Automated Editing For The Music Industry

by

Peter Hoyes (JE)

Fourth-year undergraduate project in Group F
2012/2013

I hereby declare that, except where specifically indicated, the work submitted herein is my own original work

Signed

Date

Automated Editing for the Music Industry

Peter Hoyes (JE)

May 2013

Technical Abstract

When a studio recording is made of a piece of music, multiple performances are always recorded, because it is very difficult to achieve a single perfect take when large numbers of people are involved. It is left to the job of the sound engineer to sort through the numerous takes, select the best sections from each one and tedious splice these bits together to produce a seamless recording. While the process of selecting the best takes is a subjective task, this project is working towards the aim of producing a software package which is capable of performing the splicing automatically, if the sound engineer provides a list of instants at which to cut between recordings.

The procedure for carrying out a cut between two recordings requires two steps: precisely aligning the two recordings irrespective of tempo difference and performing a cross-fade between the two performances without the transition being audible. Leonard Chan investigated both tasks in a 4th year project last year, and had success performing cross-fades, but there were improvements still be made to the alignment process. This report therefore looks into the music alignment problem in more depth, using, as last year, an accurate electronic representation of a musical score (a MIDI file) as a basis for the alignment.

The problem is tackled by converting the musical score into a sequence of discrete states in a hidden Markov model, the output of which is the audio recording, which is modelled using a probability distribution which is dependent on the current state. Three options for this probability distribution were considered, all based on properties of the audio in the frequency domain. The Poisson point model, which was used in last year's project, was investigated again. This detects the peaks in the frequency spectrum and models them as a Poisson point process with an inhomogeneous intensity function based on the expected peak locations (based on information derived from the score).

The other two probability models were based on summing frequency content in bins centred at every semitone in the spectrum. A chromagram method was considered, which involves summing bins corresponding to each of the 12 notes in the musical scale, producing a 12-point feature vector, which is then normalised to remove the effect of variations in loudness. A similar process is applied to the MIDI file to produced a second 12-point vector, and the likelihood is calculated by comparing these two vectors. The second spectrum-based method involved calculating the 'peak structure distance' by summing only frequency content in the bins in which one would expect peaks given the set of musical pitches at a moment the score, and again normalised.

Three incrementally more sophisticated methods were considered for efficiently calculating the most likely sequence through the set of states given the set of observations (the audio recording), all of which use dynamic programming to achieve worst case $O(n^2)$ complexity. Dynamic time warping involved simply computing the most likely path through a cost matrix consisting of the likelihood calculated at every possible combination of score and audio positions. Hidden Markov models were used to improve the alignment, by setting the expected probability of advancing to the next state to the expected duration of that state, which is the length of the corresponding note(s) in the score. Finally, a semi-Markov model was considered in which the probability of advancing the next state is always 1, but states are allowed to occupied for a a length of time. A probability distribution is applied the amount of time spent in a state, which in this case was a Gaussian model, which was centred about the expected duration of the state.

The best results were achieved using a chromagram and the Gaussian semi-Markov duration-based modelled - less than 4% of states severely were misaligned based on the small sample used. The method based on peak structures fared almost as well, and in some cases it aligned more precisely with the onset of the notes in the score. The Poisson point model performed reasonably well, but the large number of false positives caused by peaks detected in the background noise of the audio meant the path was unstable and struggled to recover from deviations. The performance differences between the path-finding methods investigated were less pronounced, but there was a clear improvement in accuracy as the modelling of the state durations improved.

While the semi-Markov model created the best alignments on the whole, it was also a significant constant factor more expensive to compute than both the standard Markov model and dynamic time warping. There were also moments when the tempo of the performance deviated significantly from that in the score, causing the alignment to be momentarily less accurate than the other methods.

As part of the project, a software framework was created for the efficient evaluation of different music-score alignment algorithms. It includes a web-based visualisation tool which consists of a cursor which moves along a piano-roll representation of the score, which was very useful for qualitatively analysing the generated alignments, especially in the absence of accurate ‘ground truth’ data.

The main suggestions for future work are to investigate modifying the likelihood probabilities to take into account the ‘attack’ at the start of each note rather than just the transient characteristics, and to implement methods such as path pruning to improve the efficiency of the Viterbi-based algorithms.

Contents

1 Introduction

1.1 Motivation

When a studio recording is made of a piece of music, due to the time constraints involved it is very difficult to perform a single perfect performance. Therefore, in order to save time and expense, several performances or 'takes' are recorded in the hope that the same errors are not made in the same place is every recording. The subsequent editing process then involves evaluating each take in turn and selecting which sections of which takes contain to include to construct a 'perfect' performance. Once this has been done, the best sections must be tediously spliced together. While the question of selecting the best performances is, and will most likely remain, a largely subjective problem, the process of creating the cuts between performances could feasibly be aided by signal processing algorithms.

The process of automatically cutting between takes involves two, fairly separate tasks. Firstly, all the takes must be aligned to a common time base in order to compensate for differences in tempo. Secondly, a method is needed to smoothly transition between two takes at the desired moment. The ultimate aim of the project is aid the development of a tool, which, given a number of takes of the same piece of music, is able to warp them all to the same time frame (with the help of a score), and to cleanly cut between takes at any moment of the sound engineer's choosing.

In practice, particularly with long pieces of music, takes are often recorded which start and end mid-way through a piece of music (perhaps to cover a specific, known error in a previous take). We will assume for the basis of this project that all takes are manually logged with their start and end points in the score so that we do not need to develop additional procedures to search for the start/end points automatically.

A previous 4th year project by Leonard Chan [2] produced some effective cross-fading algorithms but had more limited success with the alignment problem. My work will therefore solely concentrate on building upon his work in order to improve the time alignment of a musical performance to a musical score.

Practical problems involve recordings with multiple instruments playing at the same time (or multiple independent musical lines played by the same instrument in the case of keyboard instruments). Such pieces of music are described as being 'polyphonic'. Due to difficulties caused by the overlapping spectra of simultaneous tones, I shall focus initially on monophonic music, but I shall also later apply the algorithms developed to polyphonic music too.

While the algorithms considered are all designed to work off-line (as generally the editing is all done after the recording session has finished), I will consider the algorithmic complexity of the different methods, as in order to be effective in a sound engineer's work-flow they cannot take too long to run and should have linear complexity if possible.

1.2 Prior work

The 'online' alignment problem (where an algorithm is performed in real-time in order to, say, automatically sync a live performance to an accompaniment) is where most of the relevant research has been carried out, but there is much in common with offline alignments (where the algorithm has knowledge of the entire performance, as is considered in this report), especially as the training of such algorithms is generally done offline.

The first steps were carried out was about 20 years ago - [17] states that the automatic music alignment "involves pitch detection at a speed almost impossible for audio methods alone" (due to the limited computers available at the time), so additional information was gathered (using sensors - in this case optical sensors detecting when keys on a flute are pressed), to incorporate into the algorithm.

IRCAM (the Instintut de Recerche et Coordination Acoustique/Musique) in Paris is a major centre of research in the area[1], who also run MIREX (the Music Information Retrieval Evaluation Exchange), through which organisations around the world annually evaluate each others algorithms.

One of the first attempts at alignment using audio data alone happened at this institution in 1992 [12], which worked by first performing pitch detection on the audio, then comparing the reconstructed model of score with the real score. Most alignment research today uses a more probabilistic approach, modelling the probability of being the current state, given a set of audio inputs, with an appropriate distribution. Raphael [13] was one of the first people to model the score as a hidden Markov model, the primary approach used today. These are commonly combined with either Dynamic Time Warping or the Viterbi algorithm[18], methods to efficiently calculate the most likely state sequence, which became popular in speech recognition research before being adopted for score alignment purposes.

More recent work generally models the output probabilities of the models based on summing the bins in the frequency domain of the audio[10, 15], using various heuristic measures to take account of note onsets, silences and errors in the score or performance. A common choice of feature vector is the chromagram [9], which is also widely used in other areas of music-based research such as key detection and music retrieval.

There is also significant commercial interest in the area. Sinfonia[2] is a piece of software used by many professional theatrical productions to automatically embellish a small ensemble with additional instruments, so that fewer musicians have to be paid. Music Plus One[3] is piece of software written by Raphael (mentioned above) advertised for solo musicians to practice playing concertos in time with a recording of an orchestra. There is also research into audio alignment without the use of an accurate score: [5] presents a novel method of comparing two sets of audio data, and [8] details a method of directly correlating audio to an image of a musical score.

One of the most recent papers in the area is again from IRCAM [3], and describes a Markov model with variable-duration states.

1.3 Tools used

I have chosen to implement my project in Python, as it a language I am very familiar with and there are a large range of useful libraries available, for example to process MIDI and audio files. For matrix and scientific functionality, the SciPy[4] and NumPy[5] libraries were used, which also contain functions to perform FFTs and create filters. Matplotlib was used to create the graphics in this report.

All code in this report is written in pseudo-code for clarity and brevity.

[1]http://imtr.ircam.fr/imtr/Score_Following
[2]http://www.rms.biz/products/sinfonia/
[3]http://music-plus-one.com/
[4]http://www.scipy.org/
[5]http://www.numpy.org/

1.4 Report Structure

Sections 2 through 4 detail the theory behind the work carried out. First, the steps taken to convert the audio and MIDI files into a format suitable for processing are detailed. Then the alignment process is discussed in detail: section 3 investigates three models for correlating any given instant in a musical score with an instant in the recording, then section 4 uses the ability to calculate the overall likelihood of any given alignment of the audio to the score to discuss methods to find the most probable alignment.

The results of applying these methods to a selection of music are listed in section 6. They are them are analysed in section 7 and 8 contains some suggestions for further improvements and investigations.

2 Pre-processing

2.1 MIDI files

Traditional musical scores represent music as a series of symbols denoting the pitch, position and duration of each note. MIDI files are electronic representations of a score, discarding unnecessary information regarding the exact layout. Scores are not easy to analyse directly because they contain multiple independent musical lines which must be combined. Also, while scores normally only contain only a few different note lengths, individually notes can theoretically be arbitrarily short, meaning that a score can be viewed as being a continuous-time representation, which is unsuitable for computational analysis.

The score can be converted into a series of states s_i, where $i \in \{1 \dots N\}$. Each state has an array p_i of the MIDI pitches that are present, and a duration, $d(s_i) = d_i$, representing the expected length of the state, based on its length in the score. The overall scale of the durations is not important at this stage, only that they are correct relative to each other. Later, when comparing a sequence of states to a specific audio recording of the corresponding score, the durations will be scaled such that $\sum_{i=1}^{N} d_i = L$, where L is the length of the audio recording.

The notes for each state were stored using their MIDI pitch values, whereby each musical pitch is represented by an integer from 0 to 127. The difference between two MIDI pitch values is one 'semitone' using musical terminology, or $^1/_{12}$ of an octave. The frequency corresponding to a MIDI pitch value can be calculated using equation 2.1 .

$$f = 440 \times 2^{\frac{p-69}{12}} \tag{2.1}$$

Metronome markings in the score, indicating changes in 'tempo' (performance speed) measured in beats per minute, were taken into account. All other markings in the score (such as dynamics and articulation) were considered too susceptible to artistic interpretation so were therefore ignored. Also, while it is known that different musical instruments production different characteristic spectra (for example, plucked instruments have a high degree of inharmonicity), not all MIDI files contain accurate instrumentation information, and it is thought that the differences between instruments will not make a significant difference with the likelihood models used.

The Python library 'music21' [1] was used to convert the MIDI file format into a Python data structure. Two algorithms were then applied to the data structure to produce the sequence of states (based on Algorithms 1 and 2 in [2]).

Figure 2.1 shows the sequence of 5 states produced by processing the MIDI representation of the simple score shown.

A few modifications were made to the sequence of states in order to improve their effectiveness of the alignment process. Any state that was less than 0.1 s long was merged with the following state. 0.1 s is the length of the audio frames used (see section 2.2), so states with a shorter duration than this would have trouble matching uniquely with a single audio frame. Also, a single rest was added at the beginning of the sequence, which corresponds to the silence that is likely to occur at the beginning of the recording.

