Coarse-DTW for Sparse Time Series Alignment

Marc Dupont[1,2] and Pierre-François Marteau[1]

[1] IRISA, Université de Bretagne Sud, Campus de Tohannic, Vannes, France
[2] Thales Optronique, 2 Avenue Gay Lussac, Elancourt, France

Abstract. Dynamic Time Warping (DTW) is considered as a robust measure to compare numerical time series when some *time elasticity* is required. However, speed is a known major drawback of DTW due to its quadratic complexity. Previous work has mainly considered designing speed optimization based on early-abandoning strategies applied to nearest-neighbor classification, although some of these optimizations are restricted to uni-dimensional time series. In this paper, we introduce Coarse-DTW, a reinterpretation of DTW for sparse time series, which exploits adaptive downsampling to achieve speed enhancement, even when faced with multidimensional time series. We show that Coarse-DTW achieves nontrivial speedups in nearest-neighbor classification and even admits a positive-definite kernelization suitable for SVM classification, hence offering a good tradeoff between speed and accuracy.

1 Introduction

Since the Frechet's distance [8] proposed at the dawn of the previous century, *time-elastic* matching of time series or symbolic sequences has attracted much attention of the scientific community in domains such as information indexing and retrieval, pattern analysis extraction and recognition, data mining, etc, with a very large spectrum of applications spreading in almost all the socioeconomic areas (environment, industry, health, energy, defense and so on).

Among other measures, Dynamic Time Warping (DTW) has been widely popularized during the seventies with the advent of speech recognition systems [25], [21]. However, one of the main drawbacks of such a *time-elastic* measure is its quadratic computational complexity ($O(N^2)$) which, as is, prevents processing a very large amount of lengthy temporal data. Recent research has thus mainly focused on circumventing this complexity barrier. The original approach proposed in this paper is to cope directly and explicitly with the potential sparsity of the time series during their *time-elastic* alignment.

Our main contribution is three-fold: i) an on-line downsampling algorithm whose aim is to provide a sparse representation of time series in $O(N)$; ii) Coarse-DTW, a DTW variant that copes efficiently with the sparsity of time series; iii) a kernelization of Coarse-DTW that allows for the design of accurate time-elastic SVM. Furthermore, an experimentation section is provided that highlights the tradeoff between speedup and accuracy on a large number of time series datasets (mainly unidimensional with a few multidimensional).

2 Previous work

DTW speed-up approaches mostly fall into one or several of the following categories:

1. **Reducing the search space**, when looking for the optimal alignment path. In [21] and [10] the search space is reduced by using a fixed corridor with a band (resp. parallelogram) shape displayed around the main alignment diagonal. For these corridor approaches, finding the optimal alignment path is obviously not guaranteed. Recently, [1] proposed the SparseDTW algorithm that exploits the concept of sparse alignment matrix to dynamically reduce the search space without optimality loss. SparseDTW provides thus an exact DTW computation with efficiency improvement in average.
2. **Reducing the dimensionality** of the data, along the spatial or temporal axis. Reducing the time series along the temporal axis leads to a straightforward speed-up (reducing by 2 the number of samples provides a by 4 speedup). [27] and [12] have proposed a piecewise aggregate approximation (PAA) of time series using segments of constant size. In [14] a modification of DTW, called PDTW, has been proposed to cope explicitly with PAA. Adaptive Piecewise Constant Approximation (APCA) has also been used [4] to comprees furthermore the representation of time series. Symbolic representation of time series such as SAX [20] or its variants falls also in this category. Recently, in [19] authors have shown that in the context of isolated gesture recognition sampled using depth-camera or motion capture systems, drastic down-sampling along the time axis in general enhances the accuracy.
3. **Approaching DTW by a low complexity lower bounding function** in an early abandoning strategy to reduce (drastically) the number of DTW calculations. This idea has been first proposed by [26] then progressively improved by [13], [15] that have successively introduced tighter lower bounding functions.

Some mixed approaches have been also proposed such as in ID-DTW (Iterative Deepening DTW) [6] which uses multi-resolution approximations with an early abandoning strategy or Fast-DTW [23] which also exploits multi-resolution approximations on a divide-and-conquer principle. [22] and [24] have also proposed approaches mixing APCA with a lower bounding strategy.

Speeding up DTW algorithm is thus addressed either with or without accuracy loss. However, in the context of time series similarity, this simple choice may have a non trivial answer, depending on what meaning we give to accuracy. Are we interested in computing with accuracy the exact DTW measure? Or are we trying to maximize the accuracy of some task (regression, categorization, clustering or search for nearest neighbors) according to a given ground truth? Some elastic distance variants have been proposed, such as ERP [5] or TWED [17] with some gain in classification accuracy, but with no speed-up strategy designed so far.

Attempts to improve a 1-NN DTW classification accuracy using more sophisticated approaches such as Support Vector Machine (SVM) with a direct DTW distance substituting Gaussian kernel has also been investigated, but has led to relatively poor results comparatively to 1-NN classification [9]. Observing that this kind of kernel is not definite, and as such could lead to some discrepancies when used in a kernel machine, recent works [7], [18] have proposed new kernel regularization techniques for time-elastic kernels which lead, in general, to very significant classification accuracy improvements. These approaches can even be used in conjunction with a symmetric corridor that allows for a speedup based on the reduction of the search space, at the price of a lower accuracy in general.

Obviously, optimizing the accuracy/speedup tradeoff is thus a natural concern, and within this line of research, mixing speedup strategies with a dedicated DTW variant and a *time-elastic* kernel seems quite promising. This paper specifically contributes to this focus.

