CHAPTER 7

ALIGNMENT

Algorithms for the automatic alignment of sequences of different lengths — *Dynamic Time Warping (DTW)* in particular — are part of various algorithms for the analysis of audio signals. The following chapter gives an introduction to the DTW algorithm and presents the two typical synchronization applications such as audio-to-audio alignment and audio-to-score alignment.

7.1 Dynamic Time Warping

The objective of *DTW* is to align or to synchronize two sequences of different length [235]. Given two sequences $A(n_\mathrm{A})$ with $n_\mathrm{A} \in [0; \mathcal{N}_\mathrm{A} - 1]$ and $B(n_\mathrm{B})$ with $n_\mathrm{B} \in [0; \mathcal{N}_\mathrm{B} - 1]$, a distance measure can be computed between all pairs of $A(n_\mathrm{A})$ and $B(n_\mathrm{B})$, resulting in a *distance matrix* $\boldsymbol{D}_\mathrm{AB}(n_\mathrm{A}, n_\mathrm{B})$. The specific path from $\boldsymbol{D}_\mathrm{AB}(0,0)$ (the start of both sequences) to $\boldsymbol{D}_\mathrm{AB}(\mathcal{N}_\mathrm{A} - 1, \mathcal{N}_\mathrm{B} - 1)$ (the end of both sequences) which minimizes the overall distance is the most likely *alignment path* (also *warping path*) between the two sequences. Figure 7.1 shows two example sequences, the corresponding distance matrix and the resulting alignment path between the two sequences computed with standard DTW as described below.

This alignment path will be referred to as $\boldsymbol{p}(n_\mathrm{P})$ and $n_\mathrm{P} \in [0; \mathcal{N}_\mathrm{P} - 1]$; it is a direct measure of how one sequence has to be warped (scaled in time) to give the best fit to the other sequence. Each path entry is a matrix index in the range $([0; \mathcal{N}_\mathrm{A} - 1], [0; \mathcal{N}_\mathrm{B} - 1])$.

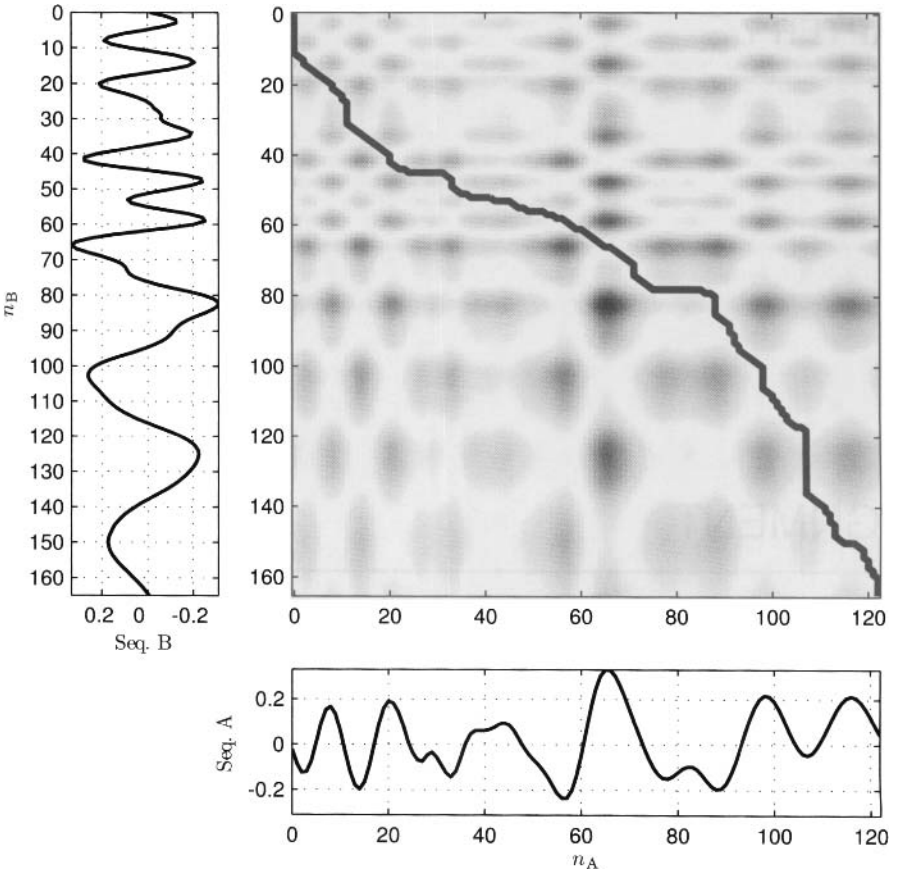

Figure 7.1 Distance matrix and alignment path for two example sequences; dark entries indicate a large distance

