
Monash University
Melbourne, Australia
matthieu.herrmann@monash.edu

Geoffrey I. Webb
Department of Data Science and AI
Monash University
Melbourne, Australia
geoff.webb@monash.edu

***Abstract*—The Dynamic Time Warping (DTW) similarity measure is widely used in many time series data mining applications. It computes the cost of aligning two series, smaller costs indicating more similar series. Most applications require tuning of DTW's Warping Window (WW) parameter in order to achieve good performance. This parameter controls the amount of warping allowed, reducing pathological alignments, with the added benefit of speeding up computation. However, since DTW is in itself very costly, learning the WW is a burdensome process, requiring days even for datasets containing only a few thousand series. In this paper, we propose ULTRAFASTWWSEARCH, a new algorithm able to learn the WW significantly faster than the state-of-the-art FASTWWSEARCH method. ULTRAFASTWWSEARCH builds upon the latter, exploiting the properties of a new efficient exact DTW algorithm which supports early abandoning and pruning (EAP). We show on 128 datasets from the UCR archive that ULTRAFASTWWSEARCH is up to an order of magnitude faster than the previous state of the art.**

***Index Terms*—Time Series, Dynamic Time Warping, Warping Window, Early Abandoning, Pruning**

I. INTRODUCTION

The Dynamic Time Warping (DTW) distance is the go-to similarity measure for a wide range of time series data mining tasks, including similarity search [1], classification [2]–[5], regression [6], clustering [7], [8], and motif discovery [9]. All these tasks rely on nearest neighbour (NN) search, and it is widely known that NN search is most effective when DTW is constrained by the right warping window (WW) [2], [5], [8], [10]. Indeed, the unconstrained DTW is subject to pathological warping, leading to unintuitive alignments [3] where a single point of a time series is aligned to a large section of another series [11].

Traditionally, learning the WW has been a time consuming leave-one-out cross-validation (LOOCV) process that requires computing DTW between every pair of training instances, for each WW [2], [8], [12]. This is only compounded by the quadratic time complexity of DTW. With a training dataset of N time series of length L, learning the WW naively requires $O(N^2.L^3)$ operations (Section II-C). This is extremely slow, even for datasets with only a thousand of time series. For instance, the naive approach took 58 hours to learn the best WW on the `StarLightCurves` dataset from the UCR archive [12], which only has 1,000 training instances with a length of 1024 (Fig. 1). In comparison, the state-of-the-art method, FASTWWSEARCH, took 30 minutes, while our proposed approach took only 4 minutes. Similar improvements can be seen across all the largest (in terms of $N \times L$) datasets from the UCR archive [12], shown in Fig. 1 (note log scale).

This work was supported by the Australian Research Council award DP210100072.

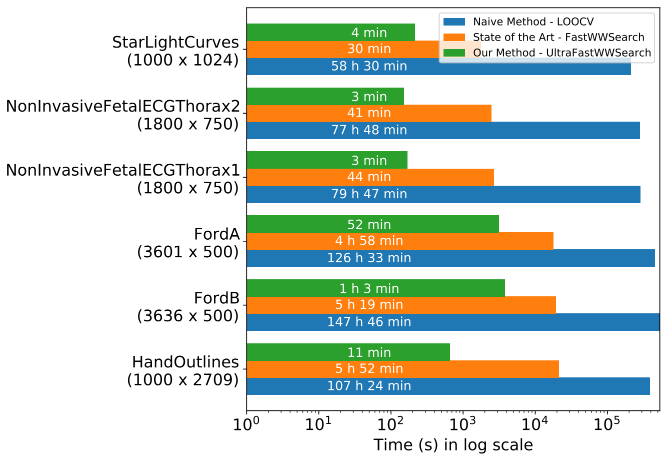


Fig. 1: ULTRAFASTWWSEARCH vs the naive LOOCV approach, and state-of-the-art FASTWWSEARCH. Total training time on the 6 largest datasets from [12] in terms of $N \times L$.

The current state-of-the-art FASTWWSEARCH is a sophisticated and intricate algorithm [3], exploiting the properties of DTW and its various lower bounds to achieve a three orders of magnitude speedup over the naive approach. Its significance is demonstrated by FASTWWSEARCH having received the SDM 2018 best paper award. Even-though FASTWWSEARCH achieves 1,000 times speedup compared to the traditional LOOCV approach, it is still undesirably slow for large datasets with long time series. This is shown in Fig. 1, where FASTWWSEARCH took almost 6 hours to train on the `HandOutlines` dataset, one of the largest and longest datasets from the UCR archive [12], compared to just 11 minutes for our proposed method.

Before FASTWWSEARCH, there were two primary strategies for speeding up LOOCV. One was by speeding up the NN search process, e.g. by using lower bounds to skip most of

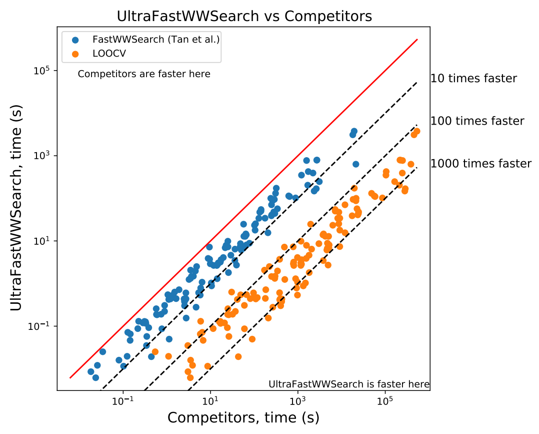


Fig. 2: Pairwise plot comparing UltraFastWWSearch to baseline methods on 128 UCR datasets. FastWWSearch is the current state of the art [3]. LOOCV is the standard method used to search for the best warping window [2].

DTW computations [1], [13]–[17]. The other was by speeding up the core computation of DTW, e.g. by approximating it [18] or pruning unnecessary operations [19]. But, scalability remains an issue for large datasets and long series [3], [20].

Recently, Herrmann and Webb [21] developed an efficient implementation strategy for six elastic distances, including DTW. This strategy, known as "Early Abandoned and Pruned" (EAP), relies on an upper bound beyond which the precise distance is not required. This is used to prune and early abandon the core computation of elastic distances. Nearest neighbor search naturally provides such an upper bound (Section III-A). EAP demonstrated more than an order of magnitude speedup for several NN-DTW search tasks.

In this paper, we propose UltraFastWWSearch, a new approach to learn the WW for DTW. We fundamentally transformed the FastWWSearch algorithm proposed in [3] to exploit the full capacity of EAP [21] and gaining a further substantial speedup by better exploiting a property called the *window validity*. Like FastWWSearch, UltraFastWWSearch is exact, i.e. produces the same results as the traditional LOOCV approach. It is, however, always faster – up to one order of magnitude – than FastWWSearch when tested on the 128 datasets from the UCR archive [12]. Fig. 2 demonstrates this by comparing UltraFastWWSearch to FastWWSearch and to the traditional LOOCV approach.

Similarly to FastWWSearch, UltraFastWWSearch systematically fills a table recording the nearest neighbor at each WW for each series in $\mathcal{T}$. However, it does so without the intricate cascading of DTW lower bounds that is critical to FastWWSearch. UltraFastWWSearch processes a new time series in a systematic order, minimizing the number of DTW computations while exploiting the strengths of EAP to speedup the required ones.. We release our code open-source[1] to ensure reproducibility, and to enable researchers and practitioners to directly use UltraFastWWSearch as a subroutine to tune DTW whatever their application.

[1]Source code available at https://github.com/ChangWeiTan/UltraFastWWS

We believe that UltraFastWWSearch serves as an important foundation to further speed up distance-based time series classification algorithms, such as the "Fast Ensemble of Elastic Distances" (FastEE) [4], that has fallen from favor due to its relatively slow compute time.

This paper is organised as follows. In Section II, we introduce some background and notation used in this work. We review some related work in Section III. Section IV describes UltraFastWWSearch in detail. Then we evaluate our method in Section V with the standard methods. Lastly, Section VI concludes our work with some future directions.