3 Presentation of Coarse-DTW

3.1 Classical DTW

Let us first review the classical formulation of DTW. If d is a fixed positive integer, we define a *dense time series* of length n as a multidimensional sequence (v_i), i.e. :

$$v : \{1, \dots, n\} \to \mathbb{R}^d.$$

Let (u_i) and (v_j) be two dense time series with respective lengths n and m. A *warping path* $\gamma = (\gamma_k)$ of length p is a sequence

$$\gamma : \{1, \dots, p\} \to \{1, \dots, n\} \times \{1, \dots, m\}$$

such that $\gamma_1 = (1, 1)$, $\gamma_p = (n, m)$, and (using the notation $\gamma_k = (i_k, j_k)$), for all k in $\{1, \dots, p-1\}$, $\gamma_{k+1} = (i_{k+1}, j_{k+1}) \in \{(i_k+1, j_k), (i_k, j_k+1), (i_k+1, j_k+1)\}$.

In other words, a warping path is required to travel along both time series from their beginnings to their ends; it cannot skip a point, but it can advance one timestep on one series without advancing the other, effectively amounting to “time-warping”.

Now let δ be a distance on $\mathbb{R}^d$; let us define the *cost* of a warping path γ as the sum of distances between pairwise elements of the time series along γ, i.e.:

$$\text{cost}(\gamma) = \sum_{(i_k, j_k) \in \gamma} \delta(v_{i_k}, w_{j_k})$$

A common choice of distance on $\mathbb{R}^d$ is the one induced by the L^2 norm:

$$\delta(x, y) = \|x - y\|_2 = \sqrt{\sum_{s=1}^{d} (x_s - y_s)^2}$$

The warping path has a finite length, and there is a finite number of possible warping paths. Hence, there is at least one path whose cost is minimal. We define $\text{DTW}(v, w)$ as the minimal cost of all warping paths.

In practice, the typical way to compute DTW leverages the recursive structure of the warping path:

Algorithm 1 DTW

```
1: procedure DTW($v, w$)                    ▷ ($v$ and $w$ are 1-indexed; $A$ is 0-indexed)
2:     $A$ = new matrix $[0..n,\ 0..m]$
3:     $A[0,.] = A[.,0] = \infty$ and $A[0,0] = 0$
4:     for $i = 1$ to $n$ do
5:         for $j = 1$ to $m$ do
6:             $A[i,j] = \delta(v_i, w_j) + \min(A[i-1,j],\ A[i,j-1],\ A[i-1,j-1])$
7:     return $A[n,m]$
```

3.2 Sparse time series

In contrast to dense time series, let us define a *sparse time series* as a pair of sequences with the same length, (s_i) and (v_i):

$$\begin{aligned} s &: \{1,\ldots,n\} \to \mathbb{R}_+ \\ v &: \{1,\ldots,n\} \to \mathbb{R}^d \end{aligned} \tag{1}$$

The sequence (v_i) represents our multidimensional signal's values, like above; the novelty resides in s_i, a number describing *how long the value v_i lasts*. We call this number s_i, the *stay* of v_i. In the following, we will also denote a sparse time series as $\{(s_1, v_1), \ldots, (s_n, v_n)\}$.

For example, every dense time series (v_i) is exactly represented by the sparse time series with the same values v_i and all stays $s_i = 1$. As another example, the 2D dense time series $\{(0.5, 1.2), (0.5, 1.2), (0.3, 1.5)\}$ is equivalent to the 2D sparse time series $\{(2, (0.5, 1.2)), (1, (0.3, 1.5))\}$.

3.3 Coarse-DTW

We may now introduce the Coarse-DTW algorithm. It takes two sparse time series: (s_i, v_i) of length n, and (t_j, w_j) of length m. The main idea behind Coarse-DTW is the 'aggregation' of very similar samples in time and space. As such, it shares similar aspects with the approach developed in [2] for symbolic time series.

Algorithm 2 Coarse-DTW

```
1: procedure Coarse-DTW($(s, v), (t, w)$)
2:     $A$ = new matrix $[0..n,\ 0..m]$
3:     $A[0,.] = A[.,0] = \infty$ and $A[0,0] = 0$
4:     for $i = 1$ to $n$ do
5:         for $j = 1$ to $m$ do
6:             $A[i,j] = \min(\ s_i.\delta(v_i, w_j) + A[i-1,j],$
7:                           $t_j.\delta(v_i, w_j) + A[i,j-1],$
8:                           $\phi(s_i, t_j).\delta(v_i, w_j) + A[i-1,j-1]\ )$
9:     return $A[n,m]$
```

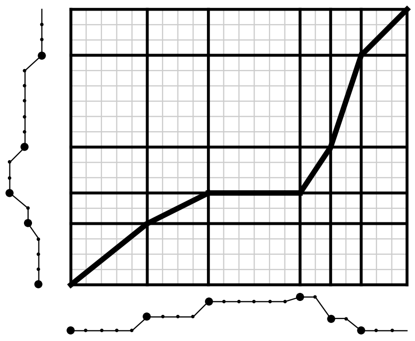


Fig. 1. A warping path in Coarse-DTW. We superimposed the sparse time series (bigger points) on top of their equivalent dense time series (smaller points). The coarse, thick grid is the Coarse-DTW matrix, whereas the underlying thin grid is the classical DTW cost matrix.

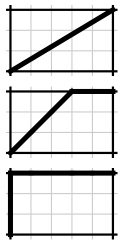
Coarse-DTW takes advantage of the sparsity in the time series to calculate costs efficiently. However, because the points last for different amount of time, we must adapt the classical DTW formulation in order to account for the stays s_i and t_j of each point into the aggregate cost calculation.

Obviously, when a point lasts for a long time, it should cost more than a point which lasts for a brief amount of time. For this reason, the pure cost $\delta(v_i, w_j)$ is multiplied by some quantity, called *weight*, linked to how long the points last, as in lines 6–8 of the algorithm. The goal of this subsection is to explain why we set those weights to s_i, t_j, and $\phi(s_i, t_j)$ respectively.

