From raw audio to a seamless mix: an Artificial Intelligence approach to creating an automated DJ system

Len Vande Veire

Supervisors: Prof. dr. Tijl De Bie, Paolo Simeone

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Department of Electronics and Information Systems
Chair: Prof. dr. ir. Rik Van de Walle
Faculty of Engineering and Architecture
Academic year 2016-2017

GHENT
UNIVERSITY

This page is intentionally left blank

From raw audio to a seamless mix: an Artificial Intelligence approach to creating an automated DJ system

Len Vande Veire

Supervisors: Prof. dr. Tijl De Bie, Paolo Simeone

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Department of Electronics and Information Systems
Chair: Prof. dr. ir. Rik Van de Walle
Faculty of Engineering and Architecture
Academic year 2016-2017

GHENT
UNIVERSITY

Copyright agreement

Len Vande Veire
02 June 2017

Preface

For over 4 years, I have been an avid Drum and Bass fan, and about 2 years ago I started creating my own DJ mixes on a hobbyist level. Like many other fans of electronic music, I greatly enjoy listening to DJ mixes, podcasts and recordings of live sets, because the masterful way in which DJs blend individual songs together turns the listening experience into a magnificent and intriguing journey. The variety and creativity in these mixes makes the Drum and Bass genre all the more exciting. In this thesis, I have been able to combine my passion for music and computer science, and it has truly been an amazingly interesting project. At the end of the road, I must say that I am very pleased with the result myself, as the first prototype developed in this thesis can indeed create a nice and continuous Drum and Bass DJ mix: it has even given me new ideas for my own mixes.

Of course, realizing this project would not have been possible without the excellent guidance and input of my thesis promotor prof. Tijl De Bie and my counselor Paolo Simeone, for which I am very grateful. I would also like to thank my family and friends for their support, in particular my fellow computer science classmates for their company and the enjoyable coffee breaks. Lastly, I would like to give a shout-out to all the amazing Drum and Bass producers and DJs. Their wonderful music has kept me motivated and energized during the many programming and writing sessions of this thesis and has inspired me throughout the years, an exciting journey which has culminated in this project.

I hope you enjoy reading my thesis.

Len Vande Veire
June 2017

From raw audio to a seamless mix: an Artificial Intelligence approach to creating an automated DJ system

Len Vande Veire

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering
Academic year 2016–2017

Supervisor: Prof. dr. Tijl De Bie
Counselor: Paolo Simeone

Department of Electronics and Information Systems, Faculty of Engineering and Architecture, Ghent University
Chair: Prof. dr. ir. Rik Van de Walle

Summary

Music is a beautiful and exciting form of art, enjoyed by many on a regular basis. However, when playing songs back to back, i.e. playing one song after the other, the listening experience is interrupted briefly between the end of one song and the beginning of the next. Furthermore, if each song has a long intro and outro, the excitement of listening might fade and the listener could get bored. It would be more interesting to have a continuous stream of music where one song seamlessly transitions into the next. In this way, the listening excitement is kept at a constant level and a more satisfying experience is obtained.

This already happens in nightclubs and online podcasts, where a DJ skillfully blends existing music into a seamless mix. However, the task of a DJ could be automated. This would allow a user to create DJ mixes on demand, using songs in his personal music library and without needing any DJing skills. The goal of this thesis is to develop an automatic DJ system that is able to generate seamless music mixes using songs from a given library. The automatic DJ is designed for Drum and Bass, a specific genre of electronic dance music. Several music information retrieval (MIR) techniques, namely beat tracking, downbeat tracking and structural segmentation, are combined to give the automatic DJ an understanding of the structural properties of the music. This knowledge is used in a rule-based framework, inspired by DJing best practices, to create transitions between the selected songs. The transition quality is evaluated by means of a machine learning model. Evaluation on a corpus of 380 Drum and Bass songs shows that the used algorithms are very accurate and enable the automatic DJ to create enjoyable mixes.

Keywords: DJ, Drum and Bass, music information retrieval, computational creativity, machine learning

From raw audio to a seamless mix: an Artificial Intelligence approach to creating an automated DJ system

Len Vande Veire
Supervisors: Tijl De Bie, Paolo Simeone

***Abstract*— We present the implementation of an automatic DJ system that is able to generate seamless music mixes using songs from a given library. The automatic DJ is designed for Drum and Bass, a specific genre of electronic dance music. Several music information retrieval (MIR) techniques, namely beat tracking, downbeat tracking and structural segmentation, are combined to give the automatic DJ an understanding of the structural properties of the music. This is used in a rule-based framework inspired by DJing best practices to create transitions between the selected songs. The transitions are optimized by means of a machine learning model. Evaluation on a corpus of 160 Drum and Bass songs and an additional held-out set of 220 songs shows that the used MIR algorithms can accurately annotate 90% of the songs, which enables the automatic DJ to create enjoyable mixes.**

***Keywords*—DJ, Drum and Bass, music, beat tracking, downbeat tracking, structural music segmentation, computational creativity, machine learning**

I. Introduction

Music is a beautiful and exciting form of art, enjoyed by many on a regular basis. However, when playing songs back to back, i.e. playing one song after the other, the listening experience is interrupted between the end of a song and the beginning of the next. Furthermore, if each song has a long intro and outro, the excitement of listening might fade and the listener could get bored. It would be more interesting to have a continuous stream of music where one song seamlessly transitions into the next. In this way, the listening excitement is kept at a constant level and a more satisfying experience is obtained.

This already happens in nightclubs for example, where a professional DJ skilfully blends songs together into a seamless music mix. However, a user might want to create his own DJ mixes on the fly using songs from his own music library, without having the required skills, equipment or time to compose and record that mix himself. For these users, a computer program that automates the DJing task would be an excellent solution. Such an automatic DJ is also very interesting for nightclubs and bars, because it alleviates the need to hire a human DJ. Thirdly, it could be used by professional DJs as an exploration tool to discover interesting song combinations and transitions. In this paper, we therefore present a system that automates the task of the DJ. It is designed for a specific genre of electronic dance music, namely Drum and Bass. A short introduction into DJing is given in Section II. Section III explores related work in scientific literature. Section IV elaborates on how the DJ discovers the musical structure in a hierarchical manner. Section V discusses the system architecture and how the song transitions are created. The structure annotation techniques and the mixing process are evaluated in Section VI. Finally, Section VII concludes this paper and gives some pointers for further improvements.

II. What is a DJ?

A DJ is a person who mixes existing music records together in a seamless way to create a continuous stream of music [20]. Doing so is a non-trivial task, and it requires executing several steps. Firstly, the DJ needs to know the music its tempo and structure. This allows him to mix songs together while respecting the musical properties during the mix. Next, he will select songs to play and determines the order to play them in. These are the *track selection* and *track listing* steps. The DJ also determines at what time instant in those songs he wants to start the transition from one song into the next, which is called *cue point selection*. These starting points, or *cue points*, are typically aligned with structurally important boundaries in the music.

When the DJ knows which songs he wants to play and when to start the transitions, he can start mixing. He plays the first song and waits until the music reaches the cue point of the second song. Then the second song is started. When playing two songs together, it is important that their beats align in time. Otherwise, the resulting mix will sound chaotic and off-tempo. To do so, the songs are slowed down or sped up by *time-stretching* them such that their tempi are equal. The beats are then aligned in a process called *beatmatching*. To establish a smooth transition between the songs, the DJ will perform a *crossfade*. This is done by gradually increasing the volume of the new song, i.e. the fade-in, while decreasing the volume of the other song, i.e. the fade-out. The DJ also adjusts the *equalization* settings of the songs while they are playing at the same time: this ensures that the mixed music does not saturate in the low- and high-frequency regions. The process of cueing, time-stretching, beatmatching and crossfading is applied for each transition between songs in the mix, effectively creating a continuous and seamless stream of music.

III. Related work

Existing research on automatic DJ system is scarce. Two types of systems often reoccur in scientific literature: automatic DJ systems and mashup systems. The former attempt to automate (parts of) the DJing task, i.e. create a continuous mix by smoothly transitioning from one song into the next. Mashup systems on the other hand create a new song by combining short fragments of existing songs. Hence, a mashup is typically much shorter than a DJ mix, and the input songs are more heavily modified by cutting and pasting fragments from them. Nevertheless, similar techniques, such as time-stretching, beat tracking and crossfading, are used in both applications.

In a section of his PhD thesis, Tristan Jehan [11] proposes a simple automatic DJ system that automatically downbeatmatches and crossfades songs. Cue points are determined by finding rhythmically similar sections in the mixed songs, but it does not consider the high-level structure of the music. In his master's thesis, David Bouckenhove [3]

describes an automatic DJ system that uses vocal activity detection to avoid overlapping vocal sections of two songs. With their Music Paste system, Lin et al. [13] propose a solution for automating the tracklisting and cue point selection process by maximizing a measure for musical similarity. The length of the transition is optimized such that the rate of tempo change remains under an acceptable threshold, which is determined in a subjective experiment. Ishizaki et al. [10] also focus on how a crossfade with a changing tempo can be optimized. They propose to use an intermediary tempo between the tempi of the original songs, such that the discomfort caused by the tempo change is spread evenly between the two songs. Finally, the MusicMixer project by Hirai et al. [9] improves the track and cue point selection process by means of two similarity measures related to the beat structure and a high-level abstraction of the chromatic content of the audio, inspired by natural language processing techniques.

Research on mashup systems typically focuses on devising an appropriate measure that determines if song extracts are compatible to be *mashed together*. A first example is AutoMashUpper by Davies et al. [5], [6]. This system features a beat tracking, downbeat tracking and a structural segmentation step to appropriately align the music. Music fragments are extracted based on a similarity measure related to the harmonic and rhythmic content of the music. Another example is the work of Lee et al. [12], which focuses on extending mashup systems to multiple overlapping songs and also considers the compatibility of music segments that occur after each other in the mashup instead of only the compatibility of the overlapped segments.

Most existing work on automatic DJ systems focuses on optimizing individual crossfades by minimizing the amount of discomfort. However, there are still some important limitations to these systems. First of all, in many of the described systems, the crossfading process remains very simple, e.g. without performing any equalization. Secondly, almost none of these systems consider the high-level structural properties of the mixed audio. Thirdly, they mainly focus on optimizing individual crossfades, but do not consider the global song progression throughout the mix. In general, there appears to be no complete integrated DJing system that elegantly combines all necessary components and considers DJing best practices to create enjoyable music mixes.

IV. Discovering the Musical Structure

One aspect that makes music different from random sound is that it exhibits a hierarchical structure [16]. At the lowest level, music consists of notes, which repeat periodically to define the tempo of the music. Certain notes (typically percussive in nature) are more prominent than others, creating the *beats*. Beats can be grouped into groups of four: such a group is called a *measure* or a *bar*. The first beat of a measure is usually more accentuated than the others: this beat is called the *downbeat* of that measure. Measures are the basic building blocks of longer musical structures such as *phrases*, which then make up the larger *sections* that determine the compositional layout of the song. The typical composition of a Drum and Bass song is illustrated in Figure 1.

Knowing the rhythmical and structural properties of the music is extremely important for a DJ. The tempo and beat positions are used to *beatmatch* the music. The high-level musical structure is important when selecting cue points: if the DJ mixes songs where the downbeats or segment boundaries are not appropriately aligned, the resulting mix will sound incoherent, leading to an inferior listening experience. In what follows is described how the automatic DJ system extracts the beat, downbeat and segment boundary locations in a hierarchical way from the input audio.

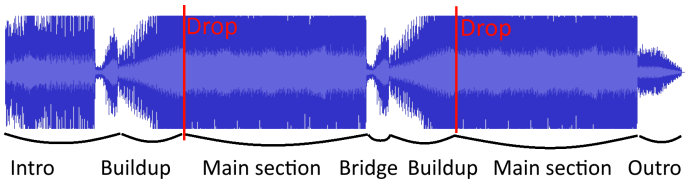


Fig. 1. Typical compositional layout of a Drum and Bass song. The song starts with an *intro*, followed by a *build-up* that gradually increases the musical tension. This tension is released in a moment called the *drop*. This is the beginning of the *main section* of the music, comparable to the chorus in pop music. After the main section, there is a musical *bridge*, also called a *breakdown*, that connects the first main section and the second build-up, drop and main section. An *outro* gradually brings the music to a close.

A. Beat tracking

To discover the beat positions in the music, an algorithm inspired by the work of Davies and Plumbley [7] is used. It assumes that the analyzed song has a constant tempo throughout the song, which is the case for most Drum and Bass music. With this assumption, only two parameters need to be determined to define the positions of all beats: the beat period, which determines the tempo, and the beat phase, i.e. the offset of the first beat of the song with respect to the beginning of the audio signal. Two observations are at the core of this algorithm. Firstly, music is a very repetitive signal, and the repetition of musical *onsets*, i.e. musical events such as notes or percussion events, typically happens after an integer number of beats. Secondly, the loudest or most prominent onsets typically occur on beat locations.

These two observations are related to the position of musical onsets. Therefore, the algorithm needs to estimate the location of onsets in the audio. This is done by means of an *onset detection function* (ODF). This is a curve that has a high value for time instants in the music where an onset is detected, and a low value when this is not the case. An excellent review on the different types of onset detection functions is given by Bello et al. [1].

Figure 2 illustrates the beat tracking algorithm. The first step is to calculate the ODF of the audio (Figure 2b). This curve is post-processed by subtracting a running mean from it and half-wave rectifying the result. Then the beat period is extracted by calculating the autocorrelation function of the ODF (Figure 2c). The autocorrelation will be high at offsets that are a multiple of the beat period. Therefore, for each possible tempo, the autocorrelation function values are summed with a step size equal to the corresponding beat period. The tempo candidate with the highest sum is selected as being correct. Next, the beat phase is determined. For each phase candidate, i.e. every possible shift between 0 and 1 times the beat period, the ODF values are summed at fixed intervals

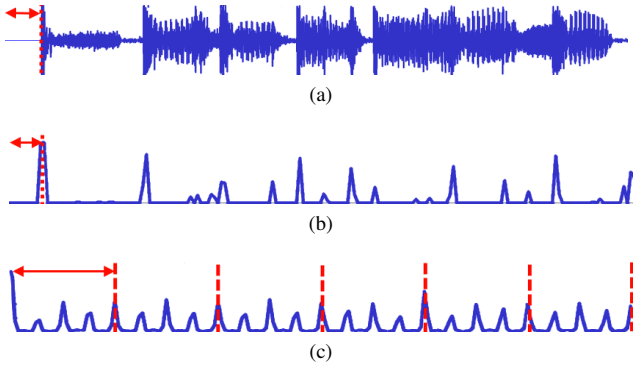


Fig. 2. Illustration of the beat tracking algorithm. (a) Audio waveform extract. The beat phase is annotated. (b) Half-wave rectified onset detection function. The beat phase is annotated. (c) Autocorrelation function of the onset detection function. The beat period is annotated.

corresponding to the detected beat period. The phase candidate with the highest sum is selected as the correct phase.

The tempo range is restricted between 160 BPM (beats per minute) and 190 BPM, because the tempo of Drum and Bass music falls between these values.The smallest increment in tempo and in phase that can be detected are 0.01 BPM and 1 ms respectively.

B. Downbeat tracking

Given the beat positions, the DJ system determines which beats are downbeats. The measures in Drum and Bass music are always 4 beats long [20], implying that there are only four options: the first downbeat of the song is either the first beat, the second beat, the third beat or the fourth beat. The downbeat tracking algorithm is summarized in Figure 3. It consists of three main steps. First, features are extracted from the beat segments. Then, a logistic regression classifier determines the probability that a beat is either the first, second, third or fourth beat in its measure. Finally, these predictions are aggregated over the entire song for each of the four options to determine the most likely downbeat positions.

To classify a beat in the input audio, useful features need to be extracted. First, the loudness [19] of each beat fragment and the energy distribution of the audio along the frequency axis, binned in 12 equally spaced bins on the Mel frequency scale [18], are calculated. Additionally, three onset detection functions are calculated of the entire song: these ODFs are split into frames of half a beat long, which are assigned to the corresponding beat they fall in, forming a second set of features. For each ODF frame, the ODF values are summed and used as a third set of features. These features describe each beat in isolation and are therefore called *isolated features*. However, a beat fragment does not contain enough information on its own to determine its position within its measure. Indeed, the notion of rhythmical structure arises by the carefully orchestrated accentuation of certain beats compared to other beats. Therefore, the machine learning classifier does not use the isolated features directly, but rather calculates the distance between the features of a given beat and those of the next 4 or 8 beats. These *contextual features*, i.e. differences of isolated features, capture the evolution of the music and are used by the classifier to predict the downbeat positions.

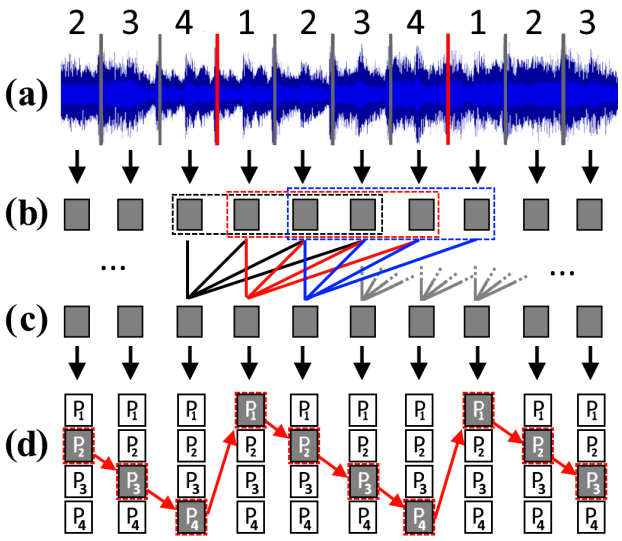
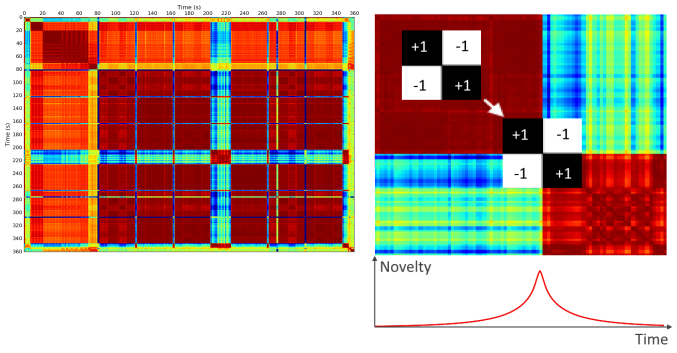


Fig. 3. Overview of the downbeat tracking algorithm. (a) The audio waveform, annotated with beats and downbeats. (b) Isolated features are extracted from the beat extracts. (c) Isolated features are combined to create contextual features. (d) The beats are classified using the machine learning model, estimating the log-probability of it being either the 1^{st}, 2^{nd}, 3^{rd} or 4^{th} beat in a measure. The downbeats are then determined by summing the log-probabilities along each possible trajectory and choosing the most likely one. In the figure, the second trajectory is highlighted.

To determine the downbeats on a song level, the algorithm works as follows. First, the audio is trimmed by removing the intro and outro of the audio using an RMS energy criterion, because the beat classifications are much less reliable in these parts of the music. Then, the features of the remaining beats are extracted and subsequently classified using the machine learning model. This results in a log-probability vector for each beat that estimates whether it is either the first, second, third or fourth beat in its measure. Then, for each of the four possible trajectories throughout the song, the corresponding log-probabilities of the beats are summed. The trajectory with the highest sum is the most likely and is selected as the correct downbeat trajectory. This prediction is extrapolated to the trimmed beats in the intro and outro, after which all downbeats can be extracted.

C. Structural segmentation

The goal of the structural segmentation is to discover the high-level compositional layout of the song. To do so, the novelty-based structural segmentation method by Foote [8] is used. This approach assumes that structural boundaries go along with significant musical changes in the audio. First, a *structural similarity matrix* (SSM) is constructed by splitting the input audio into short frames, extracting features of those frames and comparing the features of each frame with those of every other frame, resulting in a matrix of pairwise comparisons. Frames that are similar lead to a high value in the matrix, whereas dissimilar frames have low values. This is illustrated in Figure 4a. The formation of blocks in this matrix indicates the occurrence of structurally coherent musical segments: within those segments, all frames sound very similar. The algorithm by Foote determines the location of the boundaries between these segments by convolving the matrix along its main diagonal with a 'checkerboard' kernel. This leads to a *novelty curve*, which has high values for time



(a) (b)

Fig. 4. Illustration of structural segmentation concepts. (a) Example of a structural similarity matrix. (b) Illustration of the novelty curve calculation by convolving the SSM with a checkerboard kernel along its main diagonal.

instants that coincide with a structural boundary and low values elsewhere. This process is illustrated in Figure 4b.

From this novelty curve, candidates for the structural boundaries are extracted by selecting the 20 highest peaks in the curve. Several post-processing steps are applied to determine the downbeat-aligned positions of the segment boundaries. First, the detected peaks their positions are rounded to the nearest downbeat. Peaks that lie further than 0.4 times the length of a measure from a downbeat are considered unreliable and are therefore discarded. From these downbeat-aligned segment boundary candidates, a subset needs to be determined in which all candidates lie at a multiple of 8 measures from each other. This is done because in most Drum and Bass music, structural segments are a multiple of 8 measures long [20]. There are hence eight subsets to choose from. All peak values belonging to a certain subset are summed, resulting in a sum for each potential offset. However, peaks are not considered in the summation if they lie one or two measures before another peak, and if they have a novelty value smaller than 0.75 times the amplitude of that peak. This is done to reduce the number of false positives caused by *breaks*, i.e. short musical segments that *break up* the music and typically occur just before a structural boundary. Then, each sum is multiplied by the number of peaks contributing to it, giving more weight to a sum with many contributors. The structural boundary candidates that contributed to the highest sum are considered to be correct. Each segment is then assigned a label based on their RMS energy: loud segments get a label 'high', whereas more quiet segments get a label 'low'. This is a crude attempt to distinguish between different types of segments, which will be used later to determine cue points.

V. Composing a DJ mix

A. Automatic DJ system architecture

The system architecture of the automatic DJ is shown in Figure 5. Songs in the music library are first annotated by means of the beat tracker, downbeat tracker and structural segmentation modules. Additionally, the *replay gain* [17] of each song is annotated, which allows the DJ system to play back each song at an equal volume, regardless of the volume they were recorded at. Next, the automatic DJ enters the playback loop by playing the first song. The tempo of the mix is fixed at 175 BPM: hence,

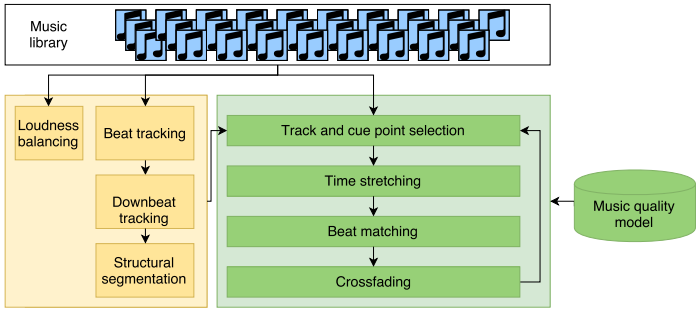


Fig. 5. Automatic DJ system architecture. On the left, in yellow, are the annotation submodules. On the right, in green, are the modules used 'live' during the mixing process.

the first song might be time-stretched before playing. Time-stretching is performed using the WSOLA algorithm [21]. Then, the next crossfade is created, which happens in two steps: first, the type of transition is chosen, and then the next song and its appropriate cue points are determined. There are three possible transition types, which differ in the type of overlapped segments ('low' or 'high') of the first song and the new song. These transition types are inspired by how professional DJs compose their mix and perform crossfades. The type is chosen using a Finite State Machine, which restricts certain transition types from happening twice in a row and hence ensures variation in the mix. The cue point in the first song is selected based on the transition type. To determine the new song and its cue point, three songs are selected from the music library at random, and three candidate cue points are chosen for each song, where their locations depend on the type of transition. Each potential crossfade is established by first time-stretching the audio to 175 BPM, beatmatching it with the first song and applying volume fading and proper equalization. The equalizer settings throughout the crossfade follow a fixed pattern, which depends on the type of crossfade. Each potential crossfade its quality is then evaluated by means of a music quality model, which is the subject of Section V-B, and the best crossfade is played to the user. This process is repeated for each song that is played, thus generating a continuous Drum and Bass mix.

B. Estimating crossfade quality using a machine learning model

To choose between different crossfades, the automatic DJ needs a way to estimate their quality. This is done using a support vector machine (SVM) that estimates the probability that a measure of Drum and Bass music is either of good quality (1), bad quality (0) or somewhere in between. For training, 'good quality' measures are extracted from the original, unmixed songs in the training set. From each song, 20 measures are extracted. 'Bad quality' examples are created using the same measure extracts, which are overlapped with random measure extracts from other songs. In this way, the resulting measures will resemble extracts from a badly mixed Drum and Bass mix.

To extract features from the audio of one measure, it is first transformed using the Constant-Q Transform (CQT) [4] with hop length 512 samples and 60 bins along the frequency axis, starting at $f \approx 32.70$ Hz (C1) and 12 bins per octave. The spectrogram amplitudes are transformed to a decibel scale and normalized between 0 and 1. The spectrogram is then split up

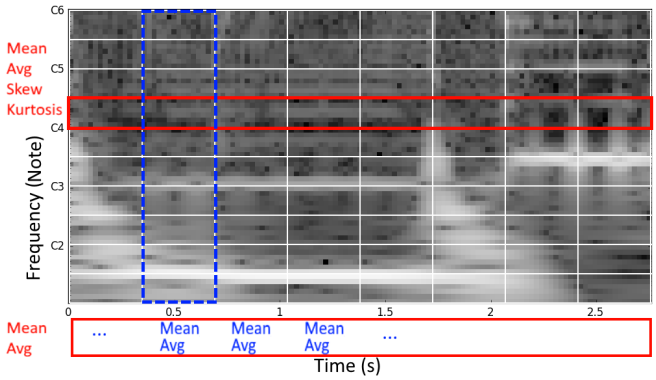


Fig. 6. Illustration of the feature calculation process for the music quality model. Features are extracted by calculating statistics of the means of rectangular, overlapping windows of the CQT spectrogram. The red features on the left of the spectrogram are used by the classifier. Note that in this illustration, the windows are wider than in the actual implementation, nor do they overlap: this is done to keep the figure clear.

into overlapping rectangular windows. Along the frequency axis, the windows have a height of a halve octave (6 CQT frequency bins) and do not overlap. Along the time axis, the windows have a width of one quarter beat, i.e. $\frac{1}{16}$th of the width of a measure, and a hop size of half the window length is used. For each of these windows, the mean of the spectrogram values is calculated. Then, the mean, variance, skew and kurtosis are calculated of the means of all windows that fall in the same halve octave, leading to 4 features per halve octave or 40 features in total. Also the mean and variance of all windows that fall in the same quarter-beat time segment are calculated. Of these 32 mean and 32 variance values, again the means and variances are calculated, giving 4 additional features for the measure extract. These features should capture the distribution of quiet and loud rectangular windows in the spectrogram and in this way describe how chaotic and cluttered the music is: badly mixed music should sound more cluttered than a good mix. Figure 6 illustrates the feature extraction process.

To estimate the quality of a crossfade of multiple measures long, the classifier first assigns a quality estimate between 0 and 1 to each measure. Then, the logarithms of these values are summed and divided by the number of measures in the crossfade. The crossfade with the highest sum is selected as the best crossfade.

VI. Experiments

The automatic DJ system has been evaluated on different aspects. First and foremost, the different annotation methods discussed in Section IV have been evaluated on their accuracy. Secondly, the music quality model its accuracy is evaluated on individual audio extracts of one measure. Apart from this, a subjective listening test has been conducted to assess its ability to discriminate between good and bad mixes. These evaluations are performed on a collection of 160 Drum and Bass songs, which were selected with care to ensure that there is a sufficient variety in Drum and Bass sub-genres and artists. The machine learning models, i.e. the downbeat tracking and the quality model, are trained on a subset of 117 songs, and evaluated on a held-out test set of 43 songs. Additionally, the accuracy of the annotation tasks is also evaluated on a second held-out test set of 220 songs to prove their robustness.

