Knowledge and Information Systems
https://doi.org/10.1007/s10115-022-01827-w

REGULAR PAPER

Ultra-fast meta-parameter optimization for time series similarity measures with application to nearest neighbour classification

Chang Wei Tan[1] · Matthieu Herrmann[1] · Geoffrey I. Webb[1]

Received: 28 January 2022 / Revised: 18 December 2022 / Accepted: 25 December 2022

Abstract
Nearest neighbour similarity measures are widely used in many time series data analysis applications. They compute a measure of similarity between two time series. Most applications require tuning of these measures' meta-parameters in order to achieve good performance. However, most measures have at least $O(L^2)$ complexity, making them computationally expensive and the process of learning their meta-parameters burdensome, requiring days even for datasets containing only a few thousand series. In this paper, we propose UltraFastMPSearch, a family of algorithms to learn the meta-parameters for different types of time series distance measures. These algorithms are significantly faster than the prior state of the art. Our algorithms build upon the state of the art, exploiting the properties of a new efficient exact algorithm which supports early abandoning and pruning for most time series distance measures. We show on 128 datasets from the UCR archive that our new family of algorithms are up to an order of magnitude faster than the previous state of the art.

Keywords Time series · Similarity measures · Early abandoning · Pruning

1 Introduction

Time series distance measures are used in a wide range of time series data mining tasks, including similarity search [5, 21, 24, 32], classification [3, 18, 34, 37, 43], regression [33], clustering [8, 23], indexing [39], and motif discovery [1]. All these tasks rely on nearest neighbour (NN) search, which is widely known to be most effective when the meta-parameters of the distance measures are learnt [3, 18, 37]. For instance, the Dynamic Time Warping (DTW) distance proves to be the most effective when constrained by the right warping window (WW) [8, 26, 34]. Indeed, the unconstrained DTW is subject to pathological warping, leading to unintuitive alignments [34] where a single point of a time series is aligned to a large section of another series [14].

✉ Chang Wei Tan
chang.tan@monash.edu

1 Department of Data Science and AI, Monash University, Melbourne, VIC 3800, Australia

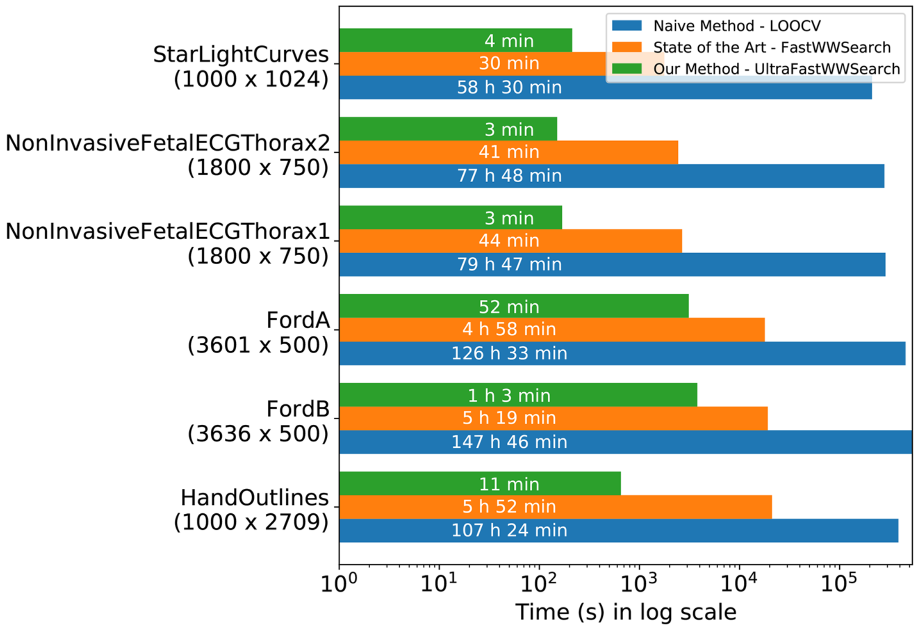


Fig. 1 UltraFastWWSearch vs the naive LOOCV approach, and state-of-the-art FastWWSearch. Total training time on the 6 largest datasets from [7] in terms of $N \times L$

Traditionally, learning the meta-parameters has been a time-consuming leave-one-out cross-validation (LOOCV) process that requires computing the distance between every pair of training instances, for each value of each meta-parameter [7, 8, 18, 35, 37]. This is only compounded by the quadratic time complexity of most distance measures. Take DTW for example, with a training dataset of N time series of length L, learning the WW naively requires $O(N^2.L^3)$ operations (Sect. 2.7). This is extremely slow, even for datasets with only a thousand of time series. For instance, the naive approach took 58 h to learn the best WW on the `StarLightCurves` dataset from the UCR archive [7], which only has 1000 training instances with a length of 1024 (Fig. 1). In comparison, the prior state-of-the-art method, FastWWSearch, took 30 min, while our proposed approach took only 4 min. Similar improvements can be seen across all the largest (in terms of $N \times L$) datasets from the UCR archive [7], shown in Fig. 1 (note log scale).

The prior state-of-the-art FastWWSearch that learns the best WW for DTW, is a sophisticated and intricate algorithm [34], 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 1000 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 h to train on the `HandOutlines` dataset, one of the largest and longest datasets from the UCR archive [7], compared to just 11 min for our proposed method. The FastWWSearch was later extended to other distance measures, forming the Fast Ensemble of Elastic Distances (FastEE) [37], that is 40 times faster than the original EE. We will refer to them in the paper as the FastEE approaches.

Before FastWWSearch and FastEE, 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 the distance computations [15–17, 24, 36, 41]. The other was by speeding up the core computation of the distances, e.g. by approximating it [28] or pruning unnecessary operations [31]. But, scalability remains an issue for large datasets and long series [34, 37, 42].

Recently, [11] 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 neighbour search naturally provides such an upper bound (Sect. 3.1). EAP demonstrated more than an order of magnitude speedup for several NN search tasks.

In this paper, we propose UltraFastMPSearch, a family of three algorithms to learn the meta-parameters for the six time series distance measures that are at the heart of the influential Ensemble of Elastic Distances (EE) algorithm [18]—Dynamic Time Warping (DTW) [27], Weighted Dynamic Time Warping (WDTW) [13], Longest Common Subsequence (LCSS) [4], Edit Distance with Real Penalty (ERP) [5, 6], Move–Split–Merge (MSM) [32], and Time Warp Edit Distance (TWE) [21]. We fundamentally transformed the FastEE algorithm proposed in [37] to exploit the full capacity of EAP [11]. UltraFastMPSearch consists of

1. UltraFastWWSearch, recently proposed in our paper IEEE ICDM2021 [35], of which the current paper is an expanded version. UltraFastWWSearch was designed specifically for DTW, exploiting a DTW property called the *window validity* to gain further substantial speedup;
2. UltraFastLocalUB, a variant of UltraFastWWSearch extended to other distance measures without the *window validity*;
3. UltraFastGlobalUB, a variant of UltraFastLocalUB that uses a global rather than local upper bound, ensuring that a distance computation is only early abandoned if it cannot provide a useful lower bound for distance computations with subsequent meta-parameter values.

Like FastWWSearch and all the FastEE approaches, UltraFastMPSearch 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 and the FastEE approaches when tested on the 128 datasets from the UCR archive [7]. Figure 2 demonstrates this by comparing UltraFastWWSearch to FastWWSearch and to the traditional LOOCV approach.

Similar to the FastWWSearch and FastEE approaches, UltraFastMPSearch systematically fills a table recording the nearest neighbour at each meta-parameter for each series in database $\mathcal{T}$. However, it does so without the intricate cascading of lower bounds that is critical to FastWWSearch and FastEE. UltraFastMPSearch processes a new time series in a systematic order, minimizing the number of distance computations while carefully 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 UltraFastMPSearch as a subroutine to tune time series distance measures whatever their application.

We believe that UltraFastMPSearch serves as an important foundation to further speed up distance-based time series classification algorithms, such as the "Fast Ensemble of Elastic Distances" (FastEE) [37], that has fallen from favor due to its relatively slow compute time.

This paper is organized as follows. In Sect. 2, we introduce some background and notation used in this work. We review some related work in Sect. 3. Section 4 describes Ultra-

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

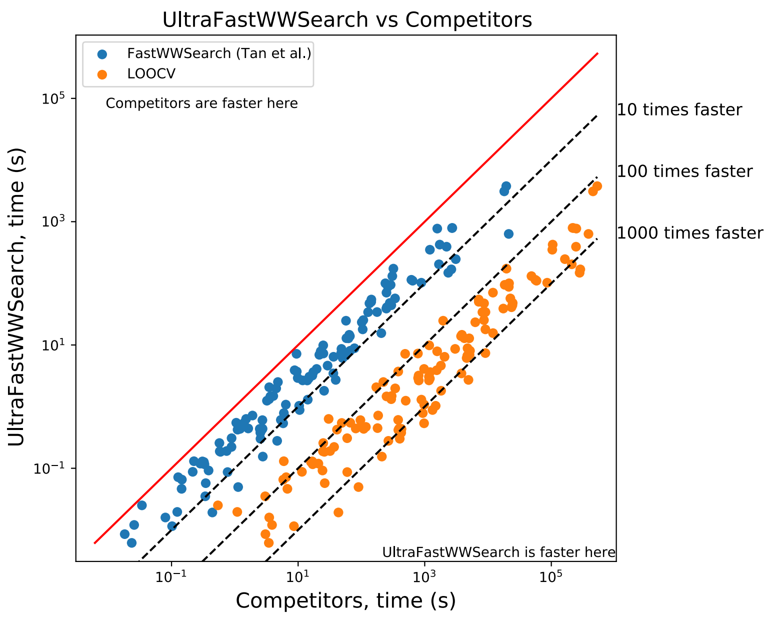


Fig. 2 Pairwise plot comparing ULTRAFASTWWSEARCH to baseline methods on 128 UCR datasets. FASTWWSEARCH is the current state of the art [34]. LOOCV is the standard method used to search for the best warping window [18]

FASTMPSEARCH in detail. Then, we evaluate our method in Sect. 5 with the standard methods. Lastly, Sect. 6 concludes our work with some future directions.

Note that this paper is an extended version of our IEEE ICDM2021 ULTRAFASTWWSEARCH paper [35].

2 Background

We consider learning from a dataset $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_N\}$ 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.

In this section, we briefly discuss the distance measures used in this work and their meta-parameters. We refer interested readers to their respective papers for a detailed overview of the measures.

2.1 Dynamic Time Warping

The DTW distance was first introduced in 1971 by [27] as a speech recognition tool. Since then, it has been one of the most widely used distance measures in NN search, supporting sub-sequence search [24], regression [33], clustering [8, 23], motif discovery [1], and classification [3, 18, 37]; nearest neighbour with DTW (NN-DTW) has been the historical approach to time series classification. The "Ensemble of Elastic Distances" (EE) [18], 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.

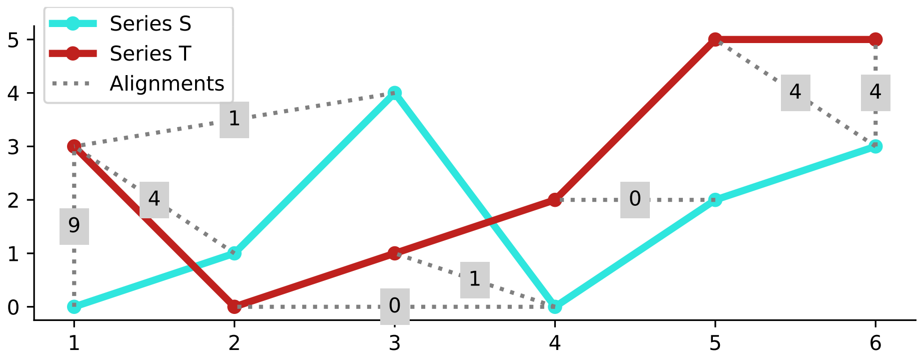


Fig. 3 Alignments associated with the warping path in Fig. 4a

classifiers embedding other classifiers as components, for time series classification. Various recent and accurate ensemble classifiers such as HIVE-COTE [19], Proximity Forest [20], and TS-CHIEF [29] 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 minimizing 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 $\text{DTW}(S, T){=}M(L, L)$.

