Published in Image Processing On Line on 2013–6–10.
ISSN 2105–1232 © 2013 IPOL CC–BY–NC–SA
This article is available online with supplement materials,
implementation, datasets and demonstration at
http://dx.doi.org/10.5201/ipol.2013.XXXXXXXX

DRAFT

Audio Files Realignment by Dynamic Time Warping Methods

Laurent Oudre, Florian Picard, Florian Tilquin

Abstract

Dynamic Time Warping (DTW) methods have been widely used to recognize and match modified and disturbed data, such as graphs and audio files. This work analyzes the differences between several DTW variants for audio files realignment that have been produced by researchers since 1978, date of the first reference article on DTW. By testing, comparing and understanding those algorithms, we seek to design a method for audio files realignment, which is robust to changes in rhythm, interpretation or speed.

1 Introduction

Two records of the same piece of music are always different. This might be due to changes in interpretation, rhythm, intensity or speed but also to the random noise which appear in the recording process. A natural question one could ask is : « How to be robust to thoses fluctuations ? ».

Let us illustrate this question with an example. Consider a musician working on several of his records and wanting to mix them. Let say that in the first record, the drums start at 2'30" and at 1'28" in the second record. If he wants to mix them, he needs the drums to be synchronized and therefore needs to realign the records. The Dynamic Time Warping (DTW) algorithm might help this musician, because it can find this corresponding time automatically and with a very good precision [4]. In this case, we want to discover how the "time flow" of a first time serie is transformed to get the one of a second time serie.

A related problem is to evaluate how two time series are close to one another, independantly of time fluctuations. This is especially useful for word recognition [10] where the speed and pronunciation might differ between speakers. DTW algorithms are also useful and widely used in this context, as they output a time-normalized distance which is independant of time fluctuations.

We will present in this article an off-line method for realigning dynamic time series (in which both series are completely known, in opposition to the on-line algorithm in which they are given sample by sample to the algorithm, as in a live recording). This algorithm relies on dynamic programming methods, as developped in [2]. Section 2 presents a warping function which transforms the time of the first time serie in the time of the second one, and then suggests a simple algorithm to solve the problem. Section 3 displays an overview of several variants for speed and performances improvement. Section 4 describes the results obtained by the algorithm on several basic toy examples. Finally, in Section 5, the algorithm and its variants are tested on several music extracts in order to study in depth the influences of the method parameters.

2 Dynamic Time Warping

This section describes the mathematical context and the details of the DTW algorithm.

2.1 Context

We consider two time series, $U \in \Omega^M$ and $V \in \Omega^N$ (where Ω is the space we work on, typically $\mathbb{R}^k$). M and N are the sizes of the two time series we are considering.

$$U = U_1, \cdots, U_M \tag{1}$$
$$V = V_1, \cdots, V_N \tag{2}$$

Given those two time series, the aim of the algorithm is to match them together, i.e. to build a correspondance between the elements of U and those of V. Its output is a sequence of points, of a certain length K, which represents the correspondance of the time line in the sequence U with the one of sequence V. This path P is defined as:

$$P = ((i(1), j(1)), \cdots, (i(K), j(K))) \in (\mathbb{N} \times \mathbb{N})^K, K \in \mathbb{N} \tag{3}$$

Such an object is called "path of length K", and if $(i(k), j(k)) \in P$ it means that $V_{j(k)}$ has been matched with $U_{i(k)}$. To illustrate this, we can represent the time series in a plane, with V on the horizontal axis, U on the vertical axis, and the path P as a sequence of points in this two-dimension space (see Figure 1).

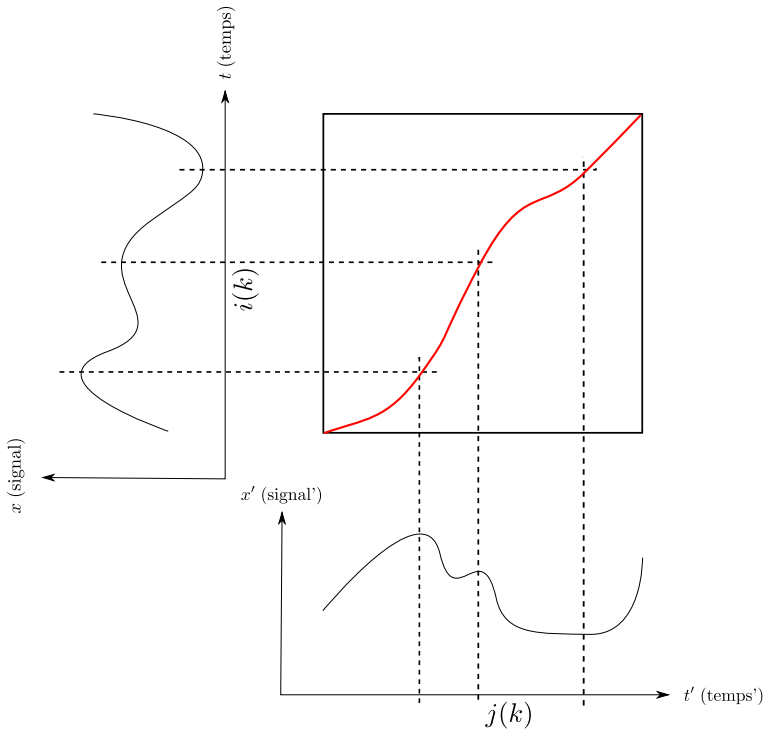


Figure 1: Two time series alignment: on the left the original signal U, at the bottom the signal V to be rematched, in the middle the path P of realignement between the two signals.

Given a path P of length K, we introduce the following cost function $W(P)$, which evaluates the accuracy of the correspondances induced by P.

$$W(P) = \sum_{k=1}^{K} d(U_{i(k)}, V_{j(k)}) \tag{4}$$

where d is a distance over Ω. Note that $W(P)$ is non-negative, and is decreasing with the intuitive closeness of the time series. In particular, if U is equal to V then, we have $N = M$ and the optimal path is $(1,1),\ldots,(M,M)$, and then $W(P)$ is equal to zero.

The best alignment between two time series corresponds to the optimal path P^* that minimizes (4) over a finite set of paths $\mathscr{P}$.

$$P^* = \arg\min_{P \in \mathscr{P}} W(P) \tag{5}$$

2.2 Application to sound signals

Since our task is audio files realignment, it induces some restrictions on the set of possible paths $\mathscr{P}$, because the path represents time fluctuation between the time series U and V. Since time variations should be *continuous*, *monotonic* (time can't go backward) and *bounded*, the path should respect the same restrictions. We will say that a path P is:

Continuous if $i(k) - i(k-1) \leq 1$ and $j(k) - j(k-1) \leq 1$ (Imagining the plane as a chessboard, we can only go to the positions a king could);

Monotonic if $i(k-1) \leq i(k)$ and $j(k-1) \leq j(k)$ (We can only go up and right, and diagonally up and right);

Bounded $(i(K), j(K)) = (M, N)$ (We impose that the path ends by matching together the last element of both signals).

In the following, we limit $\mathscr{P}$ to the set of paths respecting those three restrictions.

For $P = ((i(1), j(1)), \ldots, (M, N)) \in \mathscr{P}$, we can easily see that there is only a limited set of indexes to get to $(i(k), j(k))$: see Figure 2 and Equation (6).

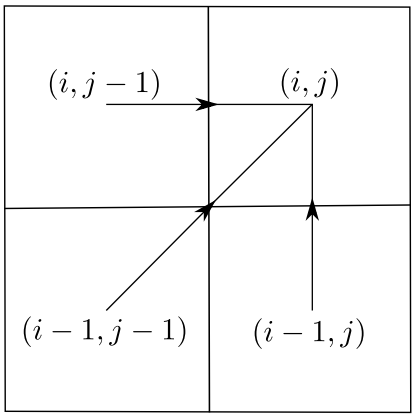


Figure 2: Continuity and monotonicity conditions.

$$(i(k-1), j(k-1)) = \begin{cases} & (i(k)-1, j(k)) \\ \text{or} & (i(k), j(k)-1) \\ \text{or} & (i(k)-1, j(k)-1) \end{cases} \tag{6}$$

2.3 DTW algorithm and implementation

Let us define the matrix of distances $D = (d_{(i,j)})$, where $d_{(i,j)} = d(U_i, V_j)$: it contains all distances between all elements of U and V.

In order to efficiently solve (5), we use a dynamic programming approach and build a "cost matrix" which will be used to trace back the path from the index (M, N). The cost matrix is build recursively: the minimum cost to pay to go to the index (i, j) is the sum of the minimum cost to reach the indexes $(i-1, j)$, $(i-1, j-1)$, $(i, j-1)$ (the indexes where you can come from - see (6)) added with the cost of the move, i.e. $d_{(i,j)}$. The initial conditions are set on the bottom and left side of the matrix, and initialized with the values of the distance matrix.

The cost matrix C is defined as follow:

$$C(i,1) = d_{(i,1)} \tag{7}$$

$$C(1,j) = d_{(1,j)} \tag{8}$$

$$C(i,j) = d_{(i,j)} + \min \begin{cases} C(i,j-1) \\ C(i-1,j-1) \\ C(i-1,j) \end{cases} \tag{9}$$

Hence, the basic DTW algorithm is define as described in Algorithm 1.