II. Background

We consider learning from a dataset $\mathcal{T} = \{\mathcal{T}_1, \cdots, \mathcal{T}_N\}$ of N time series where $\mathcal{T}_i$ are of length L. The letters S and T denote two time series, and T_i denotes the ith element of T.

A. Dynamic Time Warping

The DTW distance was first introduced in 1971 by Sakoe and Chiba [22] as a speech recognition tool. Since then, it has been one of the most widely used distance measure in NN search, supporting sub-sequence search [1], regression [6], clustering [7], [8], motif discovery [9], and classification [2], [4], [5], Nearest neighbour with DTW (NN-DTW) has been the historical approach to time series classification. The "Ensemble of Elastic Distances" (EE) [2], introduced in 2015, was one of the first classifiers to be consistently more accurate than NN-DTW over a wide variety of tasks. It relies on eleven NN classifiers including NN-DTW (and its variant with a warping window, cDTW, see below). EE opened the door to "ensemble classifiers", i.e. classifiers embedding other classifiers as components, for time series classification. Various recent and accurate ensemble classifiers such as HIVE-COTE [23], Proximity Forest [24], and TS-CHIEF [25] also embed both NN-DTW and NN-cDTW classifiers.

DTW computes in $O(L^2)$ the cost of an optimal alignment between two series (lower costs indicating more similar series) by minimising the cumulative cost of aligning their individual points. Equations 1a to 1d define the "cost matrix" M for two series S and T such that $M(i,j)$ is the minimal cumulative cost of aligning the first i points of S with the first j points of T. It follows that DTW(S,T)=$M(L,L)$.

$$M(0,0) = 0 \tag{1a}$$
$$M(i,0) = +\infty \tag{1b}$$
$$M(0,j) = +\infty \tag{1c}$$
$$M(i,j) = \mathrm{d}(S_i,T_j) + \min \begin{cases} M(i{-}1,j{-}1) \\ M(i{-}1,j) \\ M(i,j{-}1) \end{cases} \tag{1d}$$

Common functions for the cost of aligning two points are $\mathrm{d}(S_i,T_j) = |S_i,T_j|$ and $\mathrm{d}(S_i,T_j) = (S_i,T_j)^2$. In the current paper we use the latter. However, our algorithms generalize to any cost function.

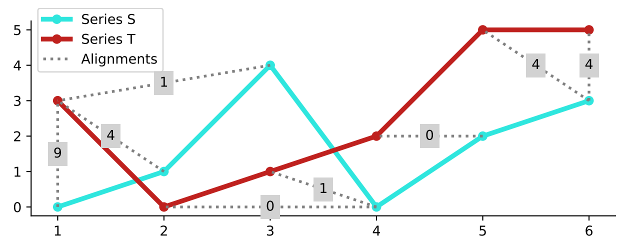


Fig. 3: Alignments associated to the warping path in Fig. 4a

The individual alignments (dotted lines in Fig. 3) form a "warping path" in the cost matrix (Fig. 4a). See how the vertical section of the path column 1 in Fig. 4a corresponds to the first point of T being aligned thrice in Fig. 3.

B. Warping window

DTW is usually associated with a "warping window" (WW) w (originally called Sakoe-Chiba band), constraining how far the warping path can deviate from the diagonal of the matrix [22]. Given a line index $1 \leq l \leq L$ and a column index $1 \leq c \leq L$, we have $|l-c| \leq w$. A WW of 0 is equivalent to the squared Euclidean distance, and a WW $\geq L-1$ is equivalent to unconstrained (or "full") DTW. DTW with a WW is often called cDTW, or simply as DTW annotated with a window w. In this paper, we focus only on the commonly used warping window (Sakoe-Chiba band) [2]–[4], [10], [13]. However it is also important to note that there are other types of constraints for DTW such as the Itakura Parallelogram [26] and the Ratanamahatana-Keogh band [27].

We can make the following observations.

1) WW have a "validity": If the warping path at a given window w deviates from the diagonal by no more that v, then it will remain the same for all windows $v \leq w' \leq w$. This is called *window validity*, v in [3], and is noted $[v, w]$, i.e. we say that $\text{DTW}_w(S,T)$ has a window validity of $[v, w]$.

2) DTW is monotonic in w: $\text{DTW}_w(S,T)$ increases as w decreases, i.e. $\text{DTW}_w(S,T) \geq \text{DTW}_{w+k}(S,T)$ for $k \geq 1$. In other words, $\text{DTW}_w(S,T)$ is a lower bound for all $\text{DTW}_{w'}(S,T)$ with $0 \leq w' < w$ (see Section III-A).

Fig. 4 illustrates these observations on the cost matrix. The full DTW (Fig. 4a) has a window validity of $[2, L-1]$, i.e. the warping path and DTW cost of 23 are the same for all warping windows from L down to 2 (Fig. 4b). The next WW of 1 actually constraints the warping path, resulting in an increased DTW cost of 25 (Fig. 4c). Fig. 5 illustrates the consequence of these observations. The DTW distance is constant for a large range of windows, and increases when w gets smaller.

C. Learning the optimal warping window

There are two main advantages of learning the optimal warping window. First, a good window provides better results (e.g. classification accuracy) with NN search by preventing spurious alignments [3], [8], [11]. Recent works [3], [5], [8] demonstrated that DTW is only competitive for classification when used with an optimized warping window. Improvements in accuracy as great as from 65% to 93% have been demonstrated [3]. Second, by reducing the time complexity down to $O(w.L)$, cDTW can be significantly faster to compute than DTW, especially for small windows. The usual approach for discovering the optimal window size is through leave-one-out cross-validation (LOOCV) [2], [5], [8], [12] by optimising a performance metric such as training accuracy. This can be seen as creating a $(N \times L)$ table as shown in Table I, giving the nearest neighbor of every time series for all windows and finding the column that gives the best training accuracy. Note that the largest effective window is $L-1$ as $\text{DTW}_L = \text{DTW}_{L-1}$. For simplicity, we use $w=L$ and $w=L-1$ interchangeably throughout the paper.

TABLE I: Table of NNs for each WW. A cell $(i, w) = \mathcal{T}_k(d)$ means $\mathcal{T}_i$ has $\mathcal{T}_k$ as its NN for window w with distance d.

	Nearest neighbor at warping windows				
	0	1	$\cdots$	$L-2$	$L-1$
$\mathcal{T}_1$	$\mathcal{T}_{24}(2.57)$	$\mathcal{T}_{55}(0.98)$	$\cdots$	$\mathcal{T}_{55}(0.98)$	$\mathcal{T}_{55}(0.98)$
$\vdots$			$\vdots$		
$\mathcal{T}_N$	$\mathcal{T}_{60}(4.04)$	$\mathcal{T}_{47}(1.61)$	$\cdots$	$\mathcal{T}_{47}(1.61)$	$\mathcal{T}_{47}(1.61)$