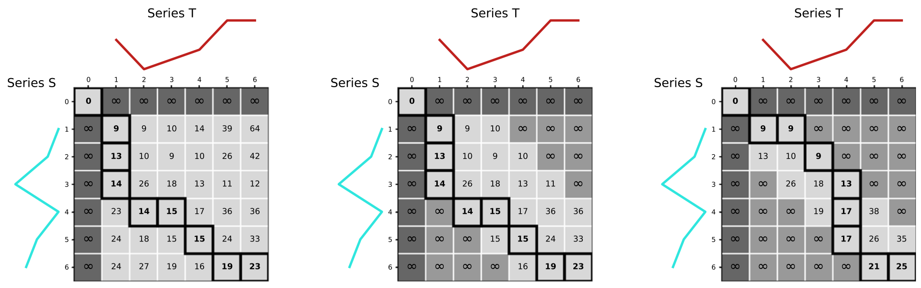
$$M(0,0) = 0 \tag{1a}$$
$$M(i,0) = +\infty \tag{1b}$$
$$M(0,j) = +\infty \tag{1c}$$
$$M(i,j) = \mathrm{d}(S_i, T_j) + \min \begin{cases} M(i{-}1, j{-}1) \\ M(i{-}1, j) \\ M(i, j{-}1) \end{cases} \tag{1d}$$

Common functions for the cost of aligning two points are $\mathrm{d}(S_i, T_j) = \left|S_i{-}T_j\right|$ 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.

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.

2.1.1 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 [27]. Given a line index $1 \leq l \leq L$ and a column index $1 \leq c \leq L$, we have $|l{-}c| \leq w$. With the cost function $\mathrm{d}(S_i, T_j){=}(S_i{-}T_j)^2$, a WW of 0 is equivalent to the squared Euclidean distance. On the other hand, 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) [15, 18, 26, 34, 37]. However, it is also important to note that there are other types of constraints for DTW such as the Itakura Parallelogram [12] and the Ratanamahatana–Keogh band [25].



(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**)

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 [34], and is noted $[v, w]$, i.e. we say that $\mathrm{DTW}_w(S, T)$ has a window validity of $[v, w]$.
2. DTW **is monotonic in** w: $\mathrm{DTW}_w(S, T)$ increases as w decreases, i.e. $\mathrm{DTW}_w(S, T) \geq \mathrm{DTW}_{w+k}(S, T)$ for $k \geq 1$. In other words, $\mathrm{DTW}_w(S, T)$ is a lower bound for all $\mathrm{DTW}_{w'}(S, T)$ with $0 \leq w' < w$ (see Sect. 3.1).
   Figure 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). Figure 5 illustrates the consequence of these observations. The DTW distance is constant for a large range of windows and increases when w gets smaller.

2.2 Weighted Dynamic Time Warping

The Weighted Dynamic Time Warping (WDTW) was proposed to reduce pathological alignments [13]. Instead of having a hard constraint like the WW for DTW, WDTW imposes a soft constraint on the warping path. The cost of aligning two points S_i and T_j is multiplied by a weight that depends on their distance in the time dimension, a= $|i - j|$. It will have a larger weight if i is far from j and reduces the chances of aligning S_i to T_j, thus preventing the alignment of two points that are too far away in the time dimension. The weights are computed using a modified logistic weight function described in Eq. 2, parameterized by the meta-parameter g that controls the level of penalization for further points [13]. The optimal range for g is distributed between 0.01 and 0.6 as suggested by the authors [13]. $\mathrm{w}_{\max}$ is the upper bound for the weight and is typically set to 1 [13].

$$\mathrm{w}_a = \frac{\mathrm{w}_{\max}}{1 + e^{-g\cdot(a-L/2)}} \tag{2}$$

We observed that WDTW monotonically decreases with increasing parameter g (Fig. 6), i.e. $\mathrm{WDTW}_g(S, T) > \mathrm{WDTW}_{g+k}(S, T)$ for $k > 0$. This means that $\mathrm{WDTW}_g(S, T)$

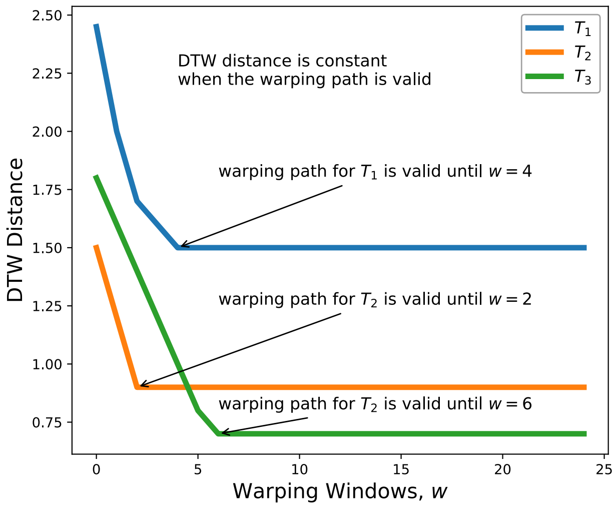


Fig. 5 DTW distances against T_0 at different w

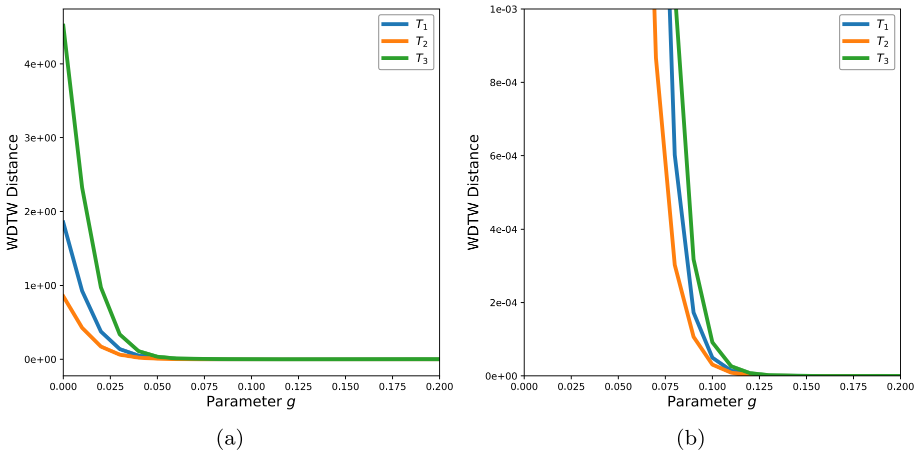


Fig. 6 WDTW distances at different **a** g values and **b** the zoomed in version showing that the distances are not constant but extremely small

is a lower bound for all $\text{WDTW}_{g'}(S, T)$ with $0 \leq g' < g$. Note that unlike DTW that stays constant within a window validity, the sigmoid weighting function makes WDTW a continuous function, preventing it from having a constant value, illustrated in Fig. 6b.

2.3 Longest Common Subsequence

The Longest Common Subsequence (LCSS) is a common measure used to compare string sequences [4, 40]. It finds the longest common subsequence that best matches the two string sequences. Using a distance threshold ε, LCSS can be extended to numeric sequences (time series) where the two points S_i and T_j are considered a match if the cost between them is less than ε. Each cell of the cost matrix $M_{\text{LCSS}}(i, j)$ indicates the number of matches between the

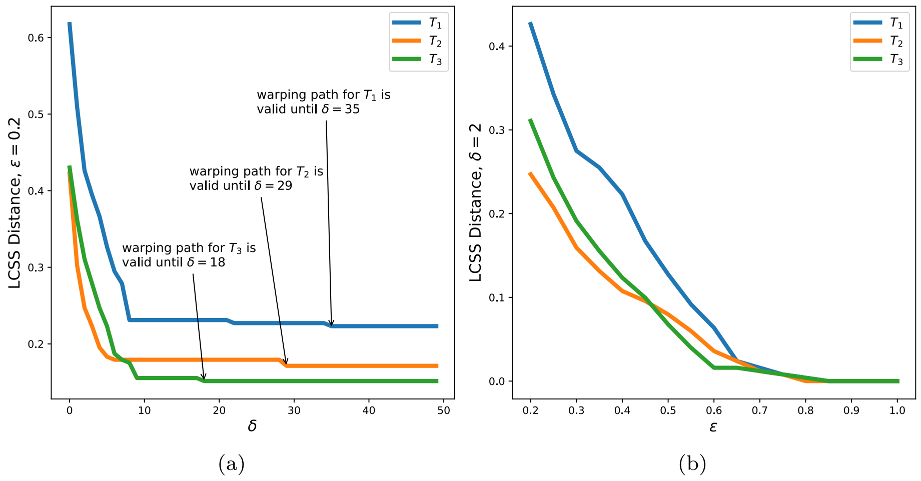


Fig. 7 LCSS distances at different **a** δ when ε=0.2 and **b** ε when δ=2

two time series, i.e. the length of the longest common subsequence. Then the LCSS distance is equal to the difference between the length of the series, L and the last cell of M_{LCSS}. Equation 3 describes the computation of LCSS. A global constraint, δ, similar to WW can also be applied to LCSS, restricting the alignment path.

$$M_{\text{LCSS}}(i, j) = \begin{cases} 0 & \text{if } i = 0, j = 0 \\ 1 + M_{\text{LCSS}}(i-1, j-1) & \text{if } |S_i - T_j| \le \varepsilon \\ \max \begin{cases} M_{\text{LCSS}}(i-1, j) \\ M_{\text{LCSS}}(i, j-1) \end{cases} & \text{otherwise} \end{cases} \tag{3}$$

Similar to WW, the δ constraint parameter has a validity, making LCSS constant for a range of δ values. By definition, LCSS is monotonic in both δ and ε as shown in Fig. 7. $\text{LCSS}_{\delta,\varepsilon}(S, T)$ decreases as δ increases, i.e. $\text{LCSS}_{\delta,\varepsilon}(S, T) \ge \text{LCSS}_{\delta+k,\varepsilon}(S, T)$ for $k \ge 1$. In addition, $\text{LCSS}_{\delta,\varepsilon}$ also decreases as ε increases, i.e. $\text{LCSS}_{\delta,\varepsilon}(S, T) \ge \text{LCSS}_{\delta,\varepsilon+k}(S, T)$ for $k > 0$. Note that $\text{LCSS}_{\delta,\varepsilon}(S, T) = \text{LCSS}_{\delta,\varepsilon+k}(S, T) = 0$ at a sufficiently large ε. This allows LCSS with a larger δ or ε to lower bound LCSS with a smaller δ or ε.