Algorithm 1 Construction of the cost matrix

Require: D
  $C = \text{matrix}(M, N)$
  $C(1 : end, 1) = D(1 : end, 1)$
  $C(1, 1 : end) = D(1, 1 : end)$
  for $i = 2$ **to** M **do**
    for $j = 2$ **to** N **do**
      $C(i,j) = D(i,j) + \min\left(C(i-1,j-1), C(i,j-1), C(i-1,j)\right)$
    end for
  end for
  return C

After constructing the cost matrix, the optimal path P^* that solves the problem is build recursively: we start from (M, N) (because the two signals match here, by assumption), we look at the minimum of the three cost values where we can move from $((M-1, N), (M-1, N-1), (M, N-1))$ and add the corresponding indexes to the path, and do so until we hit the bottom or left side. The algorithm is summarized in Algorithm 2.

3 Improvements and variants

This section describes several variants which allows to tune the algorithm, to increase its speed or to loose some of the constraints put on the path.

3.1 Troncature method

Since the signals are supposed to represent the same data with time fluctuations, our path is likely to stay near the diagonal, i.e. elements of V should be realigned with elements of U which are close in time. It is therefore possible to reduce drastically the computation time, by only computing coefficients of the matrix C which are near the diagonal (see Figure 3).

Algorithm 2 Construction of the optimal path

Require: C
  $P^* = [M; N]$
  $i = P^*(1)$
  $j = P^*(2)$
  while $i > 1$ **and** $j > 1$ **do**
    $b = \text{argmin}(C(i-1, j-1), C(i, j-1), C(i-1, j))$
    if $b = 1$ **then**
      $P^* = [[i-1; j-1], P^*]$
      $i = i - 1$
      $j = j - 1$
    else if $b = 2$ **then**
      $P^* = [[i; j-1], P^*]$
      $j = j - 1$
    else
      $P^* = [[i-1; j], P^*]$
      $i = i - 1$
    end if
  end while
  return P^*

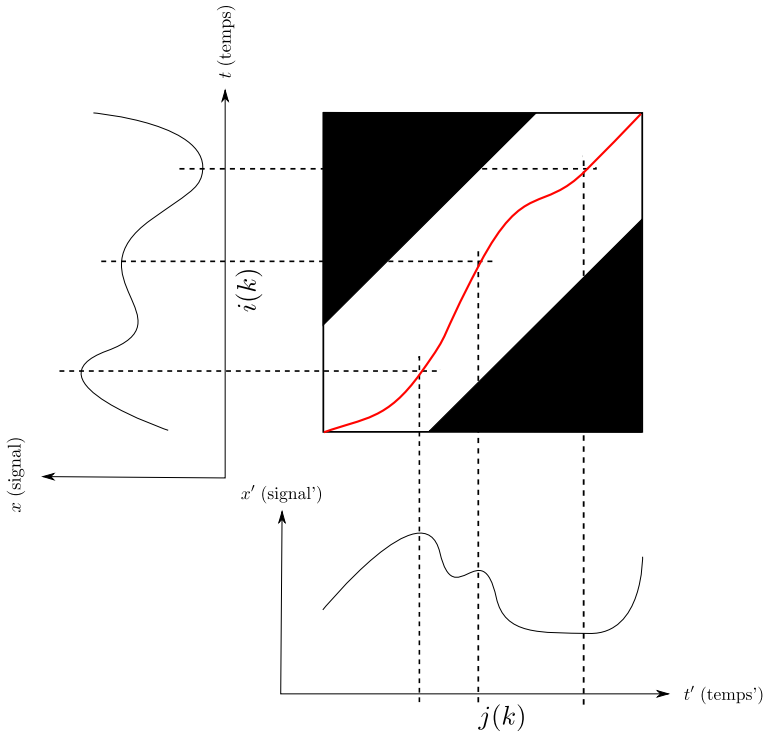


Figure 3: Illustration of the troncature method.

The upper and lower borders of the troncature are called respectively f_1 and f_2. Those two functions depend on a parameter $p \in [0, 1]$, which is the percentage of the coefficients of the total matrix we actually wish to compute (see 5.4 for discussion on the value of p). The boundary functions

f_1 and f_2 ($[\![1, M]\!]$ with values in $[\![1, N]\!]$) are defined as:

$$\begin{aligned} w &= 1 - \sqrt{1-p} \\ m &= \lfloor w \times M \rfloor \\ n &= \lfloor w \times N \rfloor \\ f_1 : k &\mapsto \mathbb{1}_{m \geq k} + \mathbb{1}_{k > m} \times (1 + \lfloor M/N \times (k-m) \rfloor) \\ f_2 : k &\mapsto \mathbb{1}_{M-m > k} \times \lfloor M/N \times k + m \rfloor + M \times \mathbb{1}_{k \geq M-m} \end{aligned}$$

In the truncated method, the coefficients of C in Algorithm 1 will only be computed for $i = 1, \cdots, M$ and $j = f_1(i), \cdots, f_2(i)$.

Note that the value of p should be chosen carrefully: if we reduce too drastically the acceptable area for the path, the path may "hit" the border of the zone, and it will not be possible to find the proper path anymore.

3.2 No end constraint

The boundary constraint imposing that the path should end at (M, N) is actually not compulsary for our algorithm to perform acurately. If we acknowledge the fact that there may be an offset at the beginning of our two signals (i.e. that one signal starts a while after the other one), it is in fact very likely that there might also be a shift at the end of the signals.

We modified the path algorithm for it to stop at the best possible indexes. However, in order to prevent abberent results, we introduced a new parameter $\delta \in [0, 1]$, which controls the acceptable ending zone. For example, if we set $\delta = \frac{1}{5}$, we constraint the end of the path to lie in the last fifth of the signals. If U finishes first ($i(K) = M$), we will contraint $j(K)$ to stay in the interval $[(1-\delta)N : N]$. Setting $\delta = 0$ is equivalent to the original $(i(K), j(K)) = (M, N)$ constraint.

In this variant, the last element of the path corresponds to the smallest cost value on either the last row or column, constrained by parameter δ:

if $\min(C((1-\delta)M, N), \ldots, C(M, N)) < \min(C(M, (1-\delta)N), \ldots, C(M, N))$ **then**
  $j(K) = N$
  $i(K) = \arg\min(C((1-\delta)M, N), \cdots, C(M, N))$
else
  $i(K) = M$
  $j(K) = \arg\min(C(M, (1-\delta)N), \cdots, C(M, N))$
end if

3.3 Weigthed variants

In order to tune the algorithm to specific configurations, we can modify our definition of C in (9) by introducing some weighting coefficients, α, β, γ, which are the respectives costs to pay for a horizontal, diagonal or vertical move: the Figure 4 shows an illustration of this principle, introduced in [10]. They can be used in order to induce difficulties for the path to go in one direction or another. With these parameters, C is now defined as:

$$C(i, j) = \min \begin{cases} C(i, j-1) + \alpha d_{(i,j)} \\ C(i-1, j-1) + \beta d_{(i,j)} \\ C(i-1, j) + \gamma d_{(i,j)} \end{cases} \tag{10}$$

Our aim is now to minimize the new cost function $\tilde{W}$ defined by (11): we sum distances along the path, impacted with some coefficient $w(k)$ equal to α, β, or γ respectively if the path has made

a horizontal, diagonal, or vertical move transition to position $(i(k), j(k))$.

$$\tilde{W}(P) = \sum_{k=1}^{K} d(U_{i(k)}, V_{j(k)}) w(k) \tag{11}$$

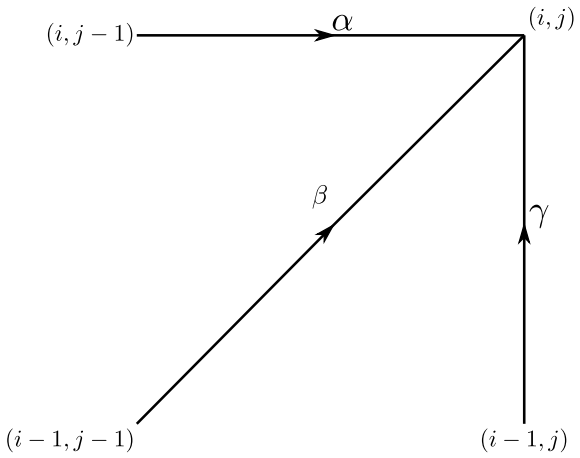


Figure 4: $\alpha\beta\gamma$ coefficients

If for instance we change the value of β while keeping $\alpha = \gamma = 1$: the greater β will be, the more disavantaged the path will be if he makes a move in the diagonal direction, because the cost of the transition will be affected by the value of β, and thus it will be incited to follow horizontal and vertical directions. On the contrary, if we reduce the value of β (or increase the one of α and γ), the path will be incited to follow the diagonal. By playing with those coefficients we can manage to tune the basic DTW algorithm. We can also introduce some asymetry by setting different values for α and γ. The DTW algorithm is the Algorithm 1 impacted with equation (10).

3.4 Final algorithm

The final algorithm relies on several parameters (see Algorithm 3):

- α, β, γ are the weight parameters for respectively a vertical, diagonal or horizontal move
- p is the truncature method parameter which controls the percentage of coefficients that are computed
- δ is the end constraint parameter which controls the area where the path can end

In the following, we shall refer to the configuration $\alpha = \beta = \gamma = p = 1 - \delta = 1$ as the *basic* algorithm.