The following conditions apply to the path:

- *Boundaries*: the path has to start at the first index of both sequences and has to end at the end of both sequences, meaning that it covers both entire sequences from beginning to end:

$$\boldsymbol{p}(0) = [0,0], \tag{7.1}$$
$$\boldsymbol{p}(\mathcal{N}_P - 1) = [\mathcal{N}_A - 1, \mathcal{N}_B - 1]. \tag{7.2}$$

- *Causality*: the path can only move forward through both sequences, meaning that it is not allowed to "go back in time":

$$n_A\big|_{\boldsymbol{p}(n_P)} \leq n_A\big|_{\boldsymbol{p}(n_P+1)}, \tag{7.3}$$
$$n_B\big|_{\boldsymbol{p}(n_P)} \leq n_B\big|_{\boldsymbol{p}(n_P+1)}. \tag{7.4}$$

- *Continuity*: no index n_A or n_B can be omitted, meaning that the path is not allowed to jump through either sequence:

$$n_A\big|_{\boldsymbol{p}(n_P+1)} \leq (n_A+1)\big|_{\boldsymbol{p}(n_P)}, \tag{7.5}$$

$$n_B\big|_{\boldsymbol{p}(n_P+1)} \leq (n_B+1)\big|_{\boldsymbol{p}(n_P)}. \tag{7.6}$$

These path restrictions result in a theoretical maximum path length of

$$\mathcal{N}_{P,max} = \mathcal{N}_A + \mathcal{N}_B - 2 \tag{7.7}$$

when the path runs along the edges of the distance matrix[1] and a minimum path length of

$$\mathcal{N}_{P,min} = \max(\mathcal{N}_A, \mathcal{N}_B) \tag{7.8}$$

when the path runs on the matrix diagonal for as long as possible.

In order to find the optimal alignment path the concept of "cost" is used. The cost of a path $\boldsymbol{p}_j$ can be computed by accumulating the values of the distance matrix at all path points:

$$\mathfrak{C}_{AB}(j) = \sum_{n_P=0}^{\mathcal{N}_P-1} \boldsymbol{D}(\boldsymbol{p}_j(n_P)). \tag{7.9}$$

The optimal alignment path is then the path that minimizes the overall cost:

$$\mathfrak{C}_{AB,min} = \min_{\forall j}\left(\mathfrak{C}_{AB}(j)\right), \tag{7.10}$$

$$j_{opt} = \operatorname*{argmin}_{\forall j}\left(\mathfrak{C}_{AB}(j)\right). \tag{7.11}$$

The optimal path can thus be found by computing all possible paths through the matrix $\boldsymbol{D}$ and determining the path with the lowest overall cost.

By means of utilizing glsIdxDP techniques this brute force approach may be discarded as the best global solution can be computed more efficiently. As an intermediate result the *cost matrix* $\boldsymbol{C}_{AB}$ is introduced; it has the same dimensions as the distance matrix, but each matrix element contains the accumulated overall cost of the best path to this specific matrix element.

The cost matrix can be computed iteratively by

$$\boldsymbol{C}_{AB}(n_A, n_B) = \boldsymbol{D}_{AB}(n_A, n_B) + \min\begin{cases}\boldsymbol{C}_{AB}(n_A-1, n_B-1)\\ \boldsymbol{C}_{AB}(n_A-1, n_B)\\ \boldsymbol{C}_{AB}(n_A, n_B-1)\end{cases}. \tag{7.12}$$

Figure 7.2 plots both the distance matrix and the corresponding cost matrix for the example sequences from Fig. 7.1.

The first entry of the cost matrix is initialized with

$$\boldsymbol{C}_{AB}(0,0) = \boldsymbol{D}_{AB}(0,0). \tag{7.13}$$

[1]The -2 originates in the automatic avoidance of the corner of the distance matrix.