[1] http://mit.edu/music21/

Algorithm 1 Generating a sequence of states from MIDI representation

```
Initialise $u_t$ = minimum note length used in the **score**
Initialise $L$ = length of **score**
Initialise array of **states** with length $L \div u_t$
for each **note** in **score** do
  for $i = \{\textbf{note}.start \div u_t \dots \textbf{note}.end \div u_t\}$ do
    **states**[i].addNote(**note**.pitch)
  end for
end for
```

Algorithm 2 Merge adjacent identical states

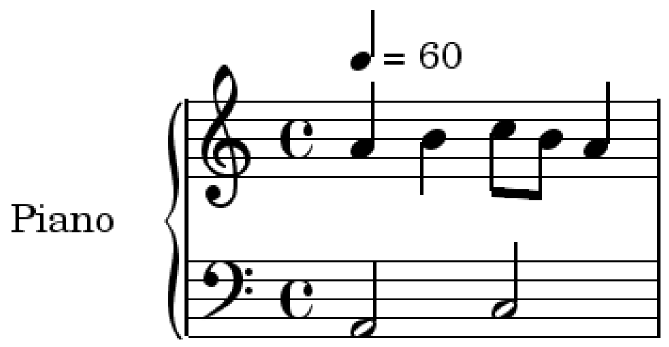
```
while $i < len(states) - 1$ do
  if states[i].notes = states[i+1].notes then
    states[i].duration += states[i+1].duration
    states.delete(i+1)
  else
    i += 1
  end if
end while
```

2.2 Audio files

The audio data was split into frames in a similar way to a past 4th year project on music transcription [1]. The audio files were sliced into frames of length 4096 samples each , giving each frame duration of about 0.1s (with CD-quality audio at 44.1 kHz), while still have a resolution of about 10 Hz in the frequency domain. Zero padding was used to increase the precision of the position of the peaks by interpolation, and data for frequencies above 5kHz were discarded.

The FFT function built into SciPy was used, and a Blackman window low-pass filter was applied to the time domain data to avoid aliasing in the frequency domain.

The resultant audio frames were stored in a matrix $\boldsymbol{A}$ with size N by T values, where the row containing the FFT of a single audio frame is $\boldsymbol{a}_t$.



i	MIDI Notes, p_i	Duration, d_i
0	45, 69	1
1	45, 71	1
2	48, 72	0.5
3	48, 71	0.5
4	48, 69	1

Figure 2.1: A simple score and its corresponding list of states

3 Likelihood calculation

3.1 Overview

The procedures defined in section 2 produce a sequence of states, s_i, $i \in \{1 \dots N\}$, representing the musical score, and a sequence of audio frames,a_t, $t \in \{1 \dots T\}$. This project will consider a probabilistic approach to finding the best matching between the two. We attempt to find an alignment $\mathbf{x} = \{x_1, x_2 \dots x_T\}$, where $x_i \in \{1 \dots N\}$ is the index of the state which correctly corresponds to audio frame a_i.

$$\underset{\mathbf{x}}{\operatorname{argmax}} \{p(\mathbf{A} \mid \mathbf{x})\} = \underset{\mathbf{x}}{\operatorname{argmax}} \left\{ \prod_{i=1}^{M} p(a_i \mid s = x_t) \right\} \tag{3.1}$$

We therefore require a method of computing $p(a_t \mid s = x_t)$. Three methods were considered, a method based on the Poisson distribution, one based on a 'chromagram' representation and one based on applying filters in the frequency domain. In practice, the quantity calculated is often a cost function rather than a probability. This could be normalised across all possibility values to create a true probability distribution, but in practice this is not necessary since the normalisation factor is constant, so it does not affect the result of the argmax operation. Also, in order to avoid underflow with the often small probabilities involved, we sum the log of the output of the cost functions:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmax}} \left\{ \sum_{i=1}^{M} \log\left[p(\boldsymbol{a}_t \mid s = x_t))\right] \right\} \tag{3.2}$$

3.2 Poisson point likelihood

[11] suggests a process whereby the set of peaks in the frequency spectrum is represented by a Poisson point process for the purposes of automatic music transcription, which was also the topic of a previous 4th year project[1]. Below is a description of how the model and how it can be used for music alignment, based on work done by Leonard Chan last year [2].

Every musical note has a corresponding fundamental frequency (e.g. the A above middle C has a frequency of 440 Hz). The frequency spectrum of a live performance of a note contains the greatest power at the fundamental frequency, but will also contain peaks at multiples of the fundamental frequency. If initially we were to imagine that a peak could occur at any point in the frequency domain, then we could model this as a Poisson point model with a constant intensity function. However, we know that peaks are most likely to occur at multiples of the fundamental frequency, with the probability decreasing as the multiplier increases. It is therefore better modelled by making the Poisson intensity a function of frequency, which results in a non-homogeneous Poisson point model.

This frequency-dependent intensity function was calculated as described in the 4th year project of music transcription [1](page 8), which is reproduced below.

$$\mu_m(k) = \sum_{h=1}^{H} \frac{A}{\sqrt{2\pi h^{2\kappa}}} \exp\left\{ \frac{-\left(f(k) - f_{m,h}\right)^2}{2h^{2\kappa}} \right\} \qquad \mu(k) = \sum_{m=1}^{M} \mu_m(k) + \mu_c$$

$$f_{m,h} = f_{0,j} h \sqrt{1 + B_j h^2}$$

$\mu(k)$ is the intensity function, where k in the index corresponding to frequency $f(k)$, h is the frequency partial and m is the note in a chord. It consists of a narrow Gaussian at each multiple of the fundamental frequency. The constants were set at $\kappa = 0.8$, $A = \sqrt{2\pi}$, $\mu_c = 0.001$ and $B_j = 0.00062$. These were all as suggested in [1], except for B_j, which was increased slightly to improved the performance with the piano music (which is percussive so has high inharmonicity).

Figure 3.1 shows the intensity function for the A above middle C (440 Hz).

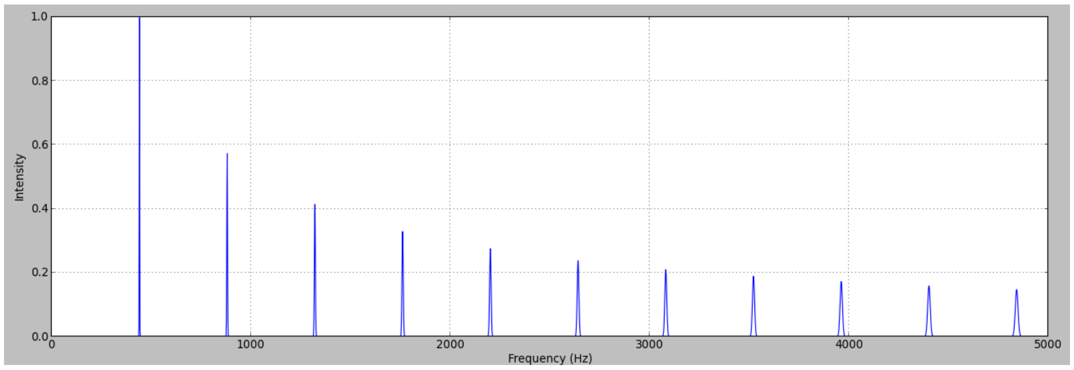


Figure 3.1: The Poisson point intensity function for a fundamental frequency of 440 Hz

For chords (where two or more notes are being played at the same time), the Gaussians for the individual notes are added together.

The probability that the number of peaks N_k in the k^{th} frequency bin is equal to n can be calculated using the Poisson formula, where $\mu(k)$ is the Poisson intensity function as described above:

$$P(N_k = n) = \frac{e^{-\mu(k)} \mu(k)^n}{n!}$$

Because there can only be at most one peak in a frequency bin, this formula reduces to a Bernoulli distribution:

$$P(N_k = n) = (1-n)e^{-\mu(k)} + n(1 - e^{-\mu(k)})$$

In order to detect where the fundamental frequency and multiples thereof lie in the spectrum (the peaks), a threshold function was created (shown as a green dashed line in Figure 3.2), which is created using the median of the surrounding 100 frequency bins, plus a constant of 6 dB. All points in the FFT were found using a simple first-order difference process, and only those peaks above the threshold function were retained (shown as yellow circles in Figure 3.2).

The likelihood function of the frame of audio given the Poisson model for the MIDI state in question can then be calculated thus:

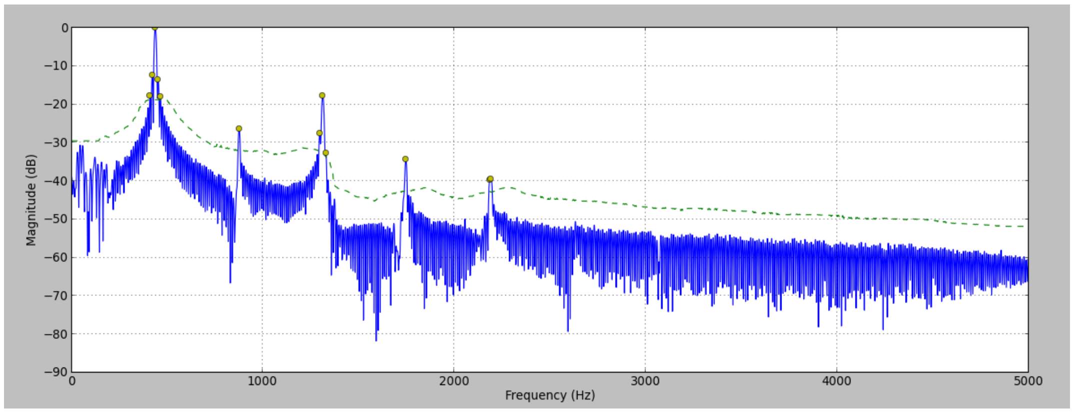


Figure 3.2: A graph showing the FFT of a frame of flute music with the peaks detected

$$P(\mathbf{a}_t|s=x_t) = \prod_{k=0}^{K} \left[(1-a_{tk})e^{-\mu(k)} + a_{tk}(1-e^{-\mu(k)})\right]$$

where and $a_{tk} = 1$ if a peak is present in frequency bin k, and 0 otherwise

In practice, the log likelihood function is used:

$$\log\left[P\left(\mathbf{a}_t|s=x_t\right)\right] = \sum_{k=0}^{K} \log\left[(1-a_{tk})e^{-\mu(k)} + a_{tk}(1-e^{-\mu(k)})\right]$$

3.3 Chromagram likelihood

A chromagram is a representation of a chord in which frequency information is grouped into 12 bins corresponding to the 12 semitones of the musical scale - i.e. specific octave information is discarded. This is an effective descriptor because the multiple frequency components belonging to each note collapse into the same bin, meaning they do not need to be considered individually. The resulting 12-point feature vector approximately describes the musical harmony or of the frame if the music is sufficiently polyphonic. They are frequently used in research into key detection and music retrieval problems due to their simplicity. The implementation described here is based on the method described in [9].

An audio frame a_t is converted into a chromagram by first summing the frequency content into bins centred on the frequencies corresponding to the 128 notes in the MIDI scale. These are split into 12 groups based on the modulus of the MIDI note value with 12, then the bins in each group are summed again to form a 12-point vector. Finally, the vector is normalised in order to remove the effect of intensity differences between frames. Some example chromagrams are depicted in Figure 3.3.

The chromagram for a state s_i is calculated similarly, except for instead of summing frequency bins, a 128-point vector is created where each value is either a 1 or 0, depending on whether that note is present in the state. This vector is then reduced and normalised as before.

The probability $P(a_t \mid s = x_i)$ is taken to be 1 - the Euclidean distance between the feature vectors.

The small size of the feature vector reduces the number of computations involved. However, it discards

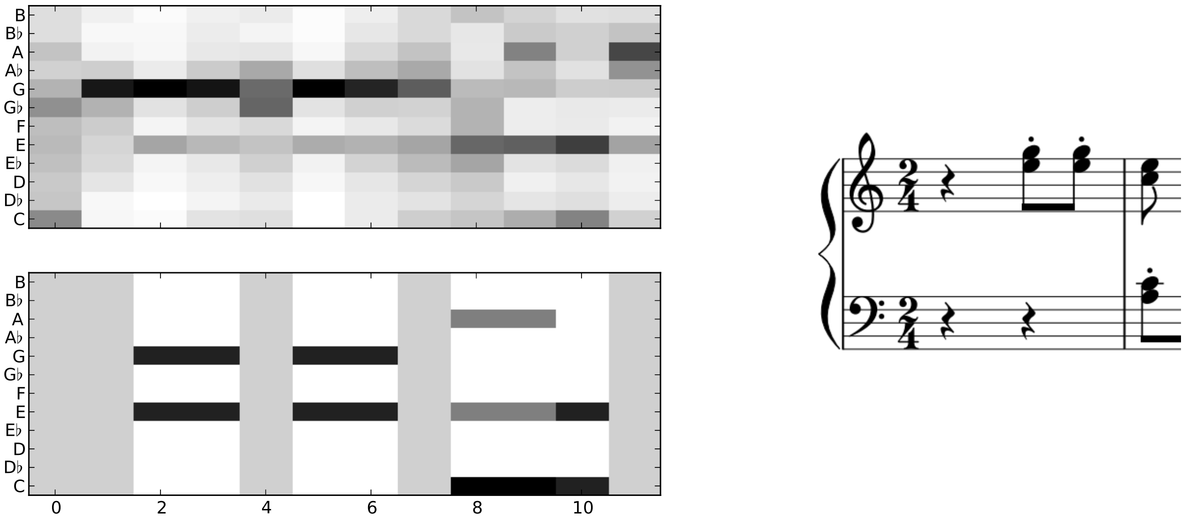


Figure 3.3: The chromagram representation of the audio (top left) and the MIDI file (bottom left) of a Rondo by Mozart, manually aligned. The corresponding musical score is on the right

information about the exact pitches involved which could lead to many false positives, particularly as tonal pieces of music, even relatively complex ones, tend to have a limited range of harmonies and much repetition.