4 Toy examples

The aim of this section is to understand how the basic algorithm ($\alpha = \beta = \gamma = p = 1 - \delta = 1$) reacts to several classical signal modifications. Let us study a basic example: our aim is to realign two sine curves in the time domain ($\Omega = \mathbb{R}$).

The reference signal S_1 (12) is a 512-sample sinusoidal signal sampled at $F_s = 44100$ Hz with amplitude equal to 1 and fundamental frequency $f_0 = 100$ Hz. The test signal S_2 (13) is also a 512-sample sinusoidal signal sampled at $F_s = 44100$ with initial time shift τ, amplitude A, fundamental frequency $f_0 + \Delta f$ and corrupted by additive Gaussian noise $e(n) \sim \mathcal{N}(0, \sigma^2)$.

$$S_1(n) = \sin(2\pi f_0 \frac{n}{F_s}) \tag{12}$$

Algorithm 3 Final complete algorithm

{Construction of the cost matrix}
Require: $D, \alpha, \beta, \gamma, p, f_1, f_2$
$C = \text{infinity}(M, N)$
$C(1, 1:m) = D(1, 1:m)$
$C(1:n, 1) = D(1:n, 1)$
for $i = 2$ **to** M **do**
  for $j = f_1(i)$ **to** $f_2(i)$ **do**
    $C(i,j) = \min(C(i,j-1) + \alpha D(i,j), C(i-1,j-1) + \beta D(i,j), C(i-1,j) + \gamma D(i,j))$
  end for
end for

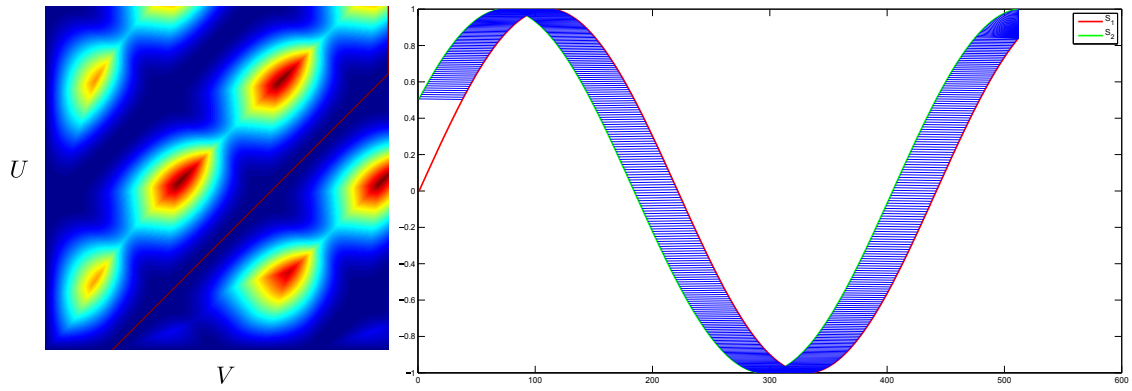
{Contruction of the optimal path}
Require: δ
$A = [\min\{C(i,N), i \in [\![\lfloor (1-\delta)M \rfloor, M]\!]\}, \min\{C(M,j), j \in [\![\lfloor (1-\delta)N \rfloor, N]\!]\}]$
if $\text{argmin}(A) = 1$ **then**
  $P^* = [A(1); N]$
else
  $P^* = [M; A(2)]$
end if
while $i > 1$ **and** $j > 1$ **do**
  $b = \text{argmin}([C(i,j-1) + \alpha D(i,j), C(i-1,j-1) + \beta D(i,j), C(i-1,j) + \gamma D(i,j)])$
  if $b = 1$ **then**
    $P^* = [[i; j-1], P^*]$
    $j = j - 1$
  else if $b = 2$ **then**
    $P^* = [[i-1; j-1], P^*]$
    $i = i - 1$
    $j = j - 1$
  else
    $P^* = [[i-1; j], P^*]$
    $i = i - 1$
  end if
end while

$$S_2(n) = A \sin(2\pi(f_0 + \Delta f)\frac{n - \tau}{F_s}) + e(t) \tag{13}$$

By changing the values of τ, A, Δf and σ^2, we will be able to study different typical signal modifications.

4.1 Influence of time shift

In this first experiment, we tried to realign the reference signal with the same sine curve, shifted by $\tau = 100$ samples. The results are presented on Figure 5: the path (in red), follows previsibly a line parallel to the diagonal, with an offset at the beginning. This is expected: the algorithm decays one of the two series to match the two together perfectly.

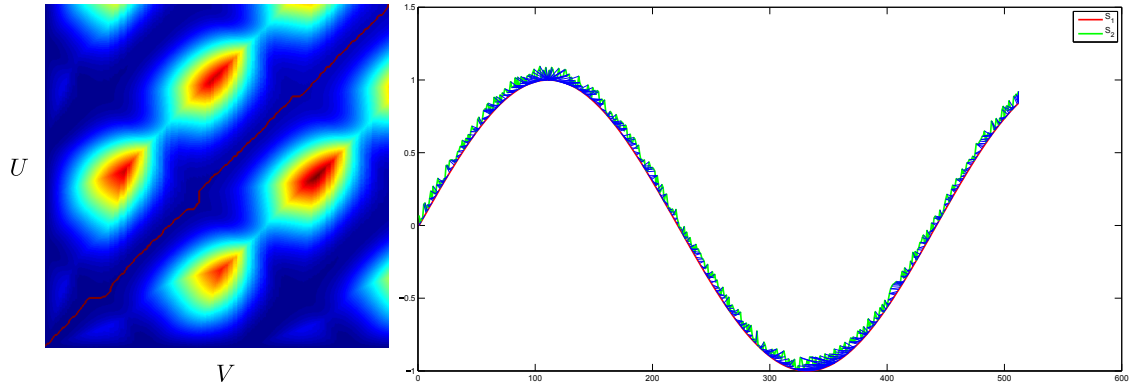


(a) Cost matrix with path (b) Matched signals

Figure 5: Realignment of two sine curves: influence of time shift ($\tau = 100$; $\sigma^2 = 0$; $A = 1$; $\Delta f = 0$)

4.2 Influence of noise

In this second experiment, we tried to realign the reference signal with the same sine curve corrupted by Gaussian noise with zero mean and variance σ^2=0.1. The results are displayed in Figure 6: the algorithm correctly matched the signals together. The path is close to the diagonal which means that the algorithm recognized that the two signals are "close".

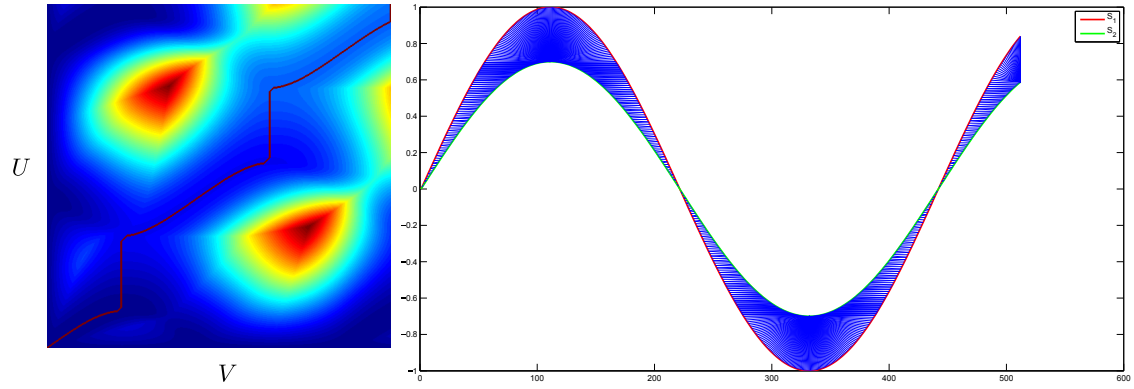


(a) Cost matrix with path (b) Matched signals

Figure 6: Realignment of two sine curves: influence of noise ($\tau = 0$; $\sigma^2 = 0.1$; $A = 1$; $\Delta f = 0$)

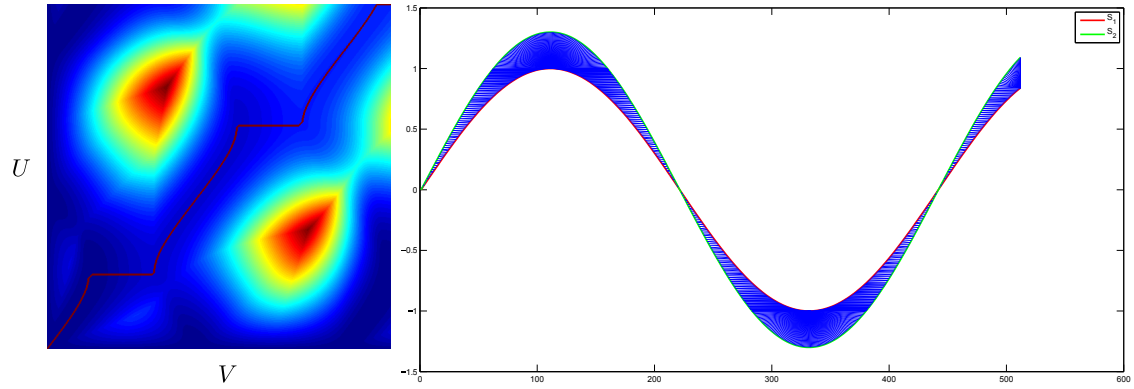
4.3 Influence of amplitude

In this third experiment, we tried to realign the reference signal with the same sine curve but with amplitude A. In order to test the robustness of the algorithms, several values for A are tested (0.7, 1.3 and 2.0). The results are respectively displayed on Figure 7, 8 and 9. We see that for $A = 0.7$ and $A = 1.3$, the path is correctly determined. For greater amplitude values, the path is completly wrong: this is simply due to the fact that if the two amplitudes are too different, since the signals we use are periodic (sinusoidal), there might exist a position further in the signal that better minimizes the distance to the coefficients far to the nods of the curves than the simple correspondances the correct path does. And by increasing the ratio between amplitudes, we increase the possibility that this happens.