A. Evaluation of the beat tracker

The beat tracking accuracy is evaluated by annotating all 160 songs and then listen if the annotated beats coincide with the beats in the audio. It is thus not done by comparing the annotations with a manually annotated ground truth, because this manual labeling is extremely time consuming and because the inherent ambiguity of the beat annotation task [14] makes this even more difficult. Several onset detection functions were evaluated, which shows that the *melflux* function, as implemented in the Essentia library [2], performs best with 159 out of 160 songs annotated correctly, i.e. an accuracy of 99.4%. On the second test set of 220 songs, an accuracy of 98.2% is achieved, i.e. 4 songs their beats are incorrectly detected. Of the incorrect songs, the tempo is still correctly detected, but the phase is wrong by half a beat period, such that the detected beats lie exactly in between the correct beats.

B. Evaluation of the downbeat tracker

On individual beat segments, the downbeat classifier achieves an accuracy of 75.4% (after trimming the intro and outro). After determining the most likely downbeat positions of each song by aggregating all individual beat predictions, the downbeat tracker achieves an accuracy of 95.3% on the test songs. On the second test set, four of the 214 songs with correct beat annotations had incorrect downbeat annotations, leading to an accuracy of 98.1%.

C. Evaluation of the structural segmentation task

The accuracy of the structural segmentation task is more difficult to assess because of the subjective nature of the problem [15]. For the automatic DJ system, annotations are considered *structurally correct* if they have the correct 8-measure offset. Additionally, the simple labeling of the segments as a 'low' or a 'high' segment is evaluated. To do so, every transition from a 'low' to a 'high' segment in each of the 160 songs is listened to and is considered correct if that transition coincides with a *drop* in the music (see Figure 1). For 3 out of 160 songs, the 8-measure offset was incorrect. For 4 out of 160 songs, the assumption that all segmentation boundaries lie at a multiple of 8 measures from each other is incorrect. This means that 95.3% of the songs received structurally correct annotations. For 82.4% of these songs, also the drop detection as a 'low' to a 'high' transition is correct. In the second test set, 94.3% of the songs with correct downbeat annotations also received structurally correct segmentation annotations.

D. Evaluation of the music quality model

The music quality classifier has been trained using cross-validation to determine the best regularization parameters. Optimizing its performance leads to an accuracy of 80% on the train set and 74% accuracy on the test set for individual downbeat segments. The automatic DJ its ability to distinguish between good and bad crossfades is furthermore evaluated by means of a subjective user test, performed with 12 participants. Ten sets of three crossfades each are created using the DJ, where the crossfades in the same set are variations on each other. The goal of the test is to assess whether the DJ can adequately choose the optimal crossfade in each of these sets using the quality model. The 12 evaluators rated each crossfade on a scale from 1 to 5, where 1 denotes a very good crossfade and 5 means its quality is very low. The ratings are averaged over all 12

users. For the DJ its selection, i.e. one selected crossfade for each of the sets of three crossfades, and for every other possible selection of 10 crossfades, these mean user ratings are averaged to give a measure for the overall quality of each of these selections. The mean user rating of the automatic DJ its selection is worse than 50.2% of the other possible selections, indicating no significant improvement over randomly choosing crossfades. However, in two of the ten crossfade sets, the DJ its choice received very bad user ratings compared to the other crossfades, and both these crossfades were very quiet and non-percussive in nature: this might indicate that the quality model seems to incorrectly estimate quiet music fragments their quality. If these crossfades are ignored, the average rating of the DJ its selection is better than 70% of the other potential selections. This test thus indicates that the quality model steers the automatic DJ towards selecting higher quality crossfades, at least for more percussive crossfades. However, its choice is clearly not optimal yet. Furthermore, this test has been conducted on a small number of users and with a small number of crossfade examples, and the users showed very varying preferences and sometimes gave very different ratings for the same crossfades. To conclude, the quality model might still need to be improved, and a test with more participants and evaluated crossfades should be conducted before any real conclusions can be drawn.

VII. Conclusion

The automatic DJ system presented in this paper is able to generate a seamless music mix from a library of Drum and Bass songs. It uses beat tracking, downbeat tracking and structural segmentation algorithms to analyze the structural properties of the music. These have been evaluated on two sets of music, one with 160 songs and a held-out test set of 220 songs. On the latter, a beat tracking accuracy of 98.2%, a downbeat tracking accuracy of 98.1% and a segmentation accuracy of 94.3% is achieved. This means that in total, 90.9% of the songs in the DJ system get structurally correct annotations. The extracted knowledge of the music structure is used in a rule-based framework, inspired by DJing best practices, to create transitions between the selected songs. The song and cue point selection is optimized by means of a model that estimates the crossfade quality. A small-scale subjective test points out that this model steers the crossfade selection in the right direction, but that it still needs improvement. Overall, the automatic DJ system is able to seamlessly join together Drum and Bass songs and create an enjoyable mix.

However, several aspects of this system can still be improved. Its main deficiency is that there is no tracklisting component, which could steer the track selection process by finding songs that are more likely to lead to an enjoyable crossfade, based on their musical properties. Professional DJs carefully select their songs to create a deliberate progression of theme and energy level throughout the mix, which the current implementation does not do. Secondly, the annotation modules do not have any built-in reliability measure. If the annotations are too uncertain, it might be better not to use that song in the mix. Finally, the crossfade quality model is too simple: it only evaluates individual measures and does not consider any long-term structures and dependencies in the music. A model that better understands what a good Drum and Bass mix sounds like, e.g. constructed using the abundance of professional Drum and Bass mixes that can be found online, might therefore lead to a more robust classification and a better sounding mix.

References

[1] Juan Pablo Bello, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B Sandler. A Tutorial on Onset Detection in Music Signals. *IEEE Transactions on Speech and Audio Processing*, 13(5):1035–1047, 2005.

[2] Dmitry Bogdanov, Nicolas Wack, Emilia Gómez, Sankalp Gulati, Perfecto Herrera, Oscar Mayor, Gerard Roma, Justin Salamon, José R Zapata, and Xavier Serra. Essentia: An Audio Analysis Library for Music Information Retrieval. *ISMIR*, pages 493–498, 2013.

[3] David Bouckenhove and Jean Martens. Automatisch Mixen Van Muzieknummers Op Basis Van Tempo, Zang, Energie En Akkoordinformatie. 2007.

[4] Judith C. Brown. Calculation of a constant-Q spectral transform. *The Journal of the Acoustical Society of America*, 89(1):425–434, 1991.

[5] Matthew E. P. Davies, Philippe Hamel, Kazuyoshi Yoshii, and Masataka Goto. AutoMashUpper: An Automatic Multi-Song Mashup System. *Proceedings of the 14th International Society for Music Information Retrieval Conference, ISMIR 2013*, pages 575—-580, 2013.

[6] Matthew E P Davies, Philippe Hamel, Kazuyoshi Yoshii, and Masataka Goto. AutoMashUpper: Automatic creation of multi-song music mashups. *IEEE/ACM Transactions on Speech and Language Processing (TASLP)*, 22(12):1726–1737, 2014.

[7] Matthew E P Davies and Mark D. Plumbley. Context-dependent beat tracking of musical audio. *IEEE Transactions on Audio, Speech and Language Processing*, 15(3):1009–1020, 2007.

[8] J. Foote. Automatic audio segmentation using a measure of audio novelty. *International Conference on Multimedia and Expo*, 1(C):452–455, 2000.

[9] Tatsunori Hirai. MusicMixer : Computer-Aided DJ System based on an Automatic Song Mixing. In *Proceedings of the 12th International Conference on Advances in Computer Entertainment Technology*, 2015.

[10] Hiromi Ishizaki, Keiichiro Hoashi, and Yasuhiro Takishima. Full-Automatic DJ Mixing System with Optimal Tempo Adjustment based on Measurement Function of User Discomfort. *ISMIR*, pages 135–140, 2009.

[11] Tristan Jehan. *Creating music by listening*. PhD thesis, Massachusetts Institute of Technology, 2005.

[12] Chuan-lung Lee, Yin-tzu Lin, Zun-ren Yao, and Feng-yi Lee. Automatic Mashup Creation by Considering both Vertical and Horizontal Mashabilities. *ISMIR*, pages 399–405, 2015.

[13] Heng-Yi Lin, Yin-Tzu Lin, Ming-Chun Tien, and Ja-Ling Wu. Music Paste: Concatenating Music Clips Based on Chroma and Rhythm Features. *ISMIR*, pages 213–218, 2009.

[14] Meinard Müller. *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer, 2015.

[15] Jouni Paulus, Meinard Müller, and Anssi Klapuri. Audio-based music structure analysis. *ISMIR*, pages 625–636, 2010.

[16] Michael Pilhofer and Holly Day. *Music theory for dummies*. John Wiley & Sons, 2015.

[17] David Robinson. ReplayGain specification., 2014.

[18] S. S. Stevens, J. Volkmann, and E. B. Newman. A Scale for the Measurement of the Psychological Magnitude Pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937.

[19] Stanley S Stevens. The Measurement of Loudness Level. *The Journal of the Acoustical Society of America*, 27(5):815—-829, 1955.

[20] John Steventon. *DJing for dummies*. John Wiley & Sons, 2014.

[21] W. Verhelst and M. Roelands. An overlap-add technique based on waveform similarity (WSOLA) for high quality time-scale modification of speech. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2(1):2–5, 1993.

Table of Contents

List of Figures

List of Abbreviations

ADSR Attack Decay Sustain Release, a model for the onset envelope of a musical sound

BPM Beats Per Minute, metronomical unit to describe the music tempo

CQT Constant Q Transform

DJ Disk Jockey

FFT Fast Fourier Transform

FSM Finite State Machine

HFC High Frequency Coefficient, an onset detection method

HMM Hidden Markov Model

IBI Inter-Beat Interval, i.e. the time between two consecutive beats

MFCC Mel-Frequency Cepstral Coefficients

MIR Music Information Retrieval

RMS Root Mean Square of a time series, calculated as $\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}$

SSM Structural Similarity Matrix

STFT Short-Time Fourier Transform

SVM Support Vector Machine

WSOLA Waveform Similarity Overlap-Add, an audio time-stretching algorithm

Chapter 1

Introduction

To listen to music, one could simply play songs back-to-back, i.e. play the first song, when that one finishes start the next and so on. However, this way of playing music is not always desirable. First of all, the listener might not want to hear each song from start to finish, and a quicker succession of songs is preferred to maintain a certain level of excitement and prevent the music from becoming boring. Secondly, playing songs back-to-back is not seamless, in the sense that in between songs, the audio is briefly interrupted. These remarks hold especially in night clubs for example, where the music should be played in such a way that the audience stays excited and can keep on dancing without interruption.

This is where the DJ, short for "disk jockey" [1], comes in. This is the person who mixes existing music records together in a seamless way. The art of mixing music like this is called DJing. He or she[1] is responsible for selecting which songs to play, and performing the seamless transition between them. Doing so is a non-trivial task, as this requires a perfect sense of rhythm, a deep knowledge of the music that is being mixed, the technical skills and delicacy to operate the DJing equipment and a good understanding of what the audience wants to hear and in what order songs should be played to achieve the desired level of excitement.

This introductory chapter provides a detailed analysis of the different tasks that a DJ needs to perform to create a seamless music mix using the songs in the input music library. This is done in Section 1.1, where the DJing task is split up into several subtask. This modular view provides a baseline for the architecture of the automatic DJ system. The scope of this thesis has been limited to one particular music genre, namely Drum and Bass. A short introduction into this genre and its properties is given in Section 1.2. An

[1] In the remainder of this master's thesis, male pronouns will be used to refer to the DJ. Please keep in mind that there are of course many talented female DJs, and that this is just a grammatical convention.

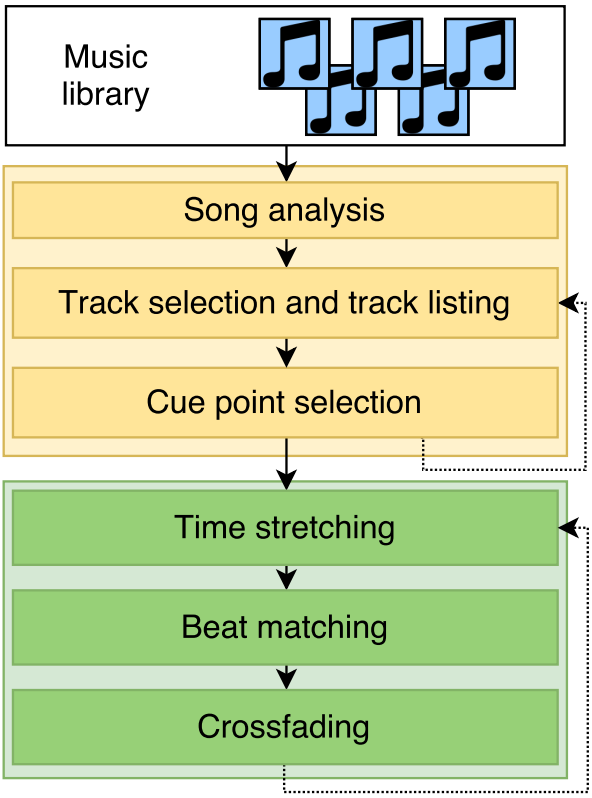


Figure 1.1: Illustration of the typical steps in the DJing process. The tasks in yellow are performed before the mixing starts. The tasks in green are performed live when mixing. Note that this is not a linear process and that tasks can be performed iteratively.

overview of the existing research on automatic DJ systems is presented in Section 1.3, and finally the requirements of the system developed in this thesis are specified in Section 1.4.

1.1 What is a DJ?

A DJ is a person who mixes existing music records together in a seamless way to create a continuous stream of music [1]. From an engineering perspective, a DJ can be abstracted as a system that takes as its input a collection of songs, and outputs a seamless mix generated using a selection of songs from the input library. To do so, the DJ needs to execute several steps, which are described below. Figure 1.1 illustrates the DJing process.

First, the DJ needs to **analyze the music** in the music library. This allows him to discover the genre and style of the song, the musical key, tempo, structure and other properties. Having a thorough understanding of the music that is available in the input library allows the DJ to make an appropriate selection of songs, as not just any pair of songs can be mixed together. Modern DJ equipment and software assists the DJ in performing this analysis by e.g. automatically annotating the tempo and rhythm of the

music, automatically discovering the musical key and so on.

After the music analysis, the DJ can start to compose the mix. A good DJ mix is more than a simple concatenation of random songs: not just any pair of songs can be mixed together, as a certain "musical compatibility" is required. This means that the songs must be similar to some extent in tempo, rhythm, musical key, instrumentation or other properties: mixing songs that are completely dissimilar will typically result in a bad mix. A good DJ mix also has a certain global structure. For example, the DJ will not start the mix with a very energetic song, but rather gradually build up the tension. Eventually, the energy in the mix will reach a climax, after which it is decreased again to give the crowd some rest. Then the tension is increased and decreased again, such that the DJ effectively manages the musical tension and energy levels throughout the mix. Some important DJing rules are described later in Section 1.2.

Selecting songs that fit together and ordering them such that the desired mix structure is obtained happens in the ***track selection*** and ***track listing*** steps. For each pair of songs that are mixed together, the timings need to be determined where they will overlap. The *cue point* is the time instant in a song where the DJ starts playing it: this can be the beginning of the song, but also somewhere in the middle. The term *cue point* will also be used to denote the time instant in the playing song where the new song is started. This is illustrated in Figure 1.2. Again, certain musical rules apply to which points in a song are appropriate cue points. This step will be called the ***cue point selection***. The track selection, track listing and cue point selection steps are closely related and their results depend on each other. For example, a DJ might have two songs that he wishes to play close to each other in the mix, but he might not find appropriate cue points to mix them. He could therefore select a new song that fits in between those songs, selects cue points in that song and then continues the process. The result of these steps is a tracklist, i.e. a list of the songs to be played, and the appropriate cue points for those songs.

The DJ has a tracklist and knows where the appropriate cue points are in the songs he wants to play: now, he can start *mixing*. The first song is played to start the mix. When the music reaches the cue point in the first song to start the second song, the second song is cued and starts playing. Here, it is important that both songs are ***beatmatched***. This means that both the songs their beats are aligned with each other. This is not possible if the songs have a different tempo: therefore, one or both songs might be ***time-stretched*** until both songs their tempi are the same. To appropriately blend together the songs, the

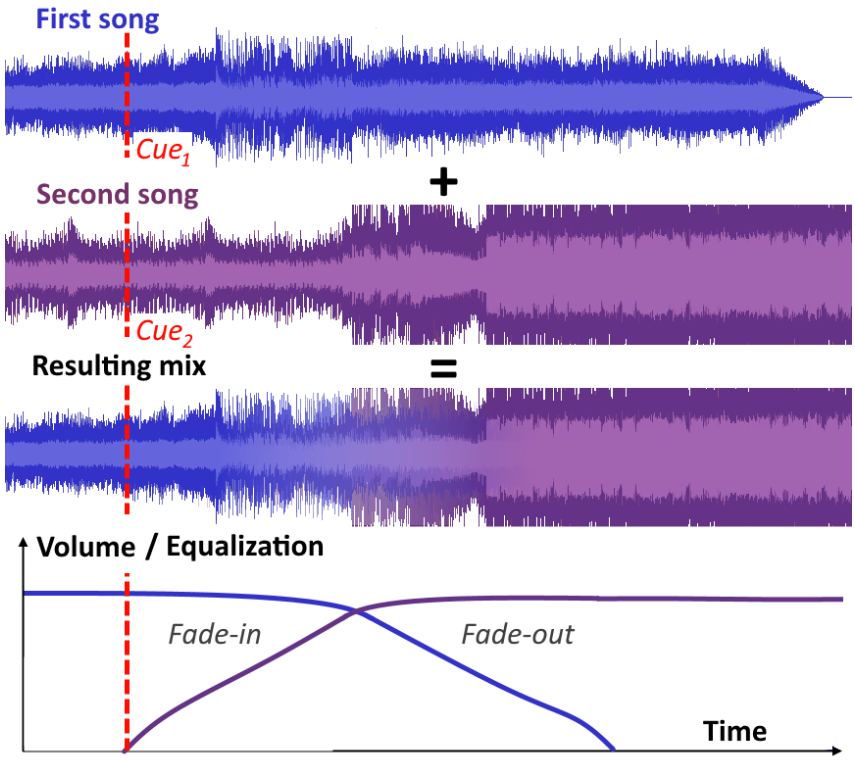


Figure 1.2: Illustration of the crossfade process. Also the two cue points are shown.

DJ uses equipment called a *DJ mixer*, with which the DJ has control over the volume and equalization settings of each individual song[2] [1]. When the second song starts playing, its volume is typically completely set to zero so that the audience cannot hear it yet. The DJ then gradually *fades in* the new song by increasing its volume and equalization settings. The first song is also *faded out* by decreasing its volume and equalization settings. Eventually, the second song is playing at full volume and the first song is completely faded out. This process of transitioning between two songs is called a ***crossfade***. The typical crossfade process is illustrated in Figure 1.2. The process of cueing, time-stretching, beatmatching and crossfading is applied for each transition between songs in the mix, effectively creating a continuous and seamless stream of music.

The steps above give an overview of everything a DJ needs to do to create a mix: music analysis, track selection and listing, cue point selection, time-stretching, beatmatching and crossfading. The DJ has many degrees of freedom to add a personal and creative touch to the mix, for example by an appropriate track and cue point selection or deciding how to perform the crossfade (fast or slow, using special effects or loops, etc.). Note that the described steps are not an exact and rigid work flow and that many steps are performed iteratively. For example, a DJ might adjust his tracklist *on the fly* because the audience is

[2]In the context of mixers, one 'song' is also often called one *track* or *channel*.

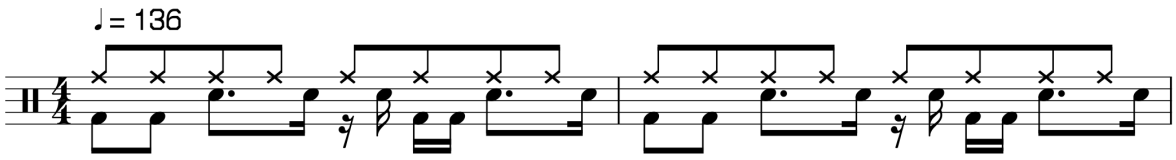


Figure 1.3: The first two bars of the Amen break, a frequently used breakbeat in Drum and Bass, in musical notation [3].

more excited than expected, so that he wants to play a more energetic song. Anticipating the expectations of the audience and being adaptive to these circumstances is also a very important aspect of DJing.

1.2 | What is Drum and Bass?

Drum and Bass (often abbreviated as *DnB*, *D'n'B* or *D&B*) is a particular branch of electronically produced music. It is characterized by fast percussive content and a prominent bass line, hence the name "*drum*" and "*bass*". Its tempo typically lies between 160 and 190 beats per minute [2]. Drum and Bass is also characterized by the heavy use of *breakbeats*. These are short samples of drum segments that are repeated as the rhythmic basis of the song. A particular example is the Amen break, of which variants are frequently used in Drum and Bass music. The Amen break in musical notation is shown in Figure 1.3. Even though many variants of breakbeats are used in practice, their use leads to the fact that Drum and Bass music has a very distinct and recognizable rhythmic pattern.

Drum and Bass is a very rich music genre. It has influences from many other genres, including (but not limited to) hip hop, jazz, rock, and other electronic genres such as electro, house, dubstep and hardcore. This leads to a great musical variety, and the Drum and Bass genre can be split up into different sub-genres. Examples of such sub-genres are liquid, neurofunk, jump up, jungle, minimal, halfstep and darkstep. Of course, the boundaries of these sub-genres are not strictly defined, and it is not always possible to classify one song as being either one genre or the other.

Drum and Bass is also very suited for creating DJ mixes. This is a consequence of the fact that most Drum and Bass has a predictable and constant tempo, prominent percussive content and many similarities between different songs such as the use of similar breakbeat patterns or instrumentation. Just like Drum and Bass is a diverse music genre, Drum and Bass mixes can widely vary in nature. The type of mix will be mainly determined by the track selection, e.g. if either energetic or rather relaxed songs are chosen.

However, also the pace of the mix and the used mixing techniques determine its dynamics and atmosphere. For example, in a more relaxed mix, the transitions are smooth and gradual, the time between two crossfades is quite long and rarely more than one song is playing at one instant (except during the crossfade, of course). However, in a more energetic mix, such as in a club environment, songs can follow each other in quick succession, and it is not uncommon to have two (or even three) songs playing at the same time. The crossfades can also be less gradual to introduce an element of surprise in the mix, which contributes to its energetic nature. Sometimes there barely is a transition, but rather one song is faded out almost instantly at a very carefully timed instant, while another is immediately faded in. This technique is known as *switching*.

Even though Drum and Bass mixes can be created in many ways, certain rules and best practices must be followed by the DJ. In general, these come from the notion that mixed music must still adhere to the musical rules and structure that define the original music. Some simple rules are defined below. These rules (and much of the information on the Drum and Bass genre in this and other chapters) originate from this master's dissertation's author his thorough knowledge of and experience with this music genre, but many of these observations are also acknowledged by other sources [1].

The first two rules apply to the crossfading process, i.e. the actions that the DJ should take when mixing two songs.

Rule 1.1. *Crossfaded songs must be beatmatched. This means that they have the same tempo during the crossfade and that their beats are aligned.*

Rule 1.2. *Appropriate equalization must be performed when playing more than one song.*

This first rule is very important: a mix that is not beatmatched sounds very messy, chaotic and off rhythm, leading to a very unpleasant result. Furthermore, when two songs are played at the same time, the bass and the treble, i.e. respectively the lower and higher frequency regions of the audio, of one of the songs should be attenuated to keep the mixed audio from saturating in these regions.

The following rules are related to the higher-level structure of the mix and the cue point selection. In short, cue points should align with structural boundaries in the music and in that way respect the hierarchical structure that music has. At the lowest level, music consists of notes. Certain notes are accentuated with a certain periodicity, defining the tempo and *beats* of the music. The beats of a song can be grouped together into

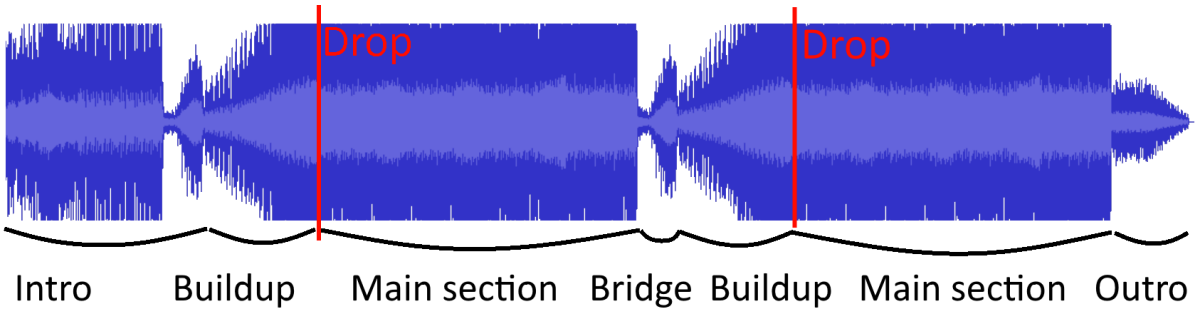


Figure 1.4: The typical structure of a Drum and Bass song

groups of four, which are called *measures*. The first beat of a measure is typically more accentuated than the others: this is called the *downbeat*. The measures form the basic building blocks for the music structure: they can again be grouped into larger segments called *phrases*, which are typically 4 or 8 measures long [1]. At the highest hierarchical level, several phrases (typically a power of two) make up the music *sections* which determine the compositional layout of the song. For example, contemporary pop music often has an intro, a chorus that is repeated several times, verses that separate the choruses, a solo, an outro, and so on. Drum and Bass music also has a typical structure, although different from pop music. The song starts with an *intro*, which typically has less expressive percussive and melodic content. During the intro, new elements, such as a bass line, drum pattern, melodies and vocals, are gradually added in order to 'build up' the music. New elements are typically added after a multiple of 4 or 8 measures. After some time, the intro will transition into the *build-up* section. This is the section where the musical tension is gradually increased. The build-up culminates in the *drop*, which is the moment where the built-up energy is released. The drop is the beginning of the *main section* of the song, comparable to the chorus in vocal-based music. Eventually, the main section will quiet down and go over into a *bridge*, i.e. a quieter section comparable to the intro that connects the first main section and a second build-up, a second drop, and a second main section. Finally, the *outro* gradually brings the music to a close. This typical structure is shown in Figure 1.4. Note that this is merely a generalization of the typical structure of Drum and Bass music, which is by no means strictly defined, and variations are always possible. However, personal experience with the genre shows that in practice, most Drum and Bass more or less follows this structure.

Related to the musical structure are the following rules:

Rule 1.3. *Songs should be downbeatmatched. This means that their downbeats should align with each other when played together.*

Rule 1.4. *Cue points should align with structural boundaries in the songs.*

If the downbeats of the mixed songs are not aligned properly, the mix will also sound off-rhythm, even though this will sound less disruptive than an inappropriately beat-matched song. Also when selecting the cue points, the DJ should take care that important structural boundaries of both mixed songs align in the output mix. For example, if a song is faded in starting at the beginning of the build-up, then that structural boundary should align with a phrase boundary in the song it is mixed with, and not with another downbeat somewhere in between two phrase boundaries. If this is not the case, the musical structure in the output mix will be inconsistent, which will not sound as nice. A technique that is often used is *double dropping* [1], where songs are aligned such that their *drops* align, i.e. they *drop* at the same time.

Rule 1.5. *When creating a tracklist, the DJ should select songs that are compatible harmonically, rhythmically and thematically.*

This rule means that not just any two songs can be mixed together: special attention must be paid to whether the songs are for example in the same or in a compatible musical key. If this is not the case, it is more likely that the mix will not sound very well. Songs should also fit together thematically: for example, a highly energetic and percussive neurofunk song will most likely not mix well with a very gentle liquid song. However, the best practices for song selection are very vague and not easily not put into fixed rules.

These five rules are an enumeration of some important best practices for creating a good DJ mix. However, it is not a complete list of all rules that a DJ should consider: for example, he might also want to avoid playing two songs with overlapping vocal sections, or he might want to have some variation in his mix instead of playing too many songs of the same artist or sub-genre in a row. Nonetheless, in this thesis, the rules specified above are considered to be the basic requirements for a good mix and most of them will be considered when implementing the automatic DJ system[3].

1.3 | Related work

In this section, an overview is presented of the existing work in scientific literature related to automatic DJing systems, which unfortunately remains very scarce. The existing

[3]Because of the limited time available in this thesis, the track selection in the prototype is still very limited, so that the last rule is not currently implemented. However, it was considered important enough to be included in the list of DJing best practices.

systems can be divided into two main categories, namely DJing systems and mashup systems.