3.4 Peak Structure Distance

The third method to be investigate is a method based on the direct analysis of the frequency content, based on a method named 'Peak Structure Distance' (PSD) [10].

A filter, H_i is created which consists of a multiple pass-bands centred on h harmonic peaks (shown in green in Figure 3.4), the first of which is centred on the fundamental frequency and the subsequent ones on multiples of the fundamental. h was to set 8, the same as in the [10], and where multiple notes are present, h pass-bands are created for each note. This filter was then multiplied by the frequency spectrum of the audio frame to produce a measure of how much of the expected frequency content is present.

In order to prevent a particularly loud frame from unfairly having a greater chance of matching a state, the filtered spectrum was normalised using equation 3.3. y_i is the ith member of the FFT vector for a given frame, and the summations are made only over the range of the minimum and maximum pass frequencies of H_i.

$$P\left(\mathbf{a}_t | s = x_t\right) = PSD(i,t) = \frac{\sum_k H_k y_k}{\sum_k y_k} \tag{3.3}$$

PSD features vectors seems similar to the chromagrams in section 3.3, except that the absolute pitch of the expected notes are taken into account to an extent. However, because all the pass-bands are of equal size, a state that expects a low-pitched note will match a higher note reasonably well, although the inverse is not so likely as the higher frequency components of a low-pitched note will be fairly low energy. Also,

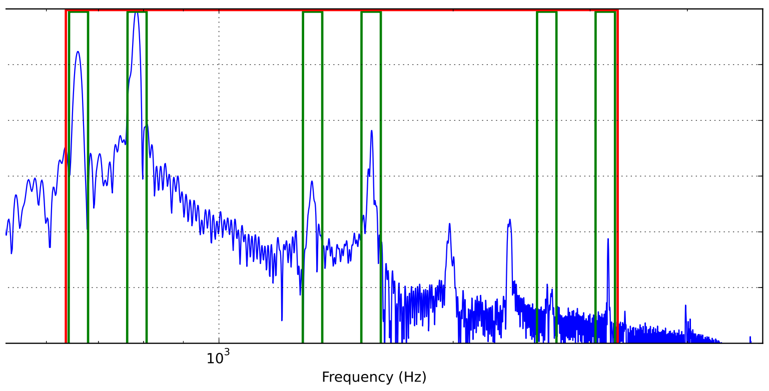


Figure 3.4: The FFT of the frame at time 2 in Figure3.3. The PSD value for this frame is the sum of the frequency content in the green rectangles divided by the sum of the frequency content inside the outer red rectangle.

the normalisation procedure is different - the feature vector is normalised across the relevant range of the frequency spectrum, rather than just based on the frequency components in question. Chromagrams are a measure of the relative power of each note in the 12-note musical scale, whereas PSD attempts to calculate how well the actual frequency spectrum matches the frequency spectrum we expect.

4 Path finding algorithms

4.1 Overview

A choice can be made of one of the methods described in section 3 in order to construct a cost matrix of size N by T, which has a 'cost' (which is proportional to log $p\left(a_t \mid s_i\right)$) for every combination of s_i and a_t. We now wish to efficiently calculate the most probable value of $\mathbf{x} = \{x_1, x_2 \ldots x_T\}$ (where x_t is the state occupied at time t), denoted $\hat{\boldsymbol{x}}$which is equivalent to finding the optimum path through this cost matrix. This path is subject to the following intuitive constraints:

- $x_{t+1} \geq x_t$ - the state at time i+1 must be the same as or greater than the state at time i, or in other words we cannot go backwards through the score.
- $x_{t+1} \leq x_t + 1$ - the state index can only increment by a maximum of one between frames (we assume there are no omissions in the recording).

A naive algorithm might exhaustively consider all possible paths, $\mathbf{x}$. However, this method clearly would not scale well with large values of T and N - better efficiency can be achieved using methods which involve dynamic programming. These use message passing between each phase of the algorithm to take into account the fact that if the globally optimal path passes through (s_i, a_t), then a subset of this path, e.g. from s_1, a_1 to s_p, a_q $(p < i, q < t)$, must also be optimal. In the general case, $O(NT) \simeq O(T^2)$ can be achieved using a message passing algorithm.

4.2 Dynamic Time Warping

'Dynamic time warping' (DTW) is an algorithm which attempts to match the music to the score by taking a 'random walk' through the cost matrix, finding the path from the bottom-left hand corner to the top-right hand corner which, when all the likelihoods along the path are added, is the most least costly (subject to the constraints listed above, such as ensuring that the state index is monotonically increasing).

DTW is a algorithm which starts at s_1, a_1 and calculates the cost to every point in the cost matrix. Formally, the cost of a given path is given by equation :

$$C_p(\mathbf{x}) = \sum_{t=1}^{T} C(x_{t,}, a_t) \tag{4.1}$$

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{argmax}\left\{C_p(\mathbf{x})\right\} = \underset{\mathbf{x}}{argmax}\left\{\sum_{t=1}^{T} C(x_{t,}, a_t)\right\} \tag{4.2}$$

However, dynamic programming can be used to avoid redundant calculations:

$$\phi_i(j) = C(s_i, a_t) + \max\left\{\phi_i\left(t-1\right), \phi_{i-1}\left(t-1\right)\right\} \tag{4.3}$$

Algorithm 3 shows the implementation of this equation.

Once the DTW matrix has been calculated, the best path can be found by starting at DTW[N, M], and tracking back through the matrix following the least costly path.

Algorithm 3 Dynamic time warping

```
Require: C: a cost matrix of size (N, T)
  Create matrix DTW with size (N, T)
  Initialise DTW[1,1] = 0
  for i = 2...N do
    for t = 2...T do
      DTW[i, t] = C[i, t] + min(DTW[i , t-1], DTW[i-1, t-1])
    end for
  end for
```

Algorithm 4 Dynamic time warping - tracing back to find the best path

```
  Initialise $i = N$
  Initialise $j = M$
  Initialise path = vector of length $M$
  while $i > 0$ do
    if DTW[$i$-1, $j$-1] < DTW[$i$, $j$-1] then
      $i = i - 1$
    end if
    $j = -1$
    path[j] = $i$
  end while
```

A major disadvantage of DTW is that the expected duration of each state is not taken into account, only the notes the state contains. An extreme example of a possible DTW path might stay in the same state for almost all the audio frames, which one would consider extremely unlikely no matter what the piece of music is. Constrained DTW (see section 5.2) prevents such paths from being produced to an extent, but the fact that the valuable duration information is unused may still affect the performance of the alignment algorithm.

4.3 Hidden Markov Models and the Viterbi algorithm

The sequence of states describing the musical score readily lends itself to a Hidden Markov Model, where each state in the score becomes one state in a Markov chain. The 'outputs' of the HMM are the feature vectors as derived from the audio frames using one of the methods described in section 3. The transition probabilities can be modelled to represent the time spent in each state - one way of doing this is presented in Figure 4.1 and explained below.

In order to satisfy the constraints detailed in 4.1, the HMM must be what is known as a 'left-right' HMM. In such a HMM, the state transition probability to past states is zero. Our model has the extra constraints that only transitions back to the same state or to the state immediately after are allowed.

One of the properties of a Markov chain is that the probability of being in the current state only depends on the previous state. We therefore require a 'memoryless' probability distribution. A homogeneous Poisson point process fits this description, as suggested by [2]. The distribution of the time between state transitions is given by:

$$p\left(d\right) = \lambda_i e^{-\lambda_i d}$$

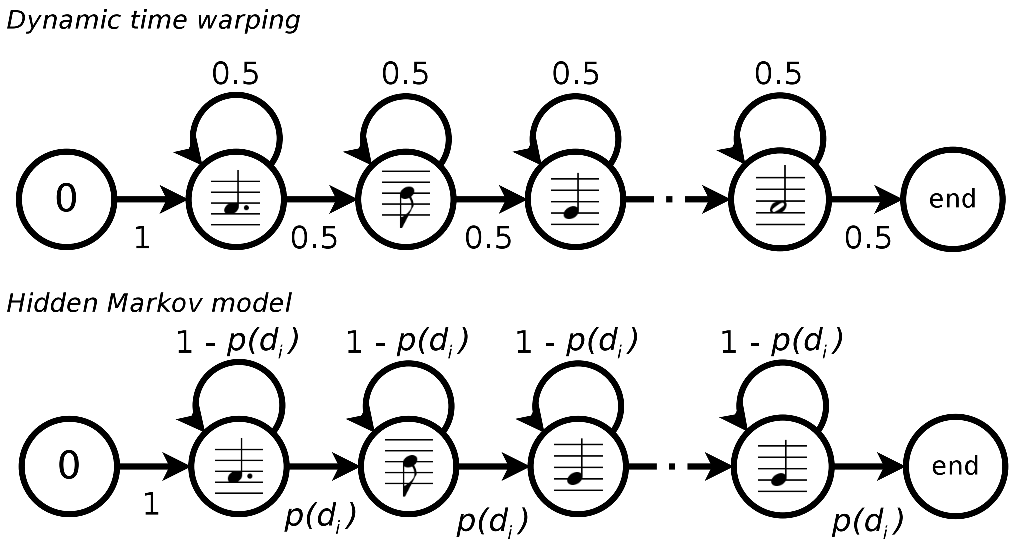


Figure 4.1: A comparison of dynamic time warping with a hidden Markov model, showing the probability of each transition

We want the probability of transitioning to the next state during a frame, which is calculated thus:

$$\begin{aligned} p\left(0 \leq d \leq l\right) &= \int_0^l p(d)dd = \int_0^l \lambda_i e^{-\lambda_i d} dd \\ &= 1 - e^{-\lambda_i l} \end{aligned}$$

where l is the duration (in seconds) of an audio frame, which can easily be derived from the sample rate and the number of samples in a frame. To calculate the Poisson intensity, λ_i, we use the fact that the expected value of the distribution must equal d_i.

$$\begin{aligned} E\left[d\right] &= \int d p\left(d\right) dd = \int d \lambda_i e^{-\lambda_i d} dd \\ &= \frac{1}{\lambda_i} \end{aligned}$$

Using the fact that the expected value of the duration $= d_i$:

$$\begin{aligned} \lambda_i &= \frac{1}{d_i} \\ p\left(x_t = i | x_{t-1} = i - 1\right) = p\left(0 \leq d \leq l\right) &= 1 - e^{-\frac{l}{d_i}} \qquad (4.4) \\ p\left(x_t = i | x_{t-1} = i\right) &= e^{-\frac{l}{d_i}} \qquad (4.5) \end{aligned}$$

Equations 4.4 and 4.5 can now be used to construct the HMM transition matrix.

The Viterbi algorithm is a standard dynamic programming algorithm for finding the most likely path through a HMM, given a sequence of outputs (which are in this case the audio feature vectors). The probability of being in state i at time j, following the best path, is given by:

$$\begin{aligned}\phi_i(t) &= \underset{j}{max}\left\{\phi_j\left(t-1\right)p\left(x_t=i|x_{t-1}=j\right)p\left(a_t|x_t=i_i\right)\right\} \\ &= p\left(a_t|x_t=i\right)\max_j\left\{\phi_j\left(t-1\right)p\left(x_t=i|x_{t-1}=j\right)\right\} \\ \phi_i(1) &= \begin{cases}1 & i=1\\ 0 & i>1\end{cases}\end{aligned} \tag{4.6}$$

Because the state transition matrix, $p\left(x_j=i|x_{j-1}\right)$, is sparse - only transitions to the same or the following state are allowed - we only need to consider two previous values of j when evaluating the max function:

$$\phi_i(j)=p\left(a_t|x_t=i\right)\max\left\{\phi_i\left(t-1\right)p\left(x_t=i|x_{t-1}=i\right),\phi_{i-1}\left(t-1\right)p\left(x_t=i|x_{t-1}=i-1\right)\right\} \tag{4.7}$$

Algorithm 5 shows the algorithm used to evaluate this expression. Note that the log domain is used in order to prevent underflow.