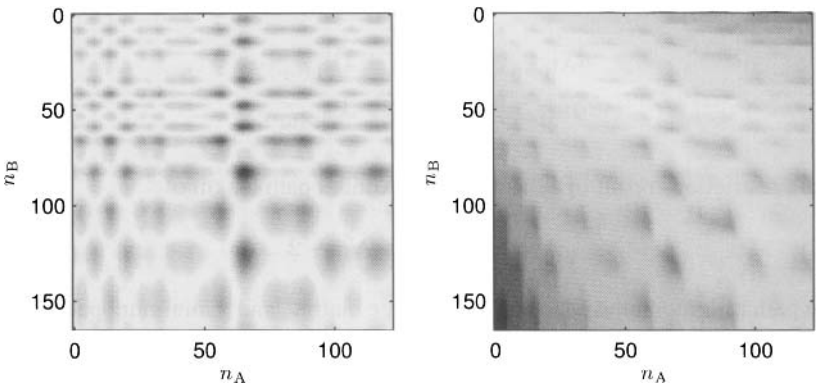

Figure 7.2 Distance matrix (left) and corresponding cost matrix (right)

Due to the alignment path restrictions given above the computation of both the first row and the first column of the cost matrix is trivial:

$$\boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}},0) = \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}},0) + \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1,0), \tag{7.14}$$

$$\boldsymbol{C}_{\mathrm{AB}}(0,n_{\mathrm{B}}) = \boldsymbol{D}_{\mathrm{AB}}(0,n_{\mathrm{B}}) + \boldsymbol{C}_{\mathrm{AB}}(0,n_{\mathrm{B}}-1). \tag{7.15}$$

This can also be interpreted as initializing the distance matrix for indices smaller than 0 with

$$\begin{aligned} \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}},-1) &= \infty, \\ \boldsymbol{C}_{\mathrm{AB}}(-1,n_{\mathrm{B}}) &= \infty, \end{aligned}$$

and applying Eq. (7.12).

Each cost matrix element contains the minimum cost to reach that element. During the calculation of the cost matrix, the indices of the preceding cell that has been selected as the minimum cost for each matrix element have to be remembered. The optimal path can then be traced back from the current element to the beginning.

The complete iterative algorithm can thus formally be summarized by

- *Initialization*:

$$\boldsymbol{C}_{\mathrm{AB}}(0,0) = \boldsymbol{D}_{\mathrm{AB}}(0,0), \tag{7.16}$$

$$\boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}},-1) = \infty, \tag{7.17}$$

$$\boldsymbol{C}_{\mathrm{AB}}(-1,n_{\mathrm{B}}) = \infty. \tag{7.18}$$

- *Recursion*:

$$\boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}) = \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}) + \min \begin{cases} \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}-1) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}-1) \end{cases}, \quad (7.19)$$

$$j = \operatorname{argmin} \begin{cases} \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}-1) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}-1) \end{cases}, \quad (7.20)$$

$$\Delta\boldsymbol{p}(n_{\mathrm{A}}, n_{\mathrm{B}}) = \begin{cases} [-1,-1] & \text{if } j = 0 \\ [-1, 0] & \text{if } j = 1 \\ [0,-1] & \text{if } j = 2 \end{cases}. \quad (7.21)$$

- *Termination*:

$$n_{\mathrm{A}} = \mathcal{N}_{\mathrm{A}} - 1 \wedge n_{\mathrm{B}} = \mathcal{N}_{\mathrm{B}} - 1. \quad (7.22)$$

- *Path backtracking*:

$$\boldsymbol{p}(n_{\mathrm{P}}) = \boldsymbol{p}(n_{\mathrm{P}}+1) + \Delta\boldsymbol{p}\big(\boldsymbol{p}(n_{\mathrm{P}}+1)\big),\ n_{\mathrm{P}} = \mathcal{N}_{\mathrm{P}} - 2, \mathcal{N}_{\mathrm{P}} - 3, \ldots, 0. \quad (7.23)$$

Note that the distance matrix is frequently be replaced by a *similarity matrix*; the algorithm will remain the same although the cost matrix (as introduced below) will have to be replaced by a *likelihood matrix* and some algorithmic details will have to be modified in other places as well, for instance, replacing the min operation by a max operation.