One Coarse-DTW iteration actually relates to a batch of DTW iterations in a *constant-cost sub-rectangle*. Indeed, suppose that we are operating on two pairs of subsequent points in a sparse time series, say, (s_i, v_i), (s_{i+1}, v_{i+1}) on one time series and (t_j, w_j), (t_{j+1}, w_{j+1}) on the other. If the time series were not sparse, as in classical DTW, there would be several repetitions of the first point v_i, namely s_i times, until it moves to the next value v_{i+1}. Similarly w_j would be repeated t_j times. In the DTW cost matrix, this would create an $s_i \times t_j$ sub-rectangle where all costs are identical because they match the same values; the constant cost here being $\delta(v_i, w_j)$; this is our constant-cost sub-rectangle (see Fig. 2).

The choice of weights s_i and t_j in lines 6 and 7 is motivated as follows: when we advance one time series without advancing the other, we want a lengthy point to cost more than a brief point. In the DTW constant-cost sub-rectangle, advancing the first time series is like following a horizontal subpath, whose aggregated cost would be $\delta(v_i, w_j)$ on each of its s_i cells. This sums up to $s_i.\delta(v_i, w_j)$, which is why the weight is chosen to be s_i in line 6. An analog interpretation holds for a vertical subpath of t_j cells.

For the third case (line 8), namely advancing *both* time series, choosing a weight for the cost is less obvious. The question is: which weight should we set for a path joining the top-right corner to the bottom-left one, in a constant-cost sub-rectangle? This question is captured by $\phi(s_i, t_j)$, the weight of the cost in this situation. We propose three choices for ϕ, all of which have a geometrical interpretation in the classical DTW cost matrix.



$$\phi_{\text{diag}}(s_i, t_j) = \sqrt{s_i^2 + t_j^2}$$

$$\phi_{\max}(s_i, t_j) = \max(s_i, t_j)$$

$$\phi_{\text{stairs}}(s_i, t_j) = s_i + t_j - 1$$

Fig. 2. Three choices for the function ϕ, in a constant-cost sub-rectangle of width s_i and height t_j.

Our first proposition for ϕ seeks to mimic the behavior of classical DTW. In the constant-cost sub-rectangle (of size $s_i \times t_j$), we know that a path minimizing the aggregated cost is the same as one minimizing the number of cells; precisely because all cells have the same cost. Furthermore, the minimal number of cells is exactly $\max(s_i, t_j)$; take, for example, a path going diagonal until it reaches the opposite size and then completing the remaining route on a line (see Fig. 2, middle).

Instead of choosing the weight inspired from DTW, which only approximates a diagonal, we can also choose the weight to be the true diagonal of the sub-rectangle, leading to $\phi_{\text{diag}}(s_i, t_j) = \sqrt{s_i^2 + t_j^2}$.

Finally, we will also consider the choice of $\phi_{\text{stairs}}(s_i, t_j) = s_i + t_j - 1$, amounting to a version of classical DTW where only horizontal and vertical paths would be allowed (like stairs, hence the name). Note that it should be used only in cases where sparse time series admit stays of 1 or more (this will be the case with our downsampling algorithm introduced in the next section).

4 Downsampling

In this section, we seek to transform a dense time series (u_i) into a sparse time series (s_i, v_i); the goal is to detect when series “move a lot” and “are rather static”, adjusting the number of emitted points accordingly. We propose a downsampling algorithm, called *Bubble*, which essentially collapses a block of several similar values into a single value, augmented with the duration of this block.

Bubble downsampling can be described in a simple form as follows:

Algorithm 3 Bubble Downsampling

```
procedure BUBBLE($v$, $\rho$)                                   ▷ $\rho \geq 0$
    $i_{\text{center}} = 1$                                      ▷ initialize bubble center
    $v_{\text{center}} = v_1$
    $v_{\text{mean}} = v_1$
    for $i = 2$ to $n$ do
        $\Delta v = \delta(v_i, v_{\text{center}})$              ▷ distance to center
        $\Delta i = i - i_{\text{center}}$                       ▷ find the stay
        if $\Delta v \geq \rho$ then                             ▷ does the bubble "burst"?
            yield $(\Delta i, v_{\text{mean}})$                  ▷ emit stay + point
            $i_{\text{center}} = i$                              ▷ update bubble center
            $v_{\text{center}} = v_i$
            $v_{\text{mean}} = v_i$
        else
            $v_{mean} = (\Delta i \times v_{mean} + v_i)/(\Delta i + 1)$   ▷ update mean
    $\Delta i = n - i_{\text{center}} + 1$                       ▷ force bursting last bubble
    yield $(\Delta i, v_{\text{mean}})$
```

The idea behind Bubble downsampling lies on the following approximation: consecutive values can be considered equal if they stay within a given radius ρ for the distance δ. We can picture a curve which makes bubbles along its path (see Fig. 4), hence the name. Concretely, the algorithm emits a sparse time series, where each stay is the number of consecutive points contained in a given bubble, and each value is the mean of the points in this bubble.

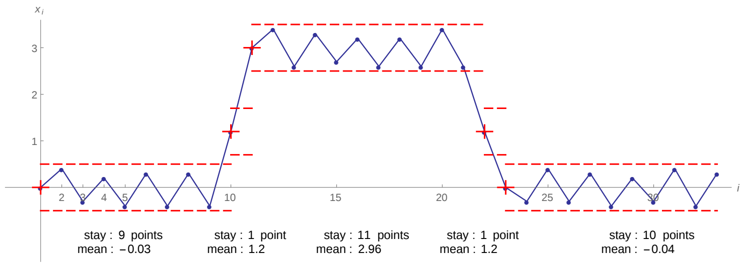


Fig. 3. Bubble downsampling applied on a 1D time series (blue, solid) with $\rho = 0.5$. The 1-bubbles are represented by their 1-centers (red crosses) and their 1-boundaries (red, dashed lines). The sparse time series emitted is $\{(9, -0.03), (1, 1.2), (11, 2.96), (1, 1.2), (10, -0.04)\}$.