Algorithm 5 The Viterbi algorithm for a left-right HMM

```
Require: trans: the state transition matrix
Require: C: the cost/log likelihood matrix
  Initialise V as matrix of size (N, T)
  for $i$ = 1...N do
    V[0, i] = $-\infty$
  end for
  for $t$ = 1...T do
    for i = 1...N do
      maxprob = $-\infty$
      maxstate = null
      for j = i, i-1 do
        prob = V[t-1][j] + log(trans[j][i])
        if prob >= maxprob then
          maxprob = prob
          maxstate = j
        end if
      end for
      max_prob += $C(s_{maxstate}, a_t)$
      V[t][i] = max_prob
    end for
  end for
```

The best path is then calculated by tracing back through the matrix V as with DTW.

A comparison can be made between the Viterbi algorithm and DTW. If we were to fix all state transition probabilities - $p\left(x_t=s_i|x_{t-1}=s_i\right)$ and $p\left(x_t=s_i|x_{t-1}=s_{i-1}\right)$ at 0.5 (i.e. make the HMM just as likely to transition to the next state as it is to stay in the same state), then the state transition probability term in equation 4.7 is a constant and can therefore be removed from the max operation. If we move to the log domain and replace the output probability with the non-normalised cost-function then we get:

$$\phi_i(j) = C(s_i, a_t) + \max\left\{\phi_i\left(t-1\right), \phi_{i-1}\left(t-1\right)\right\} \tag{4.8}$$

It can be seen that this is exactly the same as equation 4.3. This proves that DTW, in which the message-passing function depends only on the frame and state in question, can be seen as a specific case of the Viterbi algorithm, in which the cost also depends on the transition probability into the frame/state.

4.4 Hidden Semi-Markov Models

While using a standard HMM provides a simple way of taking the time we expect to spend in a particular state into account, the distribution of durations this produces is exponential, which is not ideal.

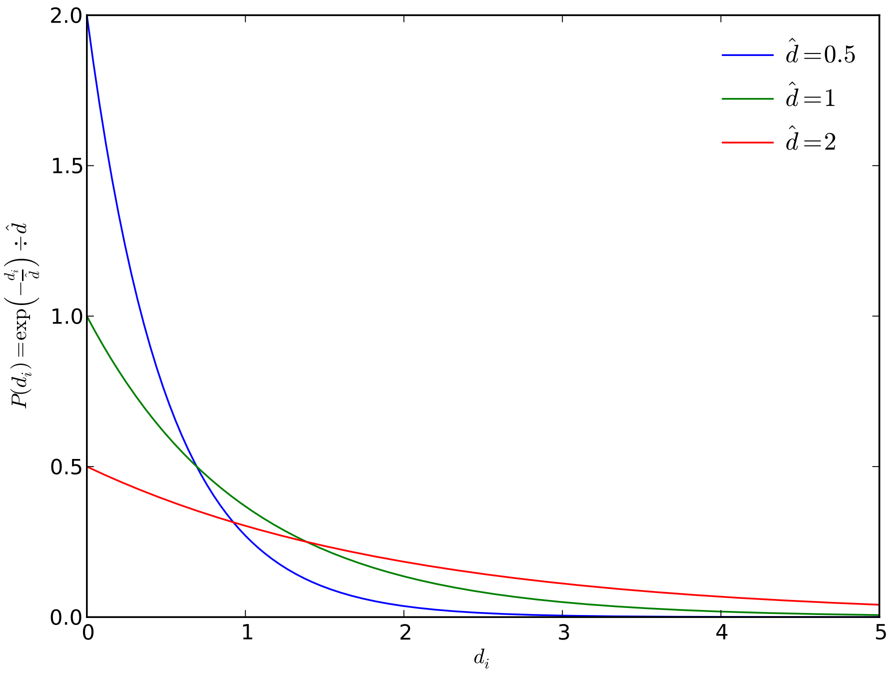


Figure 4.2: Exponentially-distributed durations

Figure 4.2 shows the probability distribution of the time before a transition to the next state. It can be seen that while the expected value of the duration is correct, the distribution does not match the the real distribution of durations in a recording. It is very right-skewed - the mode of the exponential distribution is always 0. It does however have the property of being memoryless - $P\left(d > x + a \mid d > x\right) = P\left(d > a\right)$ - ensuring that the probability of transitioning the next state is independent of how long has been spent in that state, which is required for the standard HMM.

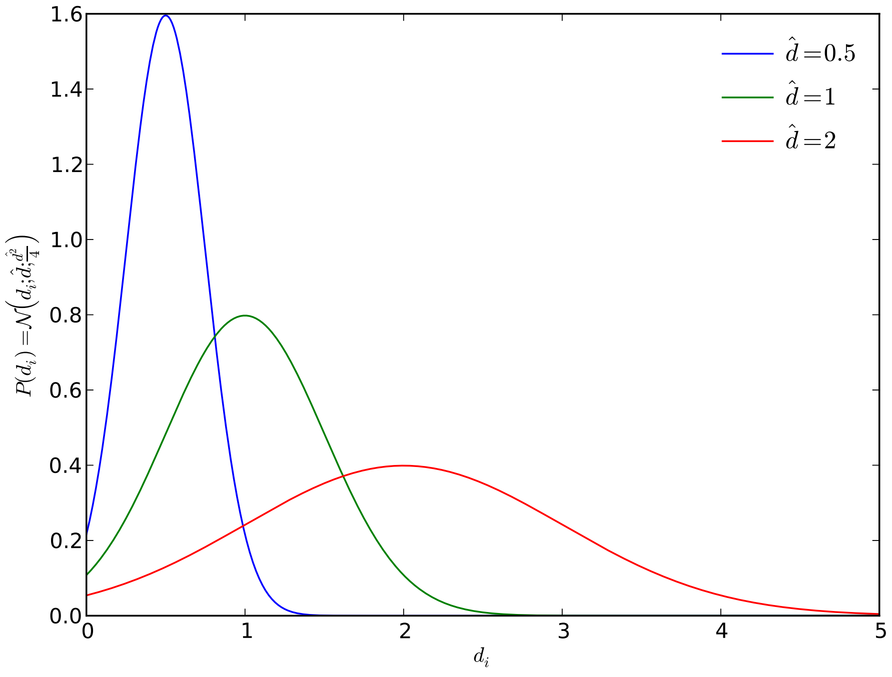


Figure 4.3: Normally-distributed durations

Figure 4.3 shows the distribution of lengths one might expect if one were asked to play a note with nominal length $\hat{d}$. The lengths are normally distributed about the expected length of the note, with a standard deviation of a half of the expected duration. The resultant model is known as a hidden semi-Markov model (HSMM); 'semi' because the process is only Markovian when transitioning between states, not in between. HSMMs were first introduced for signal processing tasks in 1980 [7], and a relatively recent paper [3] uses a HSMM for music alignment. The HSMM used, which has a constant state transition probability of 1, is depicted in Figure 4.4.

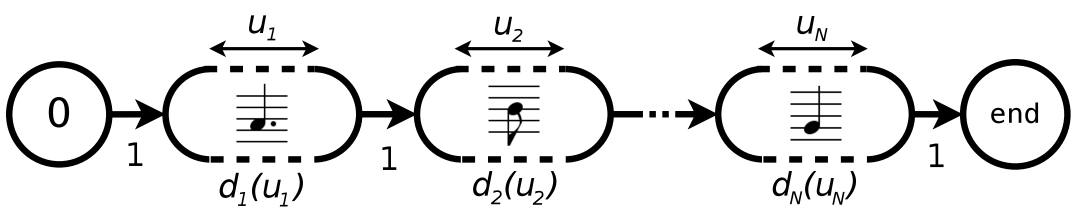


Figure 4.4: A diagram showing the construction of a hidden semi-Markov model in which only transitions to the following state are possible. u_j is the number of frames spent in the state and $d_j(u_j)$ is the probability of the duration of the state being u_j

Because a HSMM does not obey the memoryless property, we cannot model the transitions through a constant state transition matrix, so a term is required in the Viterbi probability to take into account the probability of being in that state for a certain amount of time. The probability distribution of the duration of state j is the survival distribution ($1 - F(u)$, where $F(u)$ is the cumulative distribution) of the probability density function of the duration. Because we expect u to be normally distributed, $d_j(u)$ is the normal survival function:

$$\begin{aligned} d_j(u) &= p\left(x_{t-u+1} = j \ldots x_{t-1} = j, x_t = j \mid x_{t-u} \neq j\right) \\ &= p(\hat{u_i} > u_i) \\ &= 1 - \Phi\left(u; \hat{d}; \frac{\hat{d^2}}{4}\right) = Q\left(u; \hat{d}; \frac{\hat{d^2}}{4}\right) \end{aligned}$$

Q is the ‘Q-function’, or $1-$ the cumulative Gaussian distribution. It cannot be easily expressed symbolically, but SciPy provides a function that can be used to evaluate it.

Below is a derivation (based on the derivation in the appendix of [3]) of $\phi_j(t)$, which is the probability of seeing outputs $o_1 \ldots o_t$ and the corresponding states being $x_1 \ldots x_{t-1}, x_t$. $\phi_j(t)$ depends on $\phi_i(t-1)$, meaning that as before dynamic programming can be used to decrease the algorithm’s complexity.

$$\begin{aligned} \phi_j(t) &= \max_{x_1 \ldots x_{t-1}} p\left(x_1 \ldots x_{t-1}, x_t = j, o_1 \ldots o_t\right) \\ &= \max_u \max_{i, i \neq j} p\left(x_1 \ldots x_{t-u-1}, x_{t-u} = i, x_{t-u+1} = j \ldots x_t = j, o_1 \ldots o_t\right) \\ &= \max_u \max_{i, i \neq j} \{p\left(x_1 \ldots x_{t-u}, o_1 \ldots o_{t-u}\right) p\left(x_{t-u+1} = j \mid x_{t-u} = i\right) \\ &\qquad p\left(x_{t-u+1} = j \ldots x_t = j \mid x_{t-u} \neq j\right) p\left(o_{t-u+1} \ldots o_t \mid x_{t-u+1} \ldots x_t\right)\} \\ &= \max_u \max_{i, i \neq j} \{\phi_i(t-u) p\left(x_{t-u+1} = j \mid x_{t-u} = i\right) d_j\left(u\right) p\left(o_{t-u+1} \ldots o_t \mid x_{t-u+1} \ldots x_t\right)\} \\ &= \max_u \left[d_j\left(u\right) \left\{ \prod_{v=0}^{u-1} p\left(o_{t-v} \mid x_{t-v}\right) \right\} \max_{i, i \neq j} \left(p\left(x_{t-u+1} = j \mid x_{t-u} = i\right) \phi_i(t-u)\right) \right] \end{aligned} \tag{4.9}$$

4.9 is the expression for a general HSMM. However because ours is a left-right HSMM, and only transitions to the next state (and no self-transitions) are allowed, this expression can be simplified. The state transition matrix is now defined by:

$$p\left(x_{t-u+1} = j \mid x_{t-u} = i\right) = \begin{cases} 1, & j = i + 1 \\ 0, & \text{otherwise} \end{cases}$$

4.9 therefore simplifies to equation 4.10. u has been limited so that u is always less than t.

$$\phi_j(t) = \max_{u \in [1,t]} \left[\phi_{j-1}(t-u) d_j\left(u\right) \prod_{v=0}^{u-1} p\left(o_{t-v} \mid x_{t-v}\right) \right] \tag{4.10}$$

This equation can be interpreted intuitively. For each value of t and j, it searches back through every previous state and calculates the probability that, having been in that state, it sees the next u outputs and also takes into account the probability of being in state j after u frames have passed. The overall global maximum is the join probability of observing the outputs (given the state sequence) and of the time spent in each state (given the expected state durations). While I only considered the normal distribution, $d_j(u)$ could be modified to model other distributions.

As before, the ideal path is calculated by tracing back through the matrix, and the expression is evaluated in the log domain to avoid underflow. Also, in practice u can be limited to the much less than t , - say, 5 s - as it is unlikely that a state will last longer than this, and this reduces the number of computations required.

$$\phi_j(t) = \max_{u \in [1,t]} \left[\phi_{j-1}(t-u) + \log d_j(u) + \sum_{v=0}^{u-1} \log p(o_{t-v} \mid x_{t-v}) \right] \tag{4.11}$$

It can be seen 4.6, the standard HMM model, is a special case of equation 4.9, where u is constrained to always equal 1, and $d_j(u) = 1$. Algorithm 6 shows the implementation of this formula.

Algorithm 6 The path-finding algorithm for a hidden semi-Markov model

```
Require: C: the cost/log likelihood matrix
  Initialise V as matrix of size (N, T) filled with zeros
  for $t$ = 1...T do
    for i = 1...N do
      maxprob = $-\infty$
      maxstate = null
      for u in $1 \dots t$ do
        if $y > 0$ then
          prob = V[t-u, y-1]
        else
          prob = 0.0
        end if
        prob += $d_y(u)$
        for v in $0 \dots u-1$ do
          prob += $C(s_y, a_{t-v})$
        end for
        if prob >= maxprob then
          maxprob = prob
          maxstate = j
        end if
      end for
      V[t][i] = max_prob
    end for
  end for
```

5 Implementation details

5.1 Overview

In order to enable the testing of multiple algorithms on the same piece of data, a framework was created in which the likelihood function and a path-finding algorithm could be easily swapped out . The framework outputted the path as a simple Python list contains the state index at each time frame, which could then be passed to one of the visualisation methods. Below is an overview of the main classes used.