1.3.1 | Existing research on DJ systems

The idea of an automatic DJing system is not new. In [4], Dave Cliff describes a coarse system architecture of an automatic DJing system. His proposed solution consists of a track selection and ordering step, a cue point selection step, time-stretching, beatmatching and crossfading. In a second paper [5], Cliff describes an improved system that is able to exploit crowd feedback in order to optimize the song selection throughout the mix. More specifically, it determines whether the tempo should slow down, go up or stay the same, depending on how the crowd reacts to the music. Here, he also acknowledges the possibility to improve the quality of the crossfade by allowing more complex fading patterns, e.g. by using equalization filters. However, both documents only *describe* an automatic DJ system and do not give any details on how this idea could be implemented.

In a section of his PhD thesis [6], Tristan Jehan proposes a DJ system that automatically downbeatmatches and crossfades songs. However, the system remains very simple. For example, the cue point detection does not rely on structural boundary detection, but instead chooses the fade section by detecting similar rhythmic patterns in both songs. Additionally, no specific crossfading or track selection techniques are mentioned.

In his master's thesis from 2007 [7], David Bouckenhove describes a system for automatically creating a DJ mix, with a specific focus on vocal activity detection by means of a support vector machine classifier. This allows to create a mix where the vocal segments of two songs will not be overlapped as this might disturb the listening experience. The system is not restricted to a specific music genre. Song selection in this system is based on a measure for key and tempo similarity. It uses the relation between the dominant keys of the songs and the music theory concept of the *circle of fifths* to create a harmonic mix. The algorithm that determines which parts of two songs are overlapped avoids vocal overlap and additionally takes the music its energy into account. Different crossfading patterns are allowed, but they are simple variations on linear crossfading, and no music equalization is done. Additionally, the system does not take downbeats or the higher-level structure of the song into account when beatmatching or for cue point selection.

The Music Paste paper by Lin et al. [8] focusses on optimizing the tracklisting and cue point selection process. The system determines the best cue points and time between

two crossfades by splitting up the songs in segments and maximizing the rhythmical and chromatic similarity between the transition segments. This process works in several steps. First, the songs are analyzed and musical features are extracted. Then the song collection is filtered by removing song pairs where more than half of the music segments are dissimilar based on their loudness and tempo. To determine the optimal song ordering and cue points, a distance matrix based on these features is constructed and a path between songs is determined using a greedy algorithm that attempts to minimize the average distance between features throughout the path. Finally, the songs are beatmatched and crossfaded. The length of the transition is determined by observing the smallest rate of tempo change that can be aurally detected, and by ensuring that the rate of change of the system doesn't exceed this limit. Therefore, a transition with a larger change in tempo implies a longer transition, as the rate of change is bounded. In this system, the structural properties of the song were not considered for cue point selection or when determining the length of the crossfade. Additionally, the volume fading happens in a very basic linear way. Finally, the length of the transition is chosen so that the tempo change is barely audible, but this might not be necessary, as a DJ might also use fast tempo transitions intentionally, e.g. to introduce an element of surprise in the mix.

In their work from 2009, Ishizaki et al. [9] have developed a DJ mixing system that uses a custom measurement function for user discomfort. The system is fully automatic in the sense that it can create an endless stream of mixed songs without any intervention. The proposed solution mainly addresses the problem of performing tempo changes of songs while maintaining an optimal level of user comfort during those transitions. They conducted a subjective experiment to determine a threshold for the relative tempo change at which the listening experience is disturbed. The authors then define a measurement function for user discomfort, which is used to determine an intermediary tempo that lies in between the tempi of the mixed songs such that the discomfort is spread evenly between the first and second song and the overall user discomfort has a minimal value. The first song its tempo is then adapted to this intermediary level, after which the crossfade to the second song takes place. Finally, the second song is stretched to its original tempo. Additional care was taken to deal with half-tempo or double-tempo errors. To improve the mix quality even further, the authors perform a simple beatmatching step.

Finally, MusicMixer, developed by Hirai et al. [10], improves the track and cue point selection process by means of two similarity measures, namely beat similarity and latent topic similarity. The beat similarity describes how similar the songs are in beat structure

and it is calculated by comparing the inter-beat distances. These are the distances between the peaks in a low-pass filtered version of the music, which should highlight bass drum and snare drum events. If the patterns of the inter-beat distances in the two songs are similar, then the beat similarity is high and songs are more likely to be matched together. The second measure, latent topic similarity, is the main novelty of this system. It attempts to describe the chromatic content of the music at a higher abstraction level. For this, the authors draw inspiration from natural language processing, where the latent topic of a sentence is derived from the words that appear in that sentence. In the case of music, the topic is extracted from a music extract, a 'sentence', given a bag-of-features descriptor of the musical features, which can be interpreted as the musical 'words'. These are extracted from a chromagram of the music, which describes the tonal content of the music. The authors furthermore evaluate their latent topic similarity approach in a small subjective evaluation, which shows promising results.

1.3.2 | Existing research on mashup systems

Mashups are new, individual songs that are generated by overlapping and interweaving extracted fragments from existing songs. Small extracts of those songs are played in quick succession, while one or more songs are played in the background to provide a musical "basis". This effectively creates a new song composed by "mashing together" existing music, as illustrated in Figure 1.5 [11].

Mashups and DJ mixes clearly have a different purpose: a mashup aims to create a *new* song by combining quite short fragments of the composing songs, while a DJ mix will typically play larger parts of the songs. As a consequence, a mashup is much shorter than a DJ mix, and the songs are more heavily modified by cutting and pasting fragments from them. However, there are some analogies between a DJing system and a song mashup system. Both require a thorough understanding of the music they are working with and use some form of song selection and track listing, cue point detection, beat tracking, beatmatching and crossfading to create pleasing transitions between or overlappings of the different songs. For this reason, it is still interesting to investigate mashup systems too, of which several have been proposed in literature.

A first example is AutoMashUpper, created by Davies, Hamel, Yoshii and Goto. A first version was created in 2013 [11], which was improved in 2014 [12]. The first step in the AutoMashUpper system is to divide the input song and the songs in the music library into phrase-level segments. For this, a self-similarity matrix is calculated on a

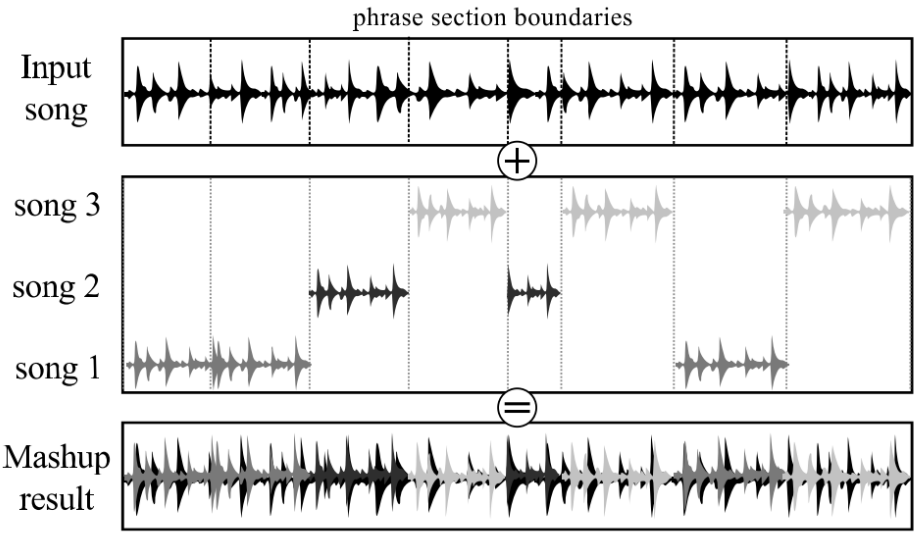


Figure 1.5: Conceptual illustration of multi-song mashups (from [11]). Short pieces of music are extracted from several songs and overlapped with one or more background songs, creating a new song.

downbeat-aligned semitone spectrogram of the song, after which the phrase boundaries are extracted using the approach of Foote [13] and aligned using a regularity-constraint. Afterwards, the system finds mashup segments that fit the segments of the input song as well as possible. This is based on the new concept of *mashability*, which is a measure based on the harmonic and rhythmic compactibility and the spectral balance of the songs. Finally, the mashup is created. This involves a time-stretching step to beatmatch the fragments, a pitch-shifting step to harmonically align them and a loudness balancing step. AutoMashUpper also includes a user interface which allows to interact with the mashup system.

The authors of the AutoMashUpper paper used similar techniques as in that project to create a real-time musical accompaniment system to accompany live musical input [14]. While this is not a DJing system, it shows that the potential of these and similar systems goes beyond DJing music itself. Another interesting application is improving a user his running or walking experience by creating an automatic mix that adapts to the user his tempo [15].

The paper by Lee et al. [16] focuses on improving the track selection in mashup systems and extending them to multiple base songs, instead of only one as in the AutoMashUpper system. It does not only consider the vertical mashability, i.e. how well songs played at the same time fit together, but also the horizontal mashability, i.e. the relation between

the consecutive song snippets. Additionally, two new mashability measures have been proposed in order to get a better estimate of the song compatibility.

Other work [17] [18] focuses on the generation of *medleys*, which are new songs generated by concatenating fragments as opposed to overlapping them as in mashups [17]. In the cited examples, the structural boundaries of the audio are considered when segmenting the music, as opposed to many mashup systems discussed in this overview. However, even though they also perform beatmatching and cross fading, they still lack many essential components to be considered an automatic DJ system.

1.3.3 | Commercial DJing software

There also exist many commercial DJing applications. Examples of professional DJing software include Serato DJ [19] and Traktor Pro 2 [20]. These applications typically come at a price of 99 euro and more. Free DJing applications also exist, such as VirtualDJ [21] and Mixxx [22, 23]. The latter is an open source project. These software solutions aid the DJ in analyzing the music in his library and discover e.g. the tempo, beat and downbeat positions and the musical key. However, most of this software is tailored to DJs who perform the mixing themselves, where they typically control the application using advanced DJing equipment such as turntables and mixers. Some of these solutions feature automatic DJing functionality, but this is often implemented in a simple and naive way. Experimentation with the VirtualDJ software has shown that, at least for Drum and Bass, the beat and especially the downbeat tracking is far from optimal, because for many songs the annotations are not correct. Furthermore, its built-in automatic DJ does not perform cue point selection based on structural segmentation techniques, but typically simply fades in the next song when the current one is almost finished. Hence, many aspects of these commercial systems can still be improved before considering these applications fully-featured automatic DJing software.

1.3.4 | Drawbacks and shortcomings of existing work

Most research focuses on optimizing the crossfade process by devising a measure for the compatibility of the mixed songs, such as AutoMashUpper with the mashability measure, MusicMixer with its measure based on latent topic similarity, or the fully automatic DJ system by Ishizaki et al. which minimizes user discomfort during transitions. However, there are still some important drawbacks to these systems:

- Crossfading in these systems is typically realized by simply fading the volume of one

song in while fading the other one out in a linear fashion. None of these approaches mention an equalization step to prevent bass or treble saturation in the output mix.

- Songs are typically mixed back-to-back, i.e. that a new song is faded in near the end of the current song. This implies that only one song will be playing at each moment except during the (typically short) crossfades. In a real DJing scenario, and especially in Drum and Bass, it is possible that the DJ plays two or more songs at the same time, for example when performing a double drop.

- Most approaches do not consider the structural boundaries when selecting cue points. Even worse, most approaches do not detect the downbeats of the song. The MusicMixer approach does not even perform any time-stretching to explicitly match the songs their tempi.

These remarks also hold for commercial DJing software that features an automatic mixing feature. In short, there appears to be no completely integrated automatic DJ system that elegantly combines all necessary components. Existing approaches only focus on improving very specific problems and do not offer a generic solution to the problem, or only offer very basic DJing capabilities and limited performance in terms of cue point selection, (down)beatmatching or crossfading techniques.

1.4 | Project scope

The goal of this thesis is to develop a fully automatic DJ system for Drum and Bass music. It uses artificial intelligence components to perform track selection and determine cue points in those tracks. In detail, the system will be able to:

- autonomously analyze songs in the input library and annotate them with tempo, beat, downbeat and structural boundary information. The songs may be in `mp3` or `wav` file format. The system does not use any metadata such as the artist name, song title or genre to perform this analysis, but it does assume that the input music its genre is Drum and Bass.

- perform time-stretching, beat and downbeatmatching of the mixed songs (rule 1.1 and 1.3)

- perform equalization of the mixed songs (rule 1.2)

- perform multiple types of crossfades to add some variety to the mix

- autonomously select tracks and cue points to create a pleasing mix, but still respect the structural properties of the music (rule 1.4)

The subsequent chapters of this thesis describe how this system is realized. The first chapters discuss how the relevant musical information can be accurately extracted from the raw audio. Chapter 2 describes how to find the tempo and beat positions. Chapter 3 builds upon this result to extract downbeat information. Chapter 4 elaborates on how the higher-level structure is discovered in the music using the downbeat annotations. Chapter 5 describes the tracklisting and cue point selection process, which is done using the provided segment annotations and a machine learning classifier that measures the music quality. It also provides an overview of the entire system and the mixing process as a whole. This thesis is then concluded and some suggestions for future work are discussed in Chapter 6.

Chapter 2

Beat tracking

Music is a very rhythmic audio signal. Musical events do not occur randomly, but follow very strict timing patterns with a certain periodicity. Certain notes are more accentuated than others, which are called the *beats*. When listening to music, people tend to automatically tap along with the beats. One can tap slowly or quickly: this is what determines the tempo of the music. The tempo need not be constant: it can vary throughout the song. The tempo is expressed in *beats per minute*, BPM in short.

For a DJ or an automatic DJ system, an excellent notion of the tempo and the beat locations is of great importance. When a DJ plays two songs at the same time, e.g. during a cross-fade or a double drop, the DJ needs to make sure that the beats of the two songs are aligned perfectly in time. If this is not done properly, the mixed music will sound cluttered, incoherent and *out-of-sync*, which is absolutely undesirable in a DJ mix.

The process of aligning the beats of the mixed songs is called *beatmatching*. There are two necessary conditions before two songs can be beatmatched. First of all, they need to have a constant tempo, at least during the period they are being mixed. This implies that the beat positions are equidistant in this portion of the music. This distance between two beats is called the *inter-beat interval*. Secondly, the songs need to have the same tempo. If this is not the case, a drift will occur, where the slower song its beats will start to lag behind those of the first song. This is illustrated in Figure 2.1a. Luckily, these requirements are not so restrictive for a DJ. Most popular music has a constant tempo, so that the first requirement is met. This especially is the case for electronically produced music such as Drum and Bass, where typically all beat positions are at exactly the same distance from each other, since there are no deviations caused by timing irregularities which can occur with human musicians. Additionally, there exist efficient algorithms with which the DJ can stretch or shrink music fragments in real time, effectively manipulating the tempo of

the song without altering its pitch. This allows the DJ to modify the tempi of both songs and correctly beatmatch them. These so-called time-stretching techniques are discussed in more detail in Chapter 5.

This chapter discusses how the presented automatic DJ system tackles the problem of beat tracking, i.e. finding the correct beat positions in a piece of music. First, an analysis of existing approaches is presented in Section 2.1. These techniques are applicable to a wide range of musical genres, but their generality appears to be a drawback, making them too inaccurate to properly beatmatch two Drum and Bass songs. Therefore, a custom algorithm has been constructed, which is the subject of Section 2.2. Even though this beat tracking algorithm is much simpler and less flexible than techniques proposed in literature, it results in very accurate beat annotations for Drum and Bass music, because it exploits the assumption of a constant tempo throughout the song. The proposed approach is evaluated in Section 2.3. Section 2.4 summarizes the obtained results and Section 2.5 discusses some possible extensions and improvements.

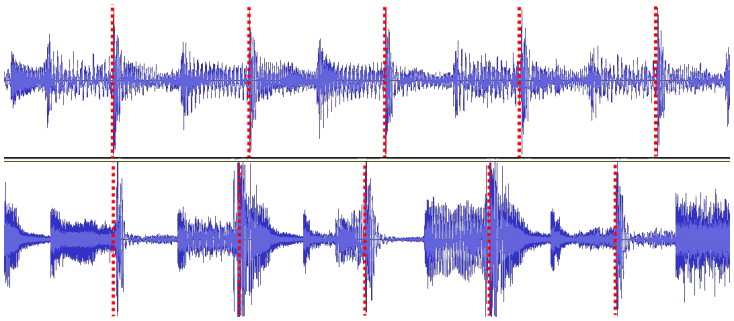
2.1 State of the Art analysis

This first section investigates what techniques already exist to perform beat tracking. First, an introduction is given into *onset detection*, a technique that is frequently used in beat tracking systems. Then, an overview of the current state of the art in beat tracking is given. Unfortunately, these techniques are not suited for an automatic DJ system implementation, which is motivated in Section 2.1.3.

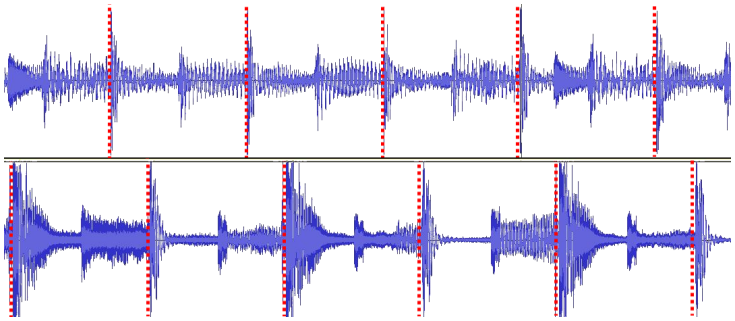
2.1.1 Onset detection functions

Music is an audio signal built from the superposition of repeating rhythmical patterns of musical 'events', i.e. the different notes or sounds being played. For example, a Drum and Bass song could be a superposition of a piano melody (a pattern of piano 'events'), a bass line (a pattern of bass 'events') and a drum rhythm (a pattern of percussive 'events'). Determining the location of these events is a first important step for many beat tracking algorithms, because the specific location of these events and the period at which they are repeated are related to the beat period and phase, as explained in Section 2.2.

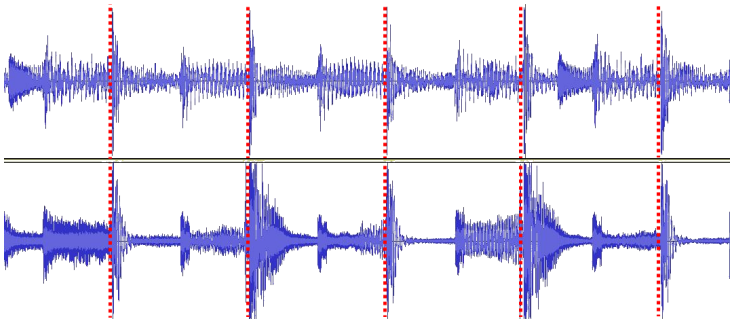
However, sounds played by an instrument can be heard during a specific time period, so it might not be immediately clear how to assign a single time annotation to it. Figure 2.2a shows the typical shape of the *envelope* of a note, i.e. the shape of the smooth



(a) Tempo error: the second song is faster and starts to run ahead of the first

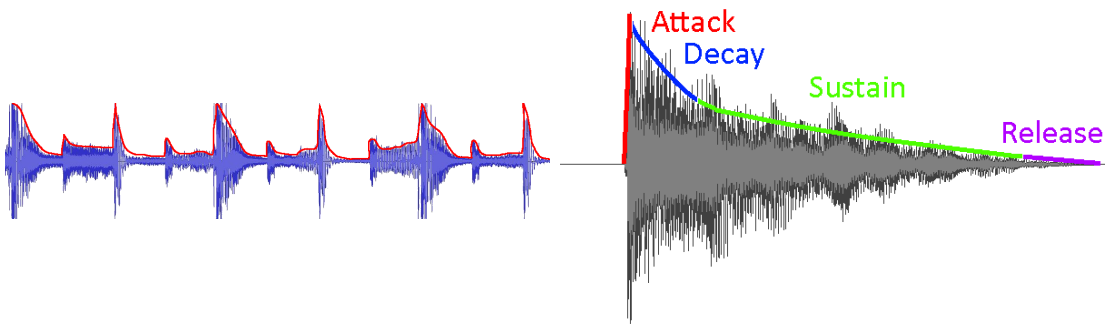


(b) Phase error: the tempi of both songs match, but they are not aligned in time



(c) Correct beatmatching of both songs

Figure 2.1: Illustration of the effects of poor beatmatching. Two waveforms are shown with their beat annotations. In the first case, the tempi of both songs do not match. This results in a drift of the beat positions of the first song with respect to those of the second song, leading to a gradual degradation of musical quality over time. The second case shows a situation where both songs have the same tempo, but where the beat positions do not match. The third case shows the result of correct beatmatching: all the beat positions of the mixed songs align perfectly.



(a) Illustration of the sound envelope of the audio waveform

(b) Illustration of the ADSR model

Figure 2.2: Illustration of the musical concepts of the sound envelope and the ADSR model.

curve that encompasses the maximum amplitudes of the waveform over time (Figure 2.2a). The envelope is typically described using the ADSR model [24], which describes it as consisting of an Attack, Decay, Sustain and Release phase. This is illustrated in Figure 2.2b. When a note is played, then typically there is a sudden burst of energy. The period in time where the sound envelope increases to a maximum is called the *attack* phase. The attack is followed by the *decay* phase, in which the amplitude envelope decreases a bit. After this, the energy of the sound can stay constant or decrease slightly over time: this is called the *sustain* phase. Eventually, the musician stops playing the note (e.g. by releasing the piano key) so that the vibration quickly stops: this is the *release* phase. Note that the relative durations and velocities of these phases differ depending on the type of instrument: for example, an instrument like a violin produces sounds with a long sustain, while percussive instruments like a drum will barely have a sustain phase at all.

The onset location is the single time instant that annotates the position of a musical event. Using the ADSR model, the beginning of the *attack* phase of the sound, or the moment where it can first reliably be detected, is considered to be a good definition for this instant [24]. Extracting the onset instants from music is not straightforward however, since music is a complex, polyphonic piece of audio with different instruments playing simultaneously. Most onset detection algorithms therefore do not extract the onset instants from the raw audio waveform. Instead, they first transform the audio into an intermediary representation that highlights significant changes in the audio. This representation is called an *onset detection function*, or sometimes a *novelty detection curve*. From the onset detection curve, the onset locations can then be determined using peak-picking [25].

To calculate an onset detection function, the input audio is typically split up into

windowed overlapping frames [25, 26]. Each frame might be transformed into another representation, e.g. by calculating the STFT or FFT of it. Afterwards, these features are used to assign a value to each frame, either based on that frame in isolation or by calculating some distance measure between successive frames. This leads to the onset detection curve, where a high value indicates that a time instant is likely to be an onset.

There exist many different types of onset detection functions, each with their specific advantages and applications. An overview of several variants is given by Bello et al [25]. They make the distinction between two major types of functions: either they are based on signal features, or they are based on a probabilistic model. As the latter type of functions is not used in this thesis, it will not be discussed here. The interested reader is referred to the discussion by Bello et al. for more details.

The simplest signal-based onset detection functions operate on **temporal features**, i.e. without transforming the audio signal to the frequency domain. These techniques assume that an onset is usually accompanied by a sudden increase in the signal its amplitude. They are typically implemented by first smoothing, i.e. low-pass filtering, the signal, followed by taking the derivative (or the first order difference for discrete time signals) to detect large differences between two time instants. These then correspond to onsets. While time-domain techniques are simple and computationally efficient, they typically do not perform well in the presence of simultaneous sounds, in which case their energy envelopes overlap so that their exact onset times are much more difficult to distinguish.

A second feature-based class therefore relies on **spectral features**, which look for significant changes in the frequency domain. An example of a spectral method is the *high frequency content* (HFC) function [27], which first calculates the short-time Fourier transform (STFT) of the signal, and then sums up the energies in sub-bands of that transform while weighting each energy component linearly depending on its frequency. High frequencies are get a higher weight whereas low frequency components are weighted less. The HFC function is successful at detecting percussive events and works well for complex music mixtures, but does not deal so well with the onsets of low-pitched and non-percussive sounds [25]. The HFC function furthermore only depends on the instantaneous short-term spectrum of the signal and does not take into account its temporal context. Alternatives are for example functions that calculate the *spectral difference* [28], also called the *spectral flux*, between adjacent short-term magnitude spectra. This hap-

pens by taking the L_1 or L_2 norm of the difference between the two spectral magnitude vectors. Sometimes the norm of the *half-wave rectified* difference is calculated, i.e. where all negative entries in the difference vector are set to zero. This has the effect that only the frequencies contribute to the norm where an increase in energy is observed, so that indeed onset events and not *offset events* are accentuated, i.e. the releases of the individual sounds. The spectral difference techniques consider both changes in the entire spectrum as well as more localized changes in certain subbands, and can therefore alleviate the shortcomings of the HFC method. They typically are a good onset detection function for many types of music, whether they are percussive or not. However, they still rely on magnitude information and can thus neglect onset events that do not show a strong energy increase, such as with low notes or transitions between harmonically related notes of bowed instruments such as violins.

All the approaches mentioned above extract their information from the magnitude of the spectrum. However, as mentioned, these techniques do not detect onset events that are not accompanied by large changes in energy. Also the phase information might be relevant for onset detection, as e.g. a sudden change in phase can also indicate an onset event. Methods exploiting both the phase and magnitude information exist [29] and they are successful at detecting both low- and high-frequency onsets regardless of their intensity. However, these methods are more computationally expensive than others and do not perform so well for percussive sounds and complex mixtures, where many phase distortions occur.

2.1.2 | Existing work on beat tracking

BeatRoot [30, 31] is an interactive beat tracking and visualization system developed by Simon Dixon and Chris Cannam. It is freely available on the project website [32] and it has been used in existing research on automatic music mixing systems [8, 33]. The BeatRoot system has been revised in 2007, improving both its onset detection method and its user interface [31]. The BeatRoot system works in two steps: a *tempo induction* step and a *beat tracking* step. The tempo induction step detects onset events in the song, which are subsequently analyzed to find multiple tempo hypotheses. The onsets locations are determined by calculating the spectral flux onset detection function of the audio and performing local peak picking. The result of this step is thus a sequence of onset times. Then, the algorithm computes calculates *inter-onset intervals* (IOIs), i.e. time intervals between pairs of (not necessarily successive) onsets. The IOIs are clustered using a greedy algorithm, so that IOIs with approximately the same length end up in the same cluster.

These clusters are given a weight, depending on the number of intervals in it and the relation of the cluster with respect to the other clusters, i.e. whether the cluster IOI lengths differ by an (approximately) integer factor. This results in a number of tempo hypotheses, which are passed to the beat tracking subsystem. The beat tracking uses a multi-agent architecture, where each agent represents a specific tempo and phase of the beats. The agents attempts to find events in the music that match their prediction of the beat locations, which are estimated using the beat tempo and phase assigned to that agent. The agreement between the predicted and actual beat times is evaluated for each agent based on how evenly spaced the beat times are, how many predicted beats correspond to actual events and how prominent the matching onsets are in the audio signal. The resulting beat sequence is then the output of the agent with the highest evaluation score.

The beat-tracking approach by Davies and Plumbley [34] uses an onset detection function to extract the beat locations, similar to many other beat trackers, including BeatRoot. However, contrary to BeatRoot, their system does not explicitly extract the onset locations from this function, as this is an error-prone step subject to false positives and missed detections. Instead, their algorithm directly works on the onset detection curve. Additionally, the algorithm does not analyze the music signal in different frequency subbands, unlike other beat tracking systems, which makes it computational more efficient. This is possible because they use the complex spectral difference onset function that already summarizes spectral changes of the audio, and therefore eliminates the need to do an explicit multi-band analysis. It furthermore accentuates both percussive and softer tonal onsets.

The beat-tracking system operates in two states. The first is called the *general state*, which is a memory-less state that operates on individual overlapping audio frames. These frames are 6 seconds in length and have a 75% overlap. The tempo is assumed to be constant in a given frame. Using this assumption, the beat period and phase are extracted in a two-step process. The first step is to extract the beat period by calculating the autocorrelation of a smoothed, half-wave rectified version of the onset detection curve. Because musical events tend to repeat at multiples of the beat period, this autocorrelation is high at offsets that are a multiple of that. For each possible tempo, the algorithm then calculates the sum of the autocorrelation function values at equidistant locations corresponding to that specific beat interval. The tempo of the fragment is then selected as the tempo that leads to the highest sum. The second step is to calculate the phase of the beats, which is extracted by applying an analogous summation approach on the

onset detection function. Searching the beat period first and then the corresponding beat phase is more efficient than doing a combined search of both parameters, as this reduces the problem from a two-dimensional search space to a one-dimensional one.