2.4 Edit distance with real penalty

Most time series distance measures such a DTW and LCSS are not metric, making it challenging to index a time series dataset or prune k-NN queries under these measures. Edit Distance with Real Penalty (ERP) is a metric that combines the L1-norm and edit distances such as DTW [5, 6]. It is parameterized by two meta-parameters, "gap value" g and WW, w. If a gap is added, the penalty will be the cost between a point S_i or T_j and g. This is described in Eq. 4. Note that our implementation uses the squared Euclidean distance as the cost function.

$$M_{\text{ERP}}(i, j) = \min \begin{cases} M_{\text{ERP}}(i-1, j-1) + \text{cost}(S_i, T_j) \\ M_{\text{ERP}}(i-1, j) + \text{cost}(S_i, g) \\ M_{\text{ERP}}(i, j-1) + \text{cost}(g, T_j) \end{cases} \tag{4}$$

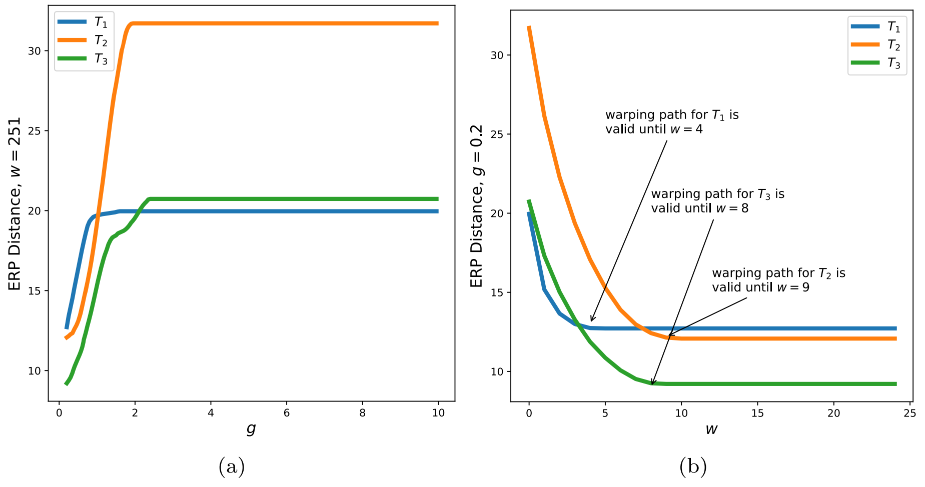


Fig. 8 ERP distances at different **a** g when w=251 and **b** w when g=0.2

Increasing g increases $\mathrm{ERP}_{g,w}(S, T)$, as illustrated in Fig. 8 We observed that $\mathrm{ERP}_{g+k,w}(S, T) \geq \mathrm{ERP}_{g,w}(S, T)$ for $k \geq 0$ and according to Eq. 4, $\mathrm{ERP}_{g+k,w}(S, T) = \mathrm{ERP}_{g,w}(S, T)$ for a sufficiently large g. Hence $\mathrm{ERP}_{g,w}(S, T)$ lower bounds $\mathrm{ERP}_{g+k,w}(S, T)$. Similarly $\mathrm{ERP}_{g,w+k}(S, T)$ also lower bounds $\mathrm{ERP}_{g,w}(S, T)$ for all $k \geq 1$. Similar to DTW with w=0, confined to the diagonal of the cost matrix, ERP is constant when w=0 regardless of the g value, i.e. $\mathrm{ERP}_{g+k,0}(S, T) = \mathrm{ERP}_{g,0}(S, T)$.

2.5 Move–Split–Merge

The Move–Split–Merge (MSM) distance is a metric proposed to overcome the limitations of existing distance measures: the Euclidean distance is not robust to temporal misalignment; edit distances such as DTW and LCSS are not metric; the ERP distance is a metric but not translation invariant due to the way the gap cost is computed. MSM is a metric, robust to temporal misalignment and translation invariant [32]. It is parameterized by an additive penalty value $\mathtt{c}$ that is used to compute the cost of aligning off-diagonal alignments, described in Eq. 5. This cost function takes in the new point (np) of the off-diagonal alignment and the two previously considered points (x and y). If the new point is within the bounds of x and y, then the penalty is only be $\mathtt{c}$.

$$\mathcal{C}(\mathrm{np}, x, y) = \begin{cases} \mathtt{c} & \text{if } x \leq \mathrm{np} \leq y \text{ or } x \geq \mathrm{np} \geq y \\ \mathtt{c} + \min \begin{cases} |\mathrm{np} - x| \\ |\mathrm{np} - y| \end{cases} & \text{otherwise} \end{cases} \tag{5}$$

$$M_{\mathrm{MSM}}(i, j) = \min \begin{cases} M_{\mathrm{MSM}}(i-1, j-1) + \left|S_i - T_j\right| \\ M_{\mathrm{MSM}}(i-1, j) + \mathcal{C}(S_i, S_{i-1}, T_j) \\ M_{\mathrm{MSM}}(i, j-1) + \mathcal{C}(T_j, S_i, T_{j-1}) \end{cases} \tag{6}$$

The MSM distance increases with increasing parameter c and stays constant after some sufficiently large c because deviating from the diagonal is too costly, becoming similar to a

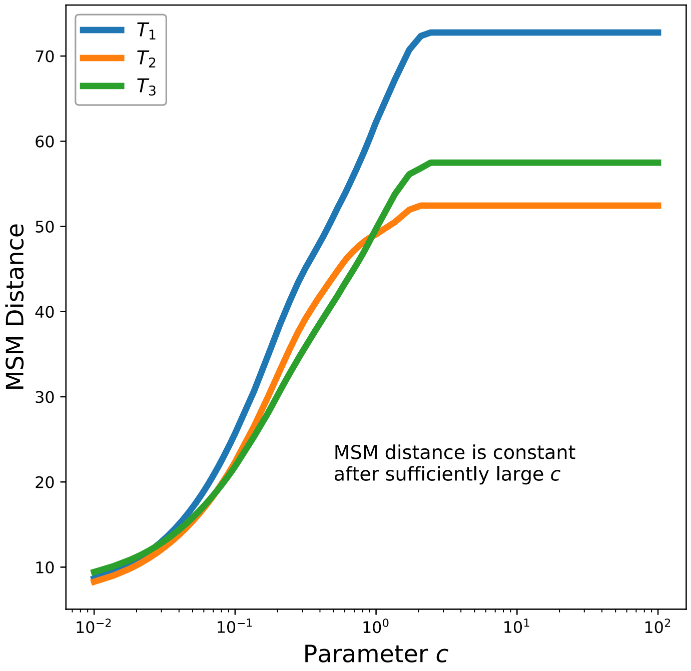


Fig. 9 MSM distances at different c parameters. Note x-axis in log scale

w=0 scenario. Figure 9 shows that $\mathrm{MSM}_c(S, T) \geq \mathrm{MSM}_{c+k}(S, T)$ for $k > 0$ and that we can use $\mathrm{MSM}_c(S, T)$ to lower bound $\mathrm{MSM}_{c+k}(S, T)$.

2.6 Time Warp Edit Distance

Existing distance measures assume that the time series are uniformly sampled and they do not consider timestamps in aligning time series. The Time Warp Edit Distance (TWE) was designed to take into account the timestamps to better align time series that are not uniformly sampled [21]. It uses three main operations (*delete*$_A$, *delete*$_B$ and *match*) to align the time series. TWE is parameterized by two meta-parameters, v and λ. v controls the "stiffness" of aligning two time series by weighing the contributions from the timestamps. v=∞ means that points that are off-diagonal of the matrix are not considered and is similar to the Euclidean distance while v=0 is similar to DTW [21]. The *match* operation is the sum of the cost between the current and previous data points and the weighted contribution of the respective timestamps using v. λ is a constant penalty added to the cost of the alternate alignments, i.e. the *delete* operations. The cost of all the three operations and the computation of each elements in the cost matrix M_{TWE} are described in Eqs. 7 and 8, respectively. Note that t_{S_i} denotes the i-th timestamp of a series S. Our implementation assumes the time series are uniformly sampled and does not use timestamp, thus t_{S_i}=i.

$$\begin{aligned} match:\ \gamma_M &= (S_i - T_j)^2 + (S_{i-1} - T_{j-1})^2 + v(\left|t_{S_i} - t_{T_j}\right| + \left|t_{S_{i-1}} - t_{T_{j-1}}\right|) \\ delete_A:\ \gamma_A &= \quad (S_i - S_{i-1})^2 + v(\left|t_{S_i} - t_{S_{i-1}}\right|) + \lambda \\ delete_B:\ \gamma_B &= \quad (T_i - T_{i-1})^2 + v(\left|t_{T_i} - t_{T_{i-1}}\right|) + \lambda \end{aligned} \tag{7}$$

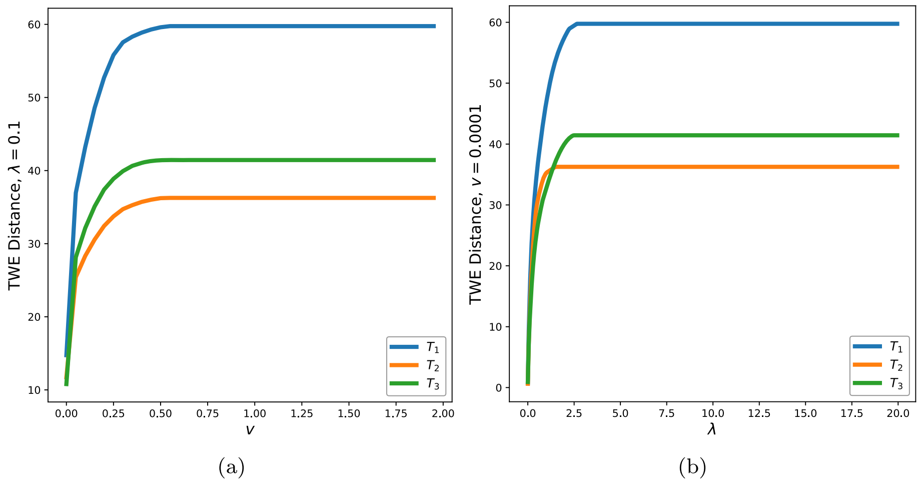


Fig. 10 TWE distances at different **a** v when λ=0.1 and **b** λ when v=0.0001

$$M_{\text{TWE}}(i,j) = \min \begin{cases} M_{\text{TWE}}(i-1,j-1)+\gamma_M & \text{match} \\ M_{\text{TWE}}(i-1,j)+\gamma_A & \text{delete}_A \\ M_{\text{TWE}}(i,j-1)+\gamma_B & \text{delete}_B \end{cases} \tag{8}$$

Figure 10 shows that TWE increases monotonically with v and λ, allowing us to use distances computed at the smaller parameters to lower bound distances at larger parameters. Similarly, TWE distance becomes constant at a sufficiently large v and λ, where the cost of doing the *delete* operations becomes too costly. Hence, $\text{TWE}_{v+k,\lambda}(S,T) \geq \text{TWE}_{v,\lambda}(S,T)$ and $\text{TWE}_{v,\lambda+k}(S,T) \geq \text{TWE}_{v,\lambda}(S,T)$ for $k > 0$.

2.7 Learning the optimal meta-parameter

There are two main advantages of learning the optimal meta-parameter. First, a good meta-parameter provides better results (e.g. classification accuracy) with NN search. For instance, learning the best DTW warping window prevents spurious alignments, which in turn improves the classification accuracy [8, 14, 34]. Recent research [3, 18, 34, 37] demonstrated that learning the optimal meta-parameter for each distance measures significantly improves classification accuracy. Improvements in accuracy as great as from 65% to 93% have been demonstrated [34]. Second, the window meta-parameter reduces the time complexity down to $O(w.L)$. By doing so, distance measures with a window meta-parameter, such as DTW, LCSS and ERP can be significantly faster to compute than without using any windows, especially for small windows. The usual approach for discovering the optimal meta-parameter is through leave-one-out cross-validation (LOOCV) [3, 7, 8, 18] by optimizing a performance metric such as training accuracy. This can be seen as creating a $(N \times L)$ table as shown in Table 1, giving the nearest neighbour of every time series for all meta-parameters and finding the column that gives the best training accuracy.

In this paper, we use 1-NN which has been widely used for benchmarking distance-based TSC algorithms [18, 37]. Note that our UltraFastMPSearch can easily be extended to cases where more than 1 neighbour ($k > 1$) is desired. This is done by adding a third

Table 1 Table of NNs for each meta-parameter. A cell $(i,p){=}\mathcal{T}_k(d)$ means $\mathcal{T}_i$ has $\mathcal{T}_k$ as its NN for meta-parameter p with distance d

	Nearest neighbour at meta-parameters				
	0	1	$\cdots$	$P-2$	$P-1$
$\mathcal{T}_1$	$\mathcal{T}_{24}(2.57)$	$\mathcal{T}_{55}(0.98)$	$\cdots$	$\mathcal{T}_{55}(0.98)$	$\mathcal{T}_{55}(0.98)$
$\vdots$			$\vdots$		
$\mathcal{T}_N$	$\mathcal{T}_{60}(4.04)$	$\mathcal{T}_{47}(1.61)$	$\cdots$	$\mathcal{T}_{47}(1.61)$	$\mathcal{T}_{47}(1.61)$

dimension, k to Table 1 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 meta-parameter search implementation has $O(N^2.L^2.P)$ time complexity (L^2 for each distance measure, repeated over P meta-parameters). To be consistent with distance-based TSC literature [18, 37], we used $P{=}100$ in our implementation. For measures with 2 meta-parameters, 10 of each meta-parameter are sampled from a specified distribution (discussed later), forming 100 parameter combinations. We leave the exploration of different parameter search space to future work. For the rest of the paper, we use a parameter ID notation, p to refer to the parameter space for each distance measure. For example for single parameter measures like DTW, $p{=}0$ refers to $w{=}0$, while for two-parameters measures like LCSS, $p{=}1$ refers to a combination of δ and ε such as $\delta{=}0$ and $\varepsilon{=}0.2$. Due to its L^2 complexity, meta-parameter search does not scale to long series [34, 35]. Note that FastWWSearch, FastEE and UltraFastMPSearch primarily tackle the impact of the L^2 part of the complexity, by minimizing the number of times the $O(L^2)$ distance 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. Figure 15a, b clearly illustrate this. When the length of the series increases (Fig. 15a), UltraFastMPSearch, especially UltraFastWWSearch scales extremely well compared to the state of the art. On the other hand, when the number of series increases (Fig. 15b), both methods suffer from the associated quadratic complexity, although UltraFastMPSearch does better.