AudioFile Its constructor takes a file-name of a WAVE file, which opens the file and reads the data into memory. It contains methods to chunk the file into *FreqFrame* objects

FreqFrame A container for the FFT of a single frame in the score. It contains methods to aid grouping into frequency bins for the PSD and Chromagram methods, and methods to generate threshold function and detect peaks for the Poisson point method.

Score Its constructor takes the file-name of a MIDI file, which performs all the pre-processing described in section 2.1. It contains an array of *State* objects.

State A single 'state' in the score, as described in section 2.1. It contains fields for the notes, its real duration and the number of beats it lasts in the score. It does not store note offset information directly as it could easily become inconsistent; instead this is calculated by the *Score* when needed by summing durations.

LikelihoodProcessor Base class for other classes able to calculate a likelihood given an audio frame and a state in the score. It contains a *calculate()* method, which is overridden by child classes to implement the different likelihood functions, which takes two arguments: an audio frame and a score state.

PoissonLikelihoodProcessor, ChromaLikelihoodProcessor, PSDLikelihoodProcessor Child classes of *LikelihoodProcessor* which implement the respective likelihood algorithms.

PathProcessor Base class for other classes able to calculate a likely path given through a cost matrix and the expected duration to spend in each state. It contains a *calculate()* method, which is overridden by child classes to implement the different path-finding algorithms, which takes a single argument - the cost matrix.

DTWLikelihoodProcessor, HMMLikelihoodProcessor, HSMMLikelihoodProcessor Child classes of *PathProcessor* which implement the respective path-finding algorithms.

AlignmentProcessor The class which ties the above together. Its constructor takes two arguments - the *LikelihoodProcessor* and *PathProcessor* to use. Its *align()* method takes 2 arguments - a *Score* object and an *AudioFile* object. It returns an *Alignment* object

Alignment A class which is returned by the *align()* method of the *AlignmentProcessor* which encapsulates data about the result of the alignment process. It can be used directly to generate graphics using *matplotlib*, serialized to JSON to send to the score-following interface (see 5.3), or serialized to disc so the results can be revisited later without having to recalculate it.

5.2 Optimisations

With the Poisson point likelihood method described in section 3.2, the calculation of the intensity functions was found to be a major bottleneck, due to the number of exponential calculations involved. A factory class was

Both DTW and the Viterbi algorithm scale as $O(NT)$, and because N is roughly proportional to T (the longer the score, the longer the recording), this means that the computational cost is $O(T^2)$, which is not ideal.

We can use the fact that we expect the path to stay close to the diagonal of the cost matrix to optimise the algorithm, as this means we only need to consider frame/states that are within a certain distance of the diagonal of the cost matrix. The search width, w, can either be set to be a constant, or a variable that is proportional to the number of states. If w is a constant then the optimum path can be found in $O(T)$ operations; otherwise it will still be $O(T^2)$, but with a smaller constant.

5.3 Visualisation of alignment

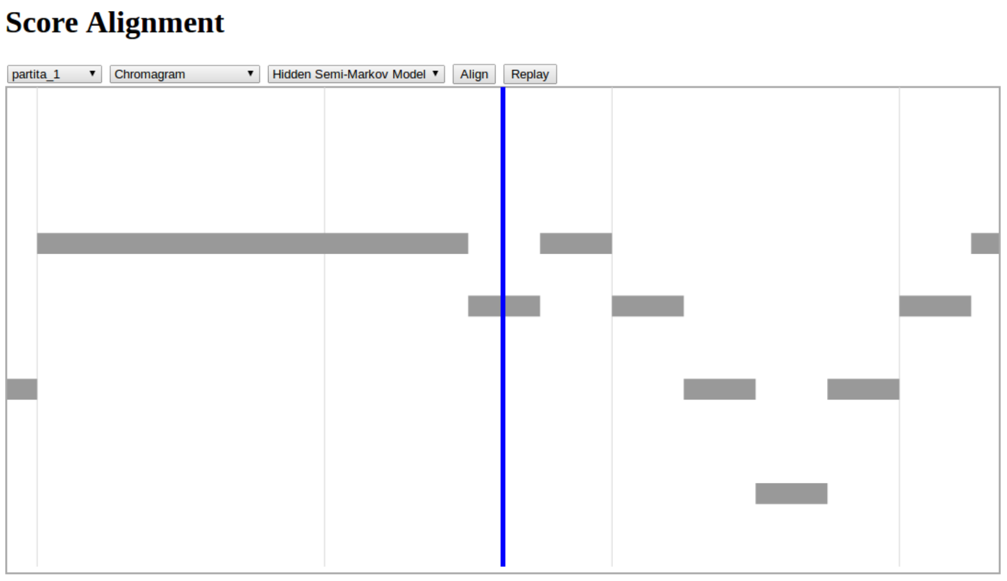


Figure 5.1: Score-following visualisation tool

A novel visualisation tool was developed for evaluating the quality of an alignment, by playing back the recording while simultaneously using a given alignment to move an cursor along a visual representation of the score.

As the standard musical score is the most intuitive written representation of a piece of music to a trained musician, I first went about creating a score-following tool which animates a cursor along such a score. However, musical scores have a very complex layout - most notably, the horizontal spacing is non-linear. For example, a bar with many short notes takes up much more space than a bar with a single semi-breve. Also, the correct placement of the cursor would be ambiguous - is a vertical cursor aligned with the onset of a crotchet when it is level with the stem or the centre of the note-head? There exists a number of Python libraries for typesetting scores (Lilypond [1], Abjad [2]), but none of them provide the necessary programming 'hooks' to discover the pixel coordinates of individual notes. The process of typesetting a musical score manually is using a graphics library is non-trivial and would have involved investing major effort in what is only a secondary part of the project.

I therefore decided to use the 'piano roll' notation. This consists of a linear-time x-axis and the musical pitch along the y-axis. All notes that are played in the score have a corresponding rectangle on the graph which starts and end at the appropriate times on the time-line. The name 'piano roll' comes from the storage medium used for old-fashioned automatic mechanical pianos, which used a similar layout except that holes were punched in the location of the notes. It also has the advantage that is it is interpretable by non-musicians.

HTML5 and Javascript were used to write the front end of the visualiser, and the Django [3] web tool-kit written in Python was used to plug the front end into the alignment framework detailed in 5.1. The score and cursor were manually drawn onto an SVG (scalable vector graphics) canvas and the HTML5 Audio API was used to accurately control the audio playback. Three simple server endpoints were written to send data to the client: one to stream an audio file to the browser, one to convert the MIDI file into a format that can easily be interpreted in Javascript and a third which provides the actual alignment data. The alignment data provided consists of a list of sync points, and the playback Javascript code interpolates between these when the audio is playing. Three drop-down boxes allow the user to select from a sample of recordings and to pick the likelihood and path-finding methods to use.

A caching system was created, whereby alignments results (which take a few minutes to calculate with the current system) can be serialized and stored to disk and retrieved at a later date. When a user requests an alignment, it is served directly from the cache if it exists, otherwise it is calculated.

A screen-shot of the tool can be seen in Figure 5.1, and a demo can be found online at http://pah58.user.srcf.net/scorealign/. It proved very useful for qualitatively analysing the output of the alignment algorithms because errors can be detected intuitively without reference to a record of the correct alignment, which may or may not exist.

5.4 Measurement of alignment accuracy

Measuring the accuracy of an alignment algorithm precisely requires a record of the 'ground truth' alignment, which may be manually transcribed or it might be the source data for the recording if it is synthesised or otherwise automatically generated.

I will evaluate the effectiveness of the algorithms using metrics described in an IRCAM paper which attempts to define a standard set of metrics for evaluating score alignment procedures [4]. Some metrics described are only of relevance to real-time score following systems and others are not useful because the methods I've used do not take into account extra or omitted notes. The metrics that I will use are:

misalign rate p_e the percentage of events with an absolute error $|e_i|$ greater than or equal to 0.3 s. This

[1] http://www.lilypond.org/
[2] http://www.projectabjad.org/
[3] https://www.djangoproject.com/

attempts to articulate the percentage of errors that are completely incorrect, rather than those that slightly miss the exact onset of a note.

average imprecision μ_e the average value of all events than are exact or closely aligned (absolute errors $|e_i|$ less than 0.3 s)

error dispersion σ_e the standard deviation of all events than are exact or closely aligned (absolute errors $|e_i|$ less than 0.3 s)

The above values will be calculated for each piece tested for each combination likelihood and path-finding method. Also, the CPU time taken by each algorithm will be logged so that the costs of computation can be analysed.

6 Results

The algorithms were tested with a number of different source materials, but I have chosen 4 to present for analysis. Two of the factors that affect alignment accuracy are the degree of polyphony and the number of short notes present (which is characterised in the table below by the 'state density' - the average number of states per second in the score), and these have been chosen to represent contrasts of these factors.

- *Sarbande* from *Partita in A minor, BWV 1013* (J.S. Bach) - Flute
- *Bourrée anglaise* from *Partita in A minor, BWV 1013* (J.S. Bach) - Flute
- *Rondo* from *Piano Sonata No.16 in C major*, K.545 (Mozart) - Piano
- *Clair de Lune* (Debussy) - Piano

The two flute pieces are, by their nature, monophonic. The *Sarabande* should be the easiest test for the system as it only contains relatively long notes, and the Rondo the most challenging as it is fast-moving with notes with a short duration.

The flute music is taken from the IMSLP/Petrucci online music library [1], and consists of a live recording and a MIDI file manually created from the originally musical score. The piano music is taken from the MAPS database [16], which is created by recording a real piano being mechanically controlled by a computer. All the music is released under the Creative Commons license [2]. $20 - 30$ s excerpts from beginning of each track were taken, as the complexity of processing the full tracks is prohibitive for the system as it stands (possible techniques for reducing the algorithm from $O(T^2)$ to nearer $O(T)$ are discussed in section 8.2).

The likelihood methods (as discussed in section 3) are the Poisson point model, chromagram and peak structure distance. The path-finding methods (as discussed in section 4) are dynamic time warping (DTW), hidden Markov model (HMM) and hidden semi-Markov model (HSMM). The statistical quantities calculated for each alignment are explained in section 5.4.

Statistics

Sample	Instrument	Length (s)	Number of frames, T	Number of states, N	State density (s^{-1})
Sarabande	flute	28.9	311	66	2.28
Bourée	flute	23.9	257	113	4.73
Rondo	piano	20.5	221	151	7.36
Clair de lune	piano	30.6	329	60	1.96

Sarabande

Misalign rate, p_e (%)

	Poisson point	Chromagram	PSD
DTW	3.03	0.00	0.0
HMM	3.03	0.00	0.0
HSMM	98.48	1.52	3.03

Average imprecision, μ_e (s)

	Poisson point	Chromagram	PSD
DTW	0.0842	0.0253	0.0324
HMM	0.0842	0.0239	0.0324
HSMM	n/a	0.0043	0.0377

[1]http://imslp.org/
[2]http://creativecommons.org/licenses/by-nc-nd/3.0/legalcode

Error dispersion σ_e (s)

	Poisson point	Chromagram	PSD
DTW	0.0842	0.0253	0.0524
HMM	0.0842	0.0466	0.0524
HSMM	n/a	0.0195	0.0485

Execution time (s) 28.9

	Poisson point	Chromagram	PSD
DTW	87.38	43.04	41.23
HMM	87.34	43.03	41.52
HSMM	162.97	121.23	120.88

Bourée

Misalign rate, p_e (%)

	Poisson point	Chromagram	PSD
DTW	7.96	0.88	0.0
HMM	7.96	0.00	0.0
HSMM	99.12	2.65	2.65

Average imprecision, μ_e (s)

	Poisson point	Chromagram	PSD
DTW	0.0634	0.0141	0.0238
HMM	0.0616	0.0164	0.0222
HSMM	n/a	0.0076	0.0439

Error dispersion σ_e (s)

	Poisson point	Chromagram	PSD
DTW	0.0807	0.0377	0.0491
HMM	0.0811	0.0164	0.0467
HSMM	n/a	0.0357	0.0799

Execution time (s) 23.9

	Poisson point	Chromagram	PSD
DTW	106.45	61.20	62.02
HMM	107.70	61.99	60.65
HSMM	217.14	171.53	171.36

Rondo

Misalign rate, p_e (%)

	Poisson point	Chromagram	PSD
DTW	0.00	0.00	2.65
HMM	1.32	1.32	3.97
HSMM	98.68	0.00	0.00

Average imprecision, μ_e (s) 20.5

	Poisson point	Chromagram	PSD
DTW	0.0431	0.0289	0.0594
HMM	0.0411	0.0318	0.0474
HSMM	0.0613	0.0111	0.0264

Error dispersion σ_e (s)

	Poisson point	Chromagram	PSD
DTW	0.0593	0.0468	0.0693
HMM	0.0583	0.0597	0.0673
HSMM	0.0929	0.0224	0.0559

Execution time (s) 20.5

	Poisson point	Chromagram	PSD
DTW	112.24	68.92	67.7
HMM	122.95	69.70	66.96
HSMM	236.49	191.20	188.26

Clair de Lune

Misalign rate, p_e (%)

	Poisson point	Chromagram	PSD
DTW	20.0	18.33	50.0
HMM	20.0	16.67	43.33
HSMM	96.67	11.67	58.33

Average imprecision, μ_e (s)

	Poisson point	Chromagram	PSD
DTW	0.0793	0.0512	0.0774
HMM	0.0793	0.0353	0.0628
HSMM	0.0464	0.0315	0.0892

Error dispersion σ_e (s)

	Poisson point	Chromagram	PSD
DTW	0.0889	0.0775	0.0900
HMM	0.0889	0.0641	0.0834
HSMM	0.0464	0.0697	0.0928

Execution time (s) 30.6

	Poisson point	Chromagram	PSD
DTW	91.34	43.49	43.20
HMM	92.01	43.85	68.64
HSMM	169.63	120.40	120.05

Overall (averaged values)

Below are the statistics averaged over the four pieces of music tested, and and again averaged per algorithm. The overall averages would ideally give an idea of the overall effectiveness of each likelihood and path-finding algorithm, but they may not be indicative because of the small sample size.