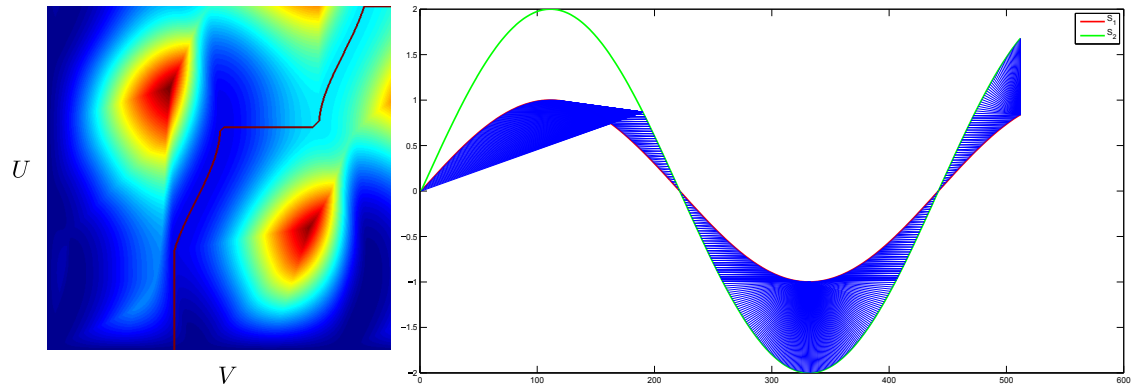
(a) Cost matrix with path (b) Matched signals

Figure 7: Realignment of two sine curves: influence of amplitude ($\tau = 0$; $\sigma^2 = 0$; $A = 0.7$; $\Delta f = 0$)



(a) Cost matrix with path (b) Matched signals

Figure 8: Realignment of two sine curves: influence of amplitude ($\tau = 0$; $\sigma^2 = 0$; $A = 1.3$; $\Delta f = 0$)



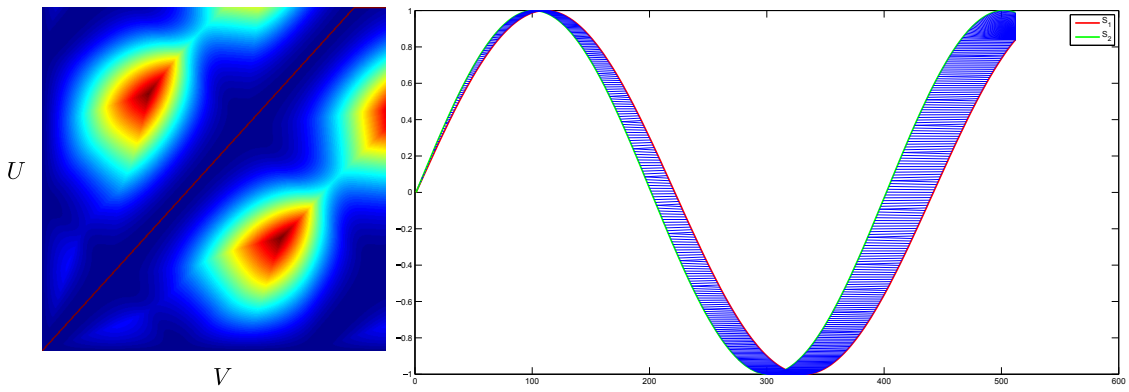
(a) Cost matrix with path (b) Matched signals

Figure 9: Realignment of two sine curves: influence of amplitude ($\tau = 0$; $\sigma^2 = 0$; $A = 2$; $\Delta f = 0$)

4.4 Influence of fundamental frequency

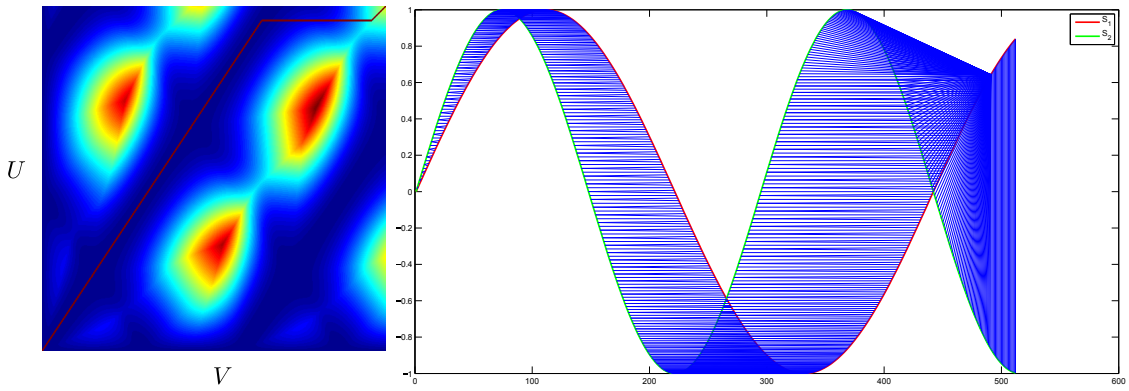
In this last experiment, we tried to realign the reference signal with the same sine curve but with a different fundamental frequency (Δf is the frequency shift). We have tested several values for Δf (10, 50 and 100) and the results are respectively displayed on Figures 10, 11 and **??**. The results are quite similar to those with the variations of amplitude: passed a certain frequency shift value, the slower signal has too many less periods over the sample we took than the reference one. Thus,

it matches its beginning with one of the next of the other signal, creating a decay between them in the rematch. For similar frequencies, the signals match more easily: the path establishes a direct correspondance of time between the signals. The paths are noteworthy, because they follow a straight line of a certain slope, which is exactly the accelaration factor of which frequency has been multiplied by.



(a) Cost matrix with path (b) Matched signals

Figure 10: Realignment of two sine curves: influence of fundamental frequency ($\tau = 0$; $\sigma^2 = 0$; $A = 1$; $\Delta f = 10$)



(a) Cost matrix with path (b) Matched signals

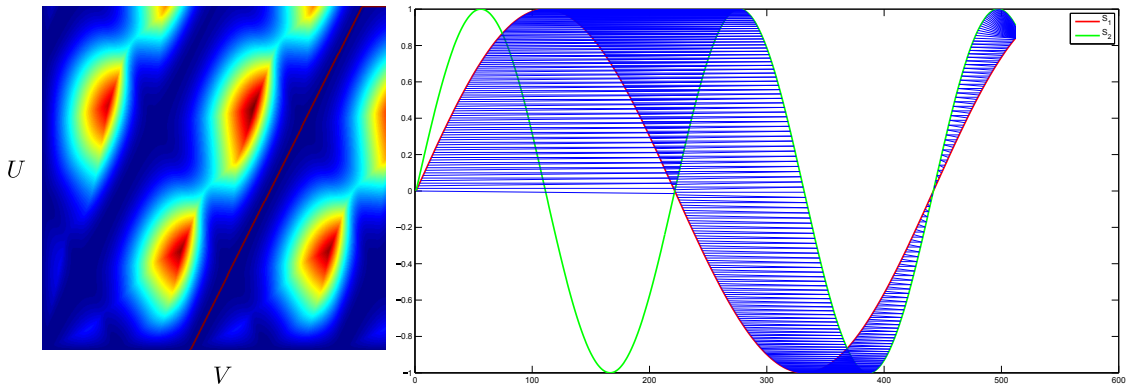
Figure 11: Realignment of two sine curves: influence of fundamental frequency ($\tau = 0$; $\sigma^2 = 0$; $A = 1$; $\Delta f = 50$)

5 Results

This section provides a qualitative and quantitative analysis of the results provided by the algorithm on several music sounds, as well as a study on the different variants.

5.1 Data

We recorded several vinyles of different pieces of music and spoken text (*Le Concerto de la Mer*, Jean-Claude Borely; *Symphony No 40 in G minor,K.550 -1. Molto allegro*, W.A.Mozart; *Le Petit Prince*, Saint-Exupéry) with a vinyle player, and applied some modifications while the vinyle was playing. We used signals of approximately 30 seconds, with a sampling frequency of 44.1 khz. Tables 1, 2, 3 sum up our modifications.