Discovering the optimal meta-parameter for all distance measures is so expensive 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 [7], and by up to 5% on some datasets [2]. The sole reason for dropping EE is its computational burden being too great for it to be considered feasible to employ. In the future, UltraFastMPSearch may allow to reinstate a more efficient implementation of EE in HIVE-COTE, providing a substantial improvement to the state of the art.

3 Related work

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

3.1 Lower bounding

Filling out the NNs table in Table 1 can be considered as finding the nearest neighbour for each time series T within $\mathcal{T}$ at each meta-parameter. It follows that one way to speed up LOOCV is to speed up NN search. A common approach to speeding up NN search is through "lower

Algorithm 1: NN search

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

Algorithm 2: Lower bounded NN search

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

bounding” [15–17, 24, 36, 37, 41]. A NN search returns the nearest neighbour $\mathcal{T}_{\text{nn}}$ of a query S among a dataset $\mathcal{T}$, i.e. we have $d_{\text{nn}} = \text{DIST}(S, \mathcal{T}_{\text{nn}})$ such that $\forall T \in \mathcal{T}, d_{\text{nn}} \leq \text{DIST}(S, T)$, where DIST denotes a time series distance measures (Algorithm 1).

It turns out that NN search naturally supports lower bounding (Algorithm 2). First, in Algorithm 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 $\text{LB}(S, T) \leq \text{DIST}(S, T)$ allows a costly DIST computation to be avoided if $\text{LB}(S, T) \geq \text{UB}$ as $\text{LB}(S, T) \geq \text{UB} \vdash \text{DIST}(S, T) \geq \text{UB}$ (Algorithm 2).

An efficient lower bound must be fast while having good approximations (“tight”) [15, 36]. Because these aims compete, lower bounds of increasing tightness and cost are used in cascade, e.g. in the UCR- Suite [24]. Various lower bounds have been developed for DTW. The most common lower bounds for DTW are LB_Kim [16] and LB_Keogh [15]. The UCR- Suite [24] 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. [34] show that although UCR- Suite is faster than naive LOOCV, it is still significantly slower than FastWWSearch in optimizing for DTW’s warping window. Lower bounds for other time series distance measures have not been well explored. Tan et al. [37] developed lower bounds for other time series distance measures and demonstrated that they can significantly speed up the meta-parameter optimization process.

3.2 Improving distance measures implementation

Speeding up the distance measure itself also speeds up the whole LOOCV process. PrunedDTW is one of the first approaches to speed up DTW computations [30]. It first computes an upper bound on the result (the Euclidean distance), then skips cells from the cost matrix M_{DTW} that are larger. It was later extended to use the upper bound UB compute by the NN search process (d_{nn} in Algorithm 2), allowing early abandoning [31]. These techniques only yielded a minimal improvement when applied to window search [34].

Recently, Herrmann and Webb [11] developed the new “EAP” (Early Abandoned and Pruned) strategy, which is tightly integrating pruning and early abandoning for the six time series distance measures considered in this paper. EAP supports the fastest known time series distance measure 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 Algorithm 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 the distance matrix M, then no warping path can exist. Note that in Algorithm 2, the initial upper bound is set to ∞, which does not allow to prune anything when using EAP. By initializing it to the diagonal of M (i.e. squared Euclidean distance for DTW), EAP can prune some cells of M from the start but not early abandon.

3.3 FastWWSearch and FastEE

FastWWSearch [34] is a window optimization algorithm for DTW, 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 1); DTW is monotonic with w (Section 2); 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 [37], a significantly faster implementation of EE. It uses the lower bounds to the distance measures and exploits their properties described in Sect. 2. We refer each of the distance measure (except cDTW) to their respective FastEE method as FastWDTW, FastLCSS, FastERP, FastMSM and FastTWE. Each distance measure in EE contains 100 parameter values in the search space. FastWWSearch was originally designed to search through L warping windows for DTW and was modified to use only 100 warping windows (percentage of the time series length) in FastEE.

FastWDTW exploits the monotonic property of WDTW by starting from the largest g value, lower bounding WDTW at smaller g using WDTW at larger g. Since WDTW is continuous with g, it does not have a constant value, preventing FastWDTW to skip any distance computations in the WDTW parameter search space. The g parameter for WDTW is chosen from an uniform distribution $U(0, 1)$ with 100 values. FastMSM takes advantage of MSM property that it increases monotonically with its parameter c and stayed constant at some sufficiently large c. It uses MSM distances computed at a smaller c to lower bound MSM at the larger c. It also skips the computations at c when the distances are constant. The parameter c is sampled from an exponential sequence in the range of $[0.01, 100]$ with 100 values.

For LCSS, 10 ε values are sampled uniformly from the range $[\sigma/5, \sigma]$ where σ is the standard deviation of the training dataset, while other 10 δ values are sampled from the range $[0, L/4]$, forming a total of 100 parameter combinations. Then, they are arranged in such a way that the overall LCSS distance decreases with an increasing parameter ID (Fig. 11a), allowing FastLCSS to start from the largest parameter ID. Similar to WW, the δ parameter also allows FastLCSS to skip the computations at δs where the distances are constant.

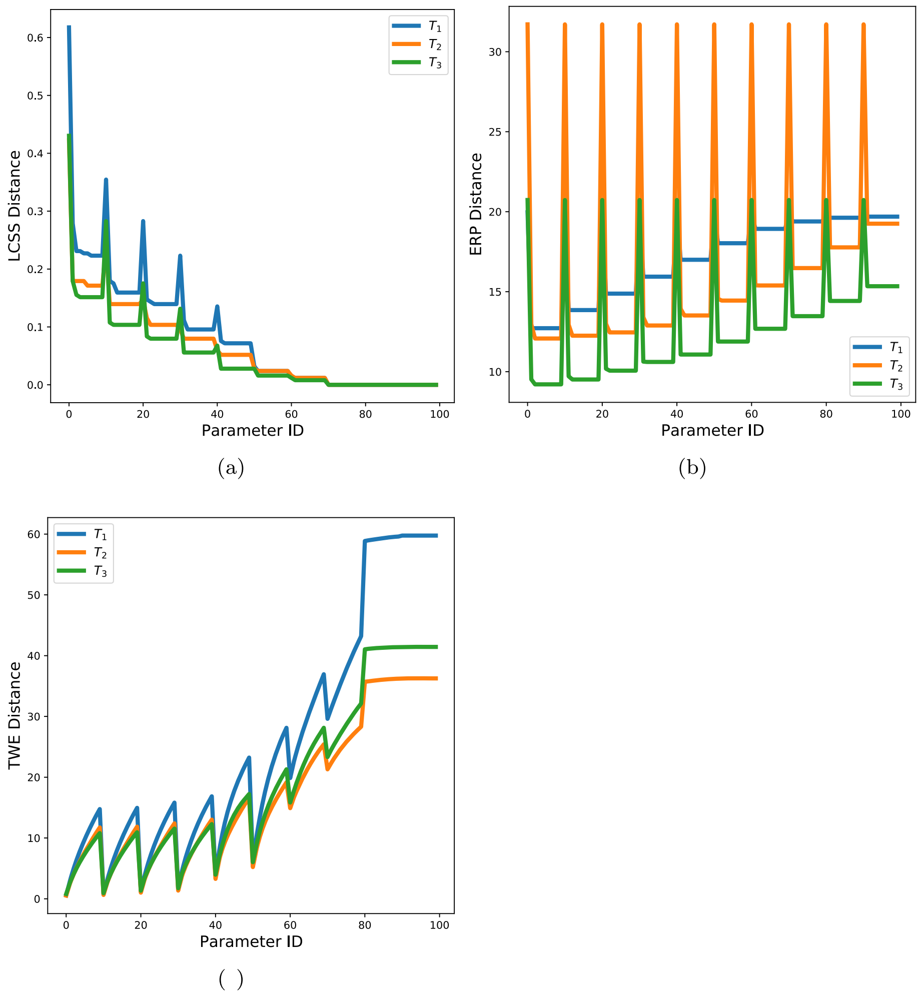


Fig. 11 **a** LCSS **b** ERP and **c** TWE distances at different parameter IDs

On the other hand, the meta-parameters combination for ERP and TWE is arranged such that the overall distances increase with an increasing parameter ID (Fig. 11b, c), allowing FastERP and FastTWE to start from the smaller parameter combination. The search space for ERP's g and w meta-parameter is chosen using the same way as LCSS. Note that regardless of the g value, the ERP distance is the same when the window parameter is 0. This means that there is a redundancy in the search space and will be explored as part of our future work. For TWE, 10 v values are sampled from an exponential distribution ranging from $[10^{-5}, 1]$, while 10 penalty parameters λ are chosen uniformly from the range $[0, 0.1]$.

4 Ultra-fast meta-parameter search

Our UltraFastMPSearch is a family of three algorithms, UltraFastWWSearch, UltraFastLocalUB and UltraFastGlobalUB. The UltraFastWWSearch algorithm was introduced in our ICDM2021 conference paper [35], designed specifically for the DTW distance measure. Both UltraFastLocalUB and UltraFastGlobalUB generalize to all distance measures that can be computed with EAP. The UltraFastLocalUB is a generalized version to UltraFastWWSearch that does not utilize the window validity property. UltraFastGlobalUB differs from UltraFastLocalUB such that a global upper bound is used instead of a local upper bound for EAP. A local upper bound refers to nearest neighbour distances at the current meta-parameter while a global upper bound refers to using nearest neighbour distances at some upper bound meta-parameter that gives an upper bound for a range of meta-parameters.

The significance of this difference lies in the consequences of EAP early abandoning a distance calculation. UltraFastMPSearch orders the meta-parameter values such that the distance at one value is a lower bound for the distance at the next. This is exploited to avoid calculating the distances at most values. However, if EAP early abandons a distance calculation, the exact distance is not known only that it is greater than the upper bound that was employed by EAP. Hence, that upper bound is the tightest lower bound available for the distance at the next parameter value. When distances only increase a small amount from one parameter value to the next, this is usually sufficient for effective lower bounding. However, when they increase more substantially, it means that opportunities to exploit lower bounding are lost. In this case, it is better to use a weaker upper bound such that if a distance turns out to be a useful lower bound for some subsequent parameter value it will be found. The strongest such weaker bound is the *global upper bound*—the minimum value that could allow the current candidate series to be a nearest neighbour at the final parameter value in the current sequence. Our experiments in Sect. 5 show that some distance measures are more effective using the local and others the global upper bound approach.

The core of UltraFastMPSearch is built upon FastEE [37] with the following key differences:

1. It replaces all calls to distance measures with the EAP variant [11];
2. It takes full advantage of EAP's early abandoning and pruning;
3. It processes the time series in an order that best exploit EAP's capabilities; and
4. It does not use any custom lower bounds.

For the rest of the paper, all distance measures should be understood as being the EAP variant unless specified otherwise.