The general state is designed to guide the second state, called the *context-dependent* state, which is used to ensure beat continuity across frames and deal with a varying tempo. The general state namely operates in a memory-less fashion without any guarantee of continuity between frames. Therefore, when the context-dependent state detects that the predictions by the general state are consistent across multiple frames, it will take over the beat tracking so that errors are prevented that could occur with the general state alone. If the continuation by the context-dependent state becomes inconsistent with the general state prediction for that part of the music, and if that inconsistency is explained by a tempo change, then the algorithm switches back to general state analysis in order to adapt to this tempo change.

The following two beat tracking approaches are beat trackers implemented in the Essentia library [35], which makes them potential candidates for an easy integration with the automatic DJ system.

The approach by Degara et al. [36] uses a first-order hidden Markov model (HMM) to perform beat tracking. Their approach exploits beat and non-beat information by means of an onset detection function, and the use of a HMM allows the system to handle changes in tempo or detect small irregularities in the individual beat positions. This algorithm has been implemented in the Essentia library as the `TempoTapDegara` function.

In a first stage of the algorithm, a frame-wise estimate of the local tempo is calculated in a similar way to the general state beat tracking of the approach by Davies and Plumbley [34], i.e. by calculating sums of the half-wave rectified autocorrelation of the onset detection function for different inter-beat interval candidates. The complex spectral difference onset detection function is used. This first tempo estimate is then used in the actual beat tracking step, where a first-order HMM models the time since the last visit to the 'beat state'. The allowed internal state transitions of the HMM are a transition to the successor state, or a transition to the beat state. The observable variable is the value of the onset detection function. The observation likelihoods and transition probabilities of the HMM are estimated using simple and intuitive rules, instead of learning them from training data. Experiments show that these simplifications still lead to adequate results. The goal of the HMM algorithm is to determine the maximum posterior likelihood of the hidden state sequence, given the observed onset detection, which is done using the Viterbi

algorithm. The sequence of beat instants is determined as the time instants where the HMM visits the 'beat state'. By using a HMM that attempts to explain the evolution of the onset function, this system effectively exploits information at non-beat states, leading to a reported equal or superior performance compared to other beat tracking methods.

Apart from creating a new beat tracking system, the authors also studied if it would be possible to predict poor performance of the beat tracker on a given piece of music. Three measures are proposed to assess the quality of the onset detection signal used for the beat tracking. These quality measures are then used to assess the reliability of the beat tracking algorithm by means of a K-nearest neighbor regression method. Estimating the expected performance of a beat tracker allows to create a system that combines different trackers depending on their expected performance, leading to a better accuracy overall.

The multi-feature beat tracker by Zapata et al. [37] is the second beat tracker implemented in the current release of Essentia, namely as the `BeatTrackerMultiFeature` function. The beat tracker is based on the result that the mutual agreement of multiple beat tracking systems surpasses the accuracy of any of these systems individually. However, running multiple beat trackers in parallel is computationally demanding, which is why the proposed system is a single, standalone solution where multiple, diverse input features pass information on to a single beat tracker. It then selects the most likely beat positions using a multiple agreement system.

Like the other systems discussed here, the system uses continuous onset detection functions as a mid-level representation of the music. However, this system does not use just one, but five onset detection functions, chosen after the careful evaluation of nine onset function with respect to their computational complexity and their complementary descriptive value, i.e. how much information they add to the system, with as little redundant information as possible. This ensures the robustness of the algorithm for many types of music. For each onset detection function, a set of beat locations is calculated using the method of Degara et al [36], which was described earlier in this section. These beat sequences are then compared using a mutual agreement method, using which the optimal beat sequence is selected. Additionally, the mutual agreement measure provides a measure for the confidence in the detected beat positions.

2.1.3 | Discussion and evaluation of existing approaches

This exploration of the state of the art shows that existing approaches attempt to deal with varying tempi and multiple music genres. However, for the development of this automatic DJing system, their generality proves to be a drawback. Informal analysis shows

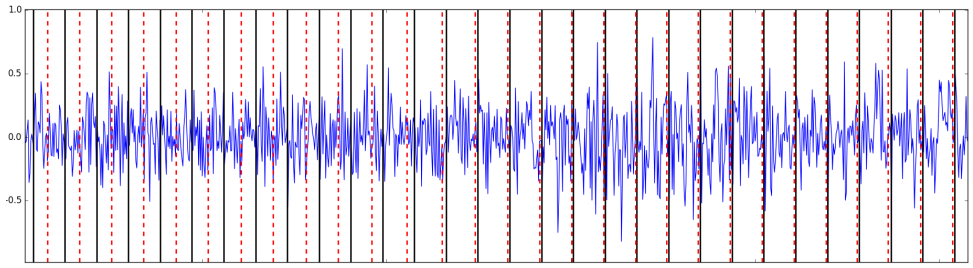


Figure 2.3: Comparison of beat annotations made using an existing beat tracking approach and the correct annotations. The annotations in red dashed lines were made using the `MultiFeature` beat tracker, implemented in the Essentia library. The black annotations are correct. The inter-beat intervals of the detected annotations fluctuate: at the beginning of the extract, the annotations are not correct, while later on they synchronize with the correct ones. However, further on in the music, the detected beats will again deviate from the correct values, making this approach unreliable for an automatic DJ system.

that the detected beats deviate from their actual position and are not exactly equidistant, even though a constant tempo is expected for Drum and Bass music. Both the BeatRoot algorithm and the algorithms implemented in the Essentia library have been experimented with and showed this drawback. Furthermore, in the parts of the song where there is no prominent beat, such as in the intro or bridge, it was found these beat tracking algorithms made significant errors. Figure 2.3 provides an illustration of this result. Attempts to fix these errors, e.g. by filtering out unlikely detections and performing some averaging and filtering steps on the remaining beat annotations to extract the constant tempo and phase of the beats, proved to be too ad-hoc and did not give any reliable results.

In the remainder of this section, a much more elegant algorithm is described, based on the work of Davies and Plumbley [34]. It exploits the assumption that the song tempi are constant throughout the song. This will lead to a much more stable and accurate beat tracking algorithm that can be used to construct the automatic DJ system.

2.2 Beat tracking system for constant-tempo Drum and Bass music

This section describes the implementation of the beat tracking system used in this project. It is based on the work by Davies and Plumbley [34]. However, this adaptation is designed with specific observations in mind that hold for (almost) all Drum and Bass music. These assumptions are the following:

Assumption 2.1. *The tempo of Drum and Bass music is constant throughout the entire song, or equivalently, all inter-beat intervals have exactly the same length.*

Assumption 2.2. *The most important music onsets, i.e. the loudest, most prominent or most accentuated musical events, typically occur on beat locations.*

Assumption 2.3. *Music shows many repeating patterns, and most patterns repeat after an integer multiple of beats.*

A thorough personal knowledge of the Drum and Bass music genre confirms that these assumptions hold. However, they also make sense from a logical point of view. Let us thus first ensure that this is indeed the case.

Assumption 2.1 makes sense when considering the production techniques and the design intentions for Drum and Bass songs. The fact that their tempo is constant[1] is not a coincidence: Drum and Bass songs are designed to be like this, so that they are 'compatible' with other Drum and Bass songs and to make it easier for DJs to mix them in their sets. Secondly, it is produced (mostly) by means of computer software. This ensures that all inter-beat intervals will have exactly the same length, since a computer does not make timing errors. Note that because of the assumption of a constant tempo, only two parameters suffice to describe all beat positions in a music fragment, namely the *period* and the *phase* of the beats. The beat period defines the tempo of the music, while the phase is the position of the first beat relative to the start of the audio fragment. This is illustrated in Figure 2.4.

Assumption 2.2 can also be rationally justified. As a matter of fact, one could reason the other way around: the most important musical onsets do not occur on beat locations by happenstance, but it is because of these accentuated events that there is a beat to observe in the first place! An analogous reasoning holds for Assumption 2.3: if the important musical repetitions did not happen at a multiple of the inter-beat interval, then there would be no rhythm or tempo to speak of (or the tempo would not be constant, but this would violate Assumption 2.1). The most prominent musical onsets in Drum and

[1]This assumption should actually be that "all Drum and Bass songs consist out of a small number of parts, usually no more than three, in which the tempo is constant". For example, some songs contain two parts belonging both to a different subgenre of electronic dance music, say Drum and Bass and Dubstep. In the first half of the song, the music is Drum and Bass at a constant 172 BPM. After this first part, the tempo slows down (which can happen quite fast) and the music transitions into the second part, which is Dubstep at 140 BPM. Both in the first and in the second part, the tempo is constant, but there clearly is a tempo change when transitioning from one segment to the other. However, only a small portion of Drum and Bass tracks have such a structure, so the detection of these segments and annotating both segments independently is left as future work.

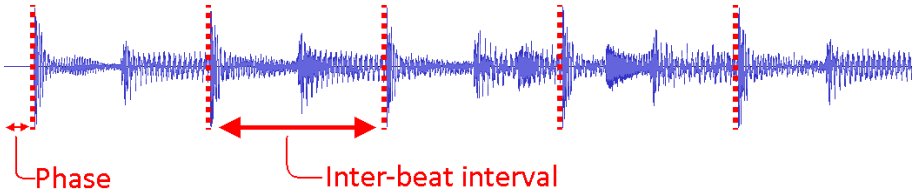


Figure 2.4: Illustration of the beat phase and inter-beat interval. The tempo in BPM is equal to 60 seconds divided by the inter-beat interval length in seconds.

Bass are usually percussive in nature.

The beat tracking algorithm is based on the exploitation of these three assumptions. Assumption 2.2 and 2.3 are related to the position of musical onsets. Hence, an onset detection function is at the core of the beat-tracking algorithm It is a simplified version of the algorithm by Davies et al [34]. More specifically, it is closely resembles the operation of the *general state* in that algorithm, but applied on the entire song instead of on 6-second windows. The different steps are described below and illustrated in Figure 2.5.

Input The input audio of the song

Output Beat annotations for the song, or equivalently the beat tempo and phase

1. First, the algorithm computes the onset detection function of the audio.

2. The onset detection function is post-processed by calculating the running mean of the signal and subtracting it from the original signal, after which it is half-wave rectified, i.e. all negative values are set to zero.

3. The autocorrelation of the onset detection function is computed. This exploits the assumption that onsets tend to repeat after an integer number of beats (Assumption 2.3). The autocorrelation will show high peaks at offsets of an integer number of inter-beat intervals. This can be seen in the third subplot of Figure 2.5.

4. Next, the tempo is extracted from the autocorrelation function. For every possible tempo and corresponding inter-beat interval, the values of the autocorrelation function are summed at fixed intervals corresponding to that inter-beat interval. This thus evaluates how much musical repetition there is after one shift of the inter-beat interval, plus how much repetition there is after two shifts of the inter-beat interval and so on, and this for every possible value of the inter-beat interval. The resulting sum is then divided by the number of values that were summed to ensure different

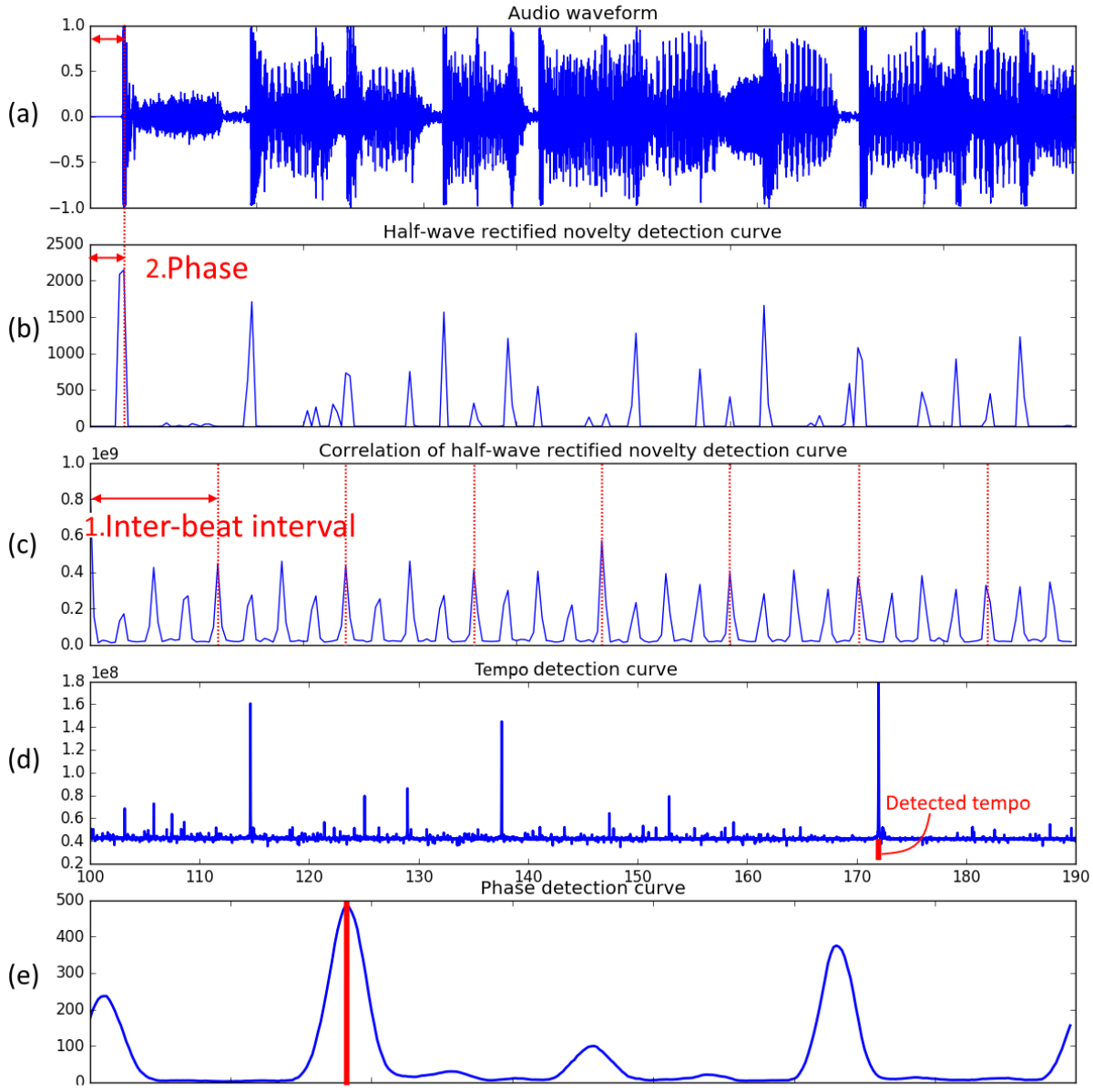


Figure 2.5: Illustration of the different steps in the beat tracking algorithm. (a) The audio waveform. (b) The half-wave rectified onset detection curve. (c) Autocorrelation function of the onset detection curve. (d) Tempo detection curve. (e) Phase detection curve.

sums are comparable. The tempo that led to the highest sum is selected as the correct BPM. The result of this step is illustrated in the fourth subplot of Figure 2.5. This curve, where the sum values are plotted for all tested BPM values, will be called the *tempo detection curve*.

5. Finally, the beat phase is extracted. The onset detection function is summed at regular intervals equal to the inter-beat interval corresponding to the detected BPM. This sum is calculated for every possible phase shift between 0 (inclusive) and 1 (exclusive) times the inter-beat interval. From Assumption 2.2, i.e. that the most significant onsets occur on beat locations, it follows that the phase value with the highest sum is most likely to be correct. The result of this step is illustrated in the fifth subplot of Figure 2.5. This curve, where all sum values are plotted for the tested phase values, will be called the *phase detection curve*.

In the current implementation of the algorithm, the *melflux* onset detection function is used, which is implemented in the Essentia library. This choice is substantiated in the next section. It is calculated on audio frames tapered with a Hanning window, using a frame length of 1024 samples and a hop size of 512 samples. The audio sample rate is 44100 Hz. The running mean is calculated on a zero-padded version of the onset detection curve to ensure that both curves have the same length. For each frame, the running mean its value is the sum of the eight values before and the eight values after that frame, divided by 16.

To extract the tempo from the autocorrelation function, the allowed tempo range is between 160 BPM and 190 BPM. The smallest increment in tempo that can be detected is 0.01 BPM. The smallest increment in phase that can be detected is 1 ms. These parameter choices are motivated in Section 2.3

2.3 | Evaluation

In this section, the performance of the proposed beat tracking algorithm is evaluated and the choice of onset detection function is justified. This is done by evaluating the following performance measures: the annotation accuracy on a song level for all four onset detection functions, the confidence of the algorithm in the selected optimal phase value and the run time for calculating the different onset detection functions.

First a note on how the accuracy of the beat annotations is evaluated. The beat positions were extracted from the song using the onset detection function under evaluation,

and the audio was overlaid with sinusoidal beep sounds at the estimated beat positions. The accuracy is then evaluated by playing an extract of approximately 20 seconds of the audio starting 20% into the song, and aurally evaluating that the annotated beats coincide with the correct beat positions in the music. Note that the beat positions are not compared with an annotated 'ground truth'. This has several reasons. First of all, beat annotations are ambiguous and subjective, since different individuals may annotate beats differently [24]. There are different ways to annotate a beat, and different manual annotations would still be correct. This is a consequence of the fact that the beat positions are spread over time, so that annotating the beginning of the beat or annotating the middle of the beat will result in two different but correct annotations. To have a similar definition of the correct beat location for all annotations, one could therefore annotate all music examples manually based on the waveform, e.g. by defining the beat position to coincide at the maximum of the envelopes of the sounds that occur on the beat position. This is very difficult, because it is often very hard to distinguish this exact position in a song with multiple instruments and a lot of musical activity. Annotating a sufficient number of files would be a very time consuming task, while offering limited additional value because of its ambiguity. For these reasons, onsets that audibly coincide with the correct position are thus considered correct.

The beat tracker can detect tempo differences up to 0.01 BPM. Let us first verify that this resolution is sufficient for the DJ application. First of all, it is often the case that the tempo is an integer BPM, e.g. 174 or 175 BPM, so for most songs no errors will be made due to an insufficient resolution. However, let us assume that the tempo is not an integer and that the error made by the algorithm is maximal, i.e. 0.005 BPM. Let us also assume that humans cannot detect two transients that occur less than 10 ms from each other [25]. If the song tempo is 170 BPM, which is slower than the tempo of most Drum and Bass, the inter-beat interval (IBI) is 353 ms, and with each minute that passes, the annotation error increases by 0.005 times the IBI or with 1.77 ms, since the incorrect annotations detected 0.005 more (or less) beats per minute than there actually are. It would thus take longer than 5 minutes before the difference is noticeable, which is longer than the typical length of a Drum and Bass song. In a DJ mix, a song is furthermore rarely played for its full length. This precision is thus considered to be sufficient because even if an error is made, it will not become noticeable to the listener. The smallest detectable phase difference is 1 ms, which is also sufficiently small if the smallest noticeable difference is 10 ms.

Let us now evaluate the effect of the BPM search range on the tempo detection accu-

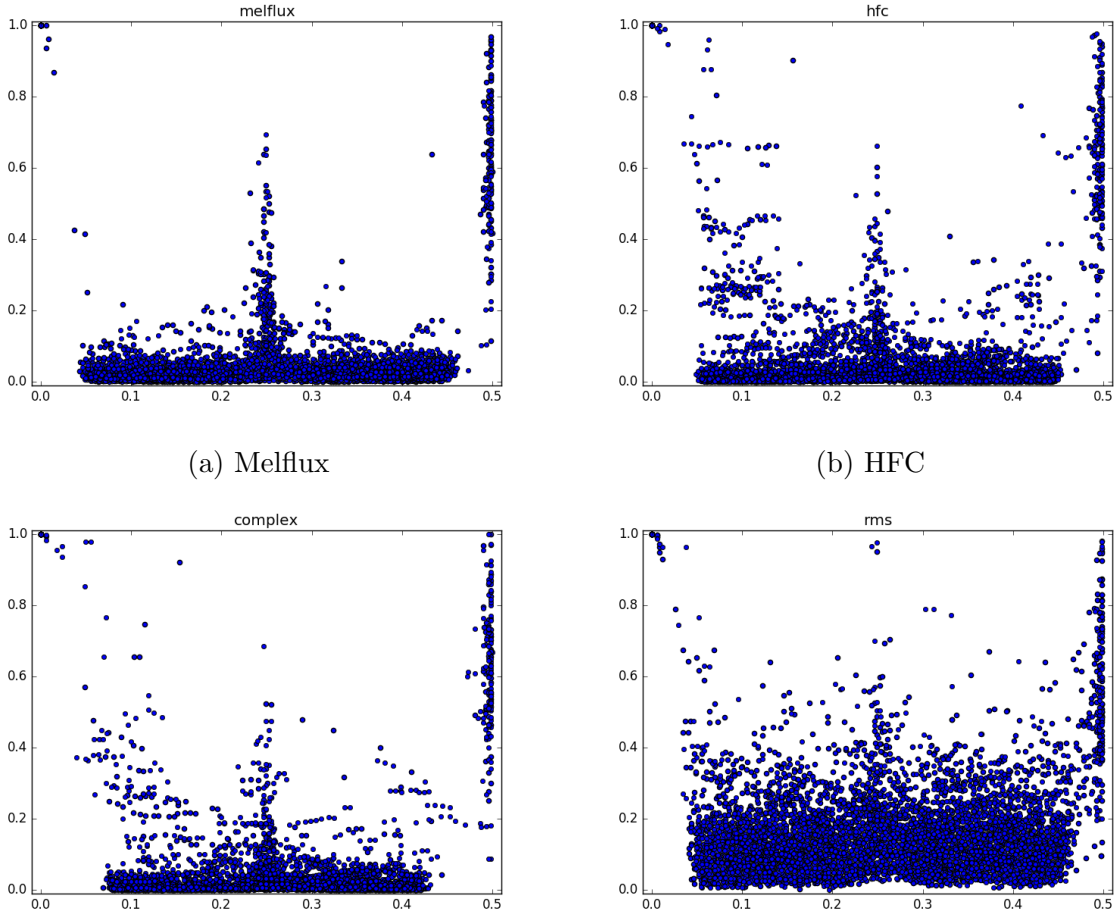
	RMS	**HFC**	**Complex**	**Melflux**
Correct songs (out of 160)	152	156	155	159
Accuracy	0.95	0.975	0.96875	0.99375
Execution time average (s)	0.701	0.724	1.778	0.789
Execution time standard deviation (s)	0.143	0.142	0.330	0.151

Table 2.1: Performance evaluation of the proposed algorithm using different onset detection functions on 160 Drum and Bass songs.

racy. In an initial implementation of the algorithm, the allowed tempo range was between 100 and 190 BPM with a resolution of 0.01 BPM. The HFC onset detection function was used. In this situation, the BPM of 2 out of 160 songs was detected incorrectly. Each time, a BPM value of 2/3 times the correct value was selected as the BPM. This happens because there also are many onsets that happen not on a beat location, but exactly in between two beat locations. This can lead to high values of the autocorrelation at shifts of $1.5 \times k$ times the inter-beat interval, because in those cases the onset detection function values of the beat onsets and the "half-beat" onsets overlap. This problem can easily be solved by exploiting the properties of Drum and Bass music: the tempo for this music lies between 160 and 190 BPM [2] (usually even between 170 and 175 BPM). As seen in the fourth subplot of Figure 2.5, there are no significant peaks within this range that can compete with the correct one. Restricting the search range thus ensures that the tempo is detected correctly for all songs, with all four evaluated onset detection functions.

Different onset detection functions have been evaluated in order to make an optimal choice for the final algorithm. Four functions are evaluated: the *high frequency coefficient* (HFC) function, the *complex spectral difference* (CSD) function, the *spectral flux* function where the spectrum is split up in equally spaced bands on the Mel scale [38] (melflux) and an onset function that calculates the root mean squared energy in a short window (RMS). All functions are available in the Essentia library [35]. Since the proposed algorithm detects the tempo correctly with all four onset detection functions after restricting the BPM search range, the difference in performance will occur when detecting the phase. Different onset detection functions accentuate different types of onset events, which affects their performance. The accuracy and execution times of the four onset functions are compared in Table 2.1. These results are obtained on a collection of 160 Drum and Bass songs, which were carefully selected to include a variety in subgenres and artists.

When an incorrect phase value is detected, then these wrong beats are typically offset by half the inter-beat interval, i.e. they lie exactly in between the correct beats. This can



(a) Melflux (b) HFC

(c) CSD (d) RMS

Figure 2.6: Scatter plot illustrating the effect of different onset detection functions on the phase detection. This figure shows the local maxima of the phase detection curves of all songs for the four tested onset detection curves.

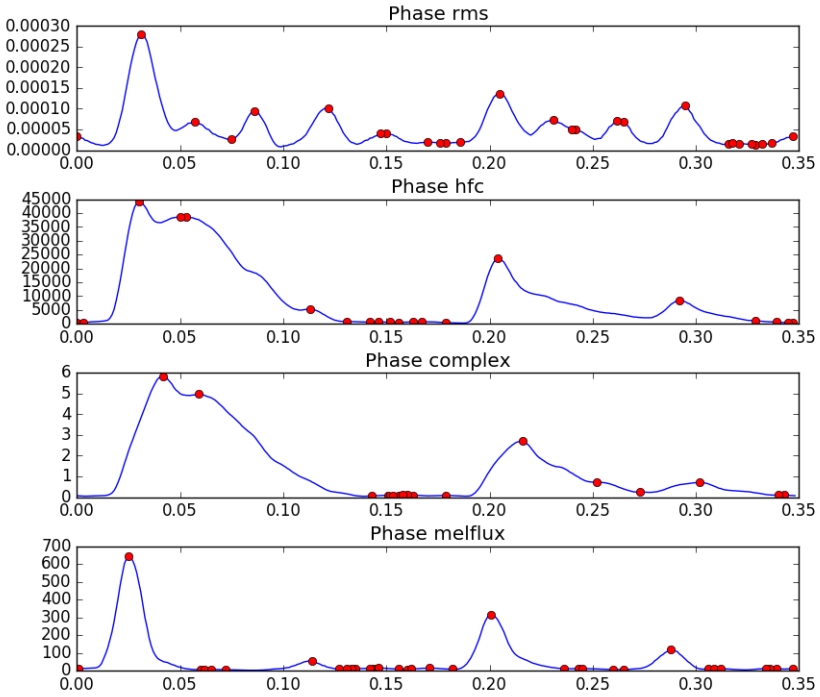


Figure 2.7: Illustration of the effect of different onset detection functions on the phase detection curve.

be explained by the fact that there are still many musical events on a half-beat position. These half-beat events have a different nature than the onsets on beat positions: usually, they are high-hats and snare drums (but other events also occur). A different onset detection curve may accentuate the type of events that occur on beat positions more than the type of events on half-beat positions and consequently improve the performance of the algorithm. Based solely on the accuracy on a song level, the melflux onset detection function appears to have the best performance on this type of music. It furthermore has an execution time comparable to the other methods, as illustrated in Table 2.1.

The reliability of the detection method depends on how much the amplitudes of the highest peak in the phase detection curve and the second-largest peak differ, and on how much the peak is spread out over different phase values. To evaluate the former, the different local maxima of the phase detection curves are be compared. The scatter plots in Figure 2.6 show the amplitude of the different local maxima of the phase detection curves of all songs in function of their distance to the absolute maximum. A local maximum in this plot is simply a value in the onset detection curve (i.e. an array of onset values) that has a greater value than both its neighbors. Note that this will lead to many local maxima if the phase detection curve is almost constant, but fortunately the phase detection curves are usually monotonically increasing or decreasing and few local maxima are detected per

song. Also note that the maximum distance is 0.5 times the inter-beat interval. This is because a time instant can never be further than half a beat period from the closest beat instant. For example, a position at three quarters of the inter-beat interval from the previous beat lies a quarter of the inter-beat interval from the next beat: its distance is therefore 0.25.

Several interesting patterns can be seen. Firstly, all significant local maxima occur at three positions: near $x = 0$, i.e. (almost) equal to the detected maximum, near $x = 0.25$, i.e. maxima at quarter-beat positions, and near $x = 0.5$, i.e. maxima at half-beat positions. For all onset detection functions, the peak in the scatter plot at the half-beat position is very close to the peak at beat-position, while the quarter-beat peak is significantly smaller. This indicates that the most significant onsets occur at beat or half-beat positions. However, for the melflux onset detection function, the difference between those peaks is the largest, indicating that the melflux function can more reliably distinguish between the two. Additionally, all functions but the melflux function show many local maxima with quite a high amplitude, mostly around beat positions ($x = 0$) and half-beat positions ($x = 0.5$). This is a consequence of the fact that the peaks of these phase detection curves are less peaked and sometimes have local maxima close to the real maximum. This can be seen in Figure 2.7. These onset detection functions therefore give a more ambiguous and spread-out localization of the possible onset locations, which is not desirable.