7.1.1 Example

In order to allow a better understanding of the initially rather abstract concept of DTW we will present a small example. The (one-dimensional) input sequences are

$$\begin{aligned} A &= [1,\ 2,\ 3,\ 0], \\ B &= [1,\ 0,\ 2,\ 3,\ 1], \end{aligned}$$

and we will simply use the magnitude of the difference as the distance measure. The distance matrix is then

$$\boldsymbol{D}_{\mathrm{AB}} = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 2 & 3 & 0 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 3 \\ 0 & 1 & 2 & 1 \end{bmatrix}, \quad (7.24)$$

and the corresponding cost matrix is

$$\boldsymbol{C}_{\mathrm{AB}} = \begin{bmatrix} 0 & 1 & 3 & 4 \\ 1 & 2 & 4 & 3 \\ 2 & 1 & 2 & 4 \\ 4 & 1 & 1 & 4 \\ 4 & 2 & 3 & 2 \end{bmatrix}. \quad (7.25)$$

During the calculation of the cost matrix, we also memorized the direction of lowest cost for each matrix element:

$$\begin{bmatrix} \cdot & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & \nwarrow & \nwarrow & \nwarrow \\ \uparrow & \nwarrow & \leftarrow & \leftarrow \\ \uparrow & \uparrow & \nwarrow & \leftarrow \\ \uparrow & \uparrow & \nwarrow & \nwarrow \end{bmatrix}$$

in order to be able to backtrack the optimal path through the distance matrix:

$$\boldsymbol{D}_{\mathrm{AB}} = \begin{bmatrix} 0 & & & \\ 1 & & & \\ & 0 & & \\ & & 0 & \\ & & & 1 \end{bmatrix}. \tag{7.26}$$

7.1.2 Common Variants

The standard DTW algorithm as described above is frequently modified; the reasons for this modification can be either the adaption to specific use cases or the optimization of its workload requirements. Several such modifications are described in detail by Müller [236].

7.1.2.1 Transition Weights

In order to favor vertical, horizontal, or diagonal path movement, weighting factors can be applied to Eq. (7.12)

$$\boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}) = \min \begin{cases} \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}} - 1, n_{\mathrm{B}} - 1) & + & \lambda_{\mathrm{d}} \cdot \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}} - 1, n_{\mathrm{B}}) & + & \lambda_{\mathrm{v}} \cdot \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}} - 1) & + & \lambda_{\mathrm{h}} \cdot \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}) \end{cases}. \tag{7.27}$$

In the default DTW approach all these weights had been set to 1. Another typical set of weights is

$$\begin{aligned} \lambda_{\mathrm{d}} &= 2, \\ \lambda_{\mathrm{v}} &= 1, \\ \lambda_{\mathrm{h}} &= 1 \end{aligned}$$

to prevent the implicit preference of the diagonal since one diagonal step corresponds to one horizontal plus one vertical step.

An algorithm closely related to DTW is the *Viterbi algorithm* which finds a path through a set of (observed and hidden) states [231].

7.1.2.2 Different Step Sizes

Instead of forcing the algorithm to only increment each index by one, one can optionally allow larger step sizes or jumps. The arguments of the minimum operation of Eq. (7.12)

could, for example, be replaced by

$$\min \begin{cases} \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}-1) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-2, n_{\mathrm{B}}-1) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}-2) \end{cases} \tag{7.28}$$

which constrains the slope of the warping path to avoid the path containing many consecutive horizontal or vertical steps. This modification will only work if the sequence lengths $\mathcal{N}_{\mathrm{A}}$ and $\mathcal{N}_{\mathrm{B}}$ differ not by more than a factor of 2. Another alternative enforcing the alignment of all elements of both sequences is to replace the arguments of the minimum operation of Eq. (7.12) by

$$\min \begin{cases} \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}-1) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-2, n_{\mathrm{B}}-1) + \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-3, n_{\mathrm{B}}-1) + \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}) + \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}-2, n_{\mathrm{B}}) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}-2) + \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}-1) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}-3) + \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}-1) + \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}-2) \end{cases} . \tag{7.29}$$

7.1.3 Optimizations

If the two sequences to be aligned are long, the size of the distance matrix increases drastically as the number of matrix elements is the multiplication of the length of the sequences $\mathcal{N}_{\mathrm{A}} \cdot \mathcal{N}_{\mathrm{B}}$. This results in both high memory requirements and a large overall number of operations. The effects can be alleviated by using different approaches to optimization.

One simple approach to reduce the amount of memory without changing the (standard DTW) algorithm or its results is to replace the cost matrix with two vectors, a row and a column vector of cost entries, as the cost of more distant elements is not used by the algorithm.