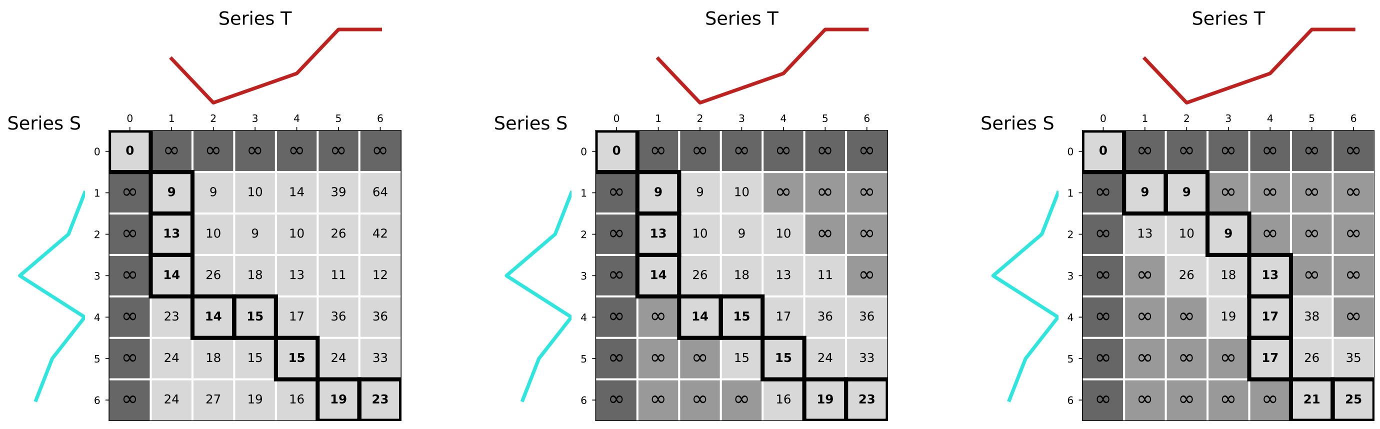
In this paper, we use 1-NN which has been widely used for benchmarking DTW algorithms [2], [4]. Note that our UltraFastWWSearch can easily be extended to cases where more than 1 neighbour ($k > 1$) is desired. This is done by adding a third dimension, k to Table I and keeping track of the distance to the k-th nearest neighbour. For simplicity, we describe our work in this paper using $k = 1$.

A straightforward LOOCV WW search implementation has $O(N^2.L^3)$ time complexity (L^2 for each DTW, repeated over L windows). Due to its L^3 complexity, it does not scale to long series [3]. Note that both FastWWSearch and UltraFastWWSearch primarily tackle the impact of the L^3 part of the complexity, by minimizing the number of times the $O(L^2)$ DTW is computed. For instance, the FordA and FordB datasets in Fig. 1 have short series, but many of them: the resulting speedup is limited by the N^2 part of the complexity. Fig. 9a and 9b clearly illustrate this. When the length of the series increases (Fig. 9a), UltraFastWWSearch scales extremely well compared to the state of the art. On the other hand, when the number of series increases (Fig. 9b), both methods suffer from the associated quadratic complexity, although UltraFastWWSearch does better.

Discovering the optimal window for DTW is so expensive (and likewise for the parameters of other elastic distances [2], [4]) that a recent version of state-of-the-art classifier HIVE-COTE dropped EE, even though doing so reduced its accuracy by 0.6% on average across the UCR series archive [12], and by up to 5% on some datasets [28]. The sole reason for dropping EE is its computational burden being too great for it to be considered feasible to employ. In the future, UltraFastWWSearch may allow to reinstate a more efficient implementation of EE in HIVE-COTE, providing a substantial improvement to the state of the art.

III. Related Work

In this section, we review the state-of-the-art methods to speed up NN-DTW, focusing on LOOCV and learning the



(a) Unconstrained DTW (b) DTW_w with $w = 2$ (c) DTW_w with $w = 1$

Fig. 4: $M_{\mathrm{DTW}_w(S,T)}$ with decreasing warping window w size. We have $\mathrm{DTW}_w(S,T) = M_{\mathrm{DTW}_w(S,T)}(L, L)$. Cells cut-out by the warping window are in light grey, borders are in dark grey. The warping path computed by the full DTW (a) is valid down to w=2 (b). Hence the next required computation is with w=1, resulting in a higher DTW cost of $25 > 23$ (c).

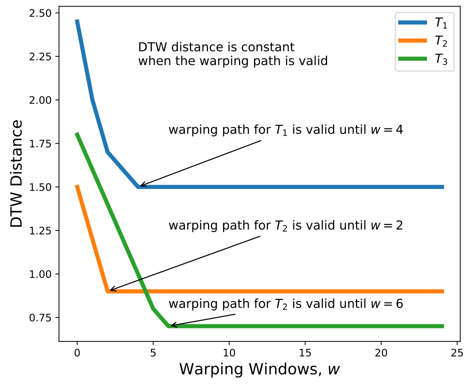


Fig. 5: DTW distances at different w

Alg. 1: NN-DTW search

```
1 $(d_{\mathrm{nn}}, \mathcal{T}_{\mathrm{nn}}) \leftarrow (\infty, \emptyset)$;
2 for $T \in \mathcal{T}$ do
3     $d \leftarrow \mathrm{DTW}(S,T)$ ;
4     if $d < d_{nn}$ then $(d_{\mathrm{nn}}, \mathcal{T}_{\mathrm{nn}}) \leftarrow (d, T)$ ;
5 return $(d_{\mathrm{nn}}, \mathcal{T}_{\mathrm{nn}})$;
```

optimal warping window efficiently.

A. Lower Bounding

Filling out the NNs table in Table I can be considered as finding the nearest neighbor for each time series T within $\mathcal{T}$ at each window size. It follows that one way to speed up LOOCV is to speed up NN-DTW. A common approach to speeding up NN-DTW is through "lower bounding" [1], [13]–[17]. A NN search returns the nearest neighbour $\mathcal{T}_{\mathrm{nn}}$ of a query S among a dataset $\mathcal{T}$, i.e. we have $d_{\mathrm{nn}} = \mathrm{DTW}(S, \mathcal{T}_{\mathrm{nn}})$ such that $\forall T \in \mathcal{T}, d_{\mathrm{nn}} \leq \mathrm{DTW}(S,T)$ (Alg. 1).

It turns out that NN search naturally supports lower bounding (Alg. 2). First, in Alg. 1, notice how d_{nn} is an upper bound (UB) on the end result: either it is the distance of the actual nearest neighbour, or it will later be replaced by a smaller value. A lower bound $\mathrm{LB}(S,T) \leq \mathrm{DTW}(S,T)$ allows

Alg. 2: Lower bounded NN-DTW search

```
1 $(d_{\mathrm{nn}}, \mathcal{T}_{\mathrm{nn}}) \leftarrow (\infty, \emptyset)$;
2 for $T \in \mathcal{T}$ do
3     if $\mathrm{LB}(S,T) < d_{nn}$ then
4         $d \leftarrow \mathrm{DTW}(S,T)$ ;
5         if $d < d_{nn}$ then $(d_{\mathrm{nn}}, \mathcal{T}_{\mathrm{nn}}) \leftarrow (d, T)$ ;
6 return $(d_{\mathrm{nn}}, C_{\mathrm{nn}})$;
```

a costly DTW computation to be avoided if $\mathrm{LB}(S,T) \geq \mathrm{UB}$ as $\mathrm{LB}(S,T) \geq \mathrm{UB} \vdash \mathrm{DTW}(S,T) \geq \mathrm{UB}$ (Alg. 2).

An efficient lower bound must be fast while having good approximations ("tight") [13], [14]. Because these aims compete, lower bounds of increasing tightness and cost are used in cascade, e.g. in the UCR-SUITE [1]. The most common lower bounds for DTW are LB_KIM [17] and LB_KEOGH [13]. The UCR-SUITE [1] is one of the fastest NN search algorithms. It uses 4 optimization techniques: early abandoning, reordering early abandoning, reversing query and candidate roles in LB_KEOGH and cascading lower bounds (LB_KIM and LB_KEOGH) to speed up NN search. Tan *et al.* [3] shows that although UCR-SUITE is faster than naive LOOCV, it is still significantly slower than FASTWWSEARCH.

B. *Improving* DTW *implementation*

Speeding up DTW itself also speeds up the whole LOOCV process. PRUNEDDTW [29] first computes an upper bound on the result (the Euclidean distance), then skip cells from the cost matrix M that are larger. It was later extended to use the upper bound UB compute by the NN search process (d_{nn} in Alg. 2), allowing early abandoning [19]. These techniques only yielded a minimal improvement when applied to window search [3].