From this analysis, the melflux onset detection function appears to be the best option to detect the beat positions in Drum and Bass music with the proposed algorithm, correctly annotating 159 out of 160 songs, i.e. an accuracy of 99.4%. As an extra verification, a new test set of 220 songs has been selected and annotated. Here, 216 out of 220 songs are annotated correctly by the algorithm, i.e. an accuracy of 98.2%. In summary, the motivation for the choice of onset detection function is threefold. First of all, it empirically gives the best accuracy on a song basis. Secondly, it discriminates beat positions from half-beat positions more clearly than other onset detection functions. Thirdly, the phase detection curve is more sharply peaked, which leads to a less ambiguous localization of the beat positions.

2.4 | Conclusion

The beat tracking algorithm devised in this chapter is inspired by the work of Davies and Plumbley [39]. It uses the melflux onset detection curve to detect musical onsets, as this proved to work best for Drum and Bass music. The beat period and phase are extracted in a two-step process. The allowed tempo range is from 160 to 190 BPM and the smallest detectable tempo difference is 0.01 BPM. The smallest detectable phase is 0.001 second. This algorithm correctly detects 159 out of 160 songs in the used music library with an average execution time of 0.789 seconds per song. In an evaluation on a second held-out test set of 220 Drum and Bass songs, a comparable accuracy of 98.2% is obtained.

2.5 | Future work

The algorithm described in this chapter has an excellent performance on Drum and Bass music and provides a solid basis for the DJ system to rely on. However, some challenges remain to be addressed, and further improvements can still be made.

A first remark is that this algorithm constrains itself to music with a constant tempo throughout the entire song. As mentioned earlier,some Drum and Bass songs consist out of several parts, for example a Drum and Bass part and an Electro part. In both parts, the tempo is constant, e.g. 172 BPM in the Drum and Bass part and 128 BPM in the Electro part. Somewhere in the song, the tempo will therefore slow down to get from the Drum and Bass part to the Electro part. The proposed algorithm could be augmented with a segmentation component that detect whether such different-tempo segments exist and that tracks the beat positions in those segments separately.

A second remark is the fact that this algorithm does not provide any warnings or uncertainty measure at this moment. The importance of this has already been highlighted by Degara et al [36]. If the algorithm makes a dubious prediction, it will now just continue and give back false results. A reliability measure could solve this and detect e.g. when the input song is a multi-genre song or when two local maxima in the tempo or phase detection curve are very close to each other, preventing the automatic DJ from using these unreliable predictions.

In Section 2.3, the performance of several onset detection functions has been evaluated. All onset detection function perform quite well, with the melflux onset function

having the superior performance. However, instead of using only one function, multiple onset functions could be used to construct a majority voting system at the expense of a higher computational cost. Indeed, informal tests show that when one function predicts the wrong result, then often other functions predict the beat positions correctly.

Alternatively, the performance of the individual onset detection functions could be improved by additional preprocessing. For example, in Drum and Bass music, the onsets on beat positions are typically kick drums while half-beat onsets are typically snare drums or hi-hats. Low-pass filtering the audio signal before applying the onset detection could increase performance at the expense of higher computational demands.

A final suggestion comes from the observation that in Drum and Bass, the most significant onsets lie on either a beat or a half-beat location. This can be exploited by first detecting the double of the tempo, so by restricting the search range from 320 till 380 BPM instead of 160 till 190 BPM. This results in twice as many annotations: half of them are on the beat positions and the other half on half-beat positions. To discriminate between both, one could then use e.g. a machine learning classifier to discriminate beats from half-beat positions based on the properties of the surrounding audio.

Chapter 3

Downbeat tracking

With the beat tracker from the previous chapter, the automatic DJ system can detect the beat positions and 'tap along with the beat'. However, apart from a sense of rhythm, a DJ needs to have a notion of the structure of the music he is working with. Music namely exhibits a hierarchical structure at different time scales [1]. At the lowest level, it consists of beats that indicate its rhythm. In Drum and Bass, the beats can be grouped into groups of four. Such a group is called a *bar* or a *measure*[1]. In music theory terms, this is called a 4/4 *time signature*. The first beat of a bar is typically accentuated, such that there is a sequence of periodic accents in the music. This makes it easier for music listeners and players to synchronize with the music and 'count along' with it (as in "*one* two three four *one* two..."). This accentuated first beat of a bar is called the *downbeat*.

Counting along with the music is very important for a DJ. When people listen to music, they typically (unconsciously) keep track of the downbeats. When a DJ mixes two songs, he should not only make sure that these songs are beatmatched, but actually that they are '*downbeatmatched*' . If this is not done properly and the downbeats of the two songs do not align, then the listener his internal counting will be disrupted during the transition.

This chapter describes how the automatic DJ system can detect which beats in the song are downbeats. This is a task known as *downbeat tracking*. First, an overview of existing approaches is presented in Section 3.1. As with beat tracking, these techniques are applicable to a wide range of music genres, but they fall short in their accuracy to properly beatmatch two Drum and Bass songs. In Section 3.2, a custom downbeat tracking solution is developed that uses a machine learning model trained on Drum and Bass music. The proposed approach is evaluated in Section 3.3. Section 3.4 summarizes

[1]Not all music has four beats in a bar. For example, waltz music has three beats per measure [40], i.e. a 3/4 time signature.

the obtained results and Section 3.5 concludes the chapter with some suggestions for future improvements.

3.1 State of the Art analysis

In comparison with the related task of beat tracking, downbeat tracking has received little attention in scientific literature [33]. Some approaches are similar to beat tracking approaches, in the sense that they assume that downbeats are more likely to occur on points of significant change in the music. For example, in [39], Davies and Plumbley use a spectral difference approach to detect downbeat positions in music. Their approach requires that the beat locations are known, for example by extracting them using a beat tracking system. It also incorporates a way to estimate the time signature of the audio, i.e. whether there are three or four beats in a measure. The audio signal is partitioned into beat-length frames, and a spectral difference function is calculated on the band-limited spectra of these frames. A band-limited spectrum is used because of the musical knowledge that downbeats are often characterized by changes in the lower parts of the spectrum. The spectral difference function is evaluated at each beat position. In order to determine the downbeat position, all function values which lie at an offset of three or four beats from each other are added together for each potential downbeat position. The downbeat position is then selected as the position leading to the highest sum. Their approach achieves an accuracy of 52.6% over a set of 181 annotated files, where a downbeat is considered correct if it falls within a certain error margin of the annotation. However, this accuracy heavily depends on the accuracy of the underlying beat tracker: on a subset of 72 songs where the beat tracker was at least 95% accurate, the accuracy of their approach becomes 81.2%.

Other approaches make use of machine learning models to estimate the probability of a beat segment being a downbeat. These probabilities are subsequently decoded using temporal models such as Hidden Markov Models (HMMs). An recent example of such an approach is the one by Durand et al. [41]. The audio is first segmented into frames, from which several features are extracted that describe the harmony, timbre, low-frequency content, rhythmic patterns and local similarities in the music. These low-level features are summarized into higher-level representations by means of several independent deep convolutional neural networks. For each type of features, such a network is constructed. In their original work, the same network architecture is used for each feature network. In an improved version of their system [42], the networks are tuned to the specific properties of the features and their relation to the downbeat tracking task. These networks then

estimate the probability that a segment is a downbeat segment or not. The probabilities over all models are averaged and a Hidden Markov Model is constructed to model the transitions between different states that represent the different metrical positions, i.e. one downbeat state and several non-downbeat states. High transition probabilities are assigned to sequential transitions between states, but to allow for variations in the time signature, small non-sequential probabilities are introduced. Finally, the most likely downbeat positions are predicted by means of the Viterbi algorithm.

The approaches mentioned above attempt to solve the downbeat tracking task for a broad corpus of music genres. However, in their *One in the Jungle paper*, Hockman, Davies and Fujinaga [33] question the usefulness of a generalized downbeat tracker, applicable on all genres of music. They argue that the properties of certain genres, such as Hardcore, Jungle and Drum and Bass, might be so complex that style-specific downbeat trackers are justified. Indeed, these genres have a fast, heavily percussive content with a high note density and often an emphasis on offbeats, which might cause the general downbeat tracking algorithms to struggle with this kind of music. The authors of this article therefore construct a custom downbeat tracking solution that uses support vector regression (SVR). The regression model is trained on percussion-only music extracts that represent several breakbeat patterns. These music extracts are segmented into an eight-note grid, and the SVR attempts to predict the metrical position in the eight-note grid of each of these segment. To determine the downbeat of a previously unseen piece of music, its beat positions are first determined using BeatRoot. The music is then segmented in a similar eight-note grid as the breakbeat patterns and the SVR model is used to infer the likely metrical position of the grid segments, giving a prediction of likely downbeat locations. The SVR model on its own is not accurate enough however, as breakbeat patterns did not appear descriptive enough to solve the downbeat detection task on their own. Therefore, several measures were taken to improve its accuracy. First of all, a coarse downbeat estimation function is constructed using a sliding buffer over the SVR predictions of eight consecutive segments, i.e. the length of a measure. This estimation function has a high value at likely downbeat locations and a low value elsewhere. It is used to scale a low-level onset detection function calculated over the lower frequency bands of the input signal. Additionally, it is also weighted by another function to give beat locations more weight, whereas onsets on non-beat locations are scaled down. The result is an onset detection function with high onset values near the estimated beat and downbeat locations. This function is used for accurate event localization, while the coarse estimation function derived from the SVR classifier informs the detection function about detected breakbeat

patterns, and the beat weighting ensures that the extracted downbeats are more likely to coincide with beats. The downbeats themselves are then determined by peak picking at regular intervals to ensure a 4/4 time signature. This algorithm achieved a mean accuracy of 74, 7% across all downbeat excerpts. Important to note is that the algorithm has been compared with the algorithm by Davies and Plumbley [39], which reportedly achieved an accuracy of 52.6% or even 81.2% on songs with accurate beat annotations. However, when evaluated on these specific genres, the accuracy drops to only 29.3%. This indicates that the simple spectral difference approach is not suited to detect downbeats in these types of music, and that a genre specific approach might be more interesting.

3.2 | Downbeat tracking for Drum and Bass music using a logistic regression classifier

The automatic DJ system developed in this thesis is designed to work specifically with Drum and Bass. The work by Hockman et al. [33] already indicated that a genre-specific approach to the downbeat tracking task is worthwhile. Furthermore, exploiting the genre-specific properties of Drum and Bass music led to excellent results for the beat tracking task in Chapter 2. Therefore, in this chapter, a downbeat tracking approach is developed that is tuned specifically to the target music genre.

More specifically, to create an accurate downbeat tracking system, a logistic regression model is constructed that distinguishes downbeats from other beats. A logistic regression model is used for its simplicity, the fast training procedure and because its output can be interpreted as a probability estimate. It classifies the feature vector of a beat-length segment of audio into one out of four classes: either it is a downbeat, the second beat, the third beat or the fourth beat of a measure. For each of these classes, the probability of belonging to that class is returned by the model. These beat-wise predictions are then aggregated to predict the downbeat positions for the entire song.

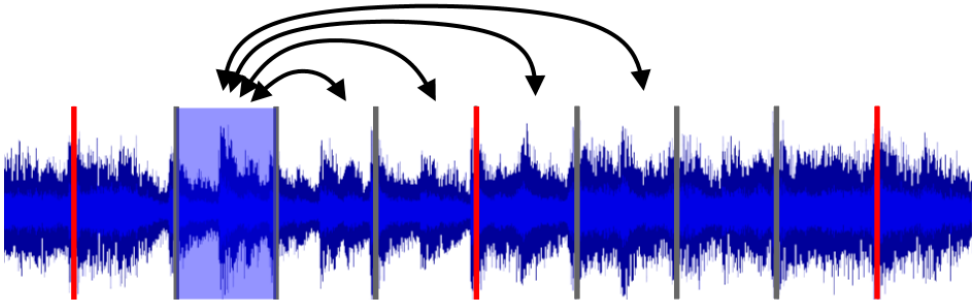
Before the machine learning classifier can work though, informative features need to be extracted from the input audio. Section 3.2.1 describes these features. Section 3.2.2 then explains how the downbeat is extracted on a song level using the classifier.

3.2.1 | Features for downbeat tracking

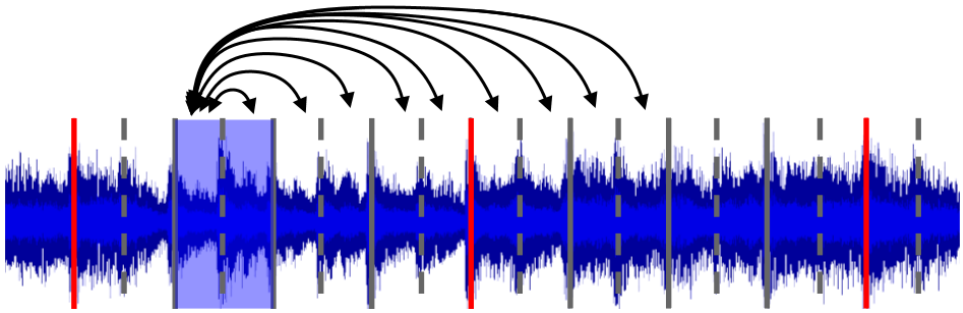
To perform the classification of a beat extract, one cannot simply give the raw audio waveform or its spectrum as an input to the machine learning algorithm, as its high dimensionality would certainly lead to poor generalizing behaviour of the classifier. Some preprocessing is required to extract relevant features that summarize the input audio in a concise and informative way. Feature engineering is performed by hand rather than performing feature learning (e.g. using a neural network) because it is simpler and to maintain a clear interpretation of the meaning of the used features.

The logistic regression model classifies beats as being a downbeat or another type of beat. The features are therefore extracted from beat-long extracts of audio. However, an important realization with respect to the downbeat tracking task is that a music segment of just one beat holds very little information on its own, and that it is very difficult to determine at which position in a measure it occurs based only on this information. Indeed, the notion of rhythmical structure in music arises by the carefully orchestrated accentuation of certain beats compared to other beats. For the downbeat tracking task, it is therefore crucial that the feature vector of a beat fragment is not just made up by features from that fragment in isolation, but rather by features that describe the relation between current beat fragment and the beats surrounding it. In what follows, features calculated on a fragment in isolation will be called the *isolated features* of that fragment. Features that describe the inter-beat relations are called *contextual features*. Because contextual features are used, the classifier is not classifying one beat fragment in isolation, but actually an audio fragment including the beat to be classified and some of the surrounding audio. The concept of contextual features is illustrated in Figure 3.1a.

For the beat tracking task, the assumption that most musical events occur on beat positions leads to an elegant and powerful beat tracking solution. However, certainly not all musical events occur on a beat position. There are also many prominent onsets on half-beat positions, i.e. positions half-way between two beats. For example, a frequently used beat pattern in Drum and Bass has a kick drum in between the third and the fourth beat of a measure. The observation that some important onsets occur on half-beats indicates that extracting features not at a beat level, but rather at a half-beat level, might aid in the classification process. Therefore, for some extracted features, the beat frames are split in half, and isolated features are calculated on those halves. Contextual features then compare two halve beats. This is illustrated in Figure 3.1b. In what follows, the



(a) The contextual features of a beat fragment are calculated by e.g. taking the difference between the isolated features of that beat with the corresponding features of the subsequent 4 beats. This leads to a contextual awareness of 1 bar.



(b) In this illustration, the features are calculated with a half-beat granularity. The isolated features of a beat fragment are calculated on both halves of that beat. Contextual features are then constructed by e.g. calculating the difference between the features of its first half with the features of the subsequent 8 halve frames. This leads to a contextual awareness of 1 bar.

Figure 3.1: Illustration of the feature calculation process using contextual features and halve-beat frames.

different features are explained in more detail.

Energy in Melbands

The spectrum of the music signal contains valuable information for downbeat tracking. Downbeats in Drum and Bass are for example typically accentuated by a kick drum, which leads to a sudden energy burst in the lower frequencies. Apart from the downbeat, there might be other repetitive musical events that are more likely to occur at other beat or inter-beat positions, leading to a specific evolution of the energy spectrum. Therefore, one of the extracted isolated features is the the energy distribution along the frequency axis, binned into 12 bins which are equally spaced along the Mel frequency scale [38]. The Mel scale transforms frequencies to a logarithmic axis, which better corresponds to the way humans perceive musical tones [26]. Contextual features are calculated by computing the differences of these 12 values with the corresponding values in the following three beats and in the previous beat, giving a contextual awareness of one measure. Hence, the total number of Melband energy features per beat segment is 48. Informal experiments showed that increasing the number of bins to 24 did not lead to an increase in performance.

Onset detection functions

In the beat tracking task, onset detection functions proved to be a very informative description of the music signal. It furthermore appeared that different onset functions lead to the detection of different music events. Therefore, three onset detection functions are used in the downbeat tracking task too: the spectral flux onset function, the high frequency coefficient (HFC) function and the complex spectral difference (CSD) function. All functions are implemented in the Essentia library [35].

First, the onset functions are calculated over the entire input song. They are then split into frames of half a beat in length, which are assigned to the corresponding beat they belong to. Isolated features are extracted by calculating the sum of the onset function values of these halves: this gives a measure of the total musical activity in that halve beat. Contextual features are created by calculating the difference of these sums with the corresponding sum values in the following 15 halve beats and the previous halve beat, giving a contextual awareness of two measures. A second type of contextual features is calculated by computing the correlation of the onset curve fragment of a beat and the onset curve fragments (originating from the same onset detection function) of the following three beats and the previous beat, giving a contextual awareness of one beat.

This gives another 60 features per beat extract (3 onset detection functions with each 20 features, i.e. 4 correlation values and 16 sum differences).

Loudness

The loudness of a piece of audio is a measure for how loud it sounds. It is related to the amplitude of the audio, but it is different because of the fact that it is a psychoacoustic measure that depends on the sound pressure level and frequency content of the audio [43]. Major changes in loudness, e.g. the transition from the intro section to the main section or from the main section to the bridge or outro, typically occur on a downbeat location. Hence, patterns in the evolution of the loudness envelope could be descriptive for the type of beat that is analyzed. It is extracted by means of the corresponding `Loudness` method in the Essentia library [35]. Contextual features are then calculated by taking the difference of the current frame its loudness value with the loudness values of the previous frame and of the next 7 frames, giving a contextual awareness of 2 measures.

3.2.2 | Downbeat tracking algorithm

Given the frame-wise features of the audio, the downbeat positions could be determined in a rule-based way, i.e. by making specific assumptions on the features and their evolution with respect to their position in a measure. Such an approach is also followed by Davies and Plumbley [39]. However, coming up with a small set of rules that are generally applicable for all Drum and Bass music and that can accurately detect downbeats is very difficult, if not impossible to do. Because of this, a machine learning approach is followed, where the algorithm discovers these 'rules' by itself.

It is furthermore important to realize that the beat classifications of all beats in the audio heavily depend on each other, since Drum and Bass follows a strict 4/4 time signature. Indeed, if a certain beat is estimated to be the downbeat, then the beat after that must be the second beat in the measure, the next is then the third beat and so on. This means that for a specific song, there are only four possible outcomes: the first downbeat of the song can be either the first, the second, the third or the fourth beat, and given the position of the first downbeat, all other downbeats are known. With this in mind, the proposed algorithm works as described below. The algorithm is furthermore illustrated in Figure 3.2.

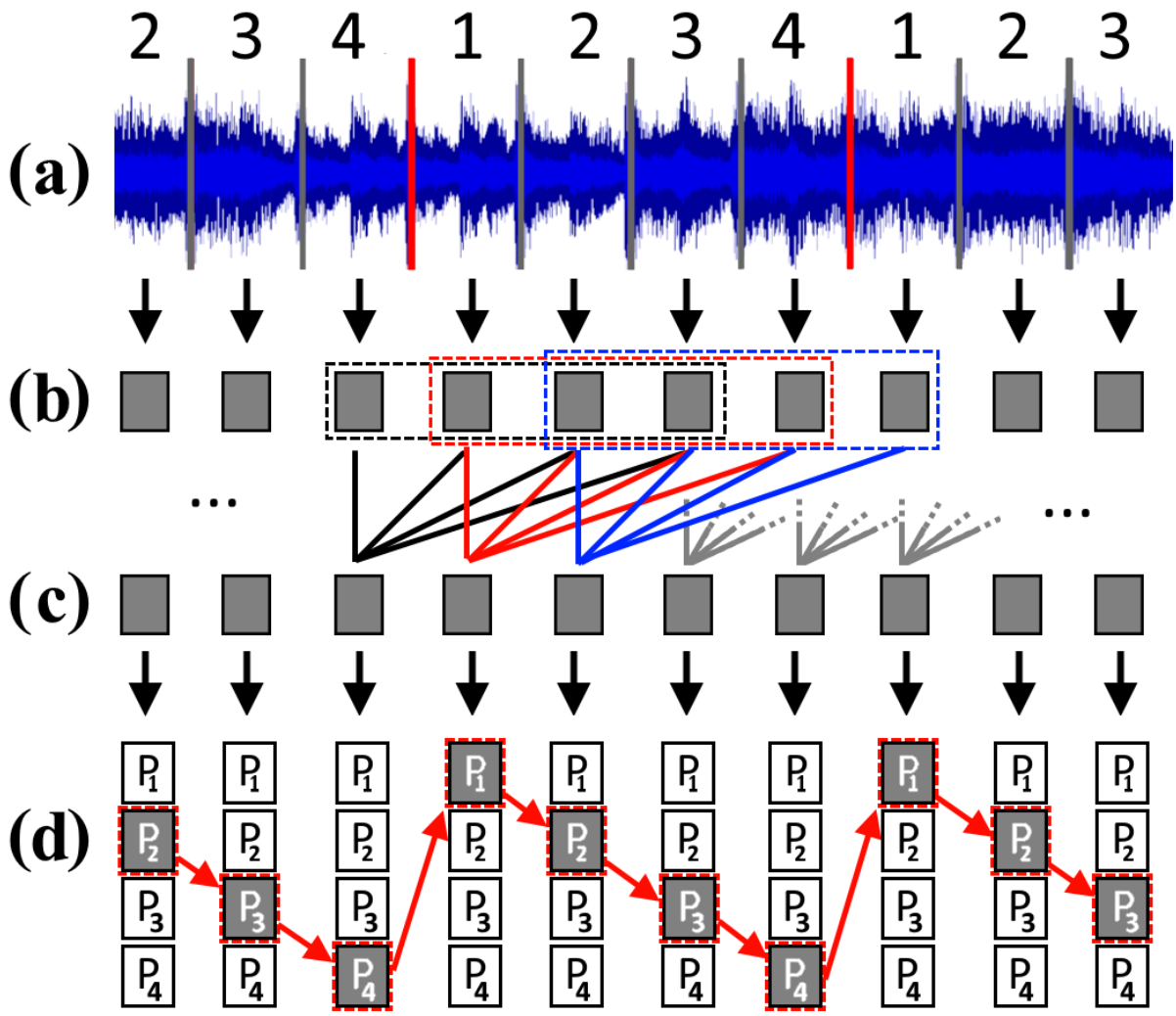


Figure 3.2: Overview of the downbeat tracking algorithm. (a) The audio waveform, annotated with the beat positions. Also the downbeats are shown in this figure. (b) Isolated features are extracted from the beat extracts. (c) Isolated features are combined to create contextual features. (d) All beats are classified using the machine learning model, estimating the log-probability of it being either the 1^{st}, 2^{nd}, 3^{rd} or 4^{th} beat in a measure. The downbeats are then determined by summing the log-probabilities along each possible trajectory and choosing the most likely one. Here, the second trajectory is highlighted.

Input The song its audio and the corresponding beat annotations

Output The downbeat annotations, which are a subset of the beat annotations

1. Segment the input audio into beat-long segments by using its beat annotations
2. Filter out the beats in the intro and outro based on their RMS value
3. Calculate the 'isolated features', i.e. features of every fragment individually.
4. Calculate the 'contextual features', e.g. differences between the isolated features of a frame and some frames before or after it.
5. Classify all feature vectors to obtain a log-probability vector for each individual beat.
6. Sum up the log-probabilities of every beat along the four possible trajectories. The trajectory with the highest sum is most likely, from which the downbeats can be extracted.

The algorithm assumes that the audio is annotated with beat annotations, e.g. obtained from the beat tracker described in the previous chapter. The audio is split up into beat-long frames of which the features are calculated. However, not every beat is considered when determining the correct downbeat positions. Initial experiments have shown that the algorithm performs much better on the louder and more percussive parts of the audio, whereas the distinction between downbeats and other beats is much less clear in the more quiet and non-percussive parts such as the intro and outro. This is illustrated in Figure 3.3. Therefore, the beats that are part of the intro and outro are trimmed. This is done using an energy criterion. First, the root mean square (RMS) value is calculated of the audio signal in each beat fragment, resulting in an RMS value for each beat. This RMS signal is smoothed by calculating a running mean of four beats over it, i.e. every value in this averaged signal is the mean of itself and the three RMS values before it. The end of the intro is then detected as the first beat that has an RMS value larger than a fraction f of the maximal RMS value of all beats. The start of the outro is detected as the last beat fragment with an RMS value larger than f times the maximal RMS value. This is first attempted for a value of 0.9 for f. If the number of remaining beats is smaller than 40% of the total number of beats, then this trimming process is repeated with values 0.8, 0.7, ..., 0.1 for f, until the number of remaining beats is large enough. This ensures that not too much beats are filtered out.

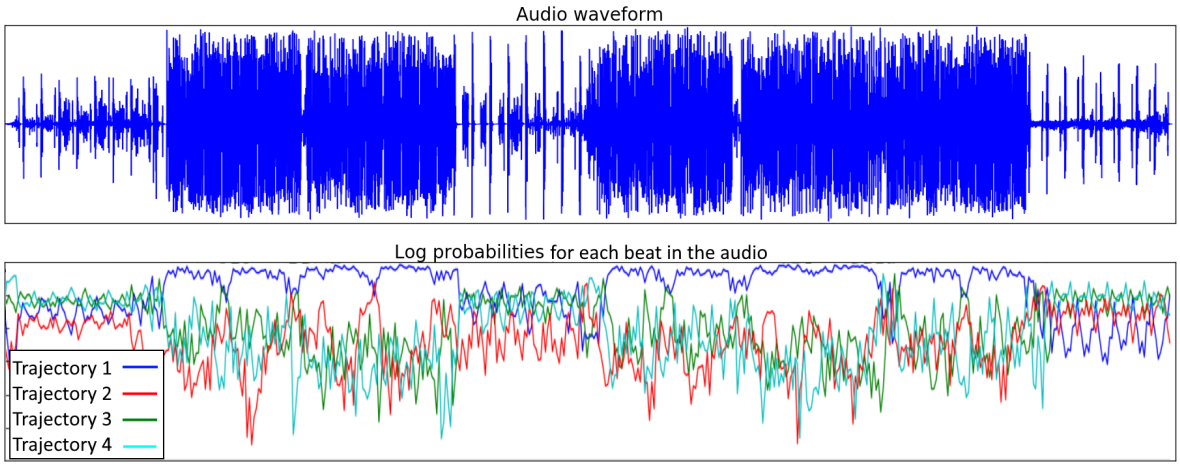


Figure 3.3: Performance of the downbeat tracker over an entire song. The first plot shows the audio waveform. The second shows the logarithm of the probabilities predicted by the classifier along each possible trajectory. The blue trajectory is correct. This illustrates that the classifier is much more certain during the main parts than in the less expressive intro, outro and bridge.

After trimming the audio, the absolute and contextual features of the beats are calculated. Note that for the calculation of the contextual features of some beats, it might be necessary to use isolated features of beat frames that are not used for the classification themselves. The feature vectors are then classified by means of the logistic regression model, which results in four log-probability estimates for each beat.

The final step is to determine the downbeat positions in the song. As argued earlier, this comes down to determining which path is the most likely: the path where the first downbeat is either the first beat (i.e. where the beats of the song are labeled as $\mathbf{1} \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \mathbf{1} \rightarrow 2 \rightarrow 3 \rightarrow \ldots$, with the downbeat shown in bold), the second beat (i.e. $4 \rightarrow \mathbf{1} \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \mathbf{1} \rightarrow 2 \rightarrow \ldots$), the third (i.e. $3 \rightarrow 4 \rightarrow \mathbf{1} \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \mathbf{1} \rightarrow \ldots$) or the fourth (i.e. $2 \rightarrow 3 \rightarrow 4 \rightarrow \mathbf{1} \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \ldots$). To do this, the log-probabilities of each beat are summed along each possible path. The path with the highest sum is most likely, from which the downbeat positions can be extracted.