7.1.3.1 Maximum Time Deviation Constraint

Under the assumption that the timing of the two sequences to be aligned does not deviate more than a certain maximum deviation $T_{\max}$, neither the distances nor the cost has to be computed for matrix entries outside a band with the width of $2T_{\max}$ centered around the diagonal from start $(0, 0)$ to stop $(\mathcal{N}_{\mathrm{A}}-1, \mathcal{N}_{\mathrm{B}}-1)$. This optimization has first been proposed by Sakoe and Chiba [235] and is visualized in Fig. 7.3 (left).

7.1.3.2 Maximum Tempo Deviation Constraint

If the slope of the alignment path is constrained or, in other words, the tempo difference between the two sequences is limited then matrix entries outside of a trapezoid spanned by this tempo relationship do not have to be computed. This optimization has first been proposed by Itakura [237] and is visualized in Fig. 7.3 (right).

If start and end points of the two sequences are not necessarily a perfect match, for example, in the case of silence frames at the beginning and end of a sequence, this approach should be combined with the optimization assuming a maximum time deviation (see above) to allow horizontal and vertical movement along the matrix edges at start and end. Otherwise the path restrictions are too narrow if the sequences do not start and stop at exactly the same point. This is a problem of the path shown in Fig. 7.3 (right) at both start and end.

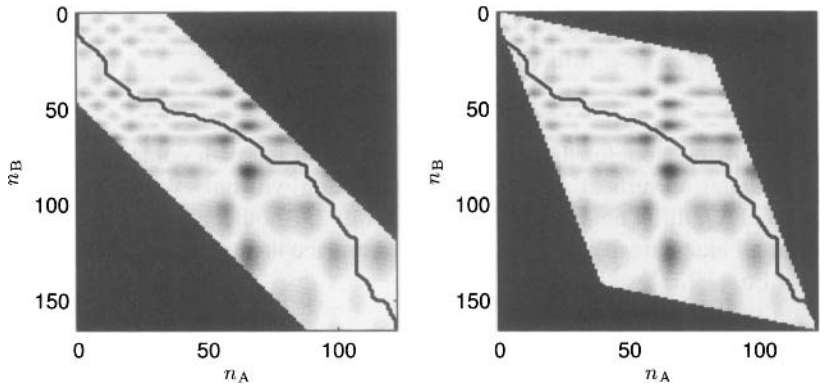

Figure 7.3 Path restrictions for performance optimizations of DTW: maximum time deviation (left) and maximum tempo deviation (right)

7.1.3.3 Sliding Window

The DTW algorithm is not a real-time algorithm as it requires the complete sequences for processing. It is possible to use DTW in a pseudo-real-time context when the alignment path is only computed within a sliding local window [238]. This results in a high algorithmic latency. Outside this window the path will not be updated anymore. The number of operations depends on the window length and is independent of the sequence length.

7.1.3.4 Multi-scale Dynamic Time Warping

Multi-scale DTW is based on the idea of processing the input sequences at different time resolution. More specifically, the time resolution of the input sequences can be reduced by both, using larger block sizes for computing the distances or low-pass filtering and down-sampling the existing sequences. The extracted path through this down-sampled distance matrix can then be used to define a sliding window to be used for a second matrix with finer resolution. This process can be done in two stages [239] or iteratively in multiple stages [236, 240].

7.2 Audio-to-Audio Alignment

Audio-to-audio alignment describes the process of retrieving corresponding points in time in between two audio signals with the same or a similar content. It requires an analysis of the audio files enabling the mapping of points in time in one signal to points in time in the other signal. The knowledge of those synchronization points enables a variety of different use cases such as

- quick browsing for certain parts in recordings in order to easily compare parts auditorily [238, 240],
- adjusting the timing of one recording to that of a second by means of a dynamic time stretching algorithm. This has practical use in a music production environment. The

different voices of a homophonic arrangement (e.g., the backing vocals in a pop song) can, for instance, be automatically synchronized to the lead voice. The same applies for different instruments playing in unison or at least in the same rhythm,

- automated synchronization of a (dubbed) studio recording with an original, possibly distorted, recording, and
- musicological analysis of the timing information contained in the alignment path of several performances of the same score in order to compare the tempo and timing to a given reference music performance.

The main difference between various publications on audio-to-audio alignment can be found in the features extracted from the audio signal as well as in the subsequent computation of the distance matrix. Differences in calculating the alignment path are mainly due to the variants and optimizations of DTW listed above.