(a) Cost matrix with path (b) Matched signals

Figure 12: Realignment of two sine curves: influence of fundamental frequency ($\tau = 0$; $\sigma^2 = 0$; $A = 1$; $\Delta f = 100$)

Track	Modifications
1	Our reference
2	Acceleration at 10", deceleration at 20"
3	A jump in the record
4	With another vinyle disc of the same music
5	Played faster
6	Played louder
7	Played backward
8	Everything (except playing backward)

Table 1: Modifications on *Le concerto de la mer*

Track	Modifications
1	Our reference
2	Deceleration at 15"
3	Acceleration at 15"
4	Played faster
5 & 6	With another vinyle disc of the same music

Table 2: Modifications on *K.550 -1.*

Track	Modifications
1	Our reference
2 & 3	With another vinyle disc of the same music
4	Speed variations

Table 3: Modifications on *Le petit prince*

5.2 Signal representation

Considering audio signals sampled at 44.1 kHz, it is impossible to use the wave forms as inputs because the number of samples would be prohibitive. Also, we want our algorithm to be robust to fluctuations so we need to represent the signals in the best possible way. A classical and compact representation of audio signals is the Short Time Fourier Transform (STFT), which consists of a discrete Fourier transform calculated over overlapping frames (see Figure 13 for illustration). In our case, we chose a Hann window of 1024 points and an overlap of 50%, where the Hann window function is defined by equation (14) ($W = 1024$ here).

$$\begin{array}{rccc} h & : & [\![0, W-1]\!] & \longrightarrow & \mathbb{R} \\ & & n & \longmapsto & 0,5.(1-\frac{cos(2\pi n)}{W-1}) \end{array} \tag{14}$$

The dimension of each STFT vector is 513 (the spectrum has Hermitian symmetry since signals are real) and hence $\Omega = \mathbb{R}^{513}$. The time series U and V are the modulus of the STFT of our audio signals. As we see in section 4.3, problems might arise from amplitudes of signals. Indeed, in order to be robust to variations of amplitudes, we will take as distance the normalized euclidian distance, defined by:

$$d(U,V) = \left\| \frac{U}{\|U\|_1} - \frac{V}{\|V\|_1} \right\|_2 \tag{15}$$

$$= \sqrt{\sum_{i=1}^{513} \left(\frac{u_i}{\sum_{k=1}^{513} \mid u_k \mid} - \frac{v_i}{\sum_{k=1}^{513} \mid v_k \mid} \right)^2} \tag{16}$$

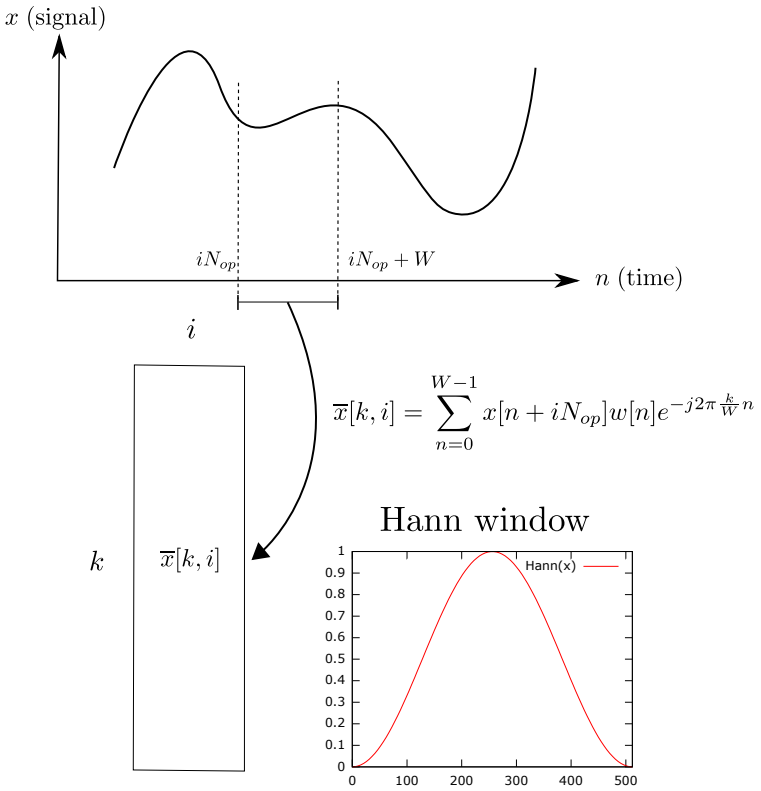


Figure 13: STFT principle, and Hann window drawing

5.3 Analyze of paths

In this section, we seek to exhibit several *typical* paths corresponding to different types of signal modifications such as changes in speed, jumps or offsets. The algorithm will be tested in its simplest variant ($\alpha = \beta = \gamma = 1 - \delta = p = 1$). The optimal path P^* is therefore the minimum of

$$W(P) = \sum_{k=1}^{K} \left\| \frac{U}{\|U\|_1} - \frac{V}{\|V\|_1} \right\|_2 \tag{17}$$

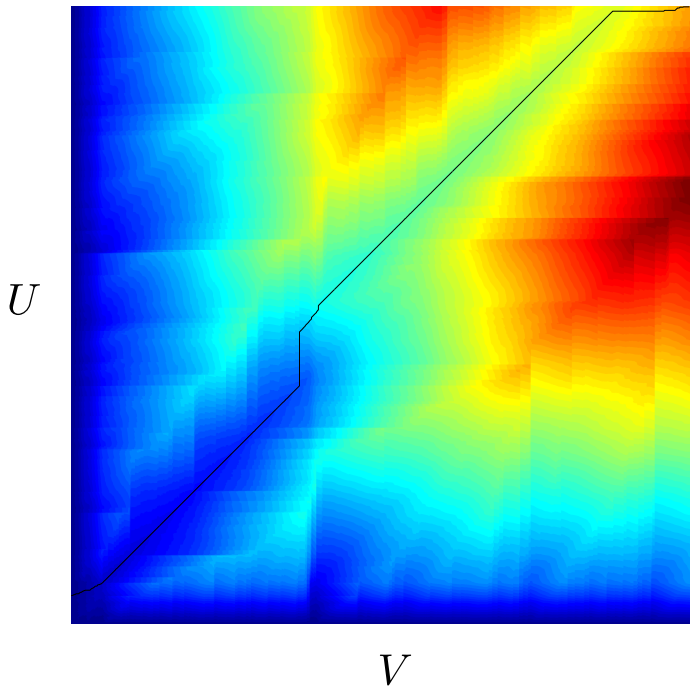


Figure 14: U = *Concerto de la mer 1*, V = *Concerto de la mer 3* ; $\alpha = \beta = \gamma = p = 1 - \delta = 1$

Jumps Figure 14 presents the results obtained by realigning *Concerto de la mer 1* and *Concerto de la mer 3*. We can see an important jump in the middle of the picture. If we think in terms of time fluctuation, it can easily be explained: while the time in the serie U do not pass, a few time in the second serie V pass, and then they flow together again. This shows us the presence of a jump in the record of the second track, which correspond to the time serie V.

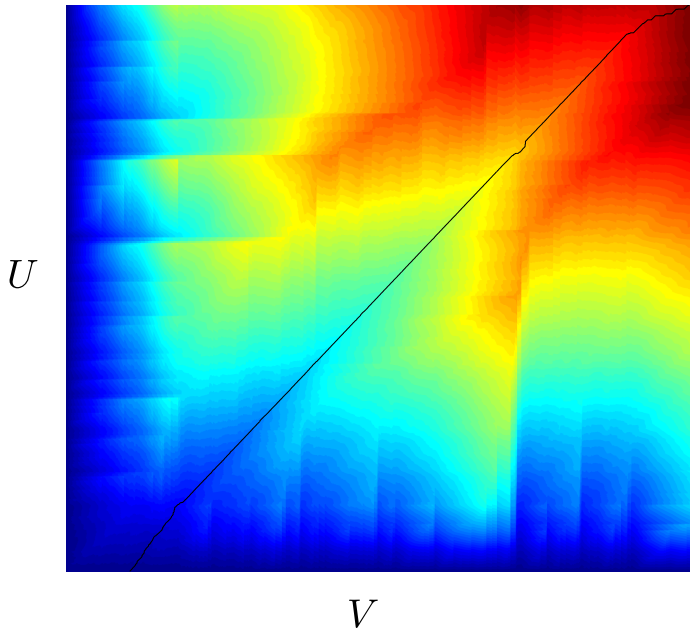


Figure 15: U = *K550-1 1*, V = *K550-1 4* ; $\alpha = \beta = \gamma = p = 1 - \delta = 1$

Change in speed Figure 15 gives us another kind of typical path (*K550-1 1* and *K550-1 4*): we can see that one signal starts later than the other one, and also that it stops earlier. The path being nearly a straight line, crossing the diagonal, we can deduce that this signal was played faster all along, with an offset at the beginning, and that the acceleration it was submit to was constant.