3.3 | Evaluation

To train and evaluate the downbeat tracker, the same set of 160 songs is used as for the evaluation of the beat tracker. A subset of 117 songs is selected for training. The remaining 43 songs are used for evaluation. The evaluation set contains several songs from artists of which no songs are in the train set, to reduce the potential of data leakage between the two sets. The songs have been annotated with the beat tracker, but the

Features	Train accuracy	Test accuracy	Train log loss	Test log loss	Train song accuracy	Test song accuracy
Loudness	40.3%	47.5%	1.28	1.24	50.4%	65.1%
Melbands	51.1%	56.2%	1.15	1.10	75.2%	88.4%
OnsetHFC	53.2%	45.0%	1.06	1.21	79.5%	60.5%
OnsetCSD	57.1%	50.4%	1.01	1.16	82.9%	76.7%
OnsetFlux	59.4%	54.3%	0.98	1.12	84.6%	79.1%
OnsetHFC, OnsetFlux, OnsetCSD, Loudness, MelEnergy	68.5%	68.6%	0.77	0.81	95.7%	95.3%
OnsetHFC, OnsetFlux, OnsetCSD, Loudness, MelEnergy (with song trimming)	**75.9%***	**75.4%***	**0.634***	**0.693***	**96.6%**	**95.3%**

Table 3.1: Evaluation of different feature combinations for downbeat classification on 117 train songs and 43 test songs (*Evaluated only on segments not trimmed away)

beat annotations of the song that was annotated incorrectly (see Section 2.3) have been adjusted manually. Two aspects of the proposed approach are evaluated. Firstly, the classifier itself is evaluated in terms of the achieved log-loss and accuracy when classifying individual beat fragments. Secondly, the algorithm its accuracy on a song basis is evaluated. This evaluation has been carried out for several feature combinations in order to demonstrate their influence. The results of this evaluation are shown in Table 3.1.

First of all, it is clear that the different features in isolation cannot distinguish downbeats from non-downbeats very well. Combining the features leads to a significant increase in both the beat-level and song level classification accuracy and a large drop in the logarithmic loss error measure. Trimming away the intro and outro gives another boost in performance. Note that the filtered set of beats is now used both for model training as for downbeat prediction. The final song-based accuracy is 96.6% for the train set (4 out of 117 songs wrong) and 95.3% for the test set (2 out of 43 songs wrong). As with the beat tracker, the accuracy of the downbeat tracker is evaluated again on a second held-out test set: of the 216 songs that received correct beat annotations, 212 songs also get correct downbeat annotations, which leads to an accuracy of 98.1%.

When the downbeat tracker fails, then two causes stand out. In two of the songs in the set of 160 songs, the music genre is *Drumstep*, a genre related to Drum and Bass. However, it uses a different drum pattern than Drum and Bass, which might be the cause of the errors. A third misclassified song is very quiet and its percussion in particular is very quiet and inexpressive, which is atypical for Drum and Bass. Two other songs use very complex breakbeats, which could also confuse the tracker. For the last misclassified song, the reason for misclassification is unfortunately unclear.

3.4 | Conclusion

The downbeat tracker described in this chapter operates in three main steps. First, features are extracted from the individual beat segments of the input audio. Then, a logistic regression classifier determines the probability that a beat is either the first, second, third or fourth beat in its measure. Finally, these predictions are aggregated over the entire song for each of the four downbeat options to determine the most likely downbeat positions. The extracted features are related to the energy content, loudness and onset information of the beat fragments. They have been constructed in such a way that they contain not only information from the currently classified beat, but rather the evolution of the features in the surrounding beats. This gives the classifier a contextual awareness and leads to better predictions, giving an accuracy of 75.4% on individual beat fragments after trimming away the intro and outro of the song using an energy criterion. The proposed algorithm correctly detects the downbeat in 95.3% of the songs in the test set. Of the 216 songs in the second test set that got correct beat annotations, 212 were annotated correctly, giving an accuracy of 98.1%.

3.5 | Future work

The downbeat tracking solution described in this chapter provides a solid basis to build the remainder of the DJ system on. However, some aspects of it can still be improved.

First of all, this downbeat tracker is tuned specifically to Drum and Bass music. This is intentional, as this allows the tracker to exploit genre-specific properties and achieve a better result. In order create a DJ system that can handle more music genres, however, it will need either a general downbeat tracking solution, or an additional genre detection module that detects the genre of the input song. A genre-specific tracker can then be

used depending on the detected genre. It is unclear if the proposed machine learning approach could be reused for other genres by simply retraining the model on a corpus of music of the desired genre, or if other features or techniques are required. However, for closely related genres (e.g. other electronic dance music genres), it could be expected that simply retraining the model would be sufficient.

The proposed solution trims the song its intro and outro, as the model performs much worse in these sections compared to the more expressive sections. However, in Drum and Bass, there often is a less expressive bridge section too that separates the first main section and the second build-up and drop. The tracker its performance might be improved even more by also filtering out the beats in this section. Alternatively, it might be possible to create a downbeat model for both percussive and non-percussive sections, so that all beats of the song can be used when predicting the downbeat position.

Some final suggested are the following. First of all, it could be investigated if the onset detection curve used for the beat tracking could be reused for the downbeat tracking to calculate frame-wise features, as reusing features would make the entire annotation process more efficient. Additionally, more feature tuning and engineering could be done to find potentially more expressive features. Finally, it could be interesting to identify repeating sections of one measure long in the audio, for example using a repetition-based segmentation technique, to detect the downbeat position. These segmentation techniques are the topic of the next chapter.

Chapter 4

Structural segmentation

With the beat tracking and downbeat tracking module in place, the automatic DJ system can keep track of the rhythm of the music. It furthermore has a basic notion of the musical structure on a low hierarchical level, in the sense that it can discriminate between downbeat and non-downbeat positions. However, a DJ also needs a thorough understanding of the musical structure of the mixed songs. This follows from Rule 1.4 in Chapter 1. He should select the cue points such that the mixed songs their structural segment boundaries appropriately align. In this way, a transition is created where the higher-level structure of the music is respected in the resulting mix.

This chapter adds a module to the automatic DJ system that detects high-level structurally coherent segments in the music. This is a task known in scientific literature as *structural segmentation* [24, 44]. Section 4.1 explores the state-of-the-art. These existing approaches are used by the DJ system to approximately determine important musical boundaries, after which the results are augmented in a rule-based algorithm to determine the correct segmentation boundaries. This algorithm is the topic of Section 4.2, and it is evaluated in Section 4.3. Section 4.4 summarizes the obtained results, and this chapter is concluded in Section 4.5 which gives some suggestions for possible extensions and future improvements.

4.1 State of the Art Analysis

In scientific literature, three general approaches to solve the structural segmentation problem are distinguished: novelty-based, repetition-based and homogeneity-based approaches [24, 44]. Novelty-based based approaches attempt to find sudden changes in the music, which typically occur on segment boundaries. Homogeneity-based approaches extract segments that show a certain consistency with respect to some musical properties,

e.g. the presence of a certain instrument or the usage of certain harmonies. Repetition-based approaches identify repeating sequences of features, e.g. notes, chord progressions and so on, in order to identify recurring melodies and themes and thus detect similar segments in the audio. These three categories will be discussed in more detail later in this section.

Before investigating these approaches in more detail, though, it is important to have an understanding of the concept of a so-called *self-similarity matrix* (SSM) [24]. Music is a self-similar signal in the sense that certain parts of it reoccur later in the signal, potentially in a slightly altered form (e.g. melodies, drum patterns, ...). A self-similarity matrix is a mid-level representation of the audio that describes these similarities between different parts. It is used by many segmentation algorithms as it adequately captures and visualizes the musical structure. To calculate such a matrix, the audio is split up into frames, and frame-wise feature vectors are extracted. The frame length and the type of features may vary depending on the task at hand, in order to capture features at a relevant time scale. The result is a sequence of frame-wise feature vectors $(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N})$. In this way, the audio is transformed into an appropriate feature space so that structural information can be more easily extracted. Then, each feature vector is compared with every other vector by calculating a similarity measure. More rigorously, given the sequence of frame-wise feature vectors $(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N})$ and a similarity measure $d(\cdot, \cdot)$, the self-similarity matrix for this sequence of feature vectors is calculated as: $D(i, j) = d(\mathbf{x_i}, \mathbf{x_j})$. In this matrix, high values will correspond to highly similar segments. Often, the cosine similarity measure is used [13, 44]:

$$d_C(\mathbf{x_i}, \mathbf{x_j}) = \frac{\mathbf{x_i} \cdot \mathbf{x_j}}{||\mathbf{x_i}||_2 \, ||\mathbf{x_j}||_2}$$

An equivalent representation is the self-distance matrix (SDM), where d represents a distance measure instead of a similarity measure, and where small values correspond to highly similar segments.

A self-similarity matrix of a music signal typically looks like the example in Figure 4.1. The structural properties of music can be seen in the matrix. Indeed, a frame extracted from a structurally coherent segment will most likely sound very similar to other frames from the same segment. This will lead to a high-valued block in the self-similarity matrix. If not all frames sound alike, but if some consecutive frames form a series of feature vectors that also occurs later (or earlier) in the audio, then diagonal stripes emerge, as illustrated in Figure 4.2. The specific appearance of the matrix and whether stripes or

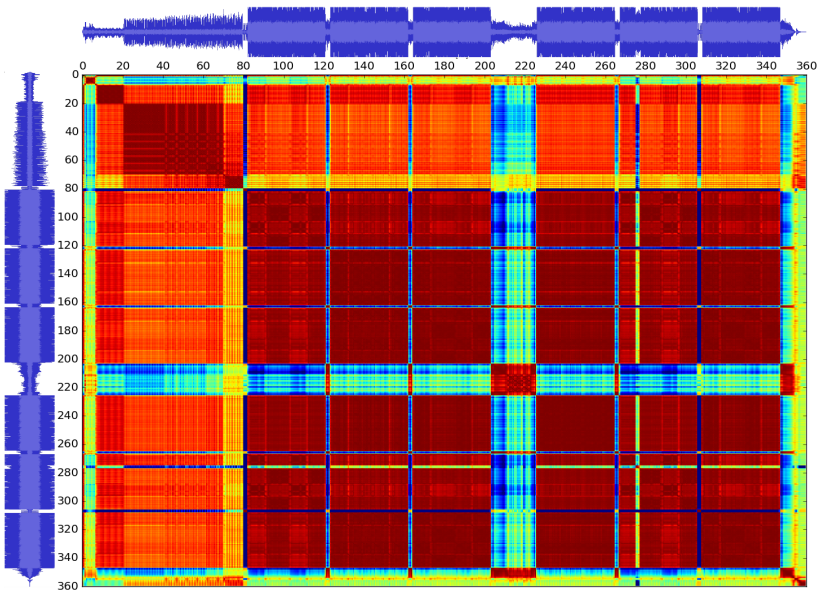


Figure 4.1: Example of a self-similarity matrix. The flow of the music is from the top left to the bottom right. One can clearly distinguish several self-similar blocks (red), separated by dissimilar sections (blue). The audio waveform is shown along both axes.

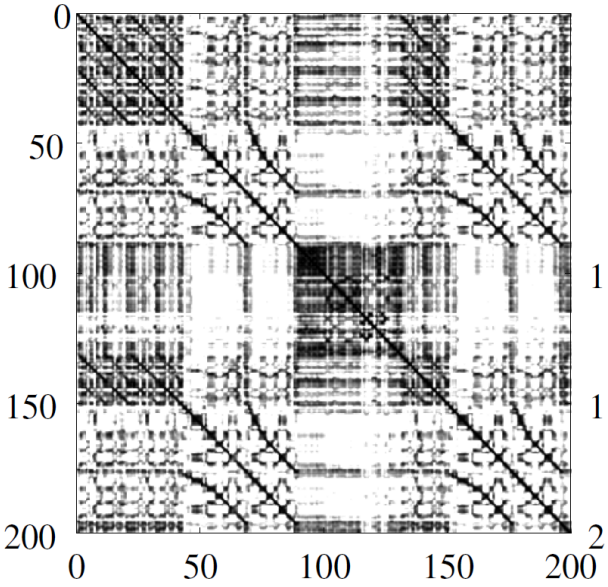
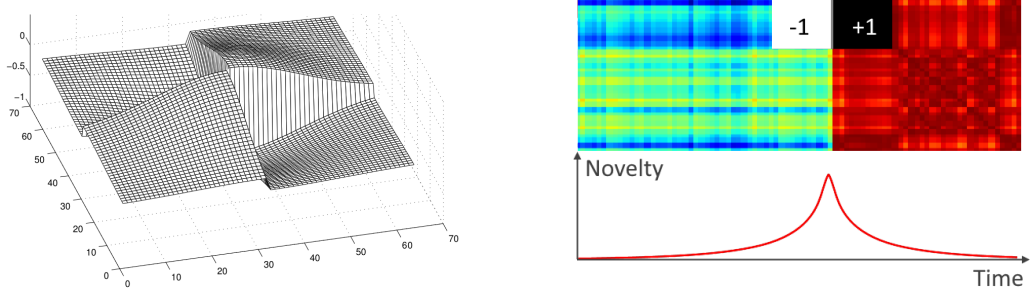


Figure 4.2: Example of a self-similarity matrix that shows striping behavior. The flow of the music is from the top left to the bottom right. (Source: [44])



(a) 64x64 Gaussian tapered checkerboard kernel [13].

(b) Illustration of the convolution operation of the self-similarity matrix with the checkerboard kernel along its main diagonal and the resulting novelty curve.

Figure 4.3: Illustration of Foote's algorithm for novelty detection.

blocks will appear depends on both the time resolution in which the signal was analyzed as well as the used features to describe the audio [44].

Novelty-based approaches attempt to locate points in time where large musical changes occur. This is motivated by the fact that these changes often occur on structurally important points, e.g. the transition from the buildup into the drop. The algorithm by Foote [13], which was the first to use the concept of a self-similarity matrix for the structural segmentation task [24], is a novelty-based approach. It exploits the observation that a significant point of change in music is one where the segment directly before is highly self-similar, as is the segment directly after, but where the segment before and after are not similar to each other. In the self-similarity matrix, this looks like a checkerboard pattern. This can be seen in Figure 4.1 as the occurrence of blue and green stripes separating red blocks.

The algorithm by Foote detects segment boundaries as follows. First, the self-similarity matrix of the audio is calculated. Foote uses the frame-wise Fast Fourier transform (FFT) power spectrum as feature vector, but other features could be used depending on the application. Then, a Gaussian tapered 'checkerboard' kernel (Figure 4.3a) is convolved with the self-similarity matrix along its main diagonal (Figure 4.3b). For each overlap, the elements of the matrix are multiplied element-wise with the overlapping kernel elements and summed together. When the kernel entirely overlaps with an approximately homogeneous

block in the matrix, the convolution value at that instant will be close to zero. However, when the kernel exactly overlaps with the checkerboard pattern that appears on points of significant change, it will lead to a high value. Correlating the checkerboard kernel with the self-similarity matrix along its main diagonal thus results in a *novelty curve*, i.e. a measure of novelty for each frame in the audio. From this novelty curve, segmentation boundaries can be found by means of a peak picking algorithm. Note that this algorithm operates solely on the input audio, and does not depend on any prior knowledge about musical concepts such as beats or downbeats, musical notes, phrases, tempo etc.

Homogeneity-based approaches are closely related to the novelty-based approaches. Indeed, the latter attempt to identify points of significant change, while the former detect the individual homogeneous sections that are separated by these points of change. The distinction between both approaches is that homogeneity-based approaches analyze the segments found by a novelty-based approach and group them into homogeneous clusters [44]. An alternative approach to homogeneity-based structural segmentation is assuming that a segment of the audio can be seen as a state in a Hidden Markov Model (HMM), and that the feature vectors in that segment are generated from the particular distribution belonging to that state. The HMM is trained to model the sequence of feature vectors of the music, after which the most likely sequence of hidden states is determined. While the idea behind this approach is very elegant, the resulting state predictions are often very fragmented as they appear to model individual sound events rather than larger musical parts [44]. Additional post-processing is thus needed to cluster the detected states into homogeneous structural segments.

Novelty-based approaches and homogeneity-based approaches both exploit the fact that within the same segment, the music will be constant in some of its properties (which change when transitioning to another segment, introducing some novelty in the audio that can be detected). One can also exploit the repetitive nature of music, where certain sequences are repeated periodically. These **repetition-based** techniques will thus typically exploit the striping behavior in self-similarity matrices, whereas the former two approaches exploit the blocking behavior. The striping behavior caused by these repetitions is shown in Figure 4.2.

An example of an approach combining repetition-based and novelty-based structural segmentation is the approach by Serrà et al [45]. It uses a type of features called *structure features* which capture global repetition-based information. These are then analyzed

using a local novelty-based procedure [24, 45]. Their algorithm works as follows. First, frame-wise feature vectors $(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n})$ are calculated. Then, for each frame, a new feature vector is constructed by incorporating the feature vectors of frames from the recent past, i.e. by concatenating for example N feature vectors. The new feature vector for the nth frame then becomes $\widetilde{\mathbf{x_n}} = (\mathbf{x_n}, \mathbf{x_{n-1}}, \ldots, \mathbf{x_{n-N}})$. This emulates some sort of short-term memory and allows the algorithm to detect repeating multi-frame patterns. The self-similarity matrix of the audio is calculated using these features. Subsequently, the matrix is thresholded with a varying threshold for each matrix cell, in order to identify the most important repetitions for each frame. From this self-similarity matrix, the *structure features* are constructed. The matrix is first 'skewed' to obtain a *time-lag matrix* instead of a *time-time matrix*. This means that a cyclic shift is performed on each row so that the elements originally on the diagonal now align with the first column: the first row is thus shifted by one element, the second row by two elements and so on. The time-lag matrix is then smoothed by convolving it with a Gaussian kernel. The structure features $(\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p_N})$ are then the rows of this matrix. A novelty curve is obtained by calculating the difference between successive structure feature vectors: $c_i = ||\mathbf{p_{i+1}} - \mathbf{p_i}||^2$. A simple peak picking strategy can then determine the segment boundaries.

One issue regarding structural segmentation research is that the **evaluation** of the proposed solutions is far from straightforward [24, 44]. The main reason for this is that the problem itself is not rigorously defined. Music does not follow strict rules, and while the structure of many musical pieces is similar in some aspects (e.g. most Drum and Bass songs contain an intro, buildup, drop etc), variations are always possible, and some songs might even have a completely different structure. Additionally, the problem is subjective in nature, as listeners tend to have different opinions about what the structure of a song is, e.g. where the segment begins, when two musical parts are dissimilar enough to be considered separate segments and so on. Usually, evaluation is done using some sort of recall rate or precision compared to some annotated ground truth, allowing some small temporal deviation [44]. However, for the reasons mentioned above, this is a far from straightforward approach, while it is very time consuming to annotate a sufficient number of audio files.

4.2 | Structural segmentation for Drum and Bass: a novelty and rule-based approach

In what follows, the structural segmentation algorithm is described that is developed for the automatic DJ system. The novelty-based approach by Foote, which has been discussed in the previous section, is used to get an initial estimate of the segment boundaries. The estimates are then augmented in a rule-based post-processing step. These rules are derived from expert knowledge of the Drum and Bass genre.

More specifically, the following assumptions are made regarding the properties of musically coherent segments, such as phrases and sections, in Drum and Bass music:

Assumption 4.1. *Large changes in the music signal often happen at (approximately) the same time as a segment boundary.*

For example, the drop, i.e. the transition from the buildup section into the main section, is often characterized by a sudden increase in energy, percussive and bass content and changes in melody or harmonic content.

Assumption 4.2. *Segments boundaries coincide with downbeats.*

A segment will thus never start in the middle of a measure: segments are an integer number of measures long.

Assumption 4.3. *Segments are a multiple of 8 measures long.*

As discussed in Section 1.2, a musical section is typically built from hierarchically smaller constructs called *phrases*. Such phrases are repeated several times, often with small variations, to create the higher-level section. In Drum and Bass, these phrases are typically 8 measures long [1], and hence all segment boundaries will lie at a multiple of 8 measures from each other.

Assumption 4.4. *Segment boundaries are sometimes preceded by a so-called break, i.e. short musical construct of one or two measures long that introduces or accentuates the newly introduced musical segment.*

Popular examples of such a break are short drum rolls or a short vocal sample just before the drop. Another example is the use of a breakbeat drum pattern of one or two measures at the end of a phrase. These breaks announce the end of the preceding phrase or section and accentuate the beginning of the new phrase after it. This observation

is important, because the beginning of the break introduces a high peak in the novelty curve one or two measures before the actual segment boundary, which might confuse the structural segmentation algorithm.

With these observations of the Drum and Bass genre in mind, the structural segmentation algorithm is constructed. The segments must be downbeat-aligned and must all lie at a multiple of 8 measures from each other. An additional requirement is that the algorithm should indicate whether a segment is part of the main section of the audio or not by labeling it with a label 'high' (for "high energy") or a label 'low' (for "low energy"). Listing 4.1 shows a high-level overview of the algorithm in pseudocode. In what follows, these steps are described in more detail.

The first step is to calculate two novelty curves using the approach by Foote [13], which is discussed above. The frames for feature extraction are half a beat long and the hop size is half the frame length (e.g. for 175 BPM and sample rate 44100 kHz, the hop size and frame length are respectively 7560 and 3780 samples). The novelty curves differ in the type of features used to analyze the audio: the first uses 13 MFCC coefficients of the frame as feature vector, the second the root mean square (RMS) value of the audio frame. The MFCC self-similarity matrix uses the cosine similarity, the RMS self-similarity matrix simply uses the absolute difference between RMS values as distance measure. The size of the checkerboard kernel is 64 frames, which corresponds to 16 beats. Choosing the kernel size this large ensures that novelties are detected at an appropriate time scale.

Both novelty curves detect points of significant change: the MFCC curve detects changes in the audio spectrum, which is related to the melodic content of the audio, whereas the RMS novelty curve detects changes in the overall energy of the audio, i.e. how quiet or loud it is. A significant segment boundary, e.g. the drop, will most likely show a change in melodic content as well as in its energy. Therefore, both novelty 'curves', which are essentially two arrays of novelty values for each frame, are multiplied element-wise to obtain a third novelty curve. Frames where both curves had a high value will now also have a high value. Frames where one of the novelty curves had a high value, but where the other one did not, will get a low value. Points of high novelty in the third curve are thus those points where the two initial novelty curves agree. These points are then extracted using peak picking, where only peaks with an amplitude of at least 1/20 times the amplitude of the highest peak are considered. Additionally, only the 20 highest peaks are kept if there are more than 20 peaks.

Listing 4.1: Structural segmentation algorithm pseudocode

```
# Step 1: calculate two novelty curves with Foote's algorithm
beat_length_samples = int(44100 * 60.0 / song.bpm)
frame_size = beat_length_samples/2
hop_size   = frame_size / 2
mfcc_features = framewise_features(audio, 'MFCC', frame_size, hop_size)
rms_features  = framewise_features(audio, 'rms', frame_size, hop_size)
ssm_mfcc = self_similarity_matrix(mfcc_features)
ssm_rms  = self_similarity_matrix(rms_features)
novelty_mfcc = convolve_kernel_along_diagonal(ssm_mfcc, N=64)
novelty_rms  = convolve_kernel_along_diagonal(ssm_rms, N=64)

# Step 2: combine the novelty curves and perform peak picking
novelty = novelty_mfcc * novelty_rms # Element-wise product
peak_amplitudes, peak_positions = PeakDetection(novelty, maxPeaks=20, threshold
    =0.05*max(novelty))

# Step 3: round the peak positions to the nearest downbeat
peak_positions, peak_amplitudes = [], []
for i in range(len(peak_amplitudes)):
        new_position = round_to_downbeat(peak_positions_[i])
        if new_position - peak_positions_[i] <= 0.4*downbeat_length:
                peak_positions.add(new_position)
                peak_amplitudes.add(peak_amplitudes[i])

# Step 4: Group the peaks at multiples of 8 measures from each other and pick
    the most likely 8-measure offset
bin_sums = [0,0,0,0,0,0,0,0]
bin_counts = [0,0,0,0,0,0,0,0]
for i in range(8):
        bin_sums[i] = sum(peak_amplitudes[j] if (peak_positions[j]-i) mod 8 == 0
             && not (exists peak 1 or 2 measures after peak_positions[j] with
            amplitude > 0.75*peak_amplitudes[j]))
        bin_counts[i] = sum(1 if (peak_positions[j]-i) mod 8 == 0 && not (exists
             peak 1 or 2 measures after peak_positions[j] with amplitude > 0.75*
            peak_amplitudes[j]))
bin_products = bin_sums * bin_counts # Element-wise product
offset = argmax(bin_products)

# Step 5: add segments and label them as 'high' or 'low'
segment_downbeats = []
segment_downbeats.add([offset])
segment_downbeats.add([song.last_downbeat - song.last_downbeat mod offset])
segment_downbeats.add([peak_positions[j] if (peak_positions[j]-offset) mod 8 ==
    0])
segment_downbeats.add([peak_positions[j]+1 if (peak_positions[j]-offset) mod 8
    == 7])
segment_downbeats.add([peak_positions[j]+2 if (peak_positions[j]-offset) mod 8
    == 6])
segment_downbeats = sort_ascending(segment_downbeats)
segment_labels = [calculate_segment_label(segment_downbeats[j])]

# Step 6: add more segments if necessary
for i in range(len(segment_downbeats)-1):
        distance = segment_downbeats[i+1] - segment_downbeats[i]
        if distance > 32:
                prev_seg_label = segment_labels[i]
                for j in range(0, distance, step=16):
                        current_label =  calculate_segment_label(
                            segment_downbeats[i]+j)
                        if current_label != prev_seg_label:
                                prev_seg_label = current_label
                                segment_downbeats.add([segment_downbeats[i]+j])
                                segment_labels.add([current_label])
```

At this point, the algorithm detected several peaks at positions $(p_1, p_2, \ldots, p_M)$ and amplitudes $(a_1, a_2, \ldots, a_M)$. However, from Assumption 4.2 follows that segment boundaries should coincide with a downbeat. For this reason, each peak position is rounded to the nearest downbeat. Peaks that are approximately in the middle of two downbeats, i.e. where the nearest downbeat is more than 0.4 times the inter-downbeat distance away, are discarded as in this case it is considered too uncertain to which downbeat the peak actually belongs. This step results in a new sequence of peaks $(\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_{\tilde{M}})$ and amplitudes $(\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_{\tilde{M}})$, where $\tilde{M}$ is potentially smaller than M. Note that it is now possible that multiple peaks lie on the same downbeat location: for example, this happens if the novelty curve had a peak just before and just after that downbeat.

Assumption 4.3 is a second restriction on the positions of the segmentation boundaries: they must not only coincide with downbeats, but all these downbeats must lie at a multiple of 8 measures from each other. To do this, the peaks are grouped into subsets where in each subset, all peaks lie at a multiple of 8 measures from each other. The most likely subset i, $0 \leq i \leq 7$, is determined, where the peaks in the ith subset lie at a multiple of 8 downbeats from the ith downbeat of the song. To determine the most likely candidates, one could sum the amplitudes of all peaks that are a member of the same subset: $sum(i) = \sum_{\tilde{p}_j \bmod i = 0} \tilde{a}_j$. The peaks in the subset with the highest sum are then selected as the correct boundaries.

This initial idea of summing the peaks in the same subset is elegant and simple, but it is flawed in two ways. Firstly, it does not take Assumption 4.4 into account, which states that a peak might occur one or two measures before the actual segment boundary because of the presence of a break pattern. For this reason, an additional filtering step is performed. A peak with downbeat position $\tilde{p}_j$ and amplitude $\tilde{a}_j$ is discarded if there is another peak one or two measures later, i.e. at position $\tilde{p}_k = \tilde{p}_j + 1$ or $\tilde{p}_k = \tilde{p}_j + 2$, and if its amplitude is smaller than 3/4 times the amplitude of that second peak, i.e. $\tilde{a}_j < 0.75\,\tilde{a}_k$. It is thus assumed that the novelty peak caused by the break preceding a segment boundary should be smaller than the peak of the boundary itself.