Hu and Dannenberg compute a simple pitch chroma as input feature [241]. The distance is the Euclidean distance between all pitch chroma pairs of the two "audio" sequences.

Turetsky and Ellis use several STFT-based features such as the power and the first order difference in time and frequency. They then use the cosine distance as similarity measure [239].

Dixon and Widmer argue that the main objective of audio-to-audio alignment is the synchronization of the onset times and propose to use a spectral flux-based feature subjected to HWR; the feature is computed in semi-tone bands [238]. The distance measure is the Euclidean distance. In order to achieve pseudo-real-time alignment, they use a modified DTW approach that estimates the optimal local path within a sliding window (see Sect. 7.1.3.3).

Müller et al. compute the pitch chroma via the energy outputs of a filterbank. They use multi-scale DTW as described in Sect. 7.1.3.4 to efficiently compute the alignment path [240].

Kirchhoff and Lerch pointed out that the type of features used for audio-to-audio alignment depend on the use case; the alignment may be done for signals with either identical pitch, envelope, or timbre while onset-related features can be used in any case [242]. They evaluated a large number of features from these four categories and did not use a vector distance but trained an LDA classifier for two classes (*on path* vs. *not on path*) for the automatic feature weighting and the distance computation.

7.2.1 Ground Truth Data for Evaluation

A set of pairs of audio files with clearly defined synchronization points is required as ground truth to evaluate the accuracy of an estimated alignment path. There exist several ways of generating a ground truth data set for audio-to-audio alignment systems. Corresponding points in time can be (manually or semi-automatically) annotated, onset times can be monitored and stored during a recording of a computer-monitored instrument, or MIDI files can be rendered to audio by means of a sample player. As pointed out in Sect. 6.3.3.2 the manual annotation of points in time is a rather arduous process and thus generally only a few points per test file can be labeled. The drawback of using piano-generated data is its restriction to solo piano music; furthermore, confining the evaluation to note onsets might be sufficient for the case of solo piano music, however, other kinds of music may also require the synchronization during the time span in between onsets. The disadvantage of using synthesized samples is that their properties might differ from "real-world" signals.

Alternatively, it is possible to artificially generate modified pairs of audio signals by using a dynamic time stretching algorithm, i.e., an audio processing algorithm able to change the tempo without changing the pitch of an audio signal [243]. This allows for high accuracy of the data set while allowing to test with a wide range of musical styles and instrumentations. In this case the validity of the data set depends (a) on the realistic dynamic use of stretch factors and (b) on the audio quality of the stretching engine. Furthermore, the test data originates from the same audio signal which may give too positive results.

7.3 Audio-to-Score Alignment

Systems for the synchronization of an audio signal with a musical score (frequently in MIDI format) are usually categorized with respect to their real-time capabilities. Real-time systems are called *score following* systems, and non-real-time (or offline) implementations are referred to as *audio-to-score alignment* systems.

Possible applications of such alignment systems (compare [244]) include

- linking notation and music performance in applications for musicologists to enable working on a symbolic notation while listening to a real performance,
- using the alignment cost as a distance measure for finding the best matching document in a database (i.e., retrieve an audio signal for a score query or vice versa),
- the musicological comparison of different performances,
- automatic accompaniment systems,
- the construction of a new score describing a selected performance by adding information on dynamics, mix information, or lyrics, and
- musical tutoring or coaching system for which the timing of a recorded performance is compared to a reference performance.

7.3.1 Real-Time Systems

Historically, the research on matching a pre-defined score automatically with a music performance goes back to the year 1984. At that time, Dannenberg and Vercoe independently presented systems for the automatic (computer-based) accompaniment of a monophonic input source in real time [245, 246].

In the following years, Dannenberg and Bloch enhanced Dannenberg's system by allowing polyphonic input sources and increasing its robustness against musical ornaments and by using multiple agent systems [247, 248]. Vercoe focused on the implementation of learning from the real performance to improve the score follower's accuracy [249].

Baird et al. proposed a score following system working with MIDI input (for the performance) based on the concept of musical segments as opposed to single musical events; the tracking algorithm itself is not described in detail [250, 251].

Heijink and Desain et al. presented a score following system that takes into account structural information as well. It uses a combination of DP and strict pitch matching between performance and score [252, 253].