The parameter ρ represents the tradeoff between information loss and density. A large ρ emits few points, thus yielding a very sparse time series, but less accurate; a smaller ρ preserves more information at the expense of a lower downsampling ratio. The degenerate case $\rho = 0$ will output a clone of the original time series with no downsampling (all stays equal to 1). Because speed is a direct consequence of sparsity in Coarse-DTW, a good middle value for ρ must be found, so that time series are as sparse as possible while retaining just the right amount of information.

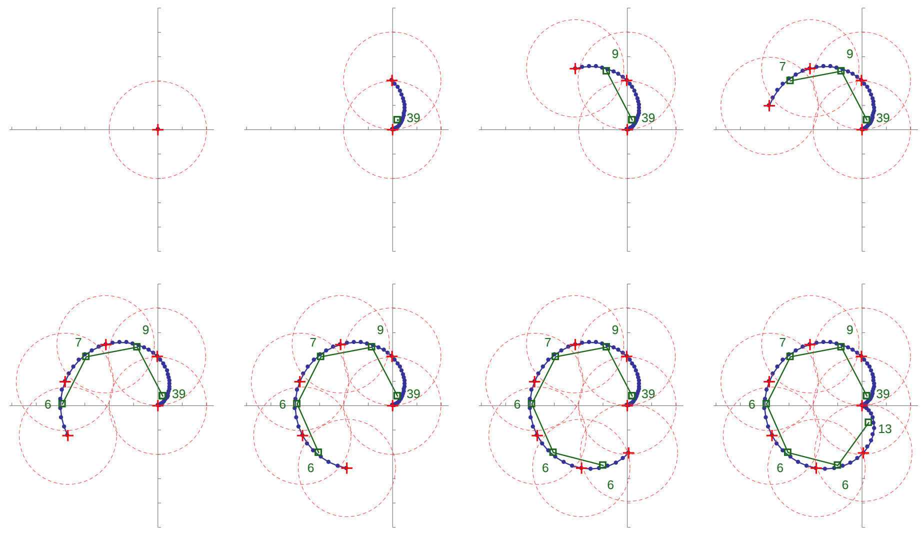


Fig. 4. Bubble downsampling progressively applied on a 2D time series (outer blue line with dots) with $\rho = 2.0$, along with the sparse time series emitted (inner green line with squares). Again, the 2-bubbles are represented by their 2-centers and their 2-boundaries (red crosses and dashed circles). Numbers indicate the stays. Notice how stays take into account the slowness at the beginning of the signal.

5 Optimizations on Coarse-DTW

DTW suffers from a slow computation time if not implemented wisely. For this reason, several optimizations have been designed [13]. The first classical optimization is to avoid a squared root computation in the distance δ. Typically, we would choose

$$\delta(x, y) = \|x - y\|_2^2 = \sum_{s=1}^{d} (x_s - y_s)^2$$

instead of its square root counterpart.

The next optimizations we considered are called *lower bounds*, designed to early-abandon computations. This kind of optimization mostly makes sense in a classification scenario along with a k-Nearest Neighbor (k-NN) classifier. In the case of 1-NN, as the classification goes on we can track the "best-so-far" distance; if a sample's lower bound exceeds the best-so-far, the computation can be stopped because it is guaranteed not to be a candidate for the nearest neighbor.

The first lower bound $\mathrm{LB_{Kim}}$ is based on the following remark: whatever the warping path found by DTW, both time series' first and last points will be matched together. More formally,

$$\begin{aligned} \mathrm{DTW}(v, w) &= \sum_{(i,j)\in\gamma} \delta(v_{i_k}, w_{j_k}) \\ &= \delta(v_1, w_1) + \cdots + \delta(v_n, w_m) \\ &\geq \delta(v_1, w_1) + \delta(v_n, w_m) \end{aligned} \tag{2}$$

where the ellipsis is a sum of positive numbers. The equation would not be satisfied if both $n \leq 1$ and $m \leq 1$, but fortunately this special case would lead to a trivial computation of DTW rendering this lower bound useless.

In the case of Coarse-DTW, the cost of matching the first points is:

$$\phi(s_1, t_1).\delta(v_1, w_1)$$

indeed, the first matching is done diagonally because $A[0{,}1] = A[1{,}0] = \infty$. Then, the cost of matching the last points is $\min(s_n, t_m, \phi(s_n, t_m))\,.\,\delta(v_n, w_m)$.

Therefore, for all pairs of time series (one of which having at least two points), the following inequality stands:

$$\text{Coarse-DTW}(v, w) \geq \phi(s_1, t_1).\delta(v_1, w_1) + \min(s_n, t_m, \phi(s_n, t_m)).\delta(v_n, w_m) \tag{3}$$

in which the right-hand side is the $\mathrm{LB_{Kim}}$ lower bound adapted to Coarse-DTW.

Another lower bound, known as $\mathrm{LB_{Keogh}}$, has enabled consequent speedup of DTW computation [13]. It is based upon the calculation of an envelope; however this calculation is not trivially transferable to the case of multidimensional time series simply by generalizing the uni-dimensional equations. Thus, we will unfortunately not consider it in our study.

However, a cheap bound can be evaluated several times as DTW progresses as follows: for any row i, the minimum of all cells $A[i,.]$ is a lower bound to the DTW result. Indeed, this result is the last cell of the last row, and the sequence mapping a row i to $\min_j A[i,j]$ is increasing, because the costs are positive. Hence, during each outer loop iteration (i.e., on index i), we can store the minimum of the current row and compare it to the best-so-far for possibly early abandoning. This can be transposed directly to Coarse-DTW without additional modifications.