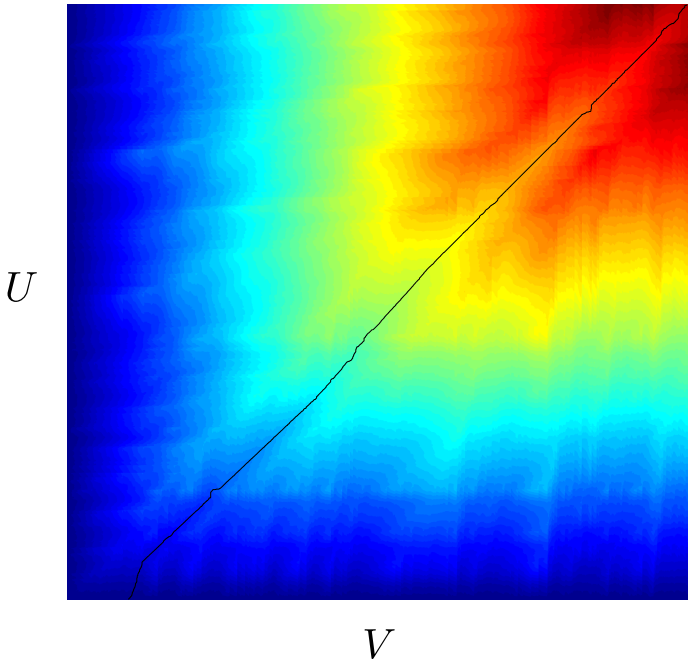


Figure 16: U = *Le petit prince 1*, V = *Le petit prince 4* ; $\alpha = \beta = \gamma = p = 1 - \delta = 1$

Multiple changes in speed Figure 16 presents us a very particular path (*Le petit prince 1* and *Le petit prince 4*): we immediatly see an offset between the two signals. But this time, the signal starting first finishes first. Thus, if ever it is accelerated, it can't be all along. What we can also see is that the path is divided in two parts, forming two segments of straight line with different gradients, both gradients being different from one (the straight lines cross the diagonal). Thereby, we can deduce that there were two different changes of rythm in the second signal, and judging by the gradients, that one is an acceleration, (the first), and the other one a deceleration nearly compensating the first acceleration.

Summary These experiments allow us to exhibit and interpret some typical paths:

- When a path follows the diagonal or a parallel to the diagonal, there is a direct correspondance between time in both series. In cases where the two signals are not quite different, the path is close to the diagonal;
- When a path follow a straigth line with some slope, it means that one of the signals has been accelerated (or decelerated) with respect to the other, with a factor equal to that slope;
- When a path has an horizontal or vertical part of great length, it means that during a certain amount of time in one of the two signals, the time was stopped and thus a lot of frames from one signal are matched with only one from the other signal. On a musical point of view it means that either there has been a great slowing in one of the two signals, or a great fastening (or even a jump) in the other one.

5.4 Influence of the troncature

We now propose to investigate the influence of the troncature parameter p on the algorithm performances. Ideally, if parameter p is correctly chosen, the performances of the algorithm should remain unchanged but the computation time should significantly decrease. We tested the influence of the troncature by doing a DTW between the reference record and each of the records of the same piece of music. We have run the algorithm with $p = 0.5$ and $p = 0.2$ while $\alpha = \beta = \gamma = 1 - \delta = 1$, and Table

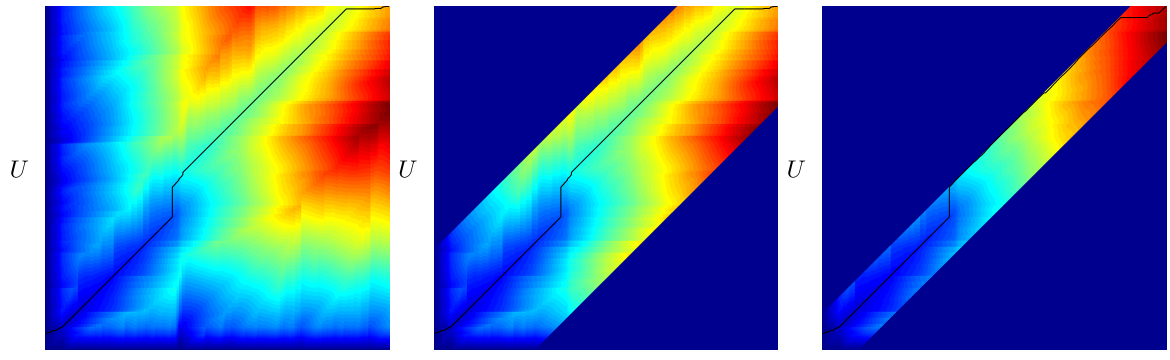
4 shows our results, which is the percentage of paths that have been changed by the troncature, for each set of records.

We verified that in each DTW where the path did not hit the troncature border, the path was identical to the one made with a classic DTW. This validates the concept of the troncature method for it gives (under this constraint) the same results that we owed to find.

- $p = 0.5$ (Figure 17(b)). The path rarely hits the border of the troncature and is hence exactly the same as in a basic DTW. Thus we understand that we are allowed to make every further DTW test with the troncature method at least with only half of the coefficients computed.
- $p = 0.2$ (Figure 17(c)). Some of the paths hit the border of the troncature, giving back an incorrect path. The unchanged paths are the ones of the signals very close to each other. Therefore, we can only get to such values of the coefficient p when we already know that the signals we work with have few differences, otherwise we will not get the expected path.

Set of record	Percentage of unchanged path	
	$p = 0.5$	$p = 0.2$
Concerto de la mer	100%	34%
K.550 -1.	66%	33%
Le petit prince	100%	60%
Total	92%	42%

Table 4: Influence of truncature parameter p



(a) $p = 1$ (b) $p = 0.5$ (c) $p = 0.2$

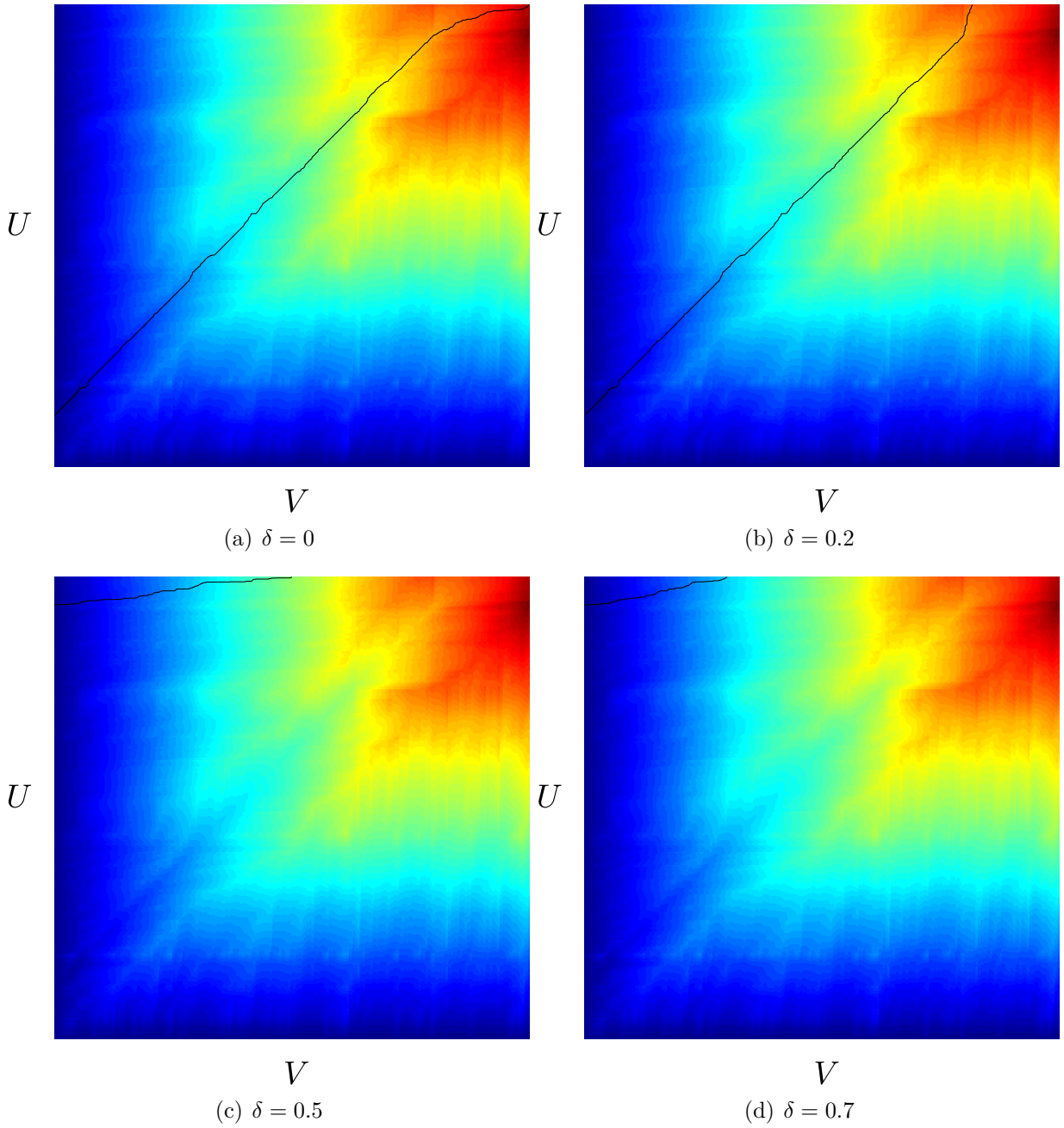
Figure 17: Influence of p. U = *Concerto de la mer 1*, V = *Concerto de la mer 3* ; $\alpha = \beta = \gamma = 1 - \delta = 1$

We timed the two kind of executions: the first, when we computed every coefficient, took on average 15 seconds; when $p = 0.5$ it took 8 seconds on average, when $p = 0.2$ it took 4 seconds on average. Thus, the reduction of the computation time is non neglictable.