While many of previously presented publications focus on the score following part rather than audio processing itself, Puckette and Lippe worked on systems with audio-only input with monophonic input signals such as clarinet, flute, or vocals [254, 255].

Vantomme proposed a monophonic score following system that uses temporal patterns from the performer as its primary information [256]. From a local tempo estimate the next event's onset time is predicted and the distance between expected onset time and extracted onset time is evaluated. In the case of an "emergency," he falls back to the use of pitch information.

Grubb and Dannenberg presented a system following a monophonic vocal performance. It uses the fundamental frequency, spectral features, as well as amplitude changes as features for the tracking process [257, 258]. The estimated score position is calculated based on a PDF conditioned on a distance computed from the previous score event, from the current observation, and from a local tempo estimate.

Raphael published several approaches to score following implementing probabilistic modeling and machine learning approaches incorporating *markov models* [259–261].

Cano et al. presented a real-time score following system for monophonic signals based on an HMM [262]. They used the features zero crossing rate, energy, and its derivative plus three features computed from the fundamental frequency.

Orio et al. introduced a score following system for polyphonic music which utilizes a two-level HMM modeling each event as a state in one level, and modeling the signal with attack, sustain, and rest phase in a lower level [263, 264]. They use a so-called *Peak Structure Distance (PSD)* that represents the energy sum of band-pass filter outputs with the filters centered around the harmonic series of the pitch of the score event under consideration.

Cont presented a polyphonic score following system using hierarchical HMMs using previously learned pitch templates for multiple fundamental frequency matching [265].

7.3.2 Non-Real-Time Systems

While the publications presented above deal with score following as a real-time application, the following publications deal with the closely related topic of non-real-time audio-to-score alignment.

The importance of reliable pattern matching methods has already been recognized in early publications on score following and alignment; in most cases DP approaches have been used [266]; see, for example, Dannenberg's publications on score following mentioned above.

Orio and Schwarz presented an alignment algorithm for polyphonic music based on DTW which combined several local distances (similarity measures) [267]. It uses the PSD as described above and a *delta of PSD* (ΔPSD) modeling a kind of onset probability; it also uses a silence model for low-energy frames [263].

Meron and Hirose proposed a similar approach with several easy-to-compute audio features and suggested post-processing of the alignment results to improve the alignment robustness [268].

The system by Arifi et al. attempts to segment the audio signal into (polyphonic) pitches and performs DP to align MIDI file to the extracted data [269, 270]. The algorithm has been tuned for polyphonic piano music.

Turetsky and Ellis avoided the problems of calculating a spectral similarity measure between score and audio by generating an audio file from the (reference) MIDI file and aligning the two audio sequences with audio-to-audio alignment [239]. For the alignment itself a DP approach is used.

Similarly, Dannenberg and Hu generated an audio file from the MIDI file to align two audio sequences [241, 271]. They calculate the distance measure based on a 12-dimensional pitch chroma. The alignment path is then calculated by a DP approach.

Shalev-Shwartz et al. presented a non-real-time system for audio-to-score alignment utilizing DP [272]. Their algorithm features a training stage for the weighting the features for the distance measure. They derived a confidence measure from audio and MIDI similarity data and trained a weight vector for these features to optimize the alignment accuracy over the training set. The audio feature set contains simple pitch-style features extracted by band-pass filtering, derivatives in spectral bands to measure onset probability, and a time deviation from the local tempo estimate.

The alignment system of Müller et al. is also based on DP [273]. It is targeted at piano music, but they claim genre independence. For the pitch feature extraction, they used a (zero phase) filterbank with each band-pass' center frequency located at a pitch of the equally tempered scale; the filter outputs are used to extract onset times per pitch.

In summary, the standard approach to audio-to-score alignment consists of three major processing steps: first, the audio feature extraction which in most cases approximates a pitch-like representation with onset information; timbre and loudness are too performance specific to be used for audio-to-score alignment. Second, a similarity or distance measure between audio and symbolic (score) features is computed. Finally, the actual alignment or path finding algorithm that is either based on DP, DTW, or on HMMs is applied. With respect to the distance measure, two distinct approaches can be identified. On the one hand the score is transformed into a more audio-like representation, ultimately by directly rendering the MIDI signal into an audio signal; on the other hand the audio signal is converted into a more score-like symbolic format, ultimately by transcribing the signal completely. A first step toward combining these two approaches has been made by Lerch by computing two distance matrices for the two approaches and combining them in a weighted average [274].