Recently, Herrmann and Webb [21] developed the new "EAP" (Early Abandoned and Pruned) strategy, which tightly integrating pruning and early abandoning. EAP supports the fastest known NN-DTW implementations, even reducing the need for lower bounds. Like any early abandoned distance, EAP takes an upper bound UB as an extra parameter (again,

d_{nn} in Alg. 2), and abandons the computation as soon as it can be established that the end result will exceed it. The novelty of EAP is that early abandoning is treated as an extreme consequence of pruning: if UB prunes a full line of M, then no warping path can exist. Note that in Alg. 2, the initial upper bound is set to ∞, which does not allow to prune anything when using EAP. By initialising it to the squared Euclidean distance (i.e. the diagonal of M), EAP can prune some cells of M from the start (but not early abandon).

C. *FastWWSearch*

FASTWWSEARCH [3] is a window optimization algorithm producing the same results as LOOCV while being significantly faster than both traditional LOOCV and UCR-SUITE. It exploits three important properties of DTW and its LB_KEOGH lower bound: warping windows have a validity (Section II-B1); DTW is monotonic with w (Section II-B2); and, like DTW, LB_KEOGH is monotonic with w.

FASTWWSEARCH exploits these properties by starting from the largest warping window, skipping all the windows where the warping path remains the same. Starting from the largest warping window has another advantage: the monotonic property of DTW and LB_KEOGH allows LB_KEOGH_{w+k} and DTW_{w+k}, for any $k \geq 1$, to be used as lower bounds for DTW_w. In other words, results obtained at larger windows provide "free" lower bounds for smaller windows.

FASTWWSEARCH was subsequently extended to other distance measures, creating FastEE [4], a significantly faster implementation of EE.

IV. ULTRA FAST WARPING WINDOW SEARCH

The core of ULTRAFASTWWSEARCH is built upon FASTWWSEARCH [3] with the following key differences: (1) it replaces all calls to DTW with the EAP variant [21]; (2) it takes full advantage of EAP's early abandoning and pruning; (3) it processes the time series in different orders that best exploit EAP's capabilities; and (4) it does not use any lower bounds. For the rest of the paper, DTW should be understood as being the EAP variant unless specified otherwise.

ULTRAFASTWWSEARCH takes advantage of the properties of DTW (Section II-B), ordering the computation from large to small windows. Similar to FASTWWSEARCH, this allows ULTRAFASTWWSEARCH to skip the computation of $\text{DTW}_{v \leq w' < w}(S,T)$ for $\text{DTW}_w(S,T)$ with a window validity of $[v, w]$, and to use DTW_w results as lower bounds for smaller window $w'' < v$ without extra expense. In practice, for most time series pairs the window validity of DTW_L extends to around 10% of L, i.e. the validity is $[\frac{L}{10}, L]$, allowing us to skip many unnecessary computations.

We present ULTRAFASTWWSEARCH as a set of algorithms. They rely on a global cache $\mathcal{C}$ indexed by a pair of series, storing a variety of information. $\mathcal{C}_{(S,T)}$`.value` stores the most recent DTW value (i.e. at a larger window); $\mathcal{C}_{(S,T)}$`.validity` stores the minimum window size for which $\mathcal{C}_{(S,T)}$.value is valid. $\mathcal{C}_{(S,T)}$`.do_euclidean` calculates the squared Euclidean distance between S and T on demand, caching the result for future uses.

Alg. 3: AssessNN(w, d, UB, S, T)

```
Input: $w$: the warping window
Input: $d$: the distance to beat
Input: UB: the upper bound to early abandon
Input: $S, T$: the time series to evaluate
Result: $\text{DTW}_w(S,T)$ if $\leq d$, else abort
1 if $\mathcal{C}_{(S,T)}$.value $\geq d$ then return abort
2 if $w \geq \mathcal{C}_{(S,T)}$.valid then return $\mathcal{C}_{(S,T)}$.value
3 else
4     $\mathcal{C}_{(S,T)} \leftarrow \text{DTW}_w(S,T,\text{UB})$
5     if $\mathcal{C}_{(S,T)}$.value $= \infty$ then $\mathcal{C}_{(S,T)}$.value $\leftarrow$ UB
6     if $\mathcal{C}_{(S,T)}$.value $\geq d$ then return abort
7     else return $\mathcal{C}_{(S,T)}$.value
```

The ASSESSNN algorithm in Alg. 3 is a function that assesses whether a given pair of time series (S,T) is less than some distance d apart for a warping window, w. ASSESSNN differs substantially from the FASTWWSEARCH function on which it is based, LASYASSESSNN, which incorporates complex management of partially completed lower bound calculations at varying windows. ASSESSNN uses DTW_{w+k} computed at a larger window as a lower bound to avoid the computation of DTW_w when possible. Unlike the complex cascade of lower bounds used in FASTWWSEARCH, this is the only lower bound used in ULTRAFASTWWSEARCH.

Alg. 3 first checks whether the previously computed DTW distance, stored in the cache $\mathcal{C}_{(S,T)}$, is larger than the current best-so-far distance to beat, d. If so, the algorithm terminates without any extra computation. This is because DTW distance increases with decreasing w (see Fig. 5), so if a distance at a larger w' is already larger than the best-so-far distance d at w, then so too is DTW_w. If not and the previously computed DTW is still valid, it is returned (line 2). Otherwise, we have to compute $\text{DTW}_w(S,T)$. Notice that on line 4, we make use of the EAP implementation of DTW, passing the upper bound UB as an argument. We will describe how UB is calculated in the following paragraphs. If we do not early abandon, then the new distance is stored in $\mathcal{C}_{(S,T)}$. Else we store UB in $\mathcal{C}_{(S,T)}$ and terminate the algorithm. Storing UB in $\mathcal{C}_{(S,T)}$ instead of ∞ provides a better ordering of $T \in \mathcal{T}$ later in the algorithm.

Recall that learning the warping window can be thought as filling up a $(N \times L)$ nearest neighbour table, illustrated in Table I. Once this table has been filled, we can easily determine the best warping window for a particular problem by looking for the column that gives the best performance. In case of ties, we take the smallest window as it is cheaper to compute at test time. Alg. 4 describes this process. The result is identical to FASTWWSEARCH and LOOCV. In general, Alg. 4 can be transformed to either FASTWWSEARCH and LOOCV by replacing line 1 with a method to fill the NNs table. For LOOCV, this is naively filling the table described in Section II-C and Alg. 3 in [3] for FASTWWSEARCH. We use Alg. 6 to fill this table efficiently.

Similar to FASTWWSEARCH, we build this table for a subset $\mathcal{T}' \subseteq \mathcal{T}$ of increasing size until $\mathcal{T}' = \mathcal{T}$. This method

Alg. 4: UltraFastWWSearch

```
    Data: $\mathcal{T}$: training data
    Result: $w^\star$: best warping window
    // Find all nearest neighbors
1   NNs ← UltraFastFillNNTable($\mathcal{T}$)
    // Find WW with fewest misclassifications
2   bestNErrors ← $|\mathcal{T}| + 1$
3   for $w \leftarrow 0$ to $L-1$ do
4     nErrors ← 0
5     foreach $T \in \mathcal{T}$ do
6       if NNs[$T$][$w$].class $\neq$ T.class then nErrors++
7     if nErrors < bestNErrors then
8       (bestNErrors, $w^\star$) ← (nErrors, $w$)
```