Recall that learning the meta-parameter can be thought as filling up a $(N \times P)$ nearest neighbour table, as shown in FastEE and illustrated in Table 1. Thus, it is important to note that FastEE and all algorithms under UltraFastMPSearch share the same space complexity. Once this table has been filled, we can easily determine the best meta-parameter for a particular problem by looking for the column that gives the best performance. In case of ties, we take the meta-parameter that is cheaper to compute at test time, for instance, smaller w for DTW. Algorithm 3 describes this process. The result is identical to FastEE and LOOCV. In general, Algorithm 3 can be transformed to either FastEE or LOOCV by replacing the InitTable algorithm in line 1 with the specified algorithm to fill the NNs table. For LOOCV, this is naively filling the table described in Sect. 2.7. Each of the UltraFastWWSearch, UltraFastLocalUB and UltraFastGlobalUB has their own InitTable method, which will be discussed later in this section.

[t!]

In the following, we describe UltraFastMPSearch using UltraFastWWSearch and explain the key differences in both UltraFastLocalUB and UltraFastGlobalUB.

Algorithm 3: UltraFastMPSearch

```
Data: $\mathcal{T}$: training data
Input: InitTable: a function to initialise the NNs table
Result: $p^\star$: best meta-parameter
// Find all nearest neighbors
1 NNs ← InitTable ($\mathcal{T}$)
// Find meta-parameter with fewest misclassifications
2 bestNErrors ← $\|\mathcal{T}\| + 1$
3 for $p \leftarrow 0$ to $P-1$ do
4     nErrors ← 0
5     foreach $T \in \mathcal{T}$ do
6         if NNs[$T$][$p$].class $\neq T$.class then nErrors++
7     if nErrors < bestNErrors then
8         (bestNErrors, $p^\star$) ← (nErrors, $p$)
```

4.1 Ultra-fast warping window search

UltraFastWWSearch takes advantage of the properties of DTW (Sect. 2.1.1), 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.

The AssessNN algorithm in Algorithm 4 is a function that assesses whether a given pair of time series (S, T) is less than some distance d apart for a meta-parameter p. For DTW, $p{=}w$ is the warping window and DIST = DTW. AssessNN differs substantially from the FastWWSearch function on which it is based, LazyAssessNN, 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.

Algorithm 4 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.

Algorithm 4: AssessNN(p, d, UB, S, T)

```
Input: $p$: the meta-parameter
Input: $d$: the distance to beat
Input: UB: the upper bound to early abandon
Input: $S, T$: the time series to evaluate
Result: DIST$_p(S, T)$ 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   $\mathcal{C}_{(S,T)} \leftarrow$ DIST$_p(S, T,$ 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
```

Algorithm 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 $\leftarrow$ max (NNs[$S$][$w$].dist, NNs[$T$][$w$].dist)
2 if UB $= \infty$ then $UB \leftarrow \mathcal{C}_{(S,T)}$.do_euclidean
  // Check if $T$ is NNs[$S$][$w$]
3 toBeat $\leftarrow$ NNs[$S$][$w$].dist
4 if AssessNN($w$, toBeat, UB, $S$, $T$) $\neq$ abort then
5   NNs[$S$][$w$] $\leftarrow (T, \mathcal{C}_{(S,T)})$
  // Check if $S$ is NNs[$T$][$w$]
6 toBeatT $\leftarrow$ NNs[$T$][$w$].dist
7 if AssessNN($w$, toBeatT, UB, $S$, $T$) $\neq$ abort then
8   NNs[$T$][$w$] $\leftarrow (S, \mathcal{C}_{(S,T)})$
```

Algorithm 3 is used to fill up the NNs table to learn the warping window for DTW. Algorithm 3 can be transformed into FastWWSearch by replacing line 1 with Algorithm 3 in [34] for FastWWSearch. We use Algorithm 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 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 $(2 \times P)$-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 neighbour of $\mathcal{T}_1$ and/or $\mathcal{T}_2$. This is described in Algorithm 5. We can then add a fourth time series $\mathcal{T}_4$ and so on until $\mathcal{T}' = \mathcal{T}$.

Algorithm 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 a local 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 minimized.

Using EAP alone has provided a significant boost to the speed of FastWWSearch, which will be shown in our experiments in Sect. 5.

Lines 1 to 2 of Algorithm 5 calculate the upper bound that will be used to early abandon and prune EAP. The upper bound is calculated as UB = max(NNs[S][w].dist, NNs[T][w].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 the Euclidean distance between S and T to use it as the UB for EAP. The Euclidean distance 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 Algorithm 4 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 NNs[S][w] and NNs[T][w].

The core of UltraFastWWSearch lies in Algorithm 6. In line 1, we start by initializing the NNs table to (_, $+\infty$), an otherwise empty table with $+\infty$ nearest neighbour distances. Then, we initialize $\mathcal{T}'$, 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 4 to 10.

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 UB = max(NNs[S][w].dist, NNs[T][w].dist). NNs[S][w].dist starts at ∞ and can only decrease with each successive T. Hence, it is most productive to pair it initially with Ts with larger NNs[T][w].dist, as the max will be large anyway, and to pair it with the smallest NNs[T][w].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 6 to 9.

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 14, 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 neighbours 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 NNs[T][L] for all $T \in \mathcal{T}'$. By keeping track of ω, we can quickly skip all the windows where the distances are constant for all $T \in \mathcal{T}'$. On line 11, 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 12 and 13, we also need to propagate the NN for all $T \in \mathcal{T}'$ if NNs[T][L]=S.

From line 16, we continue to process the windows from $\omega{-}1$ down to 0. Line 17 checks if we already have a NN for S from larger windows due to window validity. Lines 18 to 23 check whether S is the NN of any $T \in \mathcal{T}'$. Since we already have the NN for S, the process

Algorithm 6: UltraFastFillNNTable($\mathcal{T}$)

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

Algorithm 7: UPDATENN- LOCAL(S, T, NNS, p, p^{UB})

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

is the similar to lines 6-8 of Algorithm 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 24) 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 $\mathcal{T}_{\mathrm{NN}^{w+1}}$ is updated with Algorithm 5. The rest of $T \in \mathcal{T}' \setminus \mathcal{T}_{\mathrm{NN}^{w+1}}$ is processed similar to Algorithm 5, except that we need to keep track of $\mathcal{T}_{\mathrm{NN}^{w+1}}$ (lines 27–35). Finally, we have NNs[S][w], the NN of S at w; we need to propagate the information for all valid windows (line 36).

4.2 Ultra-fast meta-parameter search with local upper bound

ULTRAFASTWWSEARCH is very specific to DTW and is not applicable to other distance measures. Hence, we designed a more generic algorithm, ULTRAFASTLOCALUB that is applicable to all time series distance measures that can be calculated using EAP, including DTW. The main difference with ULTRAFASTWWSEARCH is that it that does not exploit the window validity property in DTW, as not all measures have this property.

ULTRAFASTLOCALUB requires some slight modifications to generalize the algorithms in ULTRAFASTWWSEARCH to other distance measures. Starting from Algorithm 5, Algorithm 7 replaces all w to p and takes in an additional input, p^{UB}. p^{UB} is the meta-parameter that gives the upper bound distance in the parameter search space. This parameter allows us to first compute $\mathcal{C}_{(S,T,p^{\mathrm{UB}})}$.do_upperBound= $\mathrm{DIST}_{p^{\mathrm{UB}}}(S, T)$, the distance at the upper bound meta-parameter, which is then cached for future use. This is similar to computing the Euclidean distance as the upper bound for DTW where w=0. For DTW, WDTW and ERP, this upper bound distance is equivalent to computing the diagonal of the distance matrix M and can be computed in $O(L)$ time instead of $O(L^2)$. Similar to ULTRAFASTWWSEARCH, if we do not have a best-so-far NN for either S or T yet, then we compute the upper bound distance between S and T at parameter p^{UB}. The rest of the algorithm is similar to Algorithm 5 in ULTRAFASTWWSEARCH.

Then UltraFastLocalUB replaces Algorithm 6 in Algorithm 3 with Algorithm 8 to fill the NNs table. Algorithm 8 describes the general process of filling the NNs table for any time series distance measures that support EAP. For ease of exposition, we assume that the meta-parameter p increases from 0 to P but implemented it based on the distance measure's properties described in Sect. 2. There are two main differences to Algorithm 6. First, it does not utilize the maximum window validity as per UltraFastWWSearch, because not all distance measures have the window meta-parameter. Second, it needs to keep track of the upper bound parameter ID, p^{UB}. Distance measures with two meta-parameters have multiple upper bound distances along the parameter search space and when we do not have a nearest neighbour yet, we want to use the tightest upper bound possible for EAP – the upper bound that is closest to the current distance. Hence it is important to keep updating p^{UB} while going through the search space.

The upper bound parameter p^{UB} should be updated according to the properties of each distance measure, described in Sect. 2. p^{UB} is constant for single meta-parameter measures, i.e. $p^{\text{UB}}_{\text{DTW}}=p^{\text{UB}}_{\text{WDTW}}=0$ and $p^{\text{UB}}_{\text{MSM}}=P$. In this work, we ordered the parameters for two meta-parameters measures such that every 10 meta-parameter is an upper bound to the previous 9. This means that we need to update p^{UB} at every 10-th meta-parameter. For the special case of ERP, the upper bound is when $w=0$ which gives the same ERP distance for all g, i.e. the upper bound for ERP is the same in this parameter space. When we are processing a new query, we have to reset p^{UB} to the first upper bound, as shown in line 4 of Algorithm 8. Then line 15 of Algorithm 8 checks and update p^{UB} after changing to the next parameter.

4.3 Ultra-fast meta-parameter search with global upper bound

UltraFastLocalUB takes the maximum of the nearest neighbour distance of S and T at the current meta-parameter p as the upper bound. We call this a local upper bound as this is only applicable to the current meta-parameter. Instead of using the nearest neighbour of S and T at the current meta-parameter p, UltraFastGlobalUB uses a global upper bound, i.e. the nearest neighbour distance of S and T at the parameter p^{UB}. The global upper bound is applicable to a range of previous meta-parameters. Note that by definition, the global upper bound is looser than the local upper bound.

We modify Algorithm 7 by replacing line 1 with UB = max(NNs[S][p^{UB}].dist, NNs[T][p^{UB}].dist) as presented in Algorithm 9. The rest of Algorithm 9 is the same as Algorithm 7. Similarly Algorithm 8 is also modified with respect to the global upper bound by changing lines 19 and 26. Note that as the global upper bound will be used a lot, it is important to cache it for future use.

5 Experiments

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

Our experiments use all of the 128 benchmark UCR time series datasets [7]. For each method, we perform the search using the set of 100 meta-parameters (Sect. 3.3) used in EE [18] and FastEE [37]. This allows UltraFastMPSearch to be directly used in EE. Since the

Algorithm 8: UltraFastFillNNTable- Local($\mathcal{T}$)