Misalign rate, p_e (%)

	Poisson point	Chromagram	PSD	Average
DTW	7.75	4.80	13.16	8.57
HMM	8.08	4.50	11.83	24.41
HSMM	98.24	3.96	16.00	39.40
Average	38.02	4.42	13.66	

Average imprecision, μ_e (s)

	Poisson point	Chromagram	PSD	Average
DTW	0.0675	0.0299	0.0483	0.0486
HMM	0.0666	0.0269	0.0412	0.0449
HSMM	0.0539	0.0136	0.0493	0.0390
Average	0.0626	0.0234	0.0462	

Error dispersion σ_e (s)

	Poisson point	Chromagram	PSD	Average
DTW	0.0783	0.0468	0.0652	0.0634
HMM	0.0781	0.0467	0.0625	0.0624
HSMM	0.0697	0.0368	0.0693	0.0586
Average	0.0754	0.0434	0.657	

Execution time per second of music

	Poisson point	Chromagram	PSD	Average
DTW	3.984	2.208	2.184	2.792
HMM	4.133	2.229	2.371	2.911
HSMM	7.951	6.158	6.115	6.741
Average	5.356	3.532	3.557	

Figures through show a selection of trellis graphs from the tests carried out. The green colour map behind each graph is a map of the likelihood function - the darker the colour the more likely the state and audio frame at the corresponding coordinates are related. It can be seen that the paths generally follow this maximum-likelihood path.

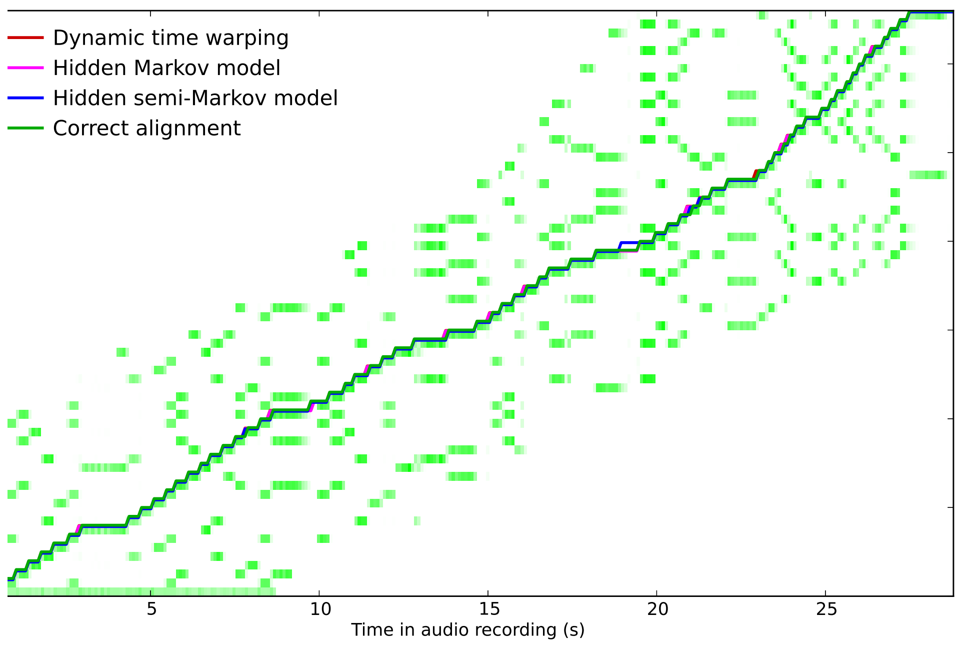


Figure 6.1: The almost perfect chromagram alignment trellises for *Sarabande*

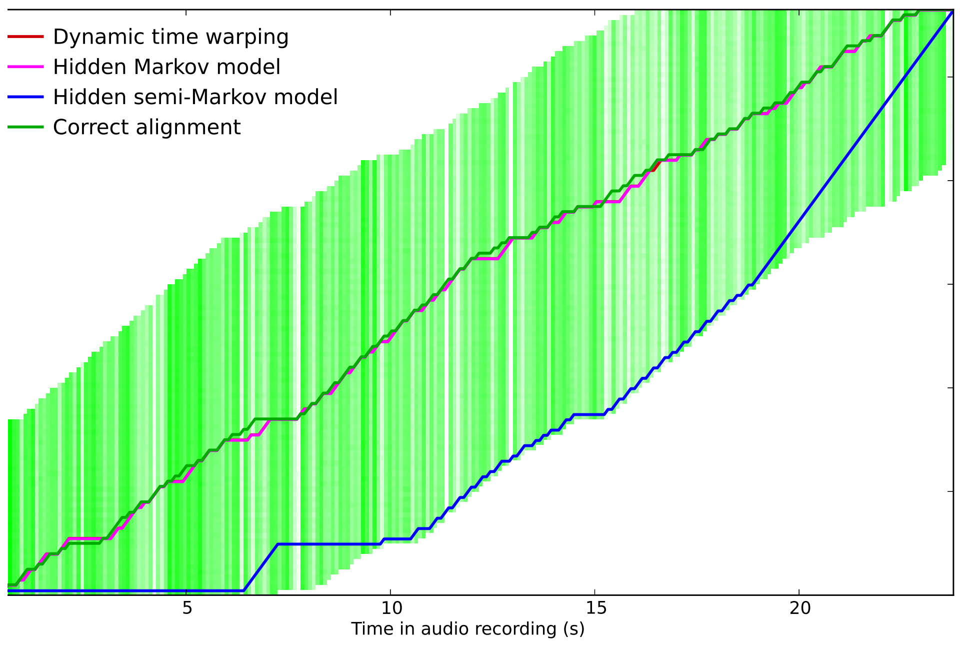


Figure 6.2: Poisson point alignment trellises for *Bourée*

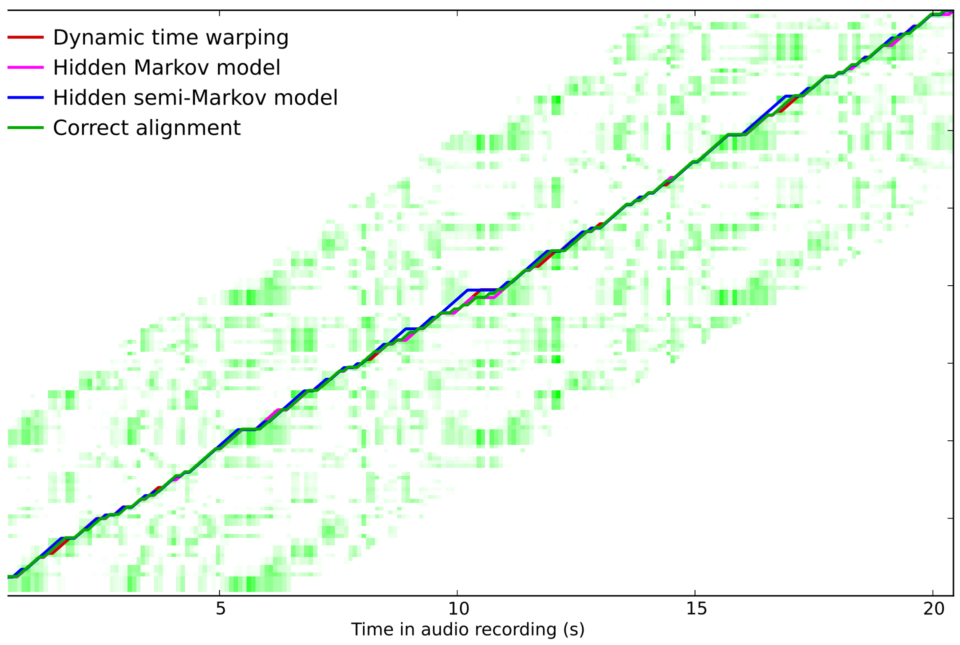


Figure 6.3: Peak structure distance (PSD) alignment trellises for *Rondo*

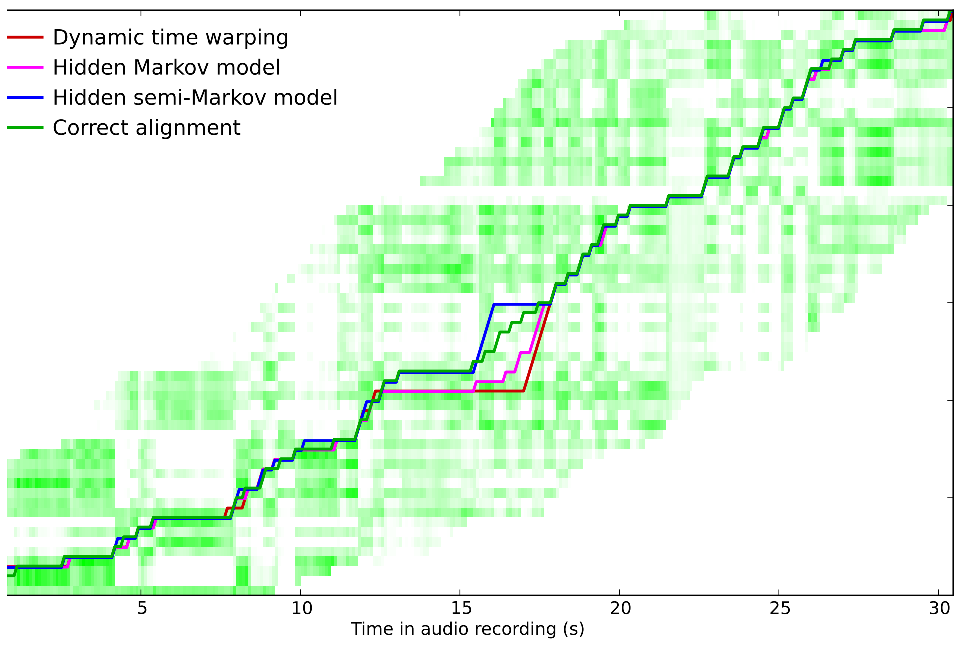


Figure 6.4: Chromagram alignment trellises for *Clair de Lune*

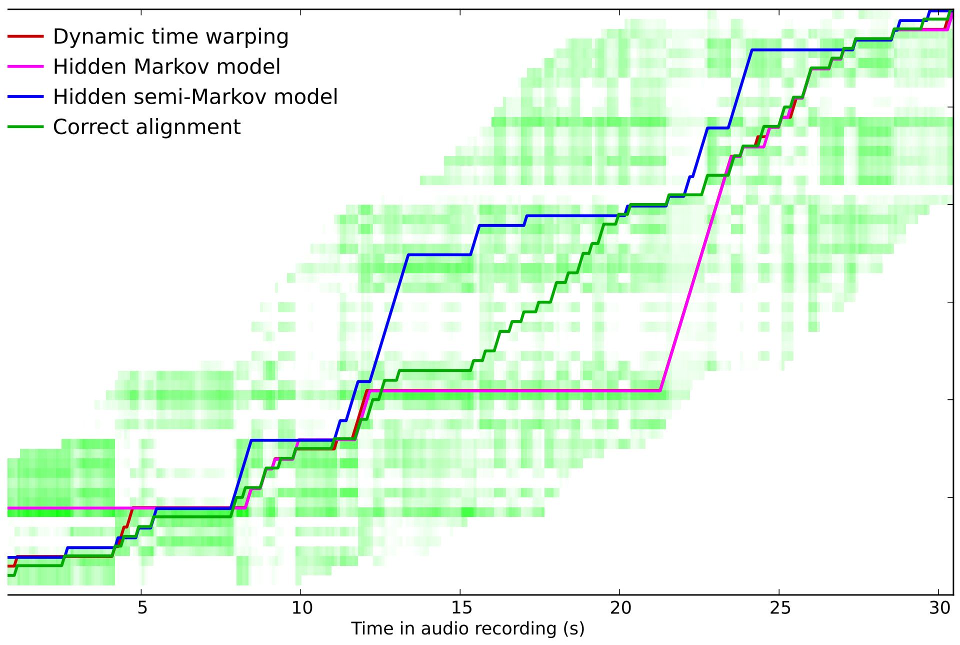


Figure 6.5: Peak structure distance (PSD) alignment trellises for *Clair de Lune*

7 Discussion

Overall, very good accuracy was achieved in most cases, with nearly all combinations of likelihood and path-finding methods achieving a misalign rate of $p_e$of 0% in certain circumstances (meaning that all alignment errors in such cases are less than 0.3 s). The chromagram proved to be the most consistently accurate, with an overall average accuracy rate of about 4.5%, dropping to about 4% with the HSMM path-finding method.