Alg. 5: UpdateNN(S, T, NNs, w)

```
    Input: $S$ the query time series
    Input: $T$ the candidate time series
    Input: NNs the nearest neighbors table
    Input: $w$ the current window
    Result: NNs updated nearest neighbors table
    // Compute the upper bound UB
1   UB ← max(NNs[$S$][$w$].dist, NNs[$T$][$w$].dist)
2   if UB = $\infty$ then UB ← $\mathcal{C}_{(S,T)}$.do_euclidean
    // Check if $T$ is NNs[$S$][$w$]
3   toBeat ← NNs[$S$][$w$].dist
4   if AssessNN($w$, toBeat, UB, $S$, $T$) $\neq$ abort then
5     NNs[$S$][$w$] ← ($T, \mathcal{C}_{(S,T)}$)
    // Check if $S$ is NNs[$T$][$w$]
6   toBeatT ← NNs[$T$][$w$].dist
7   if AssessNN($w$, toBeatT, UB, $S$, $T$) $\neq$ abort then
8     NNs[$T$][$w$] ← ($S, \mathcal{C}_{(S,T)}$)
```

allows us to process all the series in $\mathcal{T}$ in a systematic and efficient order. We start by building the table for $\mathcal{T}'$ comprising only 2 first time series $\mathcal{T}_1$ and $\mathcal{T}_2$, and fill this ($2 \times L$)-table as if $\mathcal{T}'$ was the entire dataset. At this stage it is trivial that $\mathcal{T}_2$ is the nearest neighbour of $\mathcal{T}_1$ and vice versa. We then add a third time series $\mathcal{T}_3$ 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 neighbour of $\mathcal{T}_3$ within $\mathcal{T}' \setminus \mathcal{T}_3 = \{\mathcal{T}_1, \mathcal{T}_2\}$ and (b) check whether $\mathcal{T}_3$ has become the nearest neighbor of $\mathcal{T}_1$ and/or $\mathcal{T}_2$. This is described in Alg. 5. We can then add a fourth time series $\mathcal{T}_4$ and so on until $\mathcal{T}' = \mathcal{T}$.

Alg. 5 describes the process to check whether either of a pair (S, T) is a nearest neighbour of the other and, if so, to update the NNs table accordingly. This process differs from FastWWSearch by using an UB to early abandon and prune DTW computations, exploiting EAP. It is important to have a "tight" UB especially for $w{=}L$ because DTW_L is the most expensive operation for UltraFastWWSearch and thus needs to be minimised. Using EAP alone has provided a significant boost to the speed of FastWWSearch, which will be shown in our experiments in Section V.

Lines 1 to 2 of Alg. 5 calculate the upper bound that will be used to early abandon and prune EAP. The upper bound is calculated as $\text{UB} = \max(\text{NNs}[S][w].\text{dist}, \text{NNs}[T][w].\text{dist})$. This ensures that we always and only calculate the full DTW when it can result in S being T's NN or vice versa. If we do not have a best-so-far NN for either S nor T yet, i.e. when T is the first candidate NN considered for S and hence its distance is $+\infty$, then we compute $\text{ED}(S, T)$ and use that as the UB for EAP. ED is an upper bound (UB) for DTW and provides a better UB than the commonly used $+\infty$ for us to prune EAP with DTW at the largest window, $w{=}L$. Then lines 3 to 5 check whether T can be the NN of S. The algorithm calls the AssessNN function in Alg. 3 to check whether T is able to beat (i.e. smaller than) the best-so-far NN distance of S, d. If AssessNN returns abort, it means that $\text{DTW}_{w'}(S, T) \geq d$ for all $w' \geq w$, thus T cannot be the nearest neighbour of S. Otherwise we update the nearest neighbour of S with T (line 7). Similarly lines 6 to 8 check whether S is the nearest neighbour of T. Note that we have compute $\text{DTW}_w(S, T)$ once to update both $\text{NNs}[S][w]$ and $\text{NNs}[T][w]$.

The core of UltraFastWWSearch lies in Alg. 6. In line 1, we start by initialising the NNs table to $(\_, +\infty)$, an otherwise empty table with $+\infty$ nearest neighbour distances. Then in line 2 we initialise $\mathcal{T}'$, the subset of $\mathcal{T}$ processed so far. After initializing the key components, we start with the second time series in $\mathcal{T}$, and add all the preceding time series $\mathcal{T}_{s-1}$ to $\mathcal{T}'$. We start the computation from the largest window, $w{=}L{-}1$, described from lines 6 to 12.

Recall that FastWWSearch processes the series at $w{=}L{-}1$ similarly as any other smaller w. It goes through the set $\mathcal{T}'$ in an ascending order of lower bound distance to S. For the case of $w{=}L{-}1$, $\mathcal{T}'$ is ordered on LB_Kim, which is a loose lower bound. This exploits its complex cascade lower bounds in order to minimize the number of full DTW calculations required by using the lower bounds to prune as many as possible. In contrast, UltraFastWWSearch exploits the unique properties of EAP by seeking to minimize the UB used in each call to DTW. Recall that $\text{UB} = \max(\text{NNs}[S][w].\text{dist}, \text{NNs}[T][w].\text{dist})$. $\text{NNs}[S][w].\text{dist}$ starts at ∞ and can only decrease with each successive T. Hence, it is most productive to pair it initially with Ts with larger $\text{NNs}[T][w].\text{dist}$, as the max will be large anyway, and to pair it with the smallest $\text{NNs}[T][w].\text{dist}$ last, when it is also most likely to be small and hence the max will be small. To this end we process $\mathcal{T}'$ in descending order of the NN distance of each $T \in \mathcal{T}'$ at $w{=}L{-}1$, as outlined in lines 8 to 11.

However, while this sort order is important to minimize the EAP computations at the full DTW when only loose lower bounds are possible, $\text{DTW}_{w+1}(S, T)$ provides a very tight lower bound on $\text{DTW}_w(S, T)$. Once it is available it is advantageous to exploit it. Hence, on line 16, we order $\mathcal{T}'$ in ascending order of their DTW_{L-1} distances. Note that we only do this once for each series S. In practice the order does not change substantially as the window size decreases. Rather than resorting at each window size, it is sufficient to just keep track of the nearest neighbour at $w{+}1$, process it first as it is likely to be one of the nearest neighbors at w, and then process the remaining series in the DTW_{L-1} sort order.

In addition, we also keep track of the maximum window validity, ω for all $\text{NNs}[T][L]$ for all $T \in \mathcal{T}'$. By keeping track of ω, we can quickly skip all the windows where the distances

Alg. 6: ULTRAFASTFILLNNTABLE($\mathcal{T}$)

```
   Input: $\mathcal{T}$ the set of time series
   Result: NNs the nearest neighbors table
1  NNs.fillAll(_, $+\infty$)
2  $\mathcal{T}' \leftarrow \emptyset$
3  for $s \leftarrow 2$ to $N$ do
       // Update NNs wrt adding $S$
4      $S \leftarrow \mathcal{T}_s$
5      $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{\mathcal{T}_{s-1}\}$
       // Start with full DTW
6      $\omega \leftarrow 0$      // max window validity
7      $w \leftarrow L-1$  // window for full DTW
8      foreach $T \in \mathcal{T}'$ in des. order of NNs$[T][w]$.dist do
9          $\mathcal{C}_{S,T} \leftarrow \varnothing$
10         UPDATENN($S, T,$ NNs$, w$)
11         $\omega \leftarrow \max(\omega,$ NNs$[T][w]$.valid$)$
12     $\omega \leftarrow \max(\omega,$ NNs$[S][w]$.valid$)$
       // Propagate NN for path validity
13     for $w' \in$ NNs$[S][w]$.valid do
14         NNs$[S][w'] \leftarrow$ NNs$[S][w]$
15     foreach $T \in \mathcal{T}'$ if NNs$[T][w] = S$ do
16         for $w' \in$ NNs$[T][w]$.valid do
               NNs$[T][w'] \leftarrow$ NNs$[T][w]$
       // Sort $\mathcal{T}'$ in asc order using $\mathcal{C}$ once
17     $\mathcal{T}' \leftarrow \mathcal{T}'$.sort
       // remember NN at previous window ($w$+1)
18     $\mathcal{T}_{\mathrm{NN}^{w+1}} \leftarrow \mathcal{T}'_0$
19     for $w \leftarrow \omega-1$ down to 0 do
20         if NNs$[s][w] \neq \varnothing$ then
               // Update NNs$[T][w]$ for $T \in \mathcal{T}'$
21             foreach $T \in \mathcal{T}'$ do
22                 toBeat $\leftarrow$ NNs$[T][w]$.dist
23                 UB $\leftarrow$ toBeat
24                 if UB $= \infty$ then
25                     UB $\leftarrow \mathcal{C}_{(S,T)}$.do_euclidean
26                 if AssessNN($w$, toBeat, UB, $S, T$) $\neq$ abort then
27                     NNs$[T][w] \leftarrow (S, \mathcal{C}_{(S,T)})$
28         else
               // Start from the NN at $w$+1
29             UPDATENN($S, \mathcal{T}_{\mathrm{NN}^{w+1}},$ NNs$, w$)
30             foreach $T \in \mathcal{T}' \setminus \mathcal{T}_{\mathrm{NN}^{w+1}}$ do
31                 UB $\leftarrow \max($NNs$[S][w]$.dist, NNs$[T][w]$.dist$)$
32                 if UB $= \infty$ then $UB \leftarrow \mathcal{C}_{(S,T)}$.do_euclidean
33                 toBeat $\leftarrow$ NNs$[S][w]$.dist
34                 if AssessNN($w$, toBeat, UB, $S, T$) $\neq$ abort then
35                     NNs$[S][w] \leftarrow (T, \mathcal{C}_{(S,T)})$
36                     $\mathcal{T}_{\mathrm{NN}^{w+1}} \leftarrow T$
37                 toBeatT $\leftarrow$ NNs$[T][w]$.dist
38                 if AssessNN($w$, toBeatT, UB, $S, T$) $\neq$ abort then
39                     NNs$[T][w] \leftarrow (S, \mathcal{C}_{(S,T)})$
               // Propagate NN for path validity
40             for $w' \in$ NNs$[S][w]$.valid do
                   NNs$[S][w'] \leftarrow$ NNs$[S][w]$
```