6 Kernelization of CoarseDTW

Besides the fact that DTW and CoarseDTW are not metrics (they do not comply with the triangle inequality), it is furthermore not possible to directly derive a positive definite kernel from such elastic distances. Hence, their use in kernel approaches such as Support Vector Machines (SVM) is questionable and the experience shows that directly substituting DTW into a Gaussian kernel, for instance, does not lead to satisfactory results [18].

Recent works [7], [18] propose new guidelines to regularize kernels constructed from elastic measures similar to DTW. Following the line of regularization proposed in [18], an instance of positive definite kernel deriving from the CoarseDTW (K_{cdtw}) measure can be translated into the Algorithm 4, which relies on two recursive terms, namely K_{xy} and K_{xx}.

The main idea behind this line of regularization is to replace the operators min and max (which prevent the symmetrization of the kernel) by a summation operator ($\sum$). This leads to consider, not only the best possible alignment, but also all good (or nearly best) paths by summing up their overall cost. The parameter ν is used to control what we mean by a good alignment, thus penalizing more or less alignments too far from the optimal ones. This parameter can be easily optimized through a cross-validation.

The proof for the positive definiteness of K_{cdtw} is very similar to the one given in [18] for the regularized DTW kernel, except that the local kernels $e^{-\nu \cdot \xi(s(p),t(q)) \cdot \delta_E^2(v(p),w(q))}$, where $\xi(s(p), t(q))$ stands for $s(p)$, $t(q)$ or $\phi(s(p), t(q))$ should be understood as a positive definite kernel defined on the set of constant time series of varying lengths. Note that if the two time series in argument are not sparse (this is the case when no downsampling is applied), the K_{cdtw} kernel corresponds exactly to the regularized DTW kernel described in [18].

Algorithm 4 is based on the following conventions: $\forall p \geq 0$, if $p > m = |(s, v)|$ then $s(p) = s(m)$ and $v(p) = v(m)$ and similarly $\forall q \geq 0$, if $q > n = |(t, w)|$ then $t(q) = s(n)$ and $v(q) = v(n)$.

Algorithm 4 KCoarse-DTW

```
 1: procedure KCoarse-DTW($(s, v), (t, w)$)
 2:     $K_{xy}$ = new matrix $[0..n,\ 0..m]$
 3:     $K_{xx}$ = new matrix $[0..n,\ 0..m]$
 4:     $K_{xy}[0, .] = K_{xy}[., 0] = 0$ and $K_{xy}[0, 0] = 1.0$
 5:     $K_{xx}[0, .] = K_{xx}[., 0] = 0$ and $K_{xx}[0, 0] = 1.0$
 6:     for $i = 1$ to $n$ do
 7:         for $j = 1$ to $m$ do
 8:             $K_{\text{xy}}[i, j] = \frac{1}{3}(\exp(-\nu.s_i.\delta(v_i, w_j)) \cdot K_{\text{xy}}[i-1, j]+$
 9:                     $\exp(-\nu.t_i.\delta(v_i, w_j)) \cdot K_{\text{xy}}[i, j-1]+$
10:                     $\exp(-\nu.\phi(s_i, t_j).\delta(v_i, w_j)) \cdot K_{\text{xy}}[i-1, j-1])$
11:             if $i < m$ and $j < n$ then
12:                 $K_{\text{xx}}[i, j] = \frac{1}{3}(\exp(-\nu.s_i.\delta(v_i, w_i)) \cdot K_{\text{xx}}[i-1, j]+$
13:                         $\exp(-\nu.t_i.\delta(v_i, w_i)) \cdot K_{\text{xx}}[i, j-1])$
14:                 if i == j then
15:                     $K_{\text{xx}}[i, j] += \frac{1}{3}\exp(-\nu.\phi(s_i, t_i).\delta(v_i, w_i)) \cdot K_{\text{xx}}[i-1, j-1]$
16:     return $K_{\text{xy}}[n, m] + K_{\text{xx}}[n, m]$
```

6.1 Normalization of KCoarse-DTW

As KCoarse-DTW (in short K_{cdtw}) evaluates the sum on all possible alignment paths of the products of local alignment costs $e^{-\nu \cdot \xi(s(p),t(q)) \cdot \delta_E^2(v(p),w(q))} \leq 1$, its values can be very small depending on the size of the time series and the selected value for ν. Hence, K_{cdtw} values tend to 0 when ν tends towards ∞, except when the two compared time series are identical (the corresponding Gram matrix suffers from a diagonal dominance problem). As proposed in [18], a manner to avoid numerical troubles consists in using the following *normalized* kernel:

$$\tilde{K}_{\text{cdtw}}(.,.) = exp\left(\alpha \frac{\log(K_{\text{cdtw}}(.,.)) - \log(\min(K_{\text{cdtw}}))}{\log(\max(K_{\text{cdtw}})) - \log(\min(K_{\text{cdtw}}))}\right)$$

where $\max(K_{\text{cdtw}})$ and $\min(K_{\text{cdtw}})$ respectively are the max and min values taken by the kernel on the learning dataset and $\alpha > 0$ a positive constant ($\alpha = 1$ by default). If we forget the proportionality constant, this leads to take the kernel K_{cdtw} at a power $\tau = \alpha/(\log(\max(K_{\text{cdtw}})) - \log(\min(K_{\text{cdtw}})))$, which shows that the normalized kernel $\tilde{K}_{\text{cdtw}} \propto K^{\tau}_{\text{cdtw}}$ is still positive definite ([3], Proposition 2.7).

7 Results

7.1 DTW vs. Coarse-DTW in 1-NN classification

In this first setup we considered the classification accuracy and speed of various labeled time series datasets. The classifier is 1-NN and we enabled all optimizations described earlier that apply to multidimensional time series, namely: early abandoning on LB_{Kim} and early abandoning on the minima of rows. The distance chosen is the squared version, $\delta(x, y) = \|x - y\|_2^2 = \sum_{s=1}^{d}(x_s - y_s)^2$. We report only the classification time, not the learning time.