```
Input: $\mathcal{T}$ the set of time series
Result: NNs the nearest neighbors table
1  NNs.fillAll(_, $+\infty$), $\mathcal{T}' \leftarrow \emptyset$
2  for $s \leftarrow 2$ to $N$ do
       // Update NNs wrt adding $S$
3      $S \leftarrow \mathcal{T}_s$, $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{\mathcal{T}_{s-1}\}$
4      $p^{UB}$.reset // reset upper bound parameter
5      $p \leftarrow 0$ // first meta-parameter, smallest DIST
6      foreach $T \in \mathcal{T}'$ in des. order of NNs[$T$][$w$].dist do
7          $\mathcal{C}_{S,T} \leftarrow \varnothing$
8          UpdateNN- Local($S$, $T$, NNs, $p$)
       // Propagate NN for all valid meta-parameters
9      for $p' \in$ NNs[$S$][$p$].valid do NNs[$S$][$p'$] $\leftarrow$ NNs[$S$][$p$]
10     foreach $T \in \mathcal{T}'$ if NNs[$T$][$p$] $= S$ do
11         for $p' \in$ NNs[$T$][$p$].valid do NNs[$T$][$p'$] $\leftarrow$ NNs[$T$][$p$]
       // Sort $\mathcal{T}'$ in asc order using $\mathcal{C}$ once
12     $\mathcal{T}' \leftarrow \mathcal{T}'$.sort
       // remember NN at previous param ($p-1$)
13     $\mathcal{T}_{\mathrm{NN}^{p-1}} \leftarrow \mathcal{T}'_0$
14     for $p \leftarrow 1$ to $P-1$ do
15         $p^{UB}$.update // update upper bound parameter
16         if NNs[$s$][$p$] $\neq \varnothing$ then
               // Update NNs[$T$][$p$] for $T \in \mathcal{T}'$
17             foreach $T \in \mathcal{T}'$ do
18                 toBeat $\leftarrow$ NNs[$T$][$p$].dist
19                 UB $\leftarrow$ toBeat
20                 if UB $= \infty$ then UB $\leftarrow \mathcal{C}_{(S,T,p^{\mathrm{UB}})}$.do_upperBound
21                 if AssessNN($p$, toBeat, UB, $S$, $T$) $\neq$ abort then
22                     NNs[$T$][$p$] $\leftarrow (S, \mathcal{C}_{(S,T)})$
23         else
               // Start from the NN at $p-1$
24             UpdateNN- Local($S$, $\mathcal{T}_{\mathrm{NN}^{p-1}}$, NNs, $p$)
25             foreach $T \in \mathcal{T}' \setminus \mathcal{T}_{\mathrm{NN}^{p-1}}$ do
26                 UB $\leftarrow$ max (NNs[$S$][$p$].dist, NNs[$T$][$p$].dist)
27                 if UB $= \infty$ then $UB \leftarrow \mathcal{C}_{(S,T,p^{UB})}$.do_upperBound
28                 toBeat $\leftarrow$ NNs[$S$][$p$].dist
29                 if AssessNN($p$, toBeat, UB, $S$, $T$) $\neq$ abort then
30                     NNs[$S$][$p$] $\leftarrow (T, \mathcal{C}_{(S,T)})$
31                     $\mathcal{T}_{\mathrm{NN}^{p-1}} \leftarrow T$
32                 toBeatT $\leftarrow$ NNs[$T$][$p$].dist
33                 if AssessNN($p$, toBeatT, UB, $S$, $T$) $\neq$ abort then
34                     NNs[$T$][$p$] $\leftarrow (S, \mathcal{C}_{(S,T)})$
               // Propagate NN for all valid meta-parameters
35             for $p' \in$ NNs[$S$][$p$].valid do NNs[$S$][$p'$] $\leftarrow$ NNs[$S$][$p$]
```

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 standard 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 with DTW on LOOCV. (B) Then, we explore the features of UltraFastWWSearch that

Algorithm 9: UPDATENN- GLOBAL(S, T, NNS, p, p^{UB})

```
Input: $S$ the query time series
Input: $T$ the candidate time series
Input: NNs the nearest neighbors table
Input: $p$ the current meta-parameter
Input: $p^{\text{UB}}$ the upper bound meta-parameter
Result: NNs updated nearest neighbors table
   // Compute the upper bound UB
1  UB ← max (NNs[$S$][$p^{\text{UB}}$].dist, NNs[$T$][$p^{\text{UB}}$].dist)
2  if UB = ∞ then $UB \leftarrow \mathcal{C}_{(S,T,p^{\text{UB}})}$.do_upperbound
   // Check if $T$ is NNs[$S$][$p$]
3  toBeat ← NNs[$S$][$p$].dist
4  if AssessNN($w$, toBeat, UB, $S$, $T$) ≠ abort then
5  |   NNs[$S$][$w$] ← ($T$, $\mathcal{C}_{(S,T)}$)
   // Check if $S$ is NNs[$T$][$p$]
6  toBeatT ← NNs[$T$][$p$].dist
7  if AssessNN($p$, toBeatT, UB, $S$, $T$) ≠ abort then
8  |   NNs[$T$][$p$] ← ($S$, $\mathcal{C}_{(S,T)}$)
```

help it achieves significant speed up compared to the state of the art. (C) We investigate the scalability of ULTRAFASTWWSEARCH on large and long datasets. (D) Lastly, we investigate the generalization of ULTRAFASTWWSEARCH to other distance measures using ULTRAFASTLOCALUB and ULTRAFASTGLOBALUB.

5.1 Pruning and early abandoning with EAP

Given the dramatic speedup of EAP on NN-DTW [11], 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 Sect. 2.7 using NN-DTW with early abandoning strategy described in [24] but without lower bound and UB from Sect. 3.1. 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 [24].
3. EAP_LOOCV: Replacing DTW in DTW_LOOCV with EAP [11].

Figure 12a 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 1000 h (42 days) and about 300 h (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 [36]. The work in [36] 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.

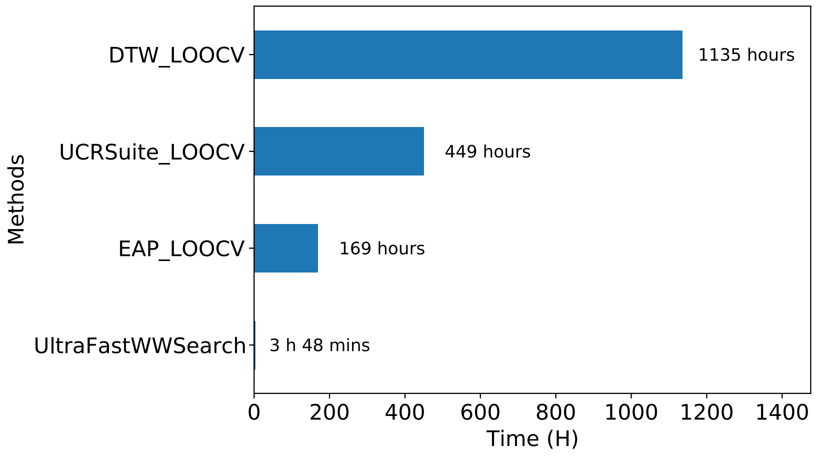
Algorithm 10: UltraFastFillNNTable- Global($\mathcal{T}$)