are constant for all $T \in \mathcal{T}'$. On line 13, once we have the NN of S at $w{=}L{-}1$, we need to propagate this information for all w', $w' \leq w$ for which the warping path is valid. Similarly in lines 14 and 15, we also need to propagate the NN for all $T \in \mathcal{T}'$ if NNs$[T][L]{=}S$.

From line 19, we continue to process the windows from $\omega{-}1$ down to 0. Line 20 checks if we already have a NN for S from larger windows due to window validity. Lines 21 to 27 check whether S is the NN of any $T \in \mathcal{T}'$. Since we already have the NN for S, the process is the similar to lines 6-8 of Alg. 5, the difference being the way UB was calculated. In this case, we can use the distance of the current NN of T (if available) as the UB instead of taking the max of the two NN distances, as we will not use the result to check whether T is S's NN. The process starting from the else case (line 28) is when we do not obtain the NN of S at w from a larger window. In this case, we need to search for the NN of S from $\mathcal{T}'$. We start from $\mathcal{T}_{\mathrm{NN}^{w+1}}$, the NN of S at w+1. The NN of both S and $\mathcal{T}_{\mathrm{NN}^{w+1}}$ are updated with Alg. 5. The rest of $T \in \mathcal{T}' \setminus \mathcal{T}_{\mathrm{NN}^{w+1}}$ is processed similar to Alg. 5, except that we need to keep track of $\mathcal{T}_{\mathrm{NN}^{w+1}}$ (lines 31 – 39). Finally we have NNs$[S][w]$, the NN of S at w, we need to propagate the information for all valid windows (line 40).

V. EXPERIMENTS

This section describes the experiments to evaluate our ULTRAFASTWWSEARCH. To ensure reproducibility, we have made our code and results available open-source at https://github.com/ChangWeiTan/UltraFastWWS. Note that ULTRAFASTWWSEARCH is exact, producing the same results as FASTWWSEARCH and LOOCV, hence we are only interested in comparing the training time.

Our experiments use all of the 128 benchmark UCR time series datasets [12]. For each method, we perform the search using the set of 100 warping windows (percentage of the time series length) used in EE [2] and FastEE [4]. This allows ULTRAFASTWWSEARCH to be directly used in EE. Note that 23 out of the 128 datasets have a length shorter than 100, incurring duplicated windows and unnecessary operations. Since the ordering of the series in the datasets might affect the training time, i.e. the speed depends on where the actual nearest neighbour is, we report the average results over 5 runs for different reshuffles of the training dataset. We conducted our experiments in Java, on a single core cluster machine with AMD EPYC Processor CPU @2.2GHz and 32GB RAM.

Our experiments are divided into three parts. (A) We first study the effect of using EAP on LOOCV. (B) Then we explore the features of ULTRAFASTWWSEARCH that help it achieves significant speed up compared to the state of the art. (C) Lastly, we investigate the scalability of ULTRAFASTWWSEARCH on large and long datasets.

A. *Pruning and Early Abandoning with* EAP

Given the dramatic speedup of EAP on NN-DTW [21], we first study the feasibility of replacing DTW in LOOCV with EAP. The following methods are compared:

1) DTW_LOOCV: Naive implementation of LOOCV described in Section II-C using NN-DTW with early abandoning strategy described in [1] but without lower bound and UB from Section III-A. The UB is computed using the best-so-far NN distance.
2) UCR-SUITE_LOOCV: Naive implementation of LOOCV using NN-DTW with optimizations from

UCR-Suite, i.e. cascading lower bounds and early abandoning as before [1].

3) EAP_LOOCV: Replacing DTW in DTW_LOOCV with EAP [21].

Fig. 6a compares the total training time of the three methods on the 128 datasets. The results show that EAP_LOOCV reduces the training time of DTW_LOOCV by almost 1,000 hours (42 days) and about 300 hours (12 days) for UCR-Suite_LOOCV. Note that EAP_LOOCV was able to achieve such significant speedup without using any lower bounds, while UCR-Suite_LOOCV uses a series of complex lower bounds. The main reason is because the LB_Kim and LB_Keogh lower bounds used in UCR-Suite are very loose at larger windows, as pointed out in [14]. The work in [14] showed that the more complex LB_Keogh can be looser than the simpler LB_Kim when $w \geq 0.5 \cdot L$, This shows that EAP is able to reduce the need for lower bounds for NN-DTW especially at larger warping windows.

B. Speeding up the state of the art

This section examines the features that make UltraFastWWSearch ultra-fast, comparing it to state-of-the-art FastWWSearch. The results are shown in Fig 6b.

Much of EAP's speed up in many NN-DTW tasks actually comes from early abandoning (see Fig. 7 of [21]). Section V-A showed that EAP, even without using lower bounds, speeds up the naive LOOCV implementation. Hence, we created two variants of FastWWSearch, (1) with early abandoning and (2) without lower bounds to study how they contribute to speeding up FastWWSearch, annotated with the suffixes "_EA" and "_NoLb" respectively. We adopt the early abandoning strategy described in [1] for the original FastWWSearch and use the UB described in Section III-A for the early abandoning process.

It is not surprising that removing lower bounds for FastWWSearch makes it slower, as it makes use of various lower bounds to achieve the huge speed up. However, it is interesting that adding early abandoning to FastWWSearch makes it the slowest. This is because if DTW is early abandoned at a larger window, then when FastWWSearch needs the DTW distance at a smaller window, because it was not fully computed, FastWWSearch needs to recalcuate DTW from scratch. Similar behaviour was observed in [3] as well.

On the other hand, the opposite is observed for the EAP variants. EAP_FastWWSearch_NoLb in Fig. 6b is EAP_FastWWSearch with the use of lower bounds removed. It shows that removing lower bounds actually improves EAP_FastWWSearch, albeit only by about 20 minutes. This is not surprising as it coincides with the results from the EAP paper [21]. This again highlights the effectiveness of the early abandoning strategy of EAP and the possibility of removing complex lower bounds.

UltraFastWWSearch incorporates six primary strategies that distinguish it from FastWWSearch. We study the effect of introducing each of these in turn with algorithms EAP_FastWWSearch: using EAP for DTW computations; EAP_FastWWSearch_NoLb: removing lower bounds; EAP_FastWWSearch_EA: using early abandoning; UltraFastWWSearch_V1: tighter upper bounds; UltraFastWWSearch_V2: sorting $\mathcal{T}'$ in ascending order of distance to nearest neighbor and then sorting on DTW_L; and UltraFastWWSearch: skipping windows from $L{-}1$ to ω, the maximum window validity at $L{-}1$.