Dataset MSRAction3D [16] consists of 10 actors executing the same gestures several times, with 60 dimensions (twenty 3D joints). To classify this dataset, we cross-validated all possible combinations of 5 actors in training and 5 in test, thus totaling 252 rounds.

The dataset uWaveGestureLibrary_[XYZ] comes from the UCR time series database [11]. It can be considered as three independent uni-dimensional datasets, but we rather used it here as a single set of 3-dimensional time series. The interest is obvious: in 1-NN DTW classification, we went from individual 1D errors of respectively 27.3 %, 36.6 % and 34.2 %, down to only 2.8 % when the three time series sets are taken together.

Finally, for the sake of comparison, we also ran our tests on the other UCR time series datasets at our disposal. It should be noted that they are all unidimensional, however we exclusively considered them as multidimensional time series which *happen* to have a dimension of $d = 1$. This means in particular that some of the traditional lower bounds such as LB_{Keogh} cannot be used, only the multidimensional-enabled ones described earlier.

For each dataset, we ran the classification once with DTW to obtain a reference value both time- and accuracy-wise. Then, we ran Coarse-DTW, with several values of ρ, as follows: the dense time series are first downsampled with

Bubble into sparse time series, according to the current ρ, and then classified with Coarse-DTW. The time and error rate was measured at every run. In Fig. 5 we show the full results for a few datasets.

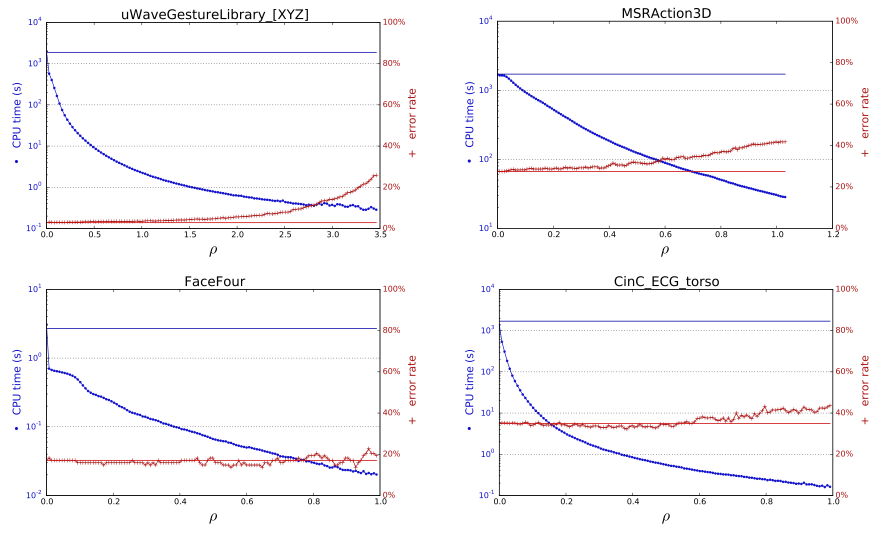


Fig. 5. 1-NN classification time and error rate of Coarse-DTW as ρ increases. For reference, DTW results are shown as horizontal bars (independent of ρ).

A general trend can be observed (Fig. 5): as ρ increases, classification time decreases. However, this comes at the expense of a higher error rate. This is expected: indeed, downsampled time series contain less information than their dense counterparts. Now, we can observe that some time series allow ρ to increase quite a bit (and therefore classification goes much faster) before the accuracy really degrades.

In order to quantify this effect, we proceed as follows. We first set a threshold on the error rate. Here, we select the threshold to be 2% (absolute error) above our reference, the DTW error rate. (For example, if the DTW error rate were 27.1%, we would set the threshold at 29.1%, which might or might not be acceptable depending on the user's constraints.) Then we find the value of

$$\rho^* = \max\{\rho \mid \forall \rho' \leq \rho, \mathrm{err}_{\rho'} \leq \mathrm{err}_{\mathrm{DTW}} + 2\%\} \quad (4)$$

which represents the last acceptable value before the error rates first goes above the threshold (the "breakout"). The CPU time associated with the run of ρ^* is likely to be below the DTW CPU time, which is why we define the speedup as their ratio:

$$\mathrm{speedup} = \frac{\text{CPU time}_{\text{ DTW}}}{\text{CPU time}_{\text{ Coarse-DTW at } \rho^*}} \quad (5)$$