The best-performing piece of music overall was the *Sarabande*, as expected. The alignment based on a chromagram model is shown in Figure 6.1. It can be seen that the most significant alignment error occurs at around 19 s, the area around which is showed zoomed in in Figure 7.1. It can be seen that the DTW and HMM paths both achieve perfect alignment in this period, however, the HSMM path skips to the next state too early. The results show that the HSMM has the lowest overall average imprecision and error dispersion, but in isolated incidences it can be seen that the reliance on the state duration data can cause unwanted errors. Here, the flautist has inserted a rest that is not marked in the score in order to take a breath, and in the absence of any strong likelihood data in that period, the HSMM model has opted to jump to the next state soon after the expected duration has elapsed (for comparison, states 38 and 39 both have the same nominal duration), rather than staying in state 39 until the onset of the next note. This shows that while the duration information improves alignment on the whole, over-reliance on this data leads to unwanted errors in cases where the performer deviates from the notes in the score, or in styles of performance where there is significant tempo fluctuation (a phenomenon referred to as 'rubato' by musicians).

Arguably, because of the deviation from the score in this instance, neither paths taken are technically correct as the performance has an extra state inserted. The models used do not allow for modelling performance additions or omissions, so this could be an area for further work. For the use case described though - music editing - all that is required is a number of useful sync points between performances, so so long as the algorithm behaves consistently on different recordings, path idiosyncrasies may not matter, especially as a more complex model which takes score errors into account would be more computationally expensive and increase the chance of further errors.

Figure 6.2 shows a Poisson point model-based alignment. Looking at the colour map, it can be seen that the Poisson point model does not appear to be as good a discriminator compared to the other likelihood methods. The colour map is vertically striped, meaning the likelihood is heavily dependent on the audio frame in question. This is most likely related to the number of peaks being detected in a frame - undue prominence is being given to false positives (peaks detected in a frame which do not correspond to harmonics in a state). An example of this is given in Figure 7.2. (a) shows an audio frame which always has a reasonable high likelihood, as there are relatively few peaks detected (many of which correspond to expected peaks); (b) shows a period which is a lighter colour on the diagram, where there are many spurious peaks. Most of these will be false positives no matter which state it is correlated with, meaning that the likelihood is dominated by these peaks. It is possible that this behaviour could be tweaked by changing the parameters of the Gaussian intensity model, in particular μ_c, which controls the minimum of the intensity function, and therefore the likelihood contribution of these incorrect peaks. A few different parameter values were tried, but an exhaustive optimisation for all the parameters is not trivial. One option for further work would be to perform maximum likelihood analysis of the Poisson point model in order to optimise its parameters, e.g. using the MAPS database [6] which contains an accurate alignment reference.

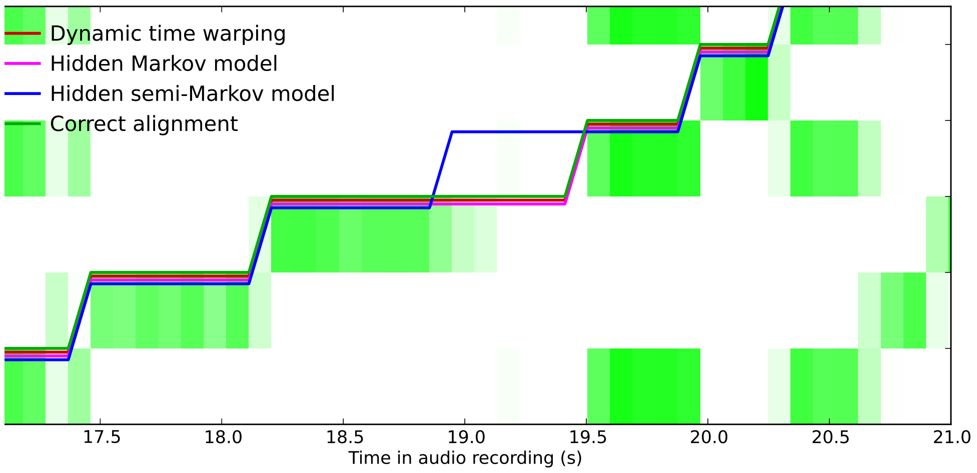


Figure 7.1: Close-up of alignment error in Figure 6.1 (chromagram alignment of *Sarabande*)

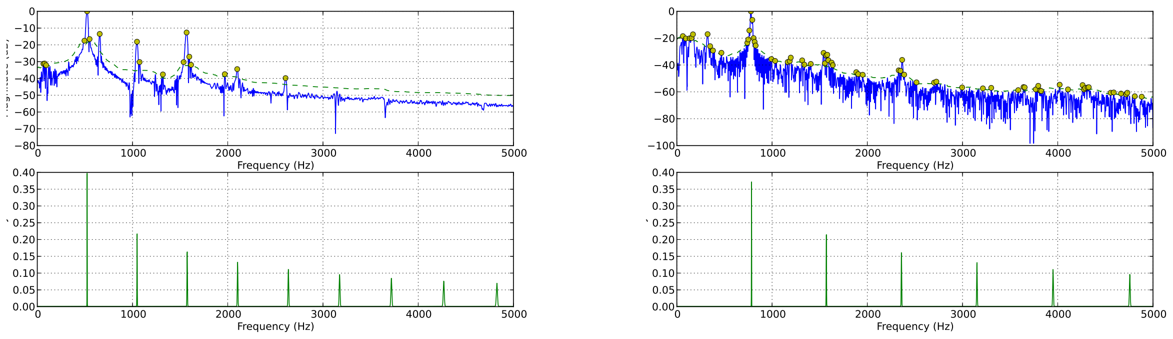


Figure 7.2: Comparison of detected peaks and corresponding Gaussian model for the alignment in Figure 6.2. (a) on the left is from around 5s, where there is a likelihood peak; (b) on the right is taking from a low-likelihood frame around 7.5s

While it is not obvious on the diagram, there is a reasonable range in the likelihood values calculated for different states within an audio frame, meaning that in most cases a reasonable path is still found. However, the variability of the likelihood along the time axis means that if an error occurs, e.g. if an extra note is played or missed out, a successful recovery is less probable.

The only consistently poor combination was the Poisson point model with a HSMM. The path is in fact following the threshold of the window around the diagonal line inside which likelihood calculations were constrained, meaning that the path would have been even further out of alignment if the complete cost matrix had been calculated. It could be that it was affected by the initial 'rest' state which was added in order to account for the silence at the beginning of the recording. Because it is hard to give a reasonable expected duration for this, it is possible that the it 'overshot' the correct path, and subsequently was not able to traverse back due to reasons described above. Another possibility is that the initially conditions for the HSMM model were incorrect, meaning it preferred to 'cling' in the initial state for as long as possible, but the other models do not seem to be affected by this so it seems unlikely.

Figure 6.3 shows an alignment using the PSD model, and 7.3 shows a close-up of the main region of misalignment. As before, the HSMM path has opted for a route which minimises the duration of state 73, which has a short duration in the score and is longer than expected in the recording. The region contains multiple rests, which are a problem for alignment algorithms because the expected spectrum is an empty one, but in practice the normalisation processes means undue prominence is given to variations in the background noise. The PSD model was particularly problematic because the the normalisation process would involve dividing by zero, so rests had to be dealt with as a special case. It was found that returning a constant value as the likelihood for such states had a tendency to cause paths to either stay in the state for too long (if it is higher than the surrounding states), or delay entering the state (if it is lower than the surrounding states). The solution I went for was to use the pitch information for the previous non-rest state to calculate the likelihood, which is justified because the major spectral content in rest states is generally caused reverberations from notes played in the past. This of course meant that the paths often pass into the rest state too soon (as can be seen with the DTW path below), but it at least ensured that the error was localised and did not have a significant effect on the alignment elsewhere.

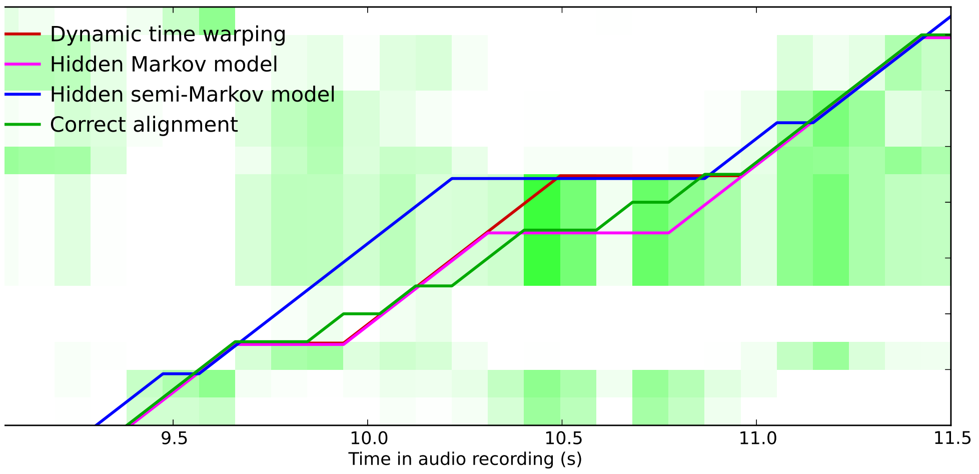
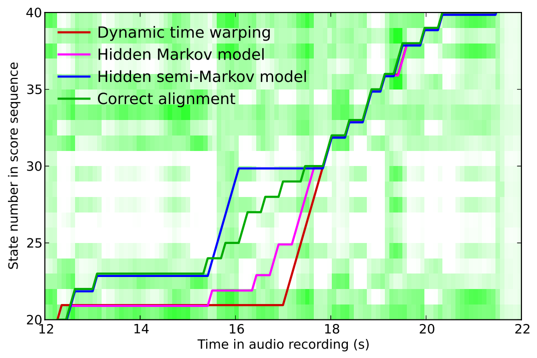


Figure 7.3: Close-up of alignment error in Figure 6.3 (PSD alignment of *Rondo*)

The score-following visualisation tool seems to show that the PSD method is better at detecting a note's onset, while the chromagram method tends to lag a bit. This can be explained because the likelihood in the PSD method largely ignores spectral information in frequency bands other than those it is expecting peaks. This means that, while the normalisation process removes the effect to an extent, a state containing a single note will match an audio frame containing any combination of notes, including the note in question, with little variation in the likelihood. Because the energy in is generally higher during the 'attack' phase of a note, it seems that this is causing a high likelihood in the audio frame at the correct start of a note, in spite of any residual reverberations from the previous state. However, this effect is less pronounced in the piano music where the amount of total amount of energy present in the spectrum is generally higher, which is why this does not seem to be reflected in the overall statistics.

Figures 6.4 and 6.5 show the alignment of *Clair de Lune* for the chromagram and PSD models respectively. It can be seen that all the alignment paths divert significantly around a region in the middle of the recording, close-ups of which can be seen in Figure 7.4. The confusion appears to arise because it consists of a fast-moving sequence of repetitive states. Additionally, because the sustain pedal is being used, notes from previous frames are still present in the following frames, meaning the spectral content is hard to pick up.

It can be seen that in general the chromagram method has produced a better alignment. In particular, the HSMM path only deviates from the correct path for about 2 s, proving, in this instance at least, the effectiveness of a duration-focused model. It is interesting that while it was predicted that the *Rondo* would perform worse, *Clair de Lune* in fact has a higher rate of misalignment. It seems that the use of the sustain pedal and variations in the tempo outweighed the advantage of all the states being reasonably long and theoretically easy to detect. It could also be because of the more complex harmonies in *Debussy's* piece increase the chances of higher-order harmonics coinciding.