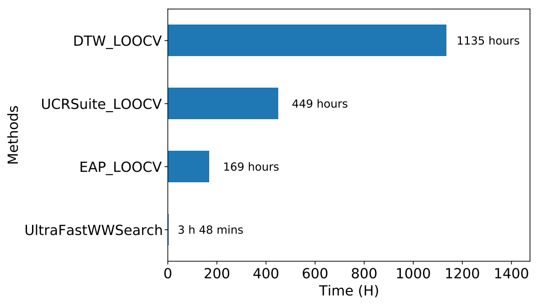
Fig. 6b shows that substituting EAP to compute DTW within FastWWSearch (even without early abandoning) (EAP_FastWWSearch) reduces the total training time for all 128 datasets by 5 hours. 3 hours and 50 minutes of this comes from 5 long and large datasets, `NonInvasiveFetalECGThorax1`, `UWaveGestureLibraryAll`, `HandOutlines`, `FordA` and `FordB`. The pairwise plot in Fig. 7 illustrates that EAP_FastWWSearch is consistently faster than the original DTW variant, although the difference between them is not large. The result shows that without early abandoning, EAP is still an efficient strategy that prunes unnecessary computations in DTW.

The effectiveness of early abandoning an EAP computation depends on the UB that was passed into it. EAP_FastWWSearch_EA uses the UB described in Section III-A. The V1 variant of UltraFastWWSearch uses the UB described in Alg. 5. The results in Fig. 6b show that this UB improves the speed of UltraFastWWSearch but by a small margin. Since we calculate UB as the maximum between the nearest neighbour distances of both S and T, it is most productive to pair S with Ts with larger $\mathrm{NNs}[T][w]\mathrm{.dist}$ (Alg. 6). This allows us to better exploit the new UB. This strategy has shown to speed up FastWWSearch substantially, as demonstrated by the V2 variant of UltraFastWWSearch in Fig. 6b.

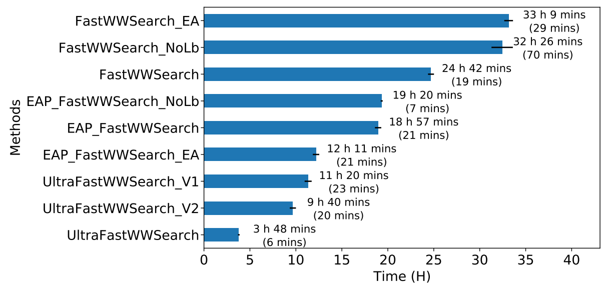
Finally we add the optimization of skipping windows from $L{-}1$ to ω. While the five previous optimizations all exploit the properties of EAP, this final optimization is a novel further exploit of the window validity property beyond those in FastWWSearch. It more than halves the total time.

Fig 6b shows that UltraFastWWSearch is able to complete all 128 datasets in under 4 hours. This is a 6 times speedup compared to 24 hours for FastWWSearch.

We performed a statistical test using the Wilcoxon signed-rank test with Holm correction as the post hoc test to the Friedman test [30] to test the significance of our results and visualise it in a critical difference diagram, illustrated in Fig. 8. Fig. 8 shows the average ranking of each method over all datasets, with a rank of 1 being the fastest and rank 9 being the slowest. Methods in the same clique (black bars) indicates that they are not significantly different from each other. Similar to the results in Fig. 6b, the optimizations for UltraFastWWSearch significantly slows down FastWWSearch. The critical difference diagram shows that all the EAP variants are faster than the original FastWWSearch with significant consistency across datasets. It is interesting to observe that although early abandoning reduces the total time on 128 datasets shown in Fig. 6b, it is ranked lower compared to



(a)



(b)

Fig. 6: Total training time on 128 datasets [12] of (a) LOOCV with DTW and EAP, UCR-SUITE, and our method for reference; (b) FASTWWSEARCH and ULTRAFASTWWSEARCH and their variants. The numbers in the round brackets represent the standard deviation over 5 runs.

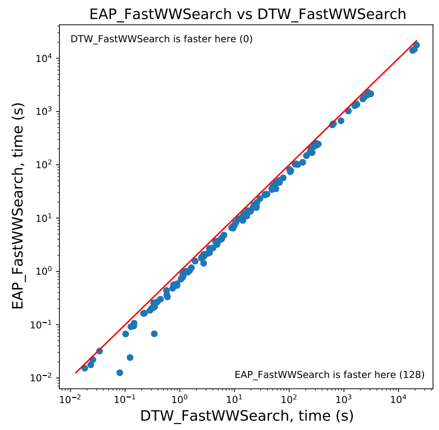


Fig. 7: Pairwise comparison on 128 UCR datasets of FASTWWSEARCH with EAP_FASTWWSEARCH.

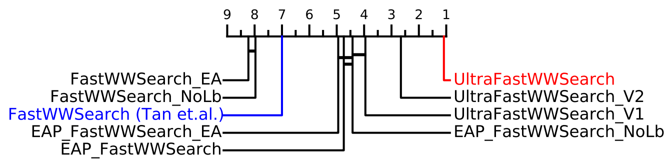
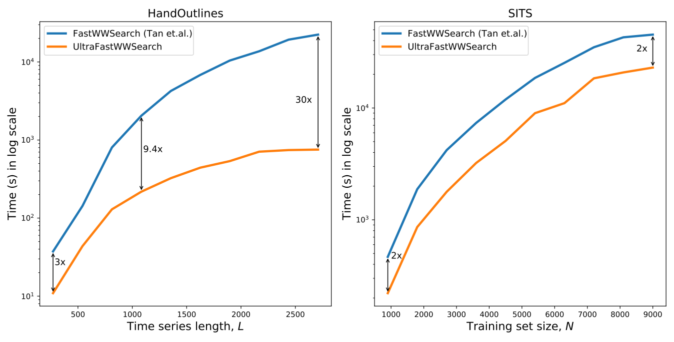


Fig. 8: Critical difference diagram comparing the training time of various methods on 128 datasets.

all other methods. The reason being the early abandoning strategy in EAP reduces the time of three largest datasets (`HandOutlines`, `FordA`, and `FordB`) by a significant amount, while the overhead of having to recalculate EAP if previously early abandoned has greater cost relative to the computation save by abandoning on smaller datasets. Then we see that ULTRAFASTWWSEARCH is the fastest among all with an average rank closed to 1 (i.e. it is faster than all methods on almost all datasets), followed by its V2 and V1 variants. Fig. 2 shows that ULTRAFASTWWSEARCH is up to one order of magnitude faster than FASTWWSEARCH.

C. Scalability to large and long datasets

We showed previously that ULTRAFASTWWSEARCH is efficient on large and long datasets. We now investigate its scalability. We first experimented using the `HandOutlines` dataset with a length of L=2709 – the longest in the UCR archive [12]. We varied the length from $0.1 \times L$ to L, recording the time to search for the best warping window. Fig. 9a shows that the training time of ULTRAFASTWWSEARCH increases slower than FASTWWSEARCH as L increases. With only L=1000, we are able to achieve 9.4 times speed up, and 30 times at L=2709. This means reducing 6 hours of compute time down to 11 minutes (Fig. 1), thus effectively tackling the L^3 part of the complexity.



(a) (b)

Fig. 9: Training time versus (a) time series length L on `HandOutlines` [12], and (b) training set size N on `SITS` [31].

We then evaluated the scalabilty to larger datasets, using the same SITS dataset as [3], taken from [31]. We chose this dataset because it has a short length of L=46, which tends to isolate the influence of N on the scalability. Fig. 9b shows that ULTRAFASTWWSEARCH is on average 2 times faster than FASTWWSEARCH for all N. This means that although ULTRAFASTWWSEARCH is faster than FASTWWSEARCH, the N^2 part of the complexity becomes a limitation of ULTRAFASTWWSEARCH. However, traditional methods LOOCV and UCR-SUITE do not even scale on this dataset as shown in [3], requiring days to complete, while ULTRAFASTWWSEARCH only takes 6 hours for N=90, 000.