```
Input: $\mathcal{T}$ the set of time series
Result: NNs the nearest neighbors table
1  NNs.fillAll(_, $+\infty$), $\mathcal{T}' \leftarrow \emptyset$
2  for $s \leftarrow 2$ to $N$ do
       // Update NNs wrt adding $S$
3      $S \leftarrow \mathcal{T}_s$, $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{\mathcal{T}_{s-1}\}$
4      $p^{UB}$.reset // reset upper bound parameter
5      $p \leftarrow 0$ // first meta-parameter, smallest DIST
6      foreach $T \in \mathcal{T}'$ in des. order of NNs[$T$][$w$].dist do
7          $\mathcal{C}_{S,T} \leftarrow \varnothing$
8          UpdateNN- Global($S$, $T$, NNs, $p$)
       // Propagate NN for all valid meta-parameters
9      for $p' \in$ NNs[$S$][$p$].valid do NNs[$S$][$p'$] $\leftarrow$ NNs[$S$][$p$]
10     foreach $T \in \mathcal{T}'$ if NNs[$T$][$p$] $= S$ do
11         for $p' \in$ NNs[$T$][$p$].valid do NNs[$T$][$p'$] $\leftarrow$ NNs[$T$][$p$]
       // Sort $\mathcal{T}'$ in asc order using $\mathcal{C}$ once
12     $\mathcal{T}' \leftarrow \mathcal{T}'$.sort
       // remember NN at previous param ($p-1$)
13     $\mathcal{T}_{\text{NN}^{p-1}} \leftarrow \mathcal{T}'_0$
14     for $p \leftarrow 1$ to $P-1$ do
15         $p^{UB}$.update // update upper bound parameter
16         if NNs[$s$][$p$] $\neq \varnothing$ then
               // Update NNs[$T$][$p$] for $T \in \mathcal{T}'$
17             foreach $T \in \mathcal{T}'$ do
18                 toBeat $\leftarrow$ NNs[$T$][$p$].dist
19                 UB $\leftarrow$ max (NNs[$S$][$p^{\text{UB}}$].dist, NNs[$T$][$p^{\text{UB}}$].dist)
20                 if UB $= \infty$ then UB $\leftarrow \mathcal{C}_{(S,T,p^{\text{UB}})}$.do_upperBound
21                 if AssessNN($p$, toBeat, UB, $S$, $T$) $\neq$ abort then
22                     NNs[$T$][$p$] $\leftarrow (S, \mathcal{C}_{(S,T)})$
23         else
               // Start from the NN at $p-1$
24             UpdateNN- Global($S$, $\mathcal{T}_{\text{NN}^{p-1}}$, NNs, $p$)
25             foreach $T \in \mathcal{T}' \setminus \mathcal{T}_{\text{NN}^{p-1}}$ do
26                 UB $\leftarrow$ max (NNs[$S$][$p^{\text{UB}}$].dist, NNs[$T$][$p^{\text{UB}}$].dist)
27                 if UB $= \infty$ then $UB \leftarrow \mathcal{C}_{(S,T,p^{UB})}$.do_upperBound
28                 toBeat $\leftarrow$ NNs[$S$][$p$].dist
29                 if AssessNN($p$, toBeat, UB, $S$, $T$) $\neq$ abort then
30                     NNs[$S$][$p$] $\leftarrow (T, \mathcal{C}_{(S,T)})$
31                     $\mathcal{T}_{\text{NN}^{p-1}} \leftarrow T$
32                 toBeatT $\leftarrow$ NNs[$T$][$p$].dist
33                 if AssessNN($p$, toBeatT, UB, $S$, $T$) $\neq$ abort then
34                     NNs[$T$][$p$] $\leftarrow (S, \mathcal{C}_{(S,T)})$
               // Propagate NN for all valid meta-parameters
35             for $p' \in$ NNs[$S$][$p$].valid do NNs[$S$][$p'$] $\leftarrow$ NNs[$S$][$p$]
```

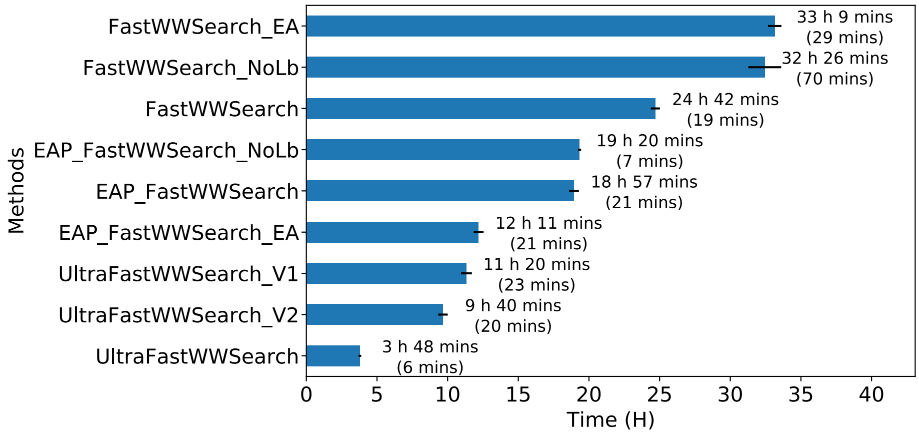
5.2 Speeding up the state of the art with UltraFastWWSearch

This section examines the features that make UltraFastWWSearch ultra-fast, comparing it to state-of-the-art FastWWSearch. The results are shown in Fig. 12b.

Much of EAP's speed up in many NN-DTW tasks actually comes from early abandoning (see Fig. 7 of [11]). Section 5.1 shows that EAP, even without using lower bounds, speeds up



(a)



(b)

Fig. 12 Total training time on 128 datasets [7] 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

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 [24] for the original FASTWWSEARCH and use the UB described in Sect. 3.1 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 [34] as well.

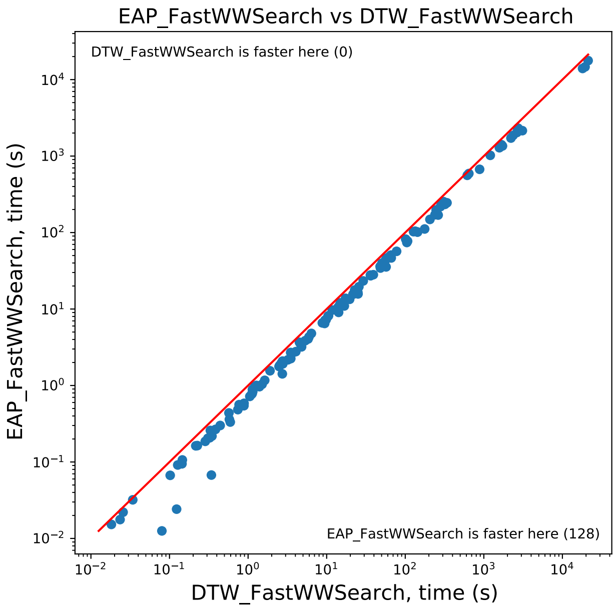


Fig. 13 Pairwise comparison on 128 UCR datasets of FastWWSearch with EAP_FastWWSearch

On the other hand, the opposite is observed for the EAP variants. EAP_FastWWSearch _NoLb in Fig. 12b 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 min. This is not surprising as it coincides with the results from the EAP paper [11]. 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 neighbour and then sorting on DTW$_L$; and UltraFastWWSearch: skipping windows from $L-1$ to ω, the maximum window validity at $L-1$.

Figure 12b 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 h. 3 h and 50 min of this comes from 5 long and large datasets, `NonInvasiveFetalECGThorax1`, `UWaveGestureLibraryAll`, `HandOutlines`, `FordA` and `FordB`. The pairwise plot in Fig. 13 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 Sect. 3.1. The V1 variant of UltraFastWWSearch uses the UB described in Algorithm 5. The results in Fig. 12b show that this UB improves the speed of UltraFastWWSearch but by a small margin.

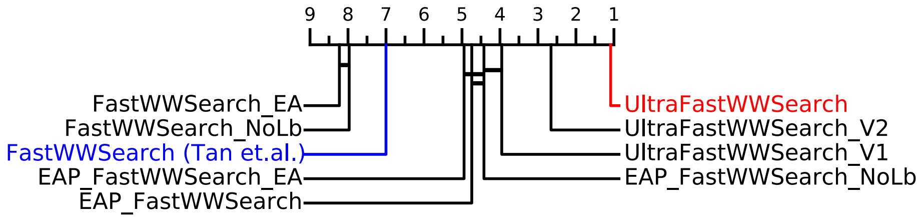


Fig. 14 Critical difference diagram comparing the training time of various methods on 128 datasets

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 NNs[T][w].dist (Algorithm 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. 12b.

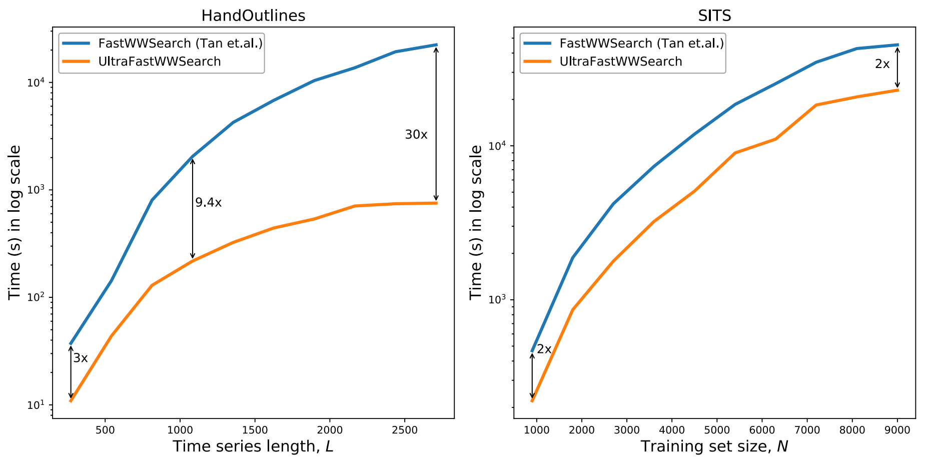
Finally, we add the optimization of skipping windows from $L-1$ to ω. While the five previous optimizations all exploit the properties of EAP, this final optimization is a novel further exploit of the window validity property beyond those in FastWWSearch. It more than halves the total time.

Figure 12b shows that UltraFastWWSearch is able to complete all 128 datasets in under 4 h. This is a 6 times speedup compared to 24 h 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 [10] to test the significance of our results and visualize it in a critical difference diagram, illustrated in Fig. 14. Figure 14 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. 12b, 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. 12b, it is ranked lower compared to 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. Figure 2 shows that UltraFastWWSearch is up to one order of magnitude faster than FastWWSearch.

5.3 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 [7]. We varied the length from $0.1 \times L$ to L, recording the time to search for the best warping window. Figure 15a 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 h of compute time down to 11 min (Fig. 1), thus effectively tackling the L^3 part of the complexity.



(a) (b)

Fig. 15 Training time versus **a** time series length L on `HandOutlines` [7], and **b** training set size N on `SITS` [38]

We then evaluated the scalability to larger datasets, using the same SITS dataset as [34], taken from [38]. We chose this dataset because it has a short length of L=46, which tends to isolate the influence of N on the scalability. Figure 15b 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 [34], requiring days to complete, while UltraFastWWSearch only takes 6 h for N=90, 000.

5.4 UltraFastMPSearch—Generalizing to other distance measures

The previous experiments showed that UltraFastWWSearch achieved a substantial speed up compared to FastWWSearch in optimizing DTW's warping window. In this section, we generalize UltraFastWWSearch to UltraFastLocalUB and UltraFastGlobalUB and apply them to all other time series distance measures. We only compare UltraFast-LocalUB and UltraFastGlobalUB to the baseline FastEE approaches for each of the measures and FastEE using EAP and early abandoning. We assume that LOOCV for each distance measure will take similar time as DTW_LOOCV as they all have the same $O(L^2)$ time complexity. Note that our proposed UltraFastWWSearch is by default using the local upper bound and we call the variant with the global upper bound, UltraFastWWSearch-Global which also uses the window validity to skip some DTW computations.

Figure 16 compares the total training time for the methods of each distance measures on the 128 datasets. Overall, we observed that using the local upper bound is more efficient for most measures except for MSM and TWE where the global upper bound is faster. This is expected because the local upper bound is by definition tighter than the global upper bound. The results show that UltraFastMPSearch significantly reduces the training time from FastEE methods for all distance measures:

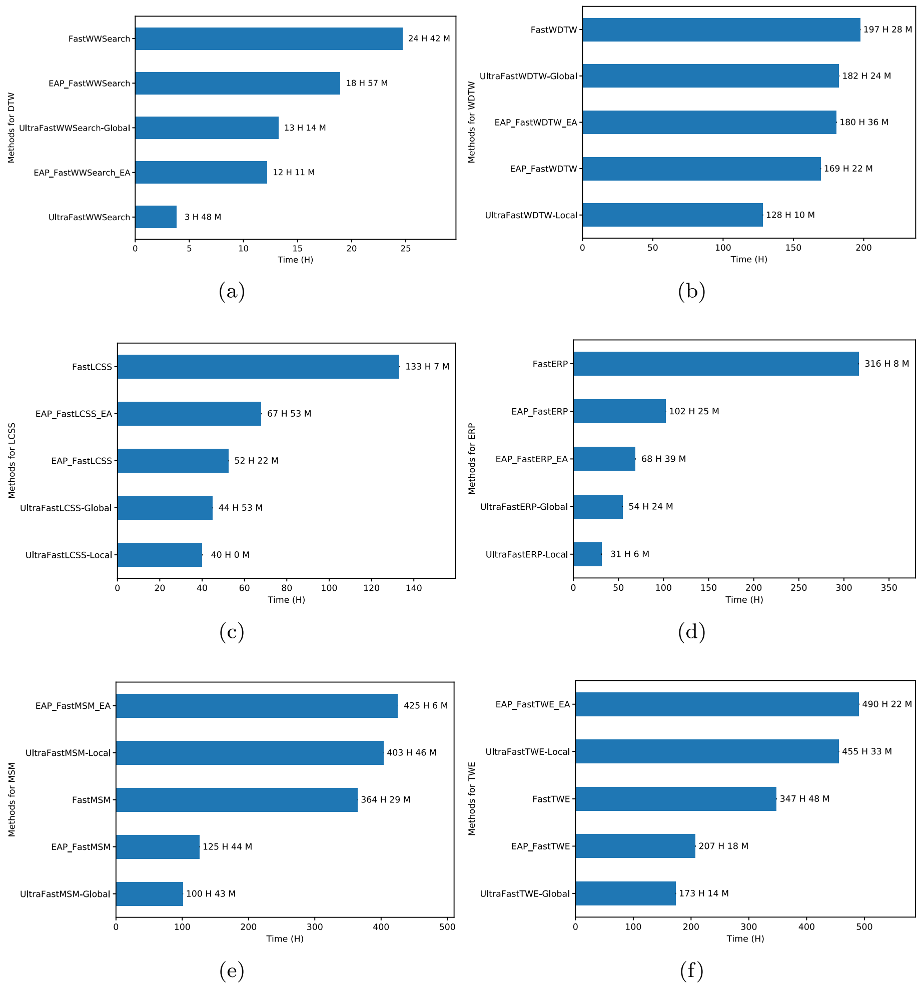


Fig. 16 Training time for **a** DTW **b** WDTW **c** LCSS **d** ERP **e** MSM and **f** TWE methods

1. DTW by 21 h ($\approx$ 1 day)
2. WDTW by 69 h ($\approx$ 3 days)
3. LCSS by 93 h ($\approx$ 4 days)
4. ERP by 285 h ($\approx$ 12 days)
5. MSM by 264 h ($\approx$ 11 days)
6. TWE by 174 h ($\approx$ 7 days)

This speed up translates to a gain of more than 38 days when they are used in EE setting (EE has 11 measures, including variations to these 6 measures).

ULTRAFASTLOCALUB is slower for TWE and MSM because they use an additive penalty for off-diagonal alignments—TWE adds a constant λ value while MSM adds a constant c value. These values are typically very small, in the range of less than 1 causing the distances in the parameter search space to be very similar. In the worst case scenario, the difference in the distance between two subsequent meta-parameters in the search space is the difference

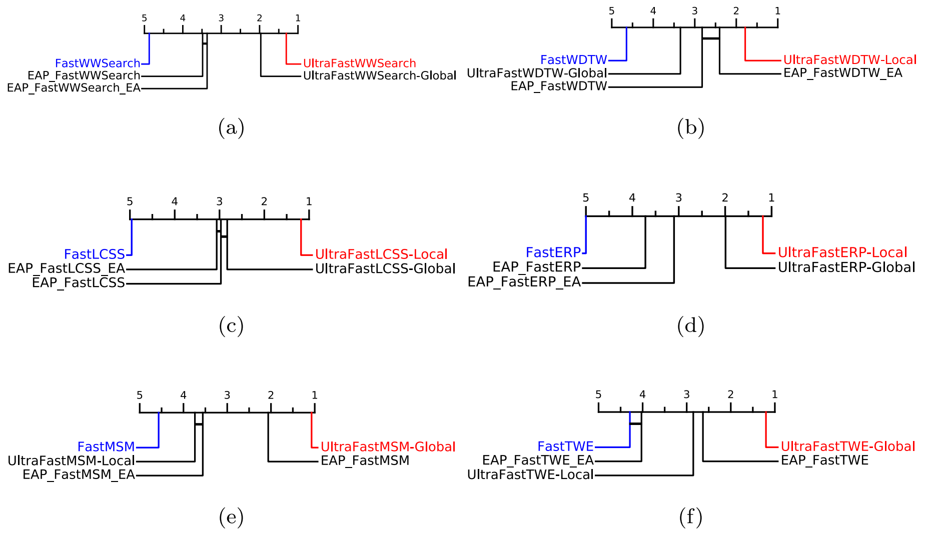


Fig. 17 Critical difference diagram comparing the training time for **a** DTW **b** WDTW **c** LCSS **d** ERP **e** MSM and **f** TWE methods

between two consecutive λ or c. As the distances will be very similar, this means that they will typically early abandon near the end of the matrix and almost computing the full distance. Recall that an early abandoned distance cannot be used as a lower bound for the subsequent meta-parameters in UltraFastMPSearch. So it becomes inefficient for the algorithm when the distance computation early abandons late too many times, and recomputing it again for the next meta-parameter. This is almost as if we are computing the distance computation twice and defeating the purpose of early abandoning.

Therefore, by using a looser global upper bound, UltraFastGlobalUB achieves a balance between the number of times the DIST function is called and the number of times it early abandons. In other words, if a distance computation early abandons using a larger global upper bound, then we know that the candidate T will never be the nearest neighbour between the current meta-parameter and p^{UB}, thus skipping all the unnecessary distance function calls. For other distance measures where the distances only grow by small amounts from one parameter value to the next, the number of times the global upper bound allows useful tighter lower bounds for subsequent parameter values is so few that the time saving is less than the time saved by the additional pruning EAP can achieve with the tighter local upper bounds.

Figure 17 shows the average ranking of each distance measure in terms of training time over all datasets using a critical difference diagram to compare the training time for all the distance measures. At a glance, UltraFastMPSearch is significantly faster than the current state of the art, FastEE methods. Similar to the results in Sect. 5.2, some of the methods like UltraFastWWSearch-Global, EAP-FastMSM-EA, UltraFastTWE-Local and EAP-FastTWE-EA have longer overall training time, as shown in Fig. 16, but ranked better in the critical difference diagram in Fig. 17. The reason is that these methods are slightly faster on most datasets, that are short and small and require little computation, but took longer on the larger and longer datasets for which overall computation is greatest.