5.5 No end constraint

We now investigate the influence of parameter δ, which controls the acceptable location for the end of the path. In order to do so, we ran a DTW between *Le petit prince 1* and *Le petit prince 2*, with $\alpha = \beta = \gamma = p = 1$, and $\delta = 0$, $\delta = 0.2$, $\delta = 0.5$, and $\delta = 0.7$. We see on Figure 18 that without

the boundary condition, the path is no longer forced to take high cost steps, and that it is now the expected path, the one of minimal cost. We verify with an audio reconstruction that with this path, the longer signal stops at the end of the shorter.



(a) $\delta = 0$ (b) $\delta = 0.2$

(c) $\delta = 0.5$ (d) $\delta = 0.7$

Figure 18: Influence of δ. U = *Le petit prince 1*, V = *Le petit prince 2* ; $\alpha = \beta = \gamma = p = 1$

We ran tests on every signal that we had recorded in order to ensure that this improvement was consistent with the result of the basic DTW at least for the major part of the path we found, and to make sure that we did not find any irregular or inacceptable paths.

5.6 Weighted variants

In this section we test the influence of the weight coefficients α, β, γ to the output paths. We propose to realign *K550-1 1* and *K550-1 3* with $p = 1 - \delta = 1$ and several values for α, β, γ. Figure 19 shows the path obtained with the reference case where $\alpha = \beta = \gamma = 1$.

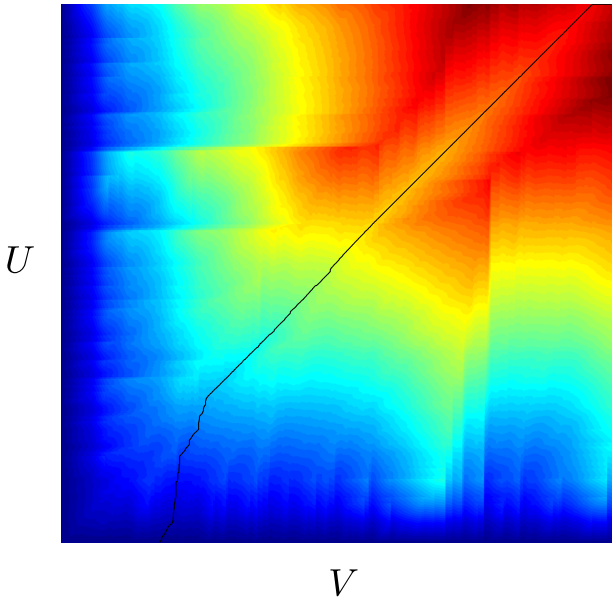
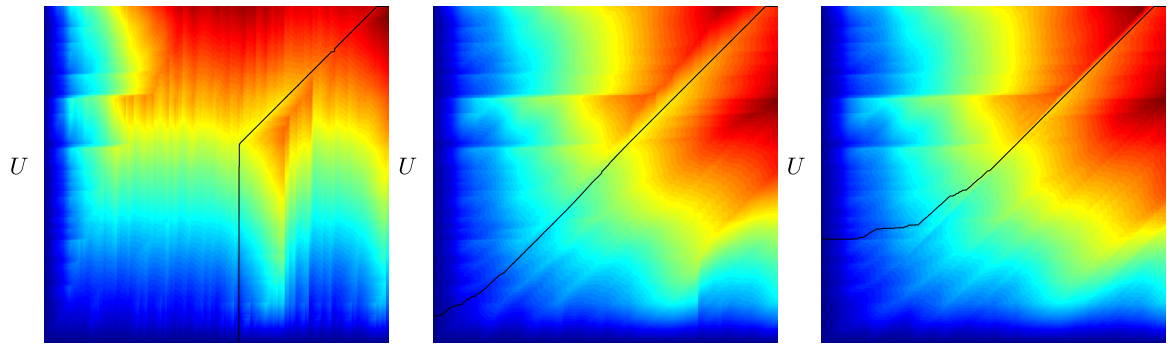


Figure 19: *U = K550-1 1*, *V = K550-1 3*; $\alpha = \beta = \gamma = p = 1 - \delta = 1$

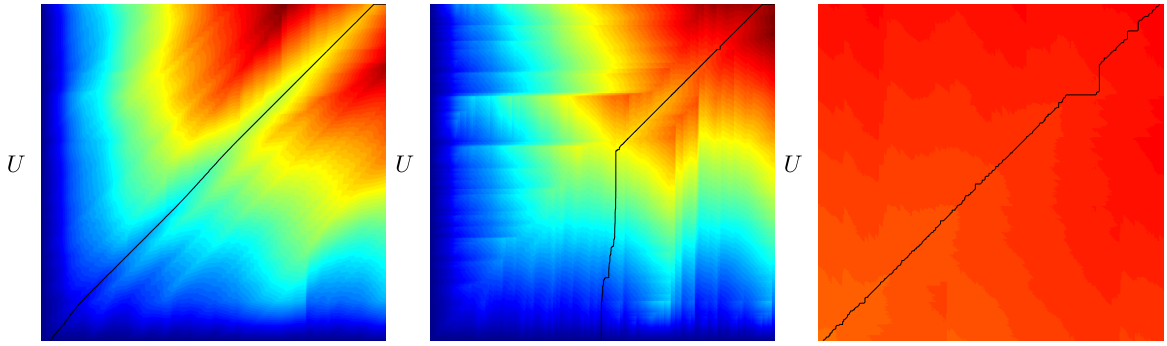
Influence of γ: variable γ, fixed $\alpha = \beta = 1$ This variant is asymmetric since α and γ are not equal. If we set $\gamma \geq 1$, the path will be more likely to follow a horizontal or diagonal direction, as seen in Figures 20(b) ($\gamma = 2$) and 20(c) ($\gamma = 10$). In the case of Figure 20(b) ($\gamma = 2$), it seems we get a better path than the one of the reference case, because it is closer from the diagonal direction, which was the expected result. With Figure 20(c) we can see the limits of this method, because if we set γ to a large value, we do not improve our path anymore, and even get degenerated ones. On the contrary, if we set $\gamma \leq 1$, we put a lower cost on vertical moves, and thus the path will be incitated to follow this direction as Figure 20(a) shows ($\gamma = 0.5$).



(a) $\gamma = 0.5$ (b) $\gamma = 2$ (c) $\gamma = 10$

Figure 20: Influence of γ. *U = K550-1 1*, *V = K550-1 3*; $\alpha = \beta = p = 1 - \delta = 1$

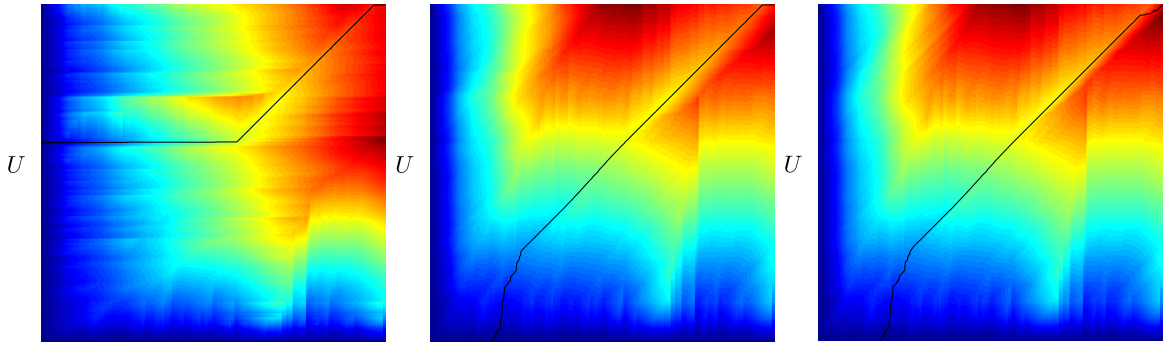
Influence of β: variable β, fixed $\alpha = \gamma = 1$ This variant is symmetric as $\alpha = \gamma$. In this case, when we set $\beta \leq 1$, the resulting path is closer from diagonal direction, as Figure 21(a) ($\beta = 0.5$) shows. This parameter can be used in order to prevent abberant paths which follow a straight vertical or horizontal line. In the case of $\beta \geq 1$, we put a higher cost on diagonal direction and hence the path will be incitated to make vertical or diagonal moves, as Figure 21(b) ($\beta = 2$) shows. In fact in this case, not only the path goes out of this way, but even the diagonal part of the path (at the end) is in fact a composition of vertical and horizontal moves, as zoom in Figure 21(c) shows.



(a) $\beta = 0.5$ (b) $\beta = 2$ (c) $\beta = 2$, zoom

Figure 21: Influence of β. U = *K550-1 1*, V = *K550-1 3*; $\alpha = \gamma = p = 1 - \delta = 1$

Influence of α: variable α, fixed $\beta = \gamma = 1$ This variant is quite similar to the first one. Results and interpretations are almost the same, vertical direction being replaced by horizontal direction. Figure 22(a) ($\alpha = 0.5$) shows the impact of a coefficient $\alpha \leq 1$, whereas Figure 22(b) ($\alpha = 2$) and Figure 22(c) ($\alpha = 4$) show the impact of a coefficient $\alpha \geq 1$.