VI. Conclusion

This paper proposes an ultra fast algorithm that is able to learn the warping window for Dynamic Time Warping efficiently. UltraFastWWSearch fundamentally transforms its predecessor FastWWSearch. It incorporates six major changes – using EAP to compute DTW; removing the use of DTW lower bounds; adding early abandoning of DTW; establishing tighter upper bounds for early abandoning; ordering the time series so as to best exploit the efficient pruning and early abandoning power of EAP; and using the window validity to skip the majority of window sizes altogether.

Our experiments show that it is up to an order of magnitude faster than the previous state of the art, with the greatest benefit achieved on long time series datasets, where it is most needed.

UltraFastWWSearch speeds up the training of NN-DTW, formerly one of the slowest time series classification (TSC) algorithms, to under 4 hours on the UCR datasets, a time close to ROCKET, one of the fastest and most accurate TSC algorithms [32]. This holds open the promise for EE to be reinstated back into HIVE-COTE, which is known to improve its classification performance to a new state-of-the-art level for TSC and only omitted due to its excessive compute time [33].

Acknowledgments

The authors would like to thank Professor Eamonn Keogh and his team at the University of California Riverside (UCR) for providing the UCR Archive, and Hassan Fawaz for providing his code for drawing the critical difference diagram.

References

[1] 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 dynamic time warping," in *Proc. 18th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2012, pp. 262–270.

[2] J. Lines and A. Bagnall, "Time series classification with ensembles of elastic distance measures," *Data Mining and Knowledge Discovery*, vol. 29, no. 3, pp. 565–592, 2015.

[3] C. W. Tan, M. Herrmann, G. Forestier, G. I. Webb, and F. Petitjean, "Efficient search of the best warping window for dynamic time warping," in *Proc. 2018 SIAM Int. Conf. Data Mining*. SIAM, 2018, pp. 225–233.

[4] C. W. Tan, F. Petitjean, and G. I. Webb, "Fastee: Fast ensembles of elastic distances for time series classification," *Data Mining and Knowledge Discovery*, vol. 34, no. 1, pp. 231–272, 2020.

[5] 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*, vol. 31, no. 3, pp. 606–660, 2017.

[6] C. W. Tan, C. Bergmeir, F. Petitjean, and G. I. Webb, "Time series extrinsic regression," *Data Mining and Knowledge Discovery*, pp. 1–29, 2021.

[7] F. Petitjean, A. Ketterlin, and P. Gançarski, "A global averaging method for dynamic time warping, with applications to clustering," *Pattern Recognition*, vol. 44, no. 3, pp. 678–693, 2011.

[8] H. A. Dau, D. F. Silva, F. Petitjean, G. Forestier, A. Bagnall, A. Mueen, and E. Keogh, "Optimizing dynamic time warping's window width for time series data mining applications," *Data Mining and Knowledge Discovery*, vol. 32, no. 4, pp. 1074–1120, 2018.

[9] S. Alaee, R. Mercer, K. Kamgar, and E. Keogh, "Time series motifs discovery under dtw allows more robust discovery of conserved structure," *Data Mining and Knowledge Discovery*, vol. 35, no. 3, pp. 863–910, 2021.

[10] C. A. Ratanamahatana and E. Keogh, "Three myths about dynamic time warping data mining," in *Proc. 2005 SIAM Int. Conf. Data Mining*. SIAM, 2005, pp. 506–510.

[11] E. J. Keogh and M. J. Pazzani, "Derivative dynamic time warping," in *Proc. 2001 SIAM Int. Conf. Data Mining*. SIAM, 2001, pp. 1–11.

[12] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, and Hexagon-ML, "The UCR time series classification archive," October 2018, https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[13] E. Keogh and C. A. Ratanamahatana, "Exact indexing of dynamic time warping," *Knowledge and Information Systems*, vol. 7, no. 3, pp. 358–386, 2005.

[14] C. W. Tan, F. Petitjean, and G. I. Webb, "Elastic bands across the path: A new framework and method to lower bound DTW," in *Proc. 2019 SIAM Int. Conf. Data Mining*. SIAM, 2019, pp. 522–530.

[15] G. I. Webb and F. Petitjean, "Tight lower bounds for dynamic time warping," *Pattern Recognition*, vol. 115, p. 107895, 2021.

[16] D. Lemire, "Faster retrieval with a two-pass dynamic-time-warping lower bound," *Pattern Recognition*, vol. 42, no. 9, pp. 2169–2180, 2009.

[17] S.-W. Kim, S. Park, and W. W. Chu, "An index-based approach for similarity search supporting time warping in large sequence databases," in *Proc. 17th Int. Conf. Data Engineering*. IEEE, 2001, pp. 607–614.

[18] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[19] D. F. Silva, R. Giusti, E. Keogh, and G. E. Batista, "Speeding up similarity search under dynamic time warping by pruning unpromising alignments," *Data Mining and Knowledge Discovery*, vol. 32, no. 4, pp. 988–1016, 2018.

[20] R. Wu and E. J. Keogh, "Fastdtw is approximate and generally slower than the algorithm it approximates," *IEEE Trans. Knowledge and Data Engineering*, 2020.

[21] M. Herrmann and G. I. Webb, "Early abandoning and pruning for elastic distances including dynamic time warping," *Data Mining and Knowledge Discovery*, pp. 1–25, 2021.

[22] H. Sakoe and S. Chiba, "A dynamic programming approach to continuous speech recognition," in *Int. Cong. Acoustics*, vol. 3, 1971, pp. 65–69.

[23] J. Lines, S. Taylor, and A. Bagnall, "Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles," *ACM Trans. Knowledge Discovery from Data*, vol. 12, no. 5, 2018.

[24] B. Lucas, A. Shifaz, C. Pelletier, L. O'Neill, N. Zaidi, B. Goethals, F. Petitjean, and G. I. Webb, "Proximity forest: an effective and scalable distance-based classifier for time series," *Data Mining and Knowledge Discovery*, vol. 33, no. 3, pp. 607–635, 2019.

[25] A. Shifaz, C. Pelletier, F. Petitjean, and G. I. Webb, "Ts-chief: a scalable and accurate forest algorithm for time series classification," *Data Mining and Knowledge Discovery*, vol. 34, no. 3, pp. 742–775, 2020.

[26] F. Itakura, "Minimum prediction residual principle applied to speech recognition," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 23, no. 1, pp. 67–72, 1975.

[27] C. A. Ratanamahatana and E. Keogh, "Making time-series classification more accurate using learned constraints," in *Proc. 2004 SIAM Int. Conf. Data Mining*. SIAM, 2004, pp. 11–22.

[28] A. Bagnall, M. Flynn, J. Large, J. Lines, and M. Middlehurst, "A tale of two toolkits, report the third: on the usage and performance of hive-cote v1. 0," *arXiv e-prints*, pp. arXiv–2004, 2020.

[29] D. F. Silva and G. E. A. P. A. Batista, "Speeding Up All-Pairwise Dynamic Time Warping Matrix Calculation," in *Proc. 2016 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, Jun. 2016, pp. 837–845.

[30] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[31] C. W. Tan, G. I. Webb, and F. Petitjean, "Indexing and classifying gigabytes of time series under time warping," in *Proceedings of the 2017 SIAM Int. Conf. Data Mining*. SIAM, 2017, pp. 282–290.

[32] A. Dempster, F. Petitjean, and G. I. Webb, "Rocket: exceptionally fast and accurate time series classification using random convolutional kernels," *Data Mining and Knowledge Discovery*, vol. 34, no. 5, pp. 1454–1495, 2020.

[33] M. Middlehurst, J. Large, M. Flynn, J. Lines, A. Bostrom, and A. Bagnall, "Hive-cote 2.0: a new meta ensemble for time series classification," *arXiv preprint arXiv:2104.07551*, 2021.