6 Conclusion

This paper proposes UltraFastMPSearch—a family of ultra-fast algorithms that are able to learn the meta-parameters for time series distance measures 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. UltraFastLocalUB generalizes UltraFastWWSearch by not using the window validity to skip the meta-parameters. Instead of using the nearest neighbour distance at the current meta-parameter for EAP, UltraFastGlobalUB uses as the upper bound the nearest neighbour distance computed at a parameter value that provides an upper bound for a series of subsequent parameter values. This achieves a better balance for measures with distances that grow substantially between successive meta-parameters, such as MSM and TWE.

Our experiments show that the UltraFastMPSearch algorithms are 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 h on the UCR datasets, a time close to ROCKET, one of the fastest and most accurate TSC algorithms [9]. Similarly UltraFastMPSearch also speeds up the training of NN search with other distance measures, although the efficiency is not as great due to the lack of the window validity property. This speedup 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 [22]

Acknowledgements This work was supported by the Australian Research Council award DP210100072. 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. Alaee S, Mercer R, Kamgar K, Keogh E (2021) Time series motifs discovery under DTW allows more robust discovery of conserved structure. Data Min Knowl Disc 35(3):863–910
2. Bagnall A, Flynn M, Large J, Lines J, Middlehurst M (2020) A tale of two toolkits, report the third: on the usage and performance of HIVE-COTE v1.0. arXiv e-prints pp. arXiv–2004
3. Bagnall A, Lines J, Bostrom A, Large J, Keogh E (2017) The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min Knowl Disc 31(3):606–660
4. Boreczky JS, Rowe LA (1996) Comparison of video shot boundary detection techniques. J Electron Imaging 5(2):122–128

5. Chen L, Ng R (2004) On the marriage of Lp-norms and edit distance. In: Proceedings of the 30th international conference on very large databases (VLDB), pp 792–803
6. Chen L, Özsu MT , Oria V (2005) Robust and fast similarity search for moving object trajectories. In: Proceedings of the 2005 ACM SIGMOD international conference on management of data (SIGMOD), pp 491–502
7. Dau HA, Keogh E, Kamgar K, Yeh C-CM, Zhu Y, Gharghabi S, Ratanamahatana CA, Yanping Hu, B, Begum N, Bagnall A, Mueen A, Batista G, Hexagon-ML (2018) The UCR time series classification archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/
8. Dau HA, Silva DF, Petitjean F, Forestier G, Bagnall A, Mueen A, Keogh E (2018) Optimizing dynamic time warping's window width for time series data mining applications. Data Min Knowl Disc 32(4):1074–1120
9. Dempster A, Petitjean F, Webb GI (2020) ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. Data Min Knowl Disc 34(5):1454–1495
10. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
11. Herrmann M, Webb GI (2021) Early abandoning and pruning for elastic distances including dynamic time warping. Data Min Knowl Discov, pp 1–25
12. Itakura F (1975) Minimum prediction residual principle applied to speech recognition. IEEE Trans Acoust Speech Signal Process 23(1):67–72
13. Jeong Y-S, Jeong MK, Omitaomu OA (2011) Weighted dynamic time warping for time series classification. Pattern Recogn 44(9):2231–2240
14. Keogh EJ , Pazzani MJ (2001) Derivative dynamic time warping. In: Proceedings of the 2001 SIAM international conference on data mining, SIAM, pp 1–11
15. Keogh E, Ratanamahatana CA (2005) Exact indexing of dynamic time warping. Knowl Inf Syst 7(3):358–386
16. Kim S-W, Park S, Chu WW (2001) An index-based approach for similarity search supporting time warping in large sequence databases. In: Proceedings 17th international conference on data engineering, IEEE, pp 607–614
17. Lemire D (2009) Faster retrieval with a two-pass dynamic-time-warping lower bound. Pattern Recogn 42(9):2169–2180
18. Lines J, Bagnall A (2015) Time series classification with ensembles of elastic distance measures. Data Min Knowl Disc 29(3):565–592
19. Lines J, Taylor S, Bagnall A (2018) Time series classification with HIVE-COTE: the Hierarchical Vote Collective of Transformation-based Ensembles. ACM Trans Knowl Discov Data 12(5)
20. Lucas B, Shifaz A, Pelletier C, O'Neill L, Zaidi N, Goethals B, Petitjean F, Webb GI (2019) Proximity Forest: an effective and scalable distance-based classifier for time series. Data Min Knowl Disc 33(3):607–635
21. Marteau P-F (2008) Time warp edit distance with stiffness adjustment for time series matching. IEEE Trans Pattern Anal Mach Intell 31(2):306–318
22. Middlehurst M, Large J, Flynn M, Lines J, Bostrom A, Bagnall A (2021) Hive-cote 2.0: a new meta ensemble for time series classification. Mach Learn 110(11):3211–3243
23. Petitjean F, Ketterlin A, Gançarski P (2011) A global averaging method for dynamic time warping, with applications to clustering. Pattern Recogn 44(3):678–693
24. Rakthanmanon T, Campana B, Mueen A, Batista G, Westover B, Zhu Q, Zakaria J, Keogh E (2012) Searching and mining trillions of time series subsequences under dynamic time warping. In: Proceedings of 18th ACM SIGKDD international conference on knowledge discovery and data mining, pp 262–270
25. Ratanamahatana CA , Keogh E (2004) Making time-series classification more accurate using learned constraints. In: Proceedings of the 2004 SIAM international conference on data mining, SIAM, pp 11–22
26. Ratanamahatana CA, Keogh E (2005) Three myths about dynamic time warping data mining. In: Proceedings of the 2005 SIAM international conference on data mining, SIAM, pp 506–510
27. Sakoe H, Chiba S (1971) A dynamic programming approach to continuous speech recognition. In: International congress on acoustics, vol 3, pp 65–69
28. Salvador S, Chan P (2007) Toward accurate dynamic time warping in linear time and space. Intell Data Anal 11(5):561–580
29. Shifaz A, Pelletier C, Petitjean F, Webb GI (2020) TS-CHIEF: a scalable and accurate forest algorithm for time series classification. Data Min Knowl Disc 34(3):742–775
30. Silva DF, Batista GEAPA (2016) Speeding up all-pairwise dynamic time warping matrix calculation. In: Proceedings of the 2016 SIAM international conference on data mining, Society for Industrial and Applied Mathematics, pp 837–845
31. Silva DF, Giusti R, Keogh E, Batista GE (2018) Speeding up similarity search under dynamic time warping by pruning unpromising alignments. Data Min Knowl Disc 32(4):988–1016

32. Stefan A, Athitsos V, Das G (2012) The Move-Split-Merge metric for Time Series. IEEE Trans Knowl Data Eng 25(6):1425–1438
33. Tan CW, Bergmeir C, Petitjean F, Webb GI (2021) Time series extrinsic regression. Data Min Knowl Discov:1032–1060
34. Tan CW, Herrmann M, Forestier G, Webb GI, Petitjean F (2018) Efficient search of the best warping window for dynamic time warping. In: Proceedings of the 2018 SIAM international conference on data mining, SIAM, pp 225–233
35. Tan CW, Herrmann M , Webb GI (2021) Ultra fast warping window optimization for dynamic time warping. In: 2021 IEEE international conference on data mining, IEEE, pp 589–598
36. Tan CW, Petitjean F, Webb GI (2019) Elastic bands across the path: a new framework and method to lower bound DTW. In: Proceedings of the 2019 SIAM international conference on data mining, SIAM, pp 522–530
37. Tan CW, Petitjean F, Webb GI (2020) FastEE: fast ensembles of elastic distances for time series classification. Data Min Knowl Disc 34(1):231–272
38. Tan CW, Webb GI, Petitjean F (2017) Indexing and classifying gigabytes of time series under time warping. In: Proceedings of the 2017 SIAM international conference on data mining, SIAM, pp 282–290
39. Vlachos M, Hadjieleftheriou M, Gunopulos D, Keogh E (2003) Indexing multi-dimensional time-series with support for multiple distance measures. In: Proceedings of the 9th ACM SIGKDD international conference on knowledge discovery and data mining, pp 216–225
40. Vlachos M, Kollios G, Gunopulos D (2002) Discovering similar multidimensional trajectories. In: Proceedings 18th international conference on data engineering, IEEE, pp 673–684
41. Webb GI, Petitjean F (2021) Tight lower bounds for dynamic time warping. Pattern Recogn 115:107895
42. Wu R, Keogh EJ (2020) FastDTW is approximate and generally slower than the algorithm it approximates. IEEE Trans Knowl Data Eng
43. Zhang D, Zuo W, Zhang D, Zhang H, Li N (2010) Classification of pulse waveforms using edit distance with real penalty. EURASIP J Adv Signal Process 2010:1–8

Dr. Chang Wei Tan is a postdoctoral research fellow at Monash University. His research focuses on learning from time series data. He has mainly worked on various time series classification (TSC) problems, where he developed scalable algorithms that are 1000 faster than the state-of-the-arts. His recent work is on the development of Multirocket, a state-of-the-art scalable TSC algorithm that is accurate and significantly faster than the state-of-the-art method. He is currently working on regression problems and applying classification algorithms to various applications such as epilepsy detection using EEG data. Last but not least, he is also actively involved in a few supply chain forecasting projects, developing forecasting models for industry clients.

Dr. Matthieu Herrmann is a research fellow at Monash University, where he works on Time Series Classification. He obtained his Ph.D. from the University of Paris Diderot in 2016, studying programming languages and formal systems. He then joined Monash University in Melbourne, switching his focus to Time Series Classification. He now mainly works on efficient computation and parameterization for instance based classifiers, and develops the C++/Python Tempo library to make his research group research easily accessible to practitioners.

Geoffrey I. Webb is Research Director of the Monash University Data Futures Institute. He was editor in chief of Data Mining and Knowledge Discovery, from 2005 to 2014. He has been Program Committee Chair of both ACM SIGKDD and IEEE ICDM, as well as General Chair of ICDM and member of the ACM SIGKDD Executive. He is a technical advisor to machine learning as a service startup BigML Inc and to recommender systems startup FROOMLE. He developed many of the key mechanisms of support–confidence association discovery in the 1980s. His OPUS search algorithm remains the state-of-the-art in rule search. He pioneered multiple research areas as diverse as black-box user modelling, interactive data analytics and statistically sound pattern discovery. He has developed many useful machine learning algorithms that are widely deployed. His many awards include IEEE Fellow and the inaugural Eureka Prize for Excellence in Data Science (2017).