(a) $\alpha = 0.5$ (b) $\alpha = 2$ (c) $\alpha = 4$

Figure 22: Influence of α. U = *K550-1 1*, V = *K550-1 3*; $\beta = \gamma = p = 1 - \delta = 1$

Summary Those variants show that coeffecients α, β, γ behave as expected on the algorithm: they can be used to induce path to follow a given direction, and to manually prevent abberant results, for the case where the path follows a straight line, for instance. However, the initial setting $\alpha = \beta = \gamma = 1$ is the only one really adapted to audio realignment, and modifying those coefficients should be done only if this configuration fails. However, some of the variants are better match recognizers than the classic DTW for speech recognition, for it provides a more fiable match score index (see [9] and [10]).

5.7 Limits of the algorithm

In this section, we investigate the limits of the algorithm by raising the issue of precision, and showing some failed examples.

5.7.1 Precision of the realignment

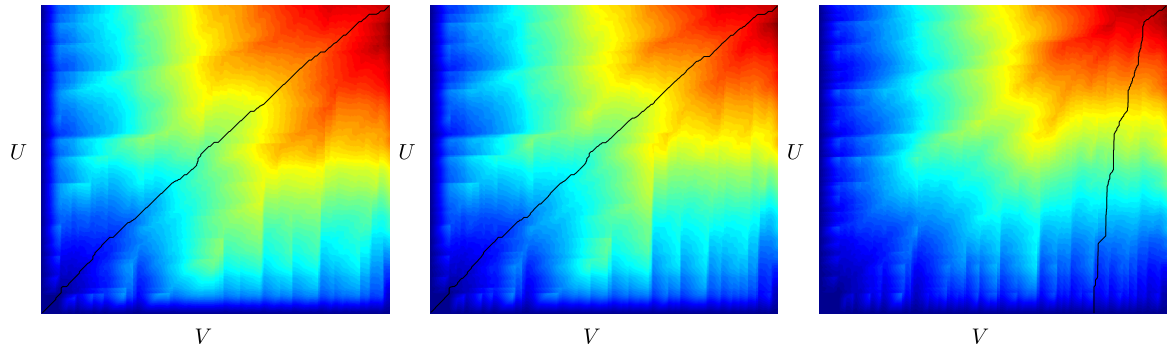
All tests presented in this article were computed with 1024-sample windows (23.2 ms). Provided that we used 50% overlap, it means that the precision of the realignment is at most around 10 ms.

We therefore decided to test the algorithm with smaller windows, hoping that the algorithm remains robust and would be able to realign with a increased precision. We therefore tried to realign two sentences pronunced by different speakers (*Baudelaire T* and *Baudelaire P*) with the basic algorithm ($\alpha = \beta = \gamma = p = 1 - \delta = 1$) but different window lengths (1024, 512 and 256). Results are presented on Figure 23 and can be summarized as follows:

- 512-sample window (11.6 ms) (Figure 23(b)): we notice that the path has the same shape that the one of the one of the 1024-sample windows (Figure 23(a)): that means that we do not get much more information by reducing window size. The drawback of this method is that the computation time is mutiplied by four (the time series involved are twice longer with this window, hence we need to compute four times more indexes in the cost matrix): the average compution time of the algorithm for a signal of 30s gets from 15s to a minute.

- 256-sample window (5.8 ms) (Figure 23(c)): not only is the computation time multiplied by 16, but the paths may not be relevant anymore. This is probably due to the fact that when using Fourier transform, the frequency resolution decreases when the window length decreases, making it more difficult to correctly match the STFT frames.

The precision of the algorithm is therefore limited by two factors: computation time and frequency resolution. If in theory it seems possible to use 512-sample window without degrading the path, we noticed on listening tests that when reconstructing the audio signals, differences between realignments using 512-sample window and 1024-sample window are nearly inaudible.



(a) Hann window size = 1024 (b) Hann window size = 512 (c) Hann window size = 256

Figure 23: Influence of the window length. U = *Baudelaire T*, V = *Baudelaire P* ; $\alpha = \beta = \gamma = p = 1 - \delta = 1$

5.7.2 Failed examples and possible solutions

This section presents some examples which appear to be challenging for the algorithm. For instance, with files *Le petit prince 1* and *Le petit prince 3* and the basic configuration $\alpha = \beta = \gamma = p = 1 - \delta = 1$, we see on Figure 24(a) that the path is likely to be unrelevant. This happens mainly with audio files where there is speech or with music presenting great variations in the amplitude of the sound: for instance, it happened when we tried to rematch our voices, or in the *K550-1*, where the amplitude of the sound can get from very low to very high. So we guess that this is due to the fact that the algorithm considers times of silence as times of noise (noise is already present in every audio file, but when the sound is very low, its amplitude is very close to the one of the noise), and that it has a great difficulty to match noise with noise (or signals very noised with signals very noised).

In these specific cases, we found out that using different distances d may improve the results and enable to find the right path.

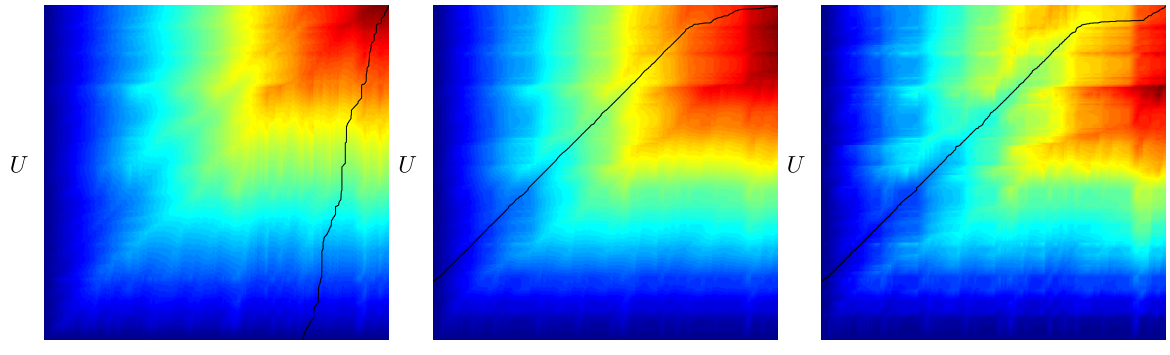
For instance, if we change the definition of our distance to the one of (18), we see an improvement of the quality of the path for *Le petit prince 1* and *Le petit prince 3* (see Figure 24(b)).

$$d(U,V) = \left\| \frac{U}{\|U\|_2} - \frac{V}{\|V\|_2} \right\|_2 \tag{18}$$

The same way we notice that using the cosine distance defined by (19) improve the results in this case, as Figure 24(c) shows.

$$d(U,V) = \frac{(U,V)_2}{\|U\|_2 \|V\|_2} \tag{19}$$

In both cases, we guess that what really improved the results is the change in the type of normalization (L-2 vs. L-1 in the basic algorithm), which was maybe more relevant for these low-amplitude examples. So, changing the type of renormalization or testing with a different distance can lead to find one more suited to the problem we are confronted to, but we miss a general improvement that would solve all the cases at once.



(a) $d(U,V) = \left\| \frac{U}{\|U\|_1} - \frac{V}{\|V\|_1} \right\|_2$ (b) $d(U,V) = \left\| \frac{U}{\|U\|_2} - \frac{V}{\|V\|_2} \right\|_2$ (c) $d(U,V) = \frac{(U,V)_2}{\|U\|_2 \|V\|_2}$

Figure 24: U = *Le petit prince 1*, V = *Le petit prince 3* ; $\alpha = \beta = \gamma = p = 1 - \delta = 1$

6 Conclusion

In this article we have presented and tested several variants and improvements of the Dynamic Time Warping algorithm for audio files realignment. We have seen on several exemples that it was often possible to realign two audio signals with the basic DTW algorithm. More complex cases can be dealt with by using the variants we have described in this article (for instance, penalisation on some moves, change in the normalisation...) In particular, when the user has an *a priori* on what he expects to find, the methods presented here allows him to intuitively design the right setup to solve his problem.

While the algorithm presented here is dedicated to audio files realignment, DTW has also been used in different fields of sound processing (mainly speech recognition), and the algorithm and implementation presented here can also be used in this context. Moreover, as it is presented the algorithm is easily adaptable to be applied to numerous different context not related to sound processing, if the time series are submitted to the right process.

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*, pages 117–127. Australian Computer Society, Inc., 2009.

[2] Richard Ernest Bellman and Stuart E Dreyfus. Applied dynamic programming. 1962.

[3] Simon Dixon. Live tracking of musical performances using on-line time warping. In *Proceedings of the 8th International Conference on Digital Audio Effects*, pages 92–97. Citeseer, 2005.

[4] Simon Dixon and Gerhard Widmer. Match: A music alignment tool chest. *Proc. ISMIR, London, GB*, 2005.

[5] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.

[6] Eamonn J Keogh and Michael J Pazzani. Derivative dynamic time warping. In *the 1st SIAM Int. Conf. on Data Mining (SDM-2001), Chicago, IL, USA*, 2001.

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

[8] Robert Macrae and Simon Dixon. Accurate real-time windowed time warping. In *Proc. ISMIR*, pages 423–428, 2010.

[9] Lawrence Rabiner, A Rosenberg, and S Levinson. Considerations in dynamic time warping algorithms for discrete word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 26(6):575–582, 1978.

[10] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 26(1):43–49, 1978.

[11] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.