Secondly, simply summing the amplitudes does not consider the number of peaks in the same subset. It can be expected that there are multiple high novelty events contributing to the correct sum as a typical song has multiple phrase and section boundaries aligning with the correct 8-measure offset. Therefore, if there are many peaks in a subset, then this subset should get a higher weight. This can be done by multiplying each sum with the number of peaks that contributed to this sum. In this way, a subset with a large

number of medium-sized peaks will get a higher value than a subset with only one very high peak. This reduces the chance that the wrong subset is selected because of a single disproportionally high peak.

Once the most likely offset i is known, segmentation boundaries still need to be selected at multiples of 8 measures from this offset. This is done in a rule-based way, where the following downbeat positions are added to the list of structurally important downbeats:

- The first downbeat of the song that aligns with the determined 8-measure offset
- The last downbeat of the song that aligns with the determined 8-measure offset
- The positions of the peaks in the ith subset (which contributed to the highest sum)
- All the downbeats that align with the determined 8-measure offset and that are preceded by a peak one or two measure before it. This follows from Assumption 4.4.

The second to last step in the segmentation algorithm is to determine the type of the segment. Two types of segments are considered: 'high' segments that are energetic and loud, and 'low' segments that are calmer and more quiet. This will be needed in the cue selection process to estimate where the main section of the music is located (see Chapter 5). First, a running mean of the frame-wise RMS features, which were used for the construction of the second structural similarity matrix, is calculated. For each frame, the running mean value is the mean of the 32 RMS values before and 32 RMS values after it, using zero padding to ensure both signals have the same length. A segment then gets the label 'high' if the running mean RMS value of the second measure after the segment boundary is higher than the mean RMS value of all frames in the song. Since the main section of Drum and Bass is typically much louder than the rest, the segments that make up the main section will typically get the label 'high', and the others will get the label 'low'. This is a crude attempt to detect the drops in the song, i.e. the transitions between a buildup and a main section, as a transition from a 'low' segment to a 'high' one.

The final step of the algorithm is to introduce more segment boundaries in between two boundaries that are too far apart, here defined as more than 32 measures. When a large gap exists in between two boundaries, then this might indicate that the onset detection algorithm skipped some important events (because they were for example not detected by the specific choice of features used for the self-similarity matrices). Therefore, the algorithm iterates over the segments detected so far, and if a gap larger than 32 measures is found, it searches that gap for new segment boundaries. Candidates for this

are downbeats that lie at a multiple of 16 measures of the start of the gap. 16 measures are chosen as this typically is the length of a phrase. A candidate is selected as a new segment boundary if it would introduce a new type of segment, for example, if the segment coming before it was labeled 'high' and the newly introduced segment would get the label 'low'. In this way, the algorithm detects as many relevant points of change in the audio as possible.

4.3 | Evaluation

The algorithm described in the previous section is evaluated on its accuracy to detect the correct 8-measure offset of the segment boundaries and whether the characterization of the drop as the transition from a 'low' to a 'high' segment makes sense. It is thus not evaluated on whether it detects all segment boundaries in the song (i.e. if the number of false negatives is low), or if all the segments that are detected are indeed segment boundaries (i.e. if the number of false positives is low). The reason for this is twofold. First of all, determining which points in the music should be considered segment boundaries is ambiguous and subjective, as mentioned at the end of Section 4.1. This makes measures such as the precision, recall and accuracy unreliable, as it is sometimes unclear if a certain time instant should be considered a boundary or not such that even 'ground truth' annotations their correctness can be disputed. Secondly, annotating a sufficiently large number of Drum and Bass songs with ground truth segment annotations is a tedious and time consuming process. Finally, it should be noted that the algorithm introduced in this chapter is rather a simple baseline approach on which the automatic DJ system can be built, instead of an advanced algorithm that can accurately handle all Drum and Bass music, including difficult songs where some of the assumptions made earlier in the chapter would not hold. Because of the limited scope of the algorithm and the vague nature of the problem, performing an extensive evaluation using manually annotated songs is therefore not considered worthwhile.

Instead, a simple evaluation procedure is performed to evaluate the accuracy of detecting the 8-measure offset of a song. Each song in the training and testing corpus, i.e. 160 Drum and Bass songs, is annotated using the algorithm, and each transition from a 'low' segment to a 'high' segment is extracted. These transitions are then evaluated by listening if the detected segment boundaries are part of the correct 8-measure subset and if they coincide with a drop. The results of this evaluation are shown in Table 4.1.

	Number of songs	Fraction
Correct 8-measure offset	153	0.95625
Incorrect 8-measure offset	3	0.01875
Incorrect Assumption 4.3	4	0.025
Total	160	1.0

Table 4.1: Evaluation of the 8-measure offset detection accuracy

For 3 out of 160 songs, the detected offset is not correct. In one song, Assumption 4.4 is violated: a break pattern was used several times just after the beginning of a phrase, rather just before. In the second song, Assumption 4.4 holds, but the onsets two measures after the breaks were not prominent enough, so that the peaks introduced by the break were not discarded in the filtering step. For the third song, the reason for incorrect detection is less clear, but it is most likely because it has a very complex bass line and drum pattern.

For 4 out of 160 songs, Assumption 4.3 does not hold, i.e. not all segment boundaries lie at a multiple of 8 measures from each other. In all cases, there namely is one small 1, 2 or 4-measure phrase used between two segments, e.g. between the buildup and the drop or between the main section and the bridge. This effectively splits the song in half: the part before this small segment, and those after it. In both halves the segment boundaries lie at a multiple of 8 measures from each other, but there is an additional offset between the boundaries of the two sets, which makes the proposed algorithm unsuited for these types of songs.

For the remainder of the songs, Assumptions 4.3 and 4.4 hold and the calculated offset is correct. This leads to an accuracy of 95.6%. The detection is furthermore very certain: for only 9 out of 153 songs (5.88% of the correctly annotated songs), the second highest sum of the subsets has a value larger than half the value of the highest sum. For only 2 songs, it is larger than 0.75 times the highest value, and only one song has a value larger than 0.9 times the highest value (0.93 times the highest value to be precises). This indicates that the detection is quite robust and can predict the correct 8-measure offset with a high certainty in most cases. As with the beat and downbeat tracking, the accuracy of the segmentation module has been evaluated on a second held-out test set of 220 songs. There, 200 of the 212 songs with correct downbeat annotations also received segment annotations with the correct 8-measure offset, giving an accuracy of 94.3%.

It has also been evaluated if the characterization of the drop as a transition between

a 'low' and a 'high' segment is an effective approximation. For this, all 'low' to 'high' transitions in the 153 correctly annotated songs are evaluated by listening. The 153 songs contained 329 'low' to 'high' transitions. 271 of these are indeed a drop: this gives a precision of 82.4%. 38 songs contained at least one mislabeled segment. This means that for 75.2% of the songs, all labeled 'low' to 'high' transitions are correct. In general, two types of errors occurred when labeling segments based on their RMS energy. Firstly, many mislabeled 'high' segments are part of a buildup with high bass content. Sometimes, the intro or buildup also contained expressive percussive content, also leading to an incorrect classification. A third type of errors, which did not occur frequently, is when the main section temporarily decreased in energy so that its energy dropped below the 'high' threshold. This segment was then labeled 'low', but the next segment (which is part of the same main segment) was labeled 'high' again. In this way, a 'low' to 'high' transition is detected in the middle of the main section. An interesting note is that while evaluating, it was sometimes unclear if a certain transition should be considered a drop or not, i.e. if the music coming after the drop should be considered part of the main section. This again illustrates the ambiguity of the segment labeling task and the fact that music cannot always be described by a clearly defined set of rules.

4.4 | Conclusion

The structural segmentation algorithm described in this chapter uses the novelty-based algorithm by Foote [13] to detect structural boundary candidates in the audio. These candidates are post-processed using a rule-based approach. First, they are aligned with downbeat positions. Then, the candidates are filtered to reduce the number of false positives. A subset of the downbeat-aligned peaks, in which all peaks lie at a multiple of eight downbeats from each other, is selected as being correct by summing the amplitudes of the peaks belonging to the same subset, weighting the sums by the number of peaks contributing to them and selecting the maximum. Additional segments are added to ensure a sufficient number of segments per song, and the segments are assigned a label based on how energetic and loud they are. The algorithm has been evaluated on a set of 160 Drum and Bass songs, where 95.6% of the songs received correctly aligned annotations. An evaluation on a second test set of 220 songs gives an accuracy of 94.3%.

4.5 | Future work

The evaluation in Section 4.3 shows that augmenting existing structural segmentation approaches with simple and musically inspired rules leads to a segmentation algorithm with a good performance. However, it is important to note that the algorithm described here is created as a simple baseline approach, and that some assumptions are made that are valid for most, but not for all Drum and Bass music. Several improvements are still possible to make this algorithm more robust and elegant.

Firstly, more or different types of novelty functions can be used in order to get a more accurate initial estimate of segment boundaries. Informal experiments with a repetition-based self-similarity matrix using structure features as in [45] show promising results, but due to time limitations this has not been implemented in the final system.

Secondly, the RMS energy of a segment might not be an appropriate measure by itself to determine whether a segment is a 'main' segment (label 'high') or another type (label 'low'). It is important to know where the drop is, i.e. where the main section begins, to guide the automatic DJ in choosing meaningful cue points. Hence, the labeling process could be improved. This can be done by using a more advanced metric for whether a segment of the song is part of a main section or not, e.g. the combination of a loudness metric with a measure for the percussive content in the music, and more fine-grained structural information from a repetition-based or homogeneity-based approach. Alternatively, a machine learning classifier could be trained to perform the labeling or to detect the position of the drop.

A related improvement is introducing more labels besides 'high' and 'low'. For example, it could be interesting to distinguish whether a 'low' segment is a part of the intro, outro or bridge or buildup. However, this is again a problematic task due to the vagueness and subjectiveness of the definition of these types of segments.

In order to make this algorithm work in a less constrained setting, the assumptions that were made need to be weakened. Whereas Assumption 4.1 and 4.2 are quite reasonable, Assumption 4.3 and 4.4 are more limiting, and the errors made by the current algorithm were often related to these assumptions. To omit Assumption 4.3, the algorithm should be made more flexible by introducing a way to detect the occurrence of the rare 1-,2- or 4-measure phrases. To remove Assumption 4.4, the system needs a way to attenuate the

influence of break segments, or a more robust detection of the correct phrase boundaries. Introducing repetition-based segmentation techniques might help achieving this.

Chapter 5

Integrated automatic DJ system

The automatic DJ system now has access to a beat tracking, downbeat tracking and structural segmentation module. With these, it has an understanding of the tempo of the song and the musical structure. This provides the DJ with the necessary tools to choose appropriate cue points and create a continuous and seamless mix.

This chapter describes how the automatic DJ does this given these annotations. Section 5.1 gives an overview of the system architecture. It then describes how the automatic DJ system selects cue points and creates crossfades by time-stretching the songs and performing equalization. To create good transitions, the automatic DJ uses a machine learning model that evaluates the quality of crossfaded audio segments. This model is the subject of Section 5.2 and its performance is evaluated in Section 5.3. Section 5.4 summarizes the results from this chapter, and some suggestions for future improvements are given in Section 5.5.

5.1 | System overview

An overview of the different components of the automatic DJ system and their interactions is shown in Figure 5.1. The system starts from a music library of Drum and Bass songs in `.mp3` or `.wav` format. These songs are first annotated by the beat tracking, downbeat tracking and structural segmentation modules. Given the annotated audio files, the DJ can start mixing. This happens in a mixing loop. First, a song is selected for playback. The tempo of the mix is kept constant at 175 BPM. Hence, the first song might need to be time-stretched. Then, the next crossfade is created, which happens in two steps: first, the type of transition is chosen, and then the next song and its appropriate cue points are determined. There are three possible transition types, which differ in the type of overlapped segments ('low' or 'high') of the first song and the new song. The cue

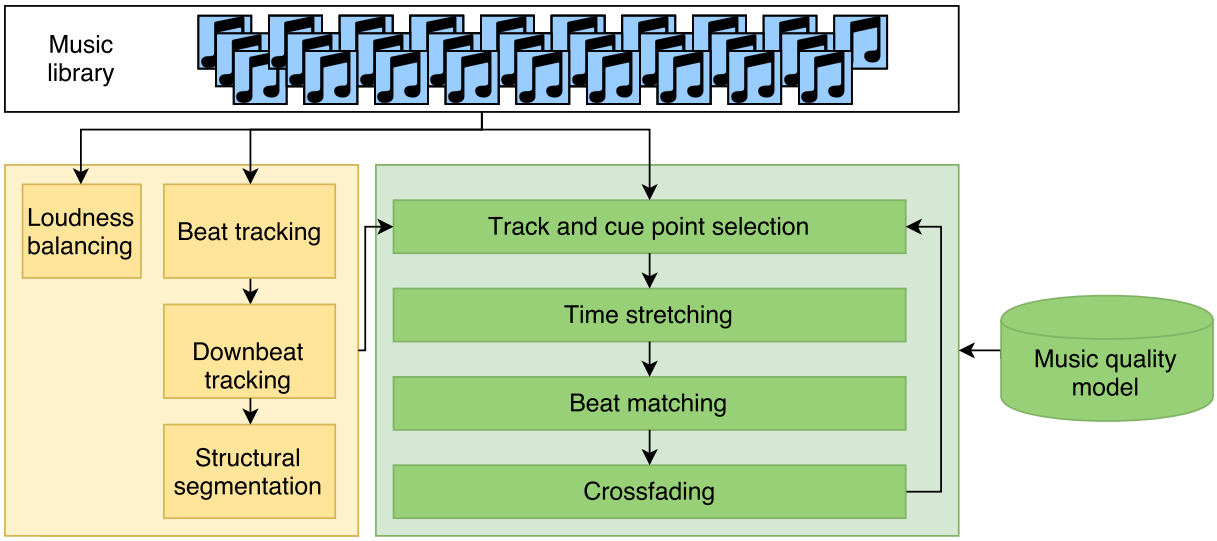


Figure 5.1: System architecture overview. On the left, in yellow, are the annotation submodules. On the right, in green, are the components used in the mixing loop.

point in the first song is selected based on the transition type. To determine the new song and its cue point, three songs are selected from the music library at random, and three candidate cue points are chosen for each song, where their location depends on the type of transition. Each potential crossfade is established by first time-stretching the new audio to 175 BPM, beatmatching it with the first song and applying volume fading and proper equalization. Each crossfade candidate its quality is then evaluated by means of the music quality model and the best crossfade is played to the user. This process is repeated for each song that is played, thus generating a continuous Drum and Bass mix. In what follows, these steps are discussed in more detail.

5.1.1 | Song preprocessing and annotating

Before the DJ can do any mixing, the songs need to be annotated with beats, downbeats and segment boundaries. This happens with the annotation modules, shown in yellow on the left of the system diagram in Figure 5.1. This preprocessing step happens in advance, and the resulting annotations are stored on disk so that they can be easily reused. Apart from beat, downbeat and segment annotations, a fourth annotation is made per song, namely the *replay gain* [46] of the audio. This is a measure for how loud the song is and it is expressed in decibel[1]. It is calculated using the corresponding method in the `Essentia` library [35]. Annotating this is necessary because not all songs are recorded at the same volume. When mixing songs together, it is important to compensate for this difference so that the resulting mix its volume is balanced. To do so, the audio of each song is amplified

[1]Not to be confused with the sound intensity, which is also expressed in decibel.

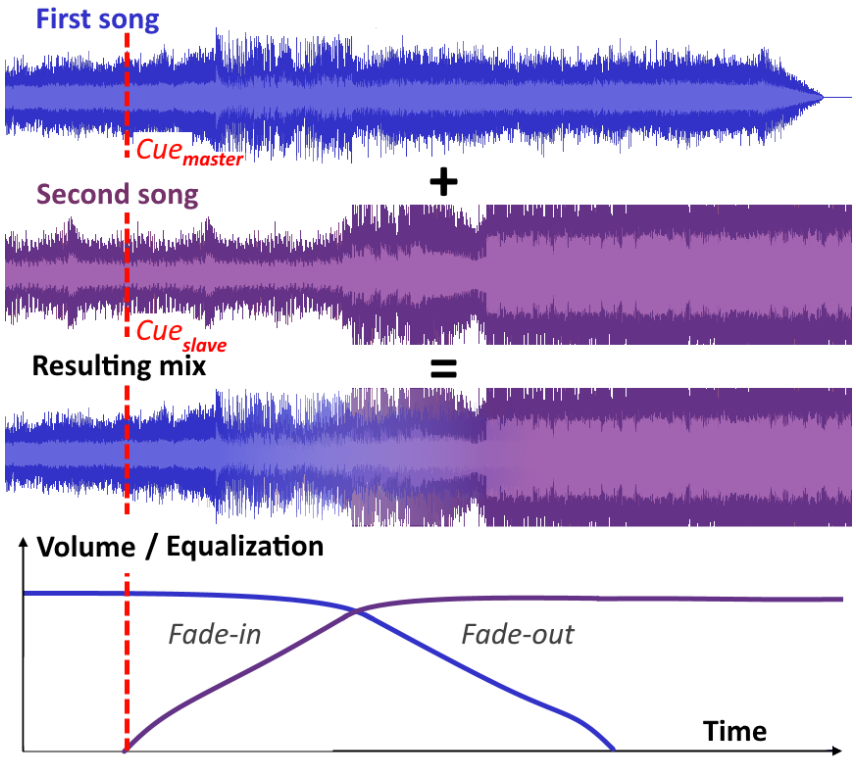


Figure 5.2: Conceptual illustration of the crossfade process.

or attenuated upon loading it into the application to obtain the same replay gain value of -16 dB.

5.1.2 | Cue points and transition types using musically informed rules

After the preprocessing step, the automatic DJ has access to a collection of annotated songs. It now has to use this information to determine which songs it wants to overlap, and how these can be overlapped. To do this, the automatic DJ chooses out of three types of transitions, which are described later in this section.

In general, the crossfading process goes as follows. Initially, the audio of a certain song (or a mix of a few songs) is playing: this will be called the *master audio*. The DJ now wants to transition to a new song, which will be called the *slave audio*. The DJ plays the slave audio beatmatched with the master audio, but with its output volume set to 0 so that the audience does not hear it yet[2]. He also turns down the equalizer settings of the slave audio to filter out low and high frequency content, to avoid saturation of the bass

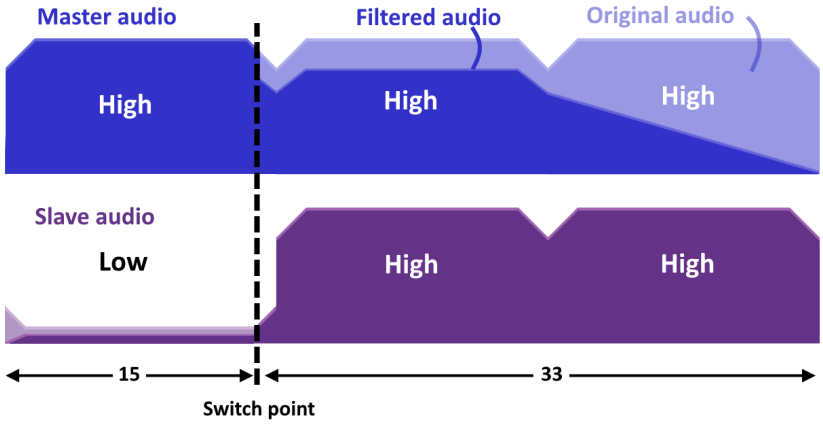
[2]A human DJ can still listen to the audio himself through his headset. This process of listening to the audio before the audience can hear it is called *pre-fade listening* [1].

and treble in the resulting mix. The DJ then starts increasing the slave audio its volume and equalizer settings to gradually introduce the new audio to the mix, while decreasing the equalizer settings and volume of the master audio. This is called the *fade-in*. Eventually, the slave audio plays at full volume and without equalization applied, while the master audio might still be playing but at a lower volume and with equalization applied. The master audio its volume and equalizer settings are then decreased further until it cannot be heard anymore. This is called the *fade-out*. A typical crossfade is illustrated schematically in Figure 5.2.

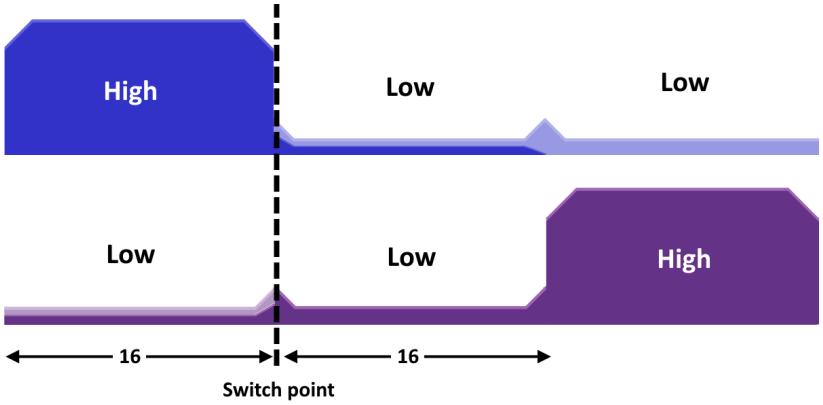
There are many ways to transition from one song into another. By carefully choosing the cue points in both songs and the speed at which the crossfade happens (i.e. whether it is a short or a long transition), a DJ can create transitions that are energetic, calm, smooth or sudden in nature. This has an important impact on the dynamics of the resulting mix. For the automatic DJ system, three types of transitions between songs are possible, characterized by the types of segments ('low' or 'high', see Chapter 4) that are overlapped. Furthermore, these transitions are assigned a downbeat-aligned cue point for each song, a fade-in length and a fade-out length. A transition is thus defined by a tuple ($type$, Q_{master}, Q_{slave}, $L_{fade-in}$, $L_{fade-out}$). The point at $L_{fade-in}$ downbeats after the start of the crossfade will be called the *switch point*. This is the point where the DJ plays the slave audio at full volume and starts fading and filtering out the master audio, i.e. where the 'dominant' audio in the mix switches from the master to the slave. Figure 5.3 illustrates the transition types, which are explained in more detail below.

- **Rolling transition**: This is a transition where the main section of one song goes over in the main section of another one. The energetic nature of the main section is thus continued by the second song. In the automatic DJ system, it is created by overlapping a phrase boundary in a 'high' segment in the master song with a 'low' to 'high' segment transition in the slave song. The switch point is one measure before the segment boundaries: switching exactly at the boundary instead of one measure before was found to give a less pleasing result. The fade-in length is 15 measures and the fade-out length is 33 measures, i.e. (about) 2 and 4 musical phrases respectively.

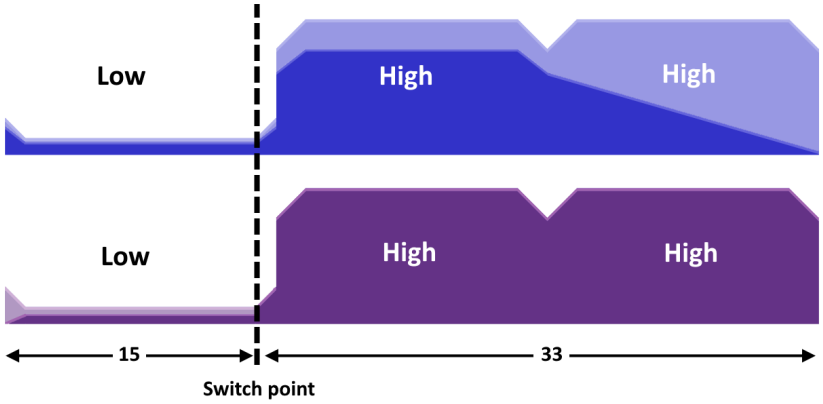
- **Relaxed transition**: This type of transition introduces a calm intermezzo in the mix. 16 measures before a 'high' to 'low' segment boundary in the master, the intro of the slave song starts playing. The switch point coincides with the 'high' to 'low' segment boundary in the master, and the fade-in and fade-out length are both 16 measures, i.e. 2 musical phrases.



(a) Rolling transition



(b) Relaxed transition



(c) Double drop

Figure 5.3: Illustration of the three transition types. In blue is the master audio, which is gradually faded out after the switch point. In purple is the slave audio, which is faded in and is at full volume from the switch point on. The light colored shape in the background represents the original audio, whereas the fully colored shape in the foreground is the volume-faded and equalized version.

	Relaxed	*Rolling*	*Double drop*
Relaxed	0	0.6	0.4
Rolling	0.5	0.5	0
Double drop	0.5	0.5	0

Figure 5.4: Transition matrix for the finite state machine controlling the transition type selection process. The element on the ith row and jth column denotes the probability of selecting type j if the previous transition its type is i.

- **Double drop**: *Double dropping* is a technique used in Drum and Bass mixing where the drops of two songs are aligned, so that both songs 'drop' simultaneously [1]. It is often used by professional DJs as an energetic climax in the mix. This type of transition is created by overlapping a 'low' to 'high' transition in the master with a 'low' to 'high' transition in the slave song. The switch point is one measure before the drop, and the fade-in and fade-out length are 15 and 33 measures long, i.e. (about) 2 and 4 phrases respectively.

These types are inspired by transitions that often occur in Drum and Bass DJ mixes. However, it should be noted that in reality, the crossfade do not comply to such simple rules, so that these crossfade types for the automatic DJ are merely simplifying abstractions of 'real-life' crossfading techniques.

The automatic DJ system selects the type of the next transition based on the type of the previous transition. Hence, the transition selection process is controlled by a *finite state machine* (FSM) with three states, i.e. one state for each transition type. The FSM transition matrix is shown in Figure 5.4. Note that certain successions of crossfade types are impossible. For example, if there just was a relaxed transition, then either a double drop or rolling transition is chosen[3]. When the previous transition was a rolling transition or a double drop, then a second double drop is impossible. This prevents the automatic DJ from playing too many songs at once and ensures that the mix is varied in nature. Using a FSM to control the transitions thus creates some progression in energy levels throughout the mix by alternating between energetic and relaxed crossfades.

5.1.3 | Time-stretching

When two songs are mixed together, it is important that their tempi match. To speed up a piece of audio, its length needs to be shrunk, and to slow it down it needs to be

[3]That is, if that is possible. It might occur in very particular situations that no double drop or rolling transition can be realized, e.g. if the remainder of the master song is too short to establish such a transition. In that case, the fall-back option is always a chill transition, as this can always be generated.

stretched. A naive approach to change the playback speed would be to sub-sample the audio to make it faster or to interpolate it to make it slower. However, this does not only change the tempo of the song, but also modifies its pitch. To change the tempo of the song without altering its pitch, more advanced *time-stretching* techniques have to be used, which can be divided into time-domain and frequency-domain approaches [47].

Frequency-domain approaches do not modify the audio signal directly, but instead transform it first into the frequency domain using e.g. the Fast Fourier Transform (FFT), perform modifications in that domain, and then transform it back to the time domain. The phase vocoder [48] is an example of such an approach. These techniques are able to deal better with larger time-stretching factors compared to time-domain approaches, as the artifacts introduced by the latter get worse with the amount of stretching applied. However, frequency-domain approaches are computationally much more demanding and more difficult to implement. Additionally, they are subject to their own specific artifacts. For example, they introduce 'transient smearing' [48], which are distortions around sounds produced by instruments with sharp attack times, such as percussion instruments and piano music, making them sound dull and reducing their expressiveness. These artifacts even occur for stretching factors close to one. These drawbacks are troublesome, as the automatic DJ system needs to perform time-stretching often, so a short computation time is desired, and it works on music with high percussive content. Therefore, a frequency-domain approach does not seem suitable to be used in this system.

Time-domain approaches are much faster compared to frequency-domain approaches as they modify the audio waveform directly, without performing any transformations to another representation domain [48, 49]. Furthermore, they do not introduce the same transient smearing artifacts. Time-domain approaches do introduce different types of artifacts, but these distortions are only noticeable at significant tempo changes. They are also more noticeable when audio is stretched out, i.e. converted from a higher to a lower tempo. The maximum amount of time-stretching in the automatic DJ system is typically limited to less than a 5% change (for example, stretching a 170 BPM Drum and Bass song to 176 BPM, i.e. a rather low Drum and Bass tempo to a rather high tempo, is a 3.5% change), so even if artifacts are present, they should not be very noticeable. To minimize the distortion even further, the tempo of the resulting mix can be chosen in such a way that it is higher than or equal to the original tempo of most Drum and Bass songs. For this reason, the current prototype operates at a constant 175 BPM, a tempo that is faster than or equal to the tempo of most Drum and Bass songs. This ensures that

only a small number of songs will be stretched out, leading to even less noticeable artifacts.