dataset	d	time DTW	time Coarse-DTW	speedup	best ϕ
uWaveGestureLibrary_[XYZ]	3	1850 s	0.769 s	**2413.3x**	ϕ_{stairs}
MSRAction3D	60	1710 s	428 s	**4.0x**	$\phi_{\max}$
Adiac	1	21.0 s	13.0 s	**1.6x**	ϕ_{stairs}
Beef	1	1.35 s	0.014 s	**93.5x**	$\phi_{\max}$
CBF	1	3.18 s	0.0393 s	**80.9x**	ϕ_{diag}
ChlorineConcentration	1	73.2 s	14.0 s	**5.2x**	$\phi_{\max}$
CinC_ECG_torso	1	1690 s	0.413 s	**4100.7x**	$\phi_{\max}$
Coffee	1	0.479 s	0.139 s	**3.5x**	$\phi_{\max}$
DiatomSizeReduction	1	3.81 s	0.241 s	**15.8x**	$\phi_{\max}$
ECG200	1	0.258 s	0.012 s	**20.7x**	$\phi_{\max}$
ECGFiveDays	1	2.34 s	0.283 s	**8.2x**	$\phi_{\max}$
FaceAll	1	73.3 s	21.5 s	**3.4x**	$\phi_{\max}$
FaceFour	1	2.69 s	0.031 s	**85.8x**	ϕ_{stairs}
FacesUCR	1	42.7 s	7.46 s	**5.7x**	$\phi_{\max}$
FISH	1	47.1 s	38.0 s	**1.2x**	$\phi_{\max}$
Gun_Point	1	0.653 s	0.108 s	**6.1x**	ϕ_{diag}
Haptics	1	445 s	0.860 s	**516.7x**	$\phi_{\max}$
InlineSkate	1	1790 s	0.548 s	**3276.1x**	ϕ_{diag}
ItalyPowerDemand	1	0.236 s	0.103 s	**2.3x**	ϕ_{stairs}
Lighting2	1	17.0 s	0.440 s	**38.6x**	ϕ_{stairs}
Lighting7	1	4.66 s	5.21s	**0.9x**	$\phi_{\max}$
MALLAT	1	1460 s	6.408 s	**228.4x**	$\phi_{\max}$
MedicalImages	1	2.92 s	0.261 s	**11.2x**	$\phi_{\max}$
MoteStrain	1	1.14 s	0.0480 s	**23.7x**	$\phi_{\max}$
NonInvasiveFetalECG_Thorax1	1	9820 s	516,s	**19.0x**	$\phi_{\max}$
NonInvasiveFetalECG_Thorax2	1	9720 s	310 s	**31.3x**	$\phi_{\max}$
OliveOil	1	3.15 s	1.43 s	**2.2x**	ϕ_{diag}
OSULeaf	1	59.3 s	2.16 s	**27.4x**	ϕ_{stairs}
SonyAIBORobot_Surface	1	0.465 s	0.245 s	**1.9x**	$\phi_{\max}$
SonyAIBORobot_SurfaceII	1	0.902 s	0.150 s	**6.0x**	ϕ_{stairs}
StarLightCurves	1	44700 s	58.6 s	**763.0x**	$\phi_{\max}$
SwedishLeaf	1	19.0 s	11.5 s	**1.7x**	$\phi_{\max}$
Symbols	1	21.8 s	1.91 s	**11.4x**	$\phi_{\max}$
synthetic_control	1	1.97 s	0.624 s	**3.2x**	$\phi_{\max}$
Trace	1	2.36 s	0.00636 s	**371.3x**	$\phi_{\max}$
TwoLeadECG	1	0.827 s	0.0869 s	**9.5x**	$\phi_{\max}$
Two_Patterns	1	371 s	0.668 s	**556.4x**	$\phi_{\max}$
wafer	1	158 s	0.402 s	**392.1x**	$\phi_{\max}$
WordsSynonyms	1	87.4 s	4.61 s	**18.9x**	$\phi_{\max}$
yoga	1	581 s	4.78 s	**121.7x**	$\phi_{\max}$

Table 1. Performance of Coarse-DTW (for a threshold at +2% abs. err.) compared to DTW, in 1-NN classification (datasets from [16] and [11]).

Furthermore, we tested each of the three possibilities for ϕ. Of all three, we selected only the ρ^* value giving the best time. The values of ρ^* and the speedup are summarized in Table 1, along with the winning ϕ.

Additionally, our study aimed to search for the best ϕ function for the diagonal weight. We can conclude from Table 1 that the most satisfactory is $\phi_{\max}$, offering the best ratio accuracy/time. Actually, it appears from our experience that ϕ_{diag} was good enough accuracy-wise but was too slow due to the square root. Thus, we recommend selecting $\phi_{\max}$ by default.

7.2 SVM classification with KCoarse-DTW

We also tested the accuracy of our regularized version, KCoarse-DTW, normalized as described in 6.1, with different values of ρ to see how accuracy degrades with approximation.

With KCoarse-DTW and with very small values of ρ, we are in general able to outperform 1-NN DTW. We must add that this was not the case for all time

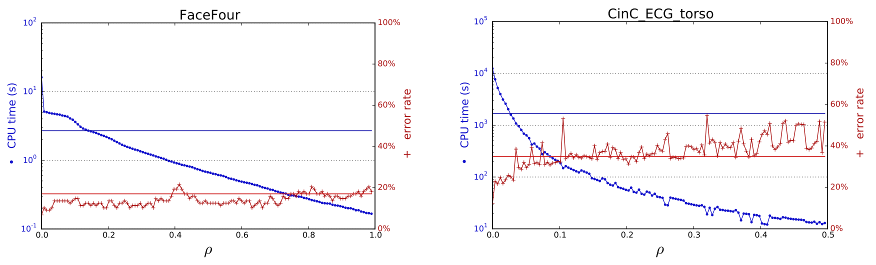


Fig. 6. SVM classification time and error rate of our regularized kernel, KCoarse-DTW. For reference, 1-NN DTW results are shown as horizontal bars (top: CPU time, bottom: error rate).

series when ρ increases, so this really depends on the nature of the time series in question.

In Fig. 6, we present two datasets where the regularized kernel performed better. (For FaceFour, the same (C, σ) was found for $\rho = 0$ and reused afterwards, whereas on CinC_ECG_Torso, there was a new grid search for each ρ.) The behavior is comparable to 1-NN classification with Coarse-DTW, except we start at $\rho = 0$ with a lower error rate. Then accuracy degrades as expected. SVM takes usually more time to run than 1-NN, but KCoarse-DTW helps by making it possible to decrease the classification time.

8 Conclusions

Not only have we transposed DTW into Coarse-DTW, a version accepting sparse time series, but we have also developed Bubble, an extremely efficient, streamable algorithm to generate such sparse time series from regular ones. By coupling those two mechanisms, we were able to discover that time series can be classified much faster in nearest-neighbor classification; the user can reach the desired tradeoff between speed and accuracy, by tuning the parameter ρ in the downsampling algorithm. Some time series are far more subject to downsampling than others, and therefore results can differ depending on which context time series originate. For example, smooth time series like gestures present a considerable ability to be downsampled, producing good results in classification speedup.