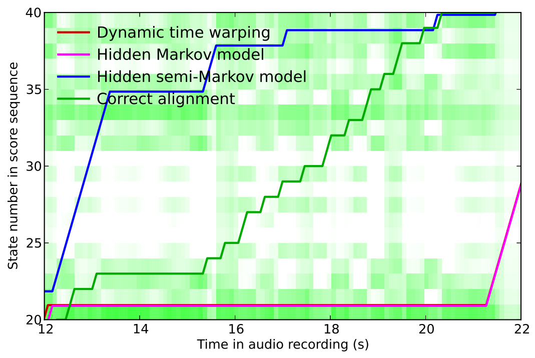


Figure 7.4: Close-up of alignment errors in Figures 6.4 and 6.5 - left is chromagram and right is PSD

The HSMM appears to be, on average, about 3 times more complicated to compute than the more basic models, but because the maximum value of u (the look-back distance) is constrained, it should never be more than a constant factor slower as T increases. Possible methods to reduce the complexity of the path-finding process are discussed in section 8.2. The Poisson model is a little slower than the other two likelihood methods due to the size of the feature vector - 3715 (the size of the FFT) compared to 12 for chromagrams and a variable size for the PSD methods, which will always be somewhat less than 3715. The Poisson model also necessitates the calculation the Gaussian intensity functions which is a major bottleneck, even though the caching implemented saves some time. This could be aided by loading pre-computed Gaussians from disk, or by using approximations (e.g. assume it equals zero outside a certain range).

Overall, the chromagram method appears to be the most reliable model for correlating the score and audio. In some ways this is surprising because it is the simplest method, but maybe this is because it effectively correlates chord progressions between audio frames rather than the exact notes per-say. While this may lead to many false positives (as can be seen by comparing the colour maps of the chromagram and PSD processes), it does not seem to affect the alignment substantially, as the best path will only be where the sequence of harmonies matches that found in the score.

The peak structure distance method is notable for its accurate detection of note onsets in certain instances, and was otherwise only slightly worse performing than the chromagram method. The Poisson point model was the least effective of the methods tested, but it is also the most complex model with a number of tunable parameters which could have been sub-optimal for the task.

The performance differences between the path-finding methods are less pronounced, but it is clear that including the duration of the states in the model does improve the rate of misalignment. Dynamic Time Warping is remarkably effective given its lack of constraints, especially on the monophonic music; the HMM shows an incremental increase in accuracy, and the semi-Markov model improves the alignment further.

8 Further Work

8.1 Alignment improvements

The work undertaken has found that taking into account the duration can lead to improvements in the accuracy of the alignment. $d_j\left(u\right)$ can take any form of probability distribution, so alternatives to the Gaussian could be considered. One possibility might be use the Poisson distribution (which is suggested in the report referenced in the HSMM section [3]), or a simple triangle function could be evaluated (centred on the expected note duration), which would be quicker to compute than a Gaussian.

A major area of current research which has not been considered in this project is accounting for the exact timing of note onsets and offsets. It is widely regarded that human perception of music is predominantly characterised by the ‘attack’ of a note rather than its transient qualities, so one would expect that an alignment which takes such features into account might perform well. Similarly, the state sequence that was used does not distinguish between a long note and two notes with the same pitch whose durations sum to the same value.

The reports that influenced the chromagram[9] and PSD[10] methods both suggest methods to take note onsets into account, generally by using the fact that note onsets often have a large burst of energy. It suggests that the full value of the chromagrams are used at the onset of the note, and thereafter the chromagram should be repeated, multiplied by a decay constant (with no regard to the note’s expected duration) . It is suggested that the PSD features are modified by using the difference between PSD values is subsequent states, which will give greater prominence to frames in which onsets occur.

Methods used to improve onset detection could also be used to improve the treatment of rests (to solve problems discussed in section 7). I took the simplistic approach of attempting to calculate the likelihood only based on the content of the score at that precise moment, but a more sophisticated approach would use data from the surrounding states to predict which notes are likely to still be reverberating.

A final area that could be researched is modelling the state transition matrices to take into account possible performance mistakes. Non-zero probabilities could be attached to state transitions for jumps greater than one state, and transitions to ‘ghost’ states can be added to take account for extra notes in the recording. The results generated show that small errors do not seem to affect the most likely path overall, but the music examples used - mechanically generated piano music and flute music with a very simple melody - were very accurate by their nature. More typical examples might less accurate, and may include ‘errors’ that would not be perceived so by musicians such as trills, vibrato and glissandi.

8.2 Further optimisations

There are a number of standard techniques to optimise the dynamic programming algorithms. FastDTW[14] is a widely used method which reduces the complexity of DTW to $O(T)$. It works by first using a low-resolution version of the cost matrix (by averaging between cells), and calculating the best path is this domain, then iteratively advancing to higher resolutions, refining the best path at each step.

It is not possible to adapt FastDTW for use with the Viterbi method because it is not possible to adequately model the state transitions in the lower resolutions, but FastDTW could be used to estimate a path, which could then be used to tightly constrain the Viterbi search path. However, speed-ups could likely be achieved more directly by path-pruning. This involves discarding paths at every iteration which have a cumulative cost less than a certain threshold. [15] suggests using a threshold of 1.1 times the minimum cost.

The system, as it is currently implemented, first calculates the complete cost matrix (within the width constraint), then subsequently works out the best path. The likelihood computation is still $O(T^2)$ (if the constraint width is not constant), and these calculations dominate the overall computation time (especially for the DTW and HMM methods). In order to take advantage of improvements to the path-finding algorithm, the system should be modified to calculate the likelihoods only when needed by the path-finding algorithm (then caching the results for later use).

It is possible that further speed-ups could be achieved by implementing the algorithms in a compiled language such as C++. However, it could only ever be faster by a constant factor, and further improvements would be more cumbersome to make due to strict typing in compiled languages and the necessary compilation process. Also, the underlying mathematical routines in the NumPy and SciPy libraries are implemented in C++ and Fortran, and care was taken to perform matrix calculations directly rather than performing loops in Python, so any improvements made by porting the rest of the system to C++ may be minimal.

8.3 Visualisation improvements

The score-following visualisation tool proved very useful for intuitively analysing the quality of alignments, and it has been built in such a way that future developments can easily be implemented and provided to the visualisation using a common API.

There are a some areas where the tool could be improved:

- Add scales to both axes to indicate the progress through the score and the absolute pitches of the notes.
- Add the ability to manually scroll through the score and start playback at an arbitrary point.
- Add ability to zoom in/out so that slow music does not pass by rapidly, and the onset of short notes in fast music can be more easily observed.
- At the moment, if the alignment requested is not cached on the server, the program has to calculate it on demand. This can take a few minutes (roughly twice the length of the audio using current implementation), during which time no feedback is provided in the browser (log messages are outputted in the server console via simple ‘print’ statements in the code). It would be preferable to make the process asynchronous, so that the browser simply submits a ‘job’ to the server, which triggers the alignment process to start in a separate process. Then the front-end can be modified to periodically query the status of the job (downloading log messages at the same time), and download the result when it is ready.
- Since the chosen platform for the visualisation tool uses a server/client model and the client can be set up to be accessed remotely, it would be nice to be able to upload audio and MIDI files via the browser rather than being restricted to a list of provided data. This would necessitate the previous point being implemented too so that feedback can be given while the alignment algorithm runs.

8.4 Integration

Once both the alignment and cross-fading algorithms have been fully refined, they could be integrated into an application with a user interface which allows sound engineers to to align multiple records of the same piece of music.

For the use case described at the beginning of this report, a 100% continuous and accurate alignment of the music to the score is not necessarily required. If a sound engineer wishes to make a cut between two takes of the same piece of music, then the exact alignment only needs to be known at the point where the cut it made. Furthermore, there will likely be a range in the audio during which a cut could be made at any point. With this in made, it would be interesting to investigate characterising the relative confidence of an alignment along its length, with the aim of suggesting a list of the most reliable 'sync points'. The likelihood of a successful cross-fade taking place at each point could be taken into account too.

9 Conclusions

Three methods were used to correlate a instant in a musical score with a frame from an audio recording. The simplest feature vector, the chromagram, proved to be the most successful, both in terms of overall accuracy and the average alignment error, because of how it directly measures the harmony content of the score rather than precise note pitches, abstracting away the effect of different timbres. A method based on summing spectral content near expected peak locations achieved similar accuracy and proved slightly better at anticipating note onsets in certain cases. A Poisson point model of peaks in the frequency domain also achieved good accuracy most of the time, but improvements need to be made to the model in order to lower the number and effect of peaks that are just a result of noise. The Poisson point model is the most sophisticated, but also the most computationally expense process, and the other two methods considered took about the same amount of time to run.

Dynamic time warping, which does not take note durations into account, proved surprisingly reliable in all but the most demanding of cases. Approaches using hidden Markov model and hidden semi-Markov model were also considered, both of which take into account the expected note durations to some degree. They both provided a small but significant reduction in errors and an increase in accuracy. While the semi-Markov model considered provided a small improvement in alignment, in some cases additional errors occurred where note lengths in the recording did not match the expected durations. The semi-Markov model is also a significant constant factor more expensive to compute than the other two methods investigated.

A software framework has been made for efficiently evaluating score-to-align alignment algorithms, including a web-based alignment visualisation tool with a score-following cursor. It is hoped that these tools can be used to aid further improvements to the alignment algorithms, with the hope of eventually having a reliable enough basis to produce an automatic music editing tool.

10 Acknowledgements

I am grateful to Prof. Simon Godsill for his advice and guidance throughout the year, as well as Leonard Chan and Peter Bunch for the pointers given by their previous 4th year projects.

11 Risk Assessment Review

As the project was entirely programming-based, the only risks identified were those to do with using a computer. Care was taken to maintain good posture and to take regular breaks while working, and no injuries occurred. I can therefore conclude that the risk assessment was adequate and sufficient.

References

[1] P. Bunch. Bayesian machine learning for musical audio signals. May 2010.

[2] L. Chan. Automated editing for the music industry. May 2012.

[3] Arshia Cont. A coupled duration-focused architecture for real-time music-to-score alignment. In *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, volume 32, June 2010.

[4] Arshia Cont, Diemo Schwarz, Norbert Schnell, and Christopher Raphael. Evaluation of real-time audio-to-score alignment. In *Proc. ISMIR*. Citeseer, 2007.

[5] S. Dixon and G. Widmer. Match: A music alignment tool chest. *6th International Conference on Music Information Retrieval (ISMIR 2005)*, pages 492–497, 2005.

[6] V. Emiya. *Transcription automatique de la musique de piano*. PhD thesis, Telecom Paris-Tech, 2008.

[7] Jack D Ferguson. Variable duration models for speech. In *Proc. Symposium on the application of hidden Markov models to text and speech*, pages 143–179, 1980.

[8] Özgür İzmirli and Gyanendra Sharma. Bridging printed music and audio through alignment using a mid-level score representation. 2012.

[9] Roger B. Dannenberg Ning Hu and George Tzanetakis. Polyphonic audio matching and alignment for music retrieval. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2003.

[10] Nicola Orio and Diemo Schwarz. Alignment of monophonic and polyphonic music to a score. In *Proceedings of the International Computer Music Conference*, pages 155–158, 2001.

[11] C. Li P. Peeling and S. Godsill. Poisson point process modeling for polyphonic music transcription. *The Journal of the Acoustical Society of America*, 121(4):168–175, 2007.

[12] Miller Puckette and Cort Lippe. Score following in practice. In *Proceedings of the International Computer Music Conference*, pages 182–182. INTERNATIONAL COMPUTER MUSIC ACCOCIATION, 1992.

[13] Christopher Raphael. Automatic segmentation of acoustic musical signals using hidden markov models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(4):360–370, 1999.

[14] S Salvador and P Chan. Fastdtw: Toward accurate dynamic time. *Warping in Linear Time and Space*, 2007.

[15] Ferréol Soulez, Xavier Rodet, Diemo Schwarz, et al. Improving polyphonic and poly-instrumental music to score alignment. In *Proceedings of International Conference on Music Information Retrieval*, 2003.

[16] R. Badeau V. Emiya and B. David. David, multipitch estimation of piano sounds using a new probabilistic spectral smoothness principle. *IEEE Transactions on Audio, Speech and Language Processing*.

[17] Barry Vercoe. The synthetic performer in the context of live performance. In *Proc. ICMC*, pages 199–200, 1984.

[18] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, 1967.

A Contents of the enclosed CD

- src/ - the source code of the project
- README.txt - details of how to use the source code
- sarabande.mpeg - a video showing the score-following visualisation tool in use