We also explored the regularization of Coarse-DTW, for use as an SVM kernel. For some time series, results were encouraging, giving classification results much better than 1-NN. As it was expected, the accuracy degraded as the downsampling radius ρ increased, giving once again the user the ability to choose a suitable tradeoff between speed and accuracy.

Coarse-DTW and Bubble have a great potential to be used in a variety of scenarios beyond offline classification; for example, they are totally suitable to reduce time series storage space, or also to reconize learnt patterns within a multidimensional stream. Finally, in an embedded context, where energy is scarce, the speedup offered by Coarse-DTW can also be interpreted as a saving in CPU cycles, which can be tremendously helpful.

9 Acknowledgements

This study was co-funded by the ANRT agency and Thales Optronique SAS, under the PhD CIFRE convention 2013/0932.

References

1. Ghazi Al-Naymat, Sanjay Chawla, and Javid Taheri. Sparsedtw: A novel approach to speed up dynamic time warping. In *Proceedings of the Eighth Australasian Data Mining Conference - Volume 101*, AusDM '09, pages 117–127, Darlinghurst, Australia, Australia, 2009. Australian Computer Society, Inc.
2. A. Apostolico, G.M. Landau, and S. Skiena. Matching for run-length encoded strings. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 348–356, Jun 1997.
3. Christian Berg, Jens Peter Reus Christensen, and Paul Ressel. *Harmonic Analysis on Semigroups: Theory of Positive Definite and Related Functions*, volume 100 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, Apr. 1984.
4. Kaushik Chakrabarti, Eamonn Keogh, Sharad Mehrotra, and Michael Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Trans. Database Syst.*, 27(2):188–228, June 2002.
5. L. Chen and R. Ng. On the marriage of lp-norm and edit distance. In *Proceedings of the 30th International Conference on Very Large Data Bases*, pages 792–801, 2004.
6. Selina Chu, Eamonn J. Keogh, David M. Hart, and Michael J. Pazzani. Iterative deepening dynamic time warping for time series. In Robert L. Grossman, Jiawei Han, Vipin Kumar, Heikki Mannila, and Rajeev Motwani, editors, *Proceedings of the Second SIAM International Conference on Data Mining, Arlington, VA, USA, April 11-13, 2002*, pages 195–212. SIAM, 2002.
7. M. Cuturi, J.-P. Vert, O. Birkenes, and T. Matsui. A kernel for time series based on global alignments. In *IEEE ICASSP 2007*, volume 2, pages II–413–II–416, April 2007.
8. Maurice Fréchet. *Sur quelques points du calcul fonctionnel.* Thèse, Faculté des sciences de Paris., 1906.
9. S. Gudmundsson, T.P. Runarsson, and S. Sigurdsson. Support vector machines and dynamic time warping for time series. In *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 2772–2776, June 2008.
10. F. Itakura. Minimum prediction residual principle applied to speech recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 23(1):67–72, Feb 1975.
11. E. J. Keogh, X. Xi, L. Wei, and C.A. Ratanamahatana. The UCR time series classification-clustering datasets, 2006. http://wwwcs.ucr.edu/ eamonn/time_series_data/.
12. Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *JOURNAL OF KNOWLEDGE AND INFORMATION SYSTEMS*, 3:263–286, 2000.
13. Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowl. Inf. Syst.*, 7(3):358–386, March 2005.

14. Eamonn J. Keogh and Michael J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 285–289, New York, NY, USA, 2000. ACM.
15. Daniel Lemire. Faster retrieval with a two-pass dynamic-time-warping lower bound. *Pattern Recognition*, 42(9):2169 – 2180, 2009.
16. W. Li, Z. Zhang, and Z. Liu. Action recognition based on a bag of 3d points. In IEEE CS Press, editor, *Proc. IEEE Int'l Workshop on CVPR for Hum. Comm. Behav. Analysis*, pages 9–14, 2010.
17. P. F. Marteau. Time warp edit distance with stiffness adjustment for time series matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(2):306–318, 2008.
18. Pierre-François Marteau and Sylvie Gibet. On Recursive Edit Distance Kernels with Application to Time Series Classification. *IEEE Trans. on Neural Networks and Learning Systems*, pages 1–14, June 2014.
19. Pierre-François Marteau, Sylvie Gibet, and Clement Reverdy. Down-Sampling coupled to Elastic Kernel Machines for Efficient Recognition of Isolated Gestures. In IAPR, editor, *ICPR 2014, International Conference on Pattern Recognition*, page pp, Stockholm, Sweden, August 2014. IEEE.
20. Pranav Patel, Eamonn Keogh, Jessica Lin, and Stefano Lonardi. Mining motifs in massive time series databases. In *In Proceedings of IEEE International Conference on Data Mining (ICDM'02*, pages 370–377, 2002.
21. H. Sakoe and S. Chiba. A dynamic programming approach to continuous speech recognition. In *Proceedings of the 7th International Congress of Acoustic*, pages 65–68, 1971.
22. Yasushi Sakurai, Masatoshi Yoshikawa, and Christos Faloutsos. Ftw: Fast similarity search under the time warping distance. In *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '05, pages 326–337, New York, NY, USA, 2005. ACM.
23. Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intell. Data Anal.*, 11(5):561–580, October 2007.
24. Yutao Shou, Nikos Mamoulis, and David W. Cheung. Fast and exact warping of time series using adaptive segmental approximations. *Mach. Learn.*, 58(2-3):231–267, February 2005.
25. V. M. Velichko and N. G. Zagoruyko. Automatic recognition of 200 words. *International Journal of Man-Machine Studies*, 2:223–234, 1970.
26. Sang wook Kim, Sanghyun Park, and Wesley W. Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In *In ICDE*, pages 607–614, 2001.
27. Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 385–394, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.