In what follows, the time-stretching algorithm used in the DJ system is explained in more detail. More specifically, the "Waveform Similarity Overlap-Add" or WSOLA algorithm is used [49]. It is one of the variants of the overlap-add (OLA) algorithm: another example is the "Synchronized Overlap-Add" or SOLA algorithm. The basic operation of the OLA algorithm is to divide the audio signal into (potentially overlapping) frames at regular intervals. These audio frames are then shifted along the time axis with an appropriate time-stretching factor and overlapped at their ends. They are then crossfaded to realize a smooth transition between frames. This effectively stretches out or shrinks the audio without altering its pitch. However, this simple implementation does not consider phase information when overlapping the separate audio frames, i.e. whether the overlapping portions of both frames match well. This can lead to a phase mismatch in the overlapping portions and inferior sound quality as a result. Adaptations of the original OLA algorithm, such as SOLA or WSOLA, therefore introduce a tolerance window when selecting or overlapping the audio frames to ensure proper phase synchronization. In the SOLA algorithm, the frames are chosen at fixed intervals, but when overlapping some additional shift is allowed to ensure that the overlapping portions of the frames have a maximal cross-correlation, which leads to a maximal 'output similarity'. The WSOLA algorithm on the other hand does not maximize the output but rather the 'input similarity': frames are not extracted at exactly the same intervals, but rather within some tolerance window around the theoretical offset. The candidate frame of which the overlapping part is maximally similar to the overlap portion of the already established output audio is then used. This ensures continuity during the overlap as the overlapping portion of the new frame will resemble the original audio as closely as possible. The difference between the SOLA and the WSOLA algorithm is thus that in the former, the choice of the next frame is fixed but the overlap with the already established audio is flexible, whereas the latter allows some flexibility when choosing the next frame. The WSOLA algorithm is illustrated in Figure 5.5.

The specific parameter settings for the WSOLA implementation have been determined by evaluating their influence on the audio quality in informal experiments. The frame length is 4410 samples, corresponding to 100 ms of audio. On both sides, the overlapping portion of a frame is 252 samples or about 5.7 ms long. The tolerance window for searching a matching frame is 400 samples wide, so 200 samples or 4.5 ms on either side of the theoretical offset. The searching window is relatively short compared to the window

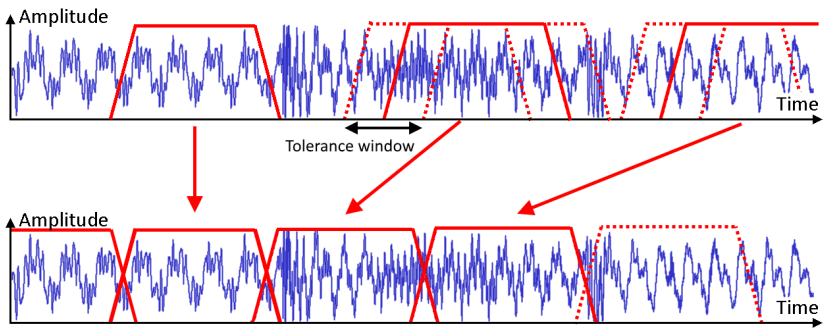


Figure 5.5: Illustration of the WSOLA algorithm

length. This constrains the algorithm when finding a matching frame, but ensures that the frames are found at approximately equal intervals in the input audio. The latter is important, as a large window length was found to introduce a 'warping' effect where the audio appears to slow down and speed up on a very short time scale because the frames that make up the output audio have been selected at non-regular intervals in the original audio. A smaller tolerance window thus guarantees a perceptually more stable output tempo. Furthermore, the tolerance window was found to be sufficiently large, because very little perceptual artifacts were observed.

5.1.4 | Music equalization using shelving filters

During a crossfade, a DJ typically performs some equalization, i.e. adjusting the balance between the frequency components of the mixed songs. To do this, DJ mixing equipment typically has three tunable knobs that can attenuate or boost the bass, mid and treble of each song separately [1]. In the automatic DJ system, similar filtering capabilities are implemented by means of a low and a high *shelving filter* [50], which are used to control the bass and treble respectively.

A shelving filter applies a constant increase or decrease in gain to all frequencies above or below a certain cut-off point. The main advantage of using such a shelving filter over a high- or low-pass filter is that the latter apply a progressive rather than a constant gain increase or decrease [50]. A digital shelving filter can be implemented as a special case of a biquad filter [51], i.e. a digital filter with a transfer function of the form:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}}$$

The parameters $a_0, a_1, a_2, b_0, b_1, b_2$ can be derived from the desired gain in decibel, the *shelf midpoint frequency* or cutoff frequency and a Q value which determines the steep-

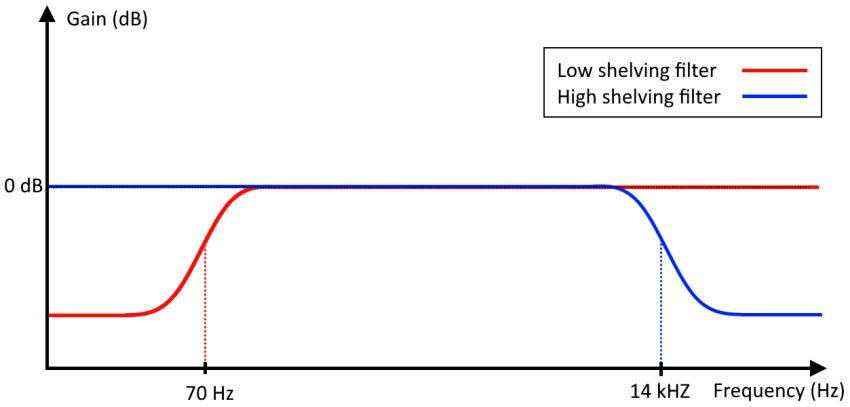
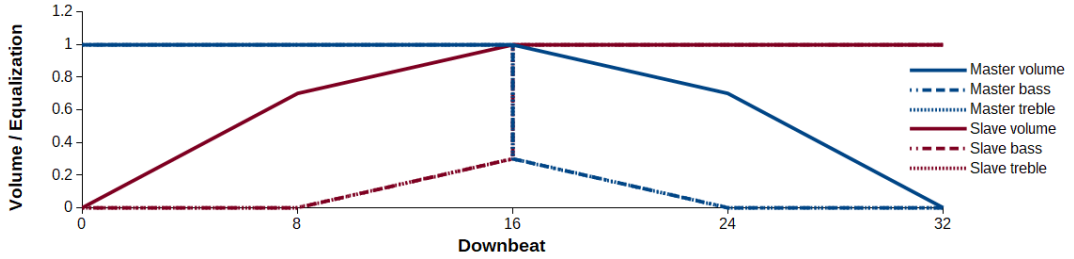


Figure 5.6: Illustration of the frequency response of the shelving filters (not to scale). The low shelving filter attenuates frequencies lower than the cutoff point but leaves higher frequencies untouched. The high shelving filter shows a mirrored behavior in the higher frequency regions.

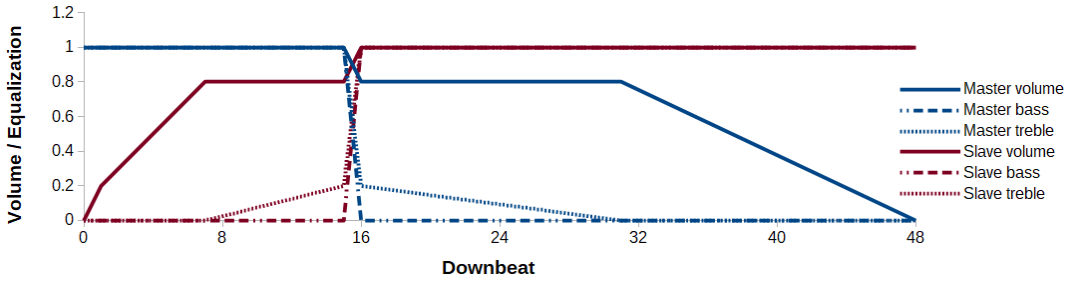
ness of the filter. The derivation of the specific parameter values for the shelving filters can be found in the "Cookbook formulae for audio EQ biquad filter coefficients" by Robert Bristow-Johnson [51]. The shelving filters their frequency response is shown in Figure 5.6. The amount of filtering or boosting can be changed by adjusting the gain and recalculating the filter values.

In the current implementation of the automatic DJ system, the filter parameters are calculated using the `yodel` python package [52]. The filters are applied to the audio using the `lfilter` function in the `scipy.signal` package, as it was found to be significantly faster than the `yodel` filtering implementation. The Q value is set to $\frac{1}{\sqrt{2}}$, as this leads to a maximum steepness of the filter [51]. The gain can vary from 0 dB (no filtering) to -26 dB, and the shelf midpoint frequency for the bass and treble filter are 70 and 13000 Hz respectively. These settings are inspired by the equalizer specifications of professional DJ mixing equipment [53].

The amount of filtering gradually decreases during the fade-in and increases during the fade-out. The specific progression of the filters depends on the type of crossfade. This is illustrated in Figure 5.7. In a relaxed transition, the fade-in and fade-out happen gradually (Figure 5.7a): the new song slowly fades in, and after the switch point the old song slowly fades out. The bass and treble of the slave song are mostly filtered out just before the switch point, and the roles of the master and slave switch instantaneously at the switch point. In the double drop and the rolling transition, the slave song is introduced more quickly (Figure 5.7b): its volume quickly increases over the span of one measure, after which the volume keeps rising until the 7th downbeat. Then, the volume of the



(a) Volume and equalization settings during a relaxed transition.



(b) Volume and equalization settings during a rolling transition or a double drop.

Figure 5.7: Volume and equalization settings throughout a crossfade.

slave song levels out and is kept constant until the switch point. At the switch point, the master and slave switch roles as 'dominant song' in the mix. The master song its volume is now held constant until 16 downbeats after the switch point. Afterwards, it fades out linearly. These transitions were designed by hand to create pleasing crossfades and mimic how crossfades are performed by professional DJs.

5.1.5 | DJ mixing loop

Selecting which songs to play, establishing the transitions between the different songs and playing back the resulting audio happens in a DJ mixing loop. Pseudocode of this process is shown in Listing 5.1. This process is clarified in this section.

First, the DJ selects a random song from the music library and starts playing it. It then enters the DJ loop that repeats for every new song that is played. At the beginning of the loop, there always is some audio playing, i.e. the master audio. This can be a single song, or the audio of several songs already mixed together. A new transition for the master audio is created by selecting a transition type using the Finite State Machine and a corresponding cue point. A more detailed description of these types has been given in Section 5.1.2. Given the new cue point in the master, it is known that the master audio up till that cue point will not be altered anymore. This piece of audio can thus be queued in a buffer for playback. After queuing, the master audio is cropped so that it begins at its cue point $Q1$. This ensures that the master audio buffer does not grow exceedingly

Listing 5.1: Pseudocode for the DJ mixing loop

```
master_song = random song from music library
master_audio = master_song.audio
Q2 = 0

repeat:
        # Determine the master cue point and the type of transition
        fade_type, Q1, L_in, L_out = create_transition_master(master_song,
            not_before=Q2)

        # Crop the master audio and play up till the cropped part
        end_sample = convert_to_sample(Q1)
        audio_to_play = master_audio[:end_sample]
        master_audio = master_audio[end_sample:]
        play(audio_to_play)

        # Choose some random successor candidates and determine the best
            cross fade
        slave_song_candidates = select random songs in music library
        best_score = -Inf
        for s in slave_song_candidates:
                # Determine cue point candidates for the given type of fade
                Q2_candidates = s.getCueCandidates(fade_type)
                for Q2 in Q2_candidates:
                        # Apply the crossfade
                        crossfaded_audio = apply_transition(fade_type,Q1,Q2
                            ,L_in,L_out, master_audio,s.audio)
                        # Estimate its quality
                        score = evaluate_audio_quality(crossfaded_audio)
                        if score > best_score:
                                best_audio = crossfaded_audio
                                best_song = s
                                best_Q2 = Q2

        master_song, master_audio, Q2 = best_song, best_audio, best_Q2
```

large.

The remainder of the master audio will now be overlapped with some new audio, so that a new transition between songs is established. First, 3 songs are chosen from the music library as candidates for the next song. Then, 3 cue point options are determined for each song. The first cue point option is chosen as described in Section 5.1.2, i.e. based on the type of transition and the annotated structural boundaries in the slave song. The second cue point is the downbeat 16 measures after the first. The third cue point is either the downbeat 16 measures before the first, or 32 measures after the first if the first cue point occurs before the 16th downbeat of the song. The candidate cue points are chosen like this because in this way, the automatic DJ might overcome segment misclassifications (e.g. a 'low' segment that is labeled as 'high') automatically by detecting that another offset from the detected boundary sounds better when overlapped. The crossfade is established for each song and cue point candidate, which happens by trimming and time-stretching the slave audio and applying the volume and filtering profiles as described in Section 5.1.2. Volume fading and filtering is also applied to the master audio its fade-out section. The slave audio is beat-aligned with the master audio and added to it, resulting in the mixed audio. Its quality is then estimated by means of the music quality machine learning model, which is described in more detail in Section 5.2. The audio that leads to the best quality is selected as the new master audio, and the loop starts again to continue the mix.

5.1.6 | Notes on the implementation of the automatic DJ system

This section gives some additional remarks with respect to the software implementation of the automatic DJ system prototype. A more detailed description of this prototype, i.e. where to download it, installation instructions and a usage manual, is provided in Appendix A.

The prototype is a command-line application developed in Python. It uses multiple threads to separate the processing of the user input, the DJ controller loop and the audio play loop. Having a separate thread to control the user input ensures that the command line interface does not block while the DJ is playing music. Also the DJ controller and the playing of the audio are two separate threads. The DJ controller loop generates audio segments that need to be played, and adds them to a thread-safe queue. The audio play

loop fetches the audio fragments and plays them. It is of course important that the queue is always filled with some audio to be played. Hence, the DJ controller loop should generate music faster than it can be played. Performing and evaluating one overlap takes 2.14 s on average with a standard deviation of 0.52 s. Determining the best crossfade, i.e. evaluating 3 songs with 3 cue points each, takes 19.3 s on average, with a standard deviation of 3.69 s. These results were obtained by performing 18 crossfades in the DJ application, i.e. 162 candidate crossfade evaluations. At 175 BPM, 19.3 seconds is less than 16 measures of music. Hence, if there are on average at least 16 measures between two transitions, then the application will have enough time to generate a new transition in time for playback. 16 measures is a rather quick succession of transitions: therefore, this should not be an issue. The automatic DJ furthermore attempts to select cue points at least 16 downbeats apart, which guarantees there is enough time between two crossfades such that the automatic DJ will never run out of audio to play.

5.2 Track and cue point selection using a machine learning model to estimate crossfade quality

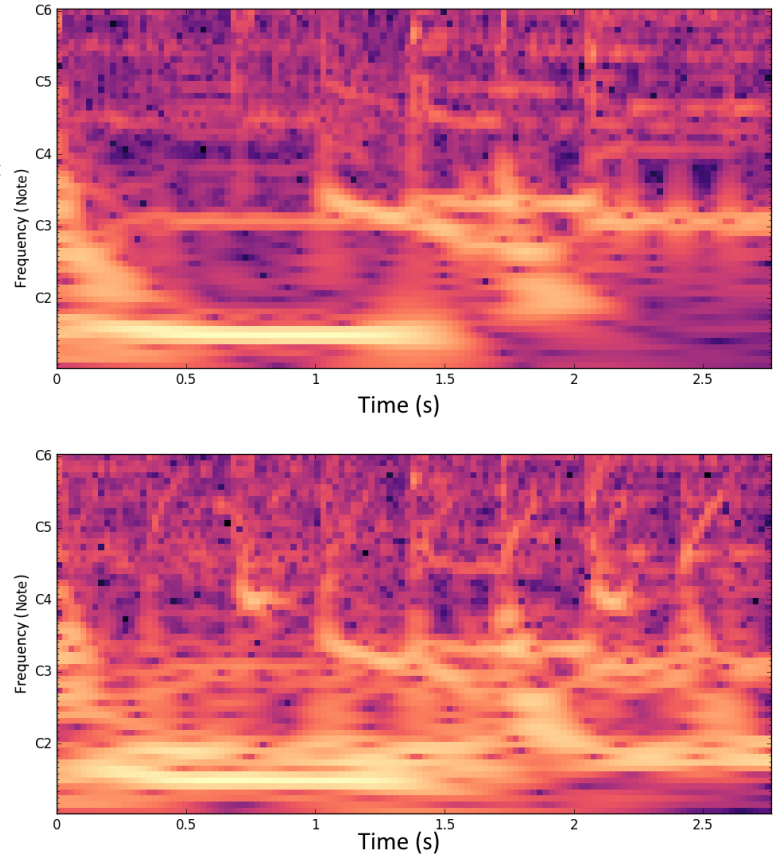
Section 5.1.5 explains how the best next song and cue point are selected. To choose from several crossfade candidates and distinguish between well mixed and badly mixed Drum and Bass music, the DJ system needs a quality measure. To this end, a machine learning classifier has been constructed, which is detailed in this section.

More specifically, a two-class support vector machine (SVM) is constructed that operates on music extracts of one measure. The binary classification by the SVM is transformed to a probability estimate using Platt scaling [54][4]. This probability can be interpreted as a quality rating between 0 and 1. It is not trivial to define when a measure-length segment of a mix sounds good or bad and to create a meaningful training set for this problem. To do so, the assumption is made that a good mix segment should sound like a measure extracted from an unmixed, 'pure' Drum and Bass song. Training examples for the positive class are thus created by simply extracting measure segments from Drum and Bass songs. To generate examples for the negative class, it is assumed that bad mix segments rather sound like the DJ performed bad equalization and overlapped random songs that do not fit together (but still beatmatches them). Two types are distinguished. The first is when the DJ overlaps two 'high' segments without any filtering of the low or

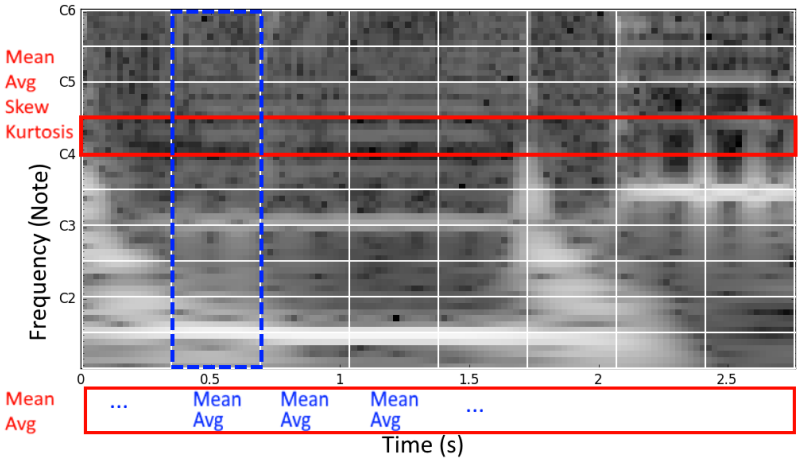
[4]The scikit-learn library is used for this machine learning task. Details on how this scaling works can be found on the user manual on `scikit-learn.org/stable/modules/svm.html`

high frequencies, which leads to a saturated bass and/or treble in the resulting mix. The second is when the DJ overlaps an unfiltered 'low' segment with a filtered 'high' segment. In this case, the listener will be able to hear artifacts of the filtered beats of the 'high' segment because the 'low' segment is not percussive or loud enough to mask these artifacts. For each song, 20 downbeat-aligned measures are extracted from the audio: 15 from these from a 'high' segment in the audio and 5 from a 'low' segment. These are used as examples for the positive class. By selecting both 'low' and 'high' segments, the classifier should learn what the main section sounds like as well as the intro, bridge and buildup. From the same 20 segments, the examples of the negative class are created. For each 'high' segment, another 'high' segment from a different song is selected and added to it. The result is divided by a factor $\sqrt{2}$ to balance out the increase in loudness. For each 'low' segment, a 'high' segment from a different song is selected and added to it after applying low- and high-shelf filtering to the 'high' segment. In total, 4680 training examples have been generated from the 117 training songs, and 1720 test examples from the 43 test songs.

From these measure segments, features are extracted. These should describe how 'cluttered' the audio fragment is: the underlying assumption is that good Drum and Bass will typically contain more (short) silences and a clear separation between different sounds, whereas randomly overlapped Drum and Bass will have much more musical activity throughout the entire segment as a result of an inappropriate overlap and will therefore sound much more cluttered and chaotic. The first step is to calculate the Constant-Q Transform (CQT) [55] spectrogram of the audio fragment. There are 60 bins along the frequency axis, with the lowest bin at $f \approx 32.70$ Hz, i.e. the frequency of the C1 note, and 12 bins per octave. The hop length is 512 samples. Two spectrograms are shown in Figure 5.8a to illustrate the difference between the spectrogram of an extract from unmixed Drum and Bass and one from a corresponding badly mixed example. The spectrogram amplitudes are transformed to a decibel scale and normalized between 0 and 1. The spectrogram is then split up into rectangular windows as illustrated in Figure 5.8b. Along the frequency axis, the windows have a height of a halve octave (6 CQT frequency bins) and do not overlap. Along the time axis, the windows have a width of one quarter beat, i.e. 1/16 times the width of a measure, and a hop size of half the window length is used. For each of these windows, the mean of the spectrogram values is calculated. Then, the mean, variance, skew and kurtosis are calculated of the means of all windows that fall in the same halve octave, leading to 4 features per halve octave or 40 features in total. Also the mean and variance of all windows that fall in the same quarter-beat time segment are calculated. Of these 32 mean and 32 variance values, again the means and variances are



(a) CQT spectrogram of unmixed Drum and Bass (top) and badly mixed, cluttered Drum and Bass (bottom).



(b) Illustration of how the features are calculated for the audio quality model. Note that in this figure, the bins along the time axis are not overlapping nor are they 1/16th measure long, as is the case in the actual implementation. This is done to keep the figure clear and concise.

Figure 5.8: Illustration of the CQT spectrogram features for the audio quality model

calculated, giving 4 additional features for the measure extract. These features capture the distribution of quiet and loud rectangular windows in the spectrogram and in this way describe how chaotic and cluttered the music is. The features are also standardized such that their distribution has zero mean and unit variance. The mean and variance of each feature are estimated using the features of the training set examples, using which the features of each example can be scaled.

To estimate the quality of an entire crossfade, each downbeat of the crossfade is evaluated using the quality model. The mean of the logarithms of all probability estimates is computed, giving a mean log-probability value for the entire crossfade. When comparing different crossfades, the one with the highest mean quality, i.e. the least negative value, is selected as the best candidate.

5.3 | Evaluation

In this section, the ability of the automatic DJ to distinguish between good and bad crossfades using the quality model from Section 5.2 is evaluated. This is done in two ways: by evaluating the machine learning model performance on individual downbeat segments and by performing a subjective user test.

The classifier has been trained using cross-validation to determine the best regularization parameters. The SVM classifier uses a soft margin and hence has a C parameter, denoting how severe errors are penalized (a high C implies a small amount of regularization). It also has a γ parameter which determines the kernel width. An radial basis function (RBF) kernel is used. The optimal values for C and γ were found to be 100 and 0.001 respectively. These lead to a logarithmic loss of 0.45 and an accuracy of 80% for the train set and 0.55 logarithmic loss and 74% accuracy for the test set.

To assess whether the proposed approach for crossfade quality estimation is able to distinguish between good and bad crossfades, a subjective test is performed. Twelve users participated in this test. Ten sets of crossfades are generated by the automatic DJ, with three variants of the same crossfade in each set. For five of these sets, the three variants in the same set are a transition between the same two songs, but other cue points are used. For the other five sets, the three variants in each set start from the same song but transition into three different songs. In this way, it is evaluated if the automatic DJ can choose between different cue points in the same song on the one hand and different

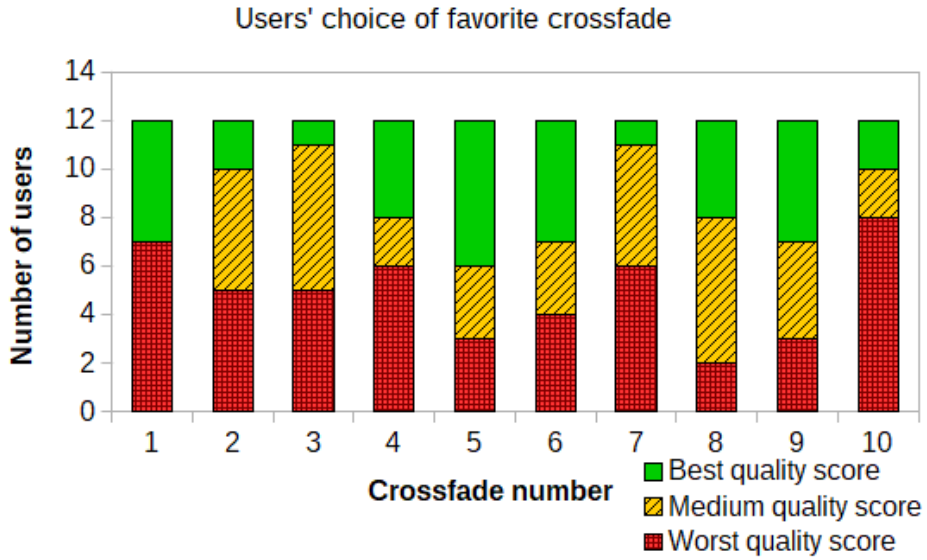
song combinations on the other hand. Recall that in the crossfade selection process, the automatic DJ evaluates all three variants in each set using the quality model and selected the best variant for playback[5]. The goal of the user test is thus to assess if the choices made by the automatic DJ are good, i.e. if he selects the 'best' crossfades from the point of view of the listeners. To do so, the twelve participants were asked to listen to all crossfades and rate them with a quality rating between 1 and 5, where 1 means that it is a crossfade of good quality and 5 means that it is of bad quality, i.e. it sounds like the DJ just overlapped two random songs without considering their musical compatibility at all. They also ranked the crossfades in each set of three from "most preferred" to "least preferred".

The user their rankings are visualized in Figure 5.9: it shows for each set of three crossfades how many times each crossfade was chosen by the users as most preferred and as least preferred. Analyzing these charts indicates that user preferences heavily differ for almost every crossfade. Hence, it is not possible to evaluate the crossfade optimization process on its accuracy of choosing the "best" crossfade candidates, since there appears to be no consensus among users on what the "best" crossfades are.

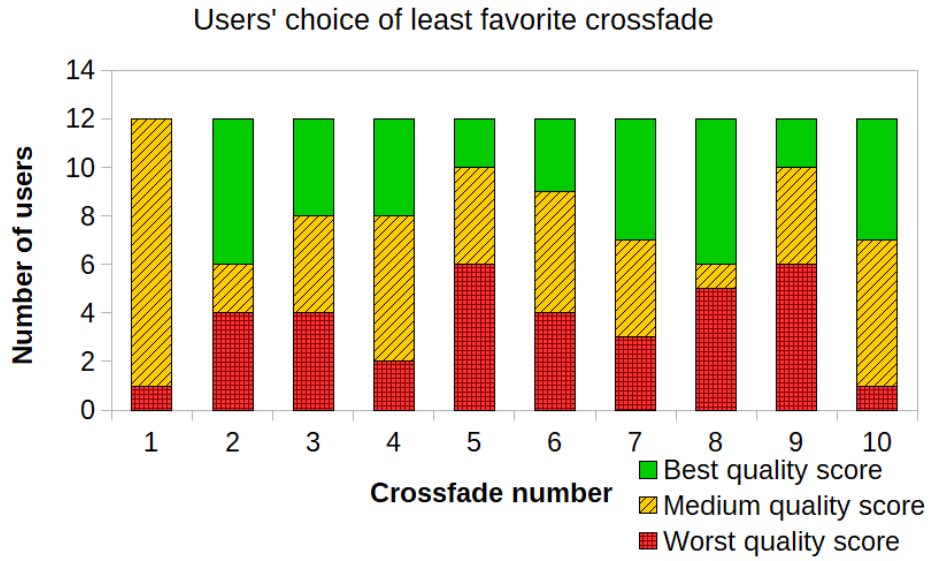
Instead, it is evaluated what the mean user rating is of the crossfades that the automatic DJ selected for playback. The goal of the automatic DJ is to select crossfades such that the mean quality rating is minimal for all users and over all crossfades in the mix (recall that a rating of 1 means a high quality crossfade). First, the mean of all 12 users their rating is calculated for each of the thirty crossfades. Then, for the DJ its selection of 10 crossfades, i.e. one selected crossfade for each of the sets of three crossfades, the mean user ratings are averaged to give a measure for the overall quality of this selection. The average rating is also calculated for every other possible selection of crossfades. There are ten sets of crossfades, with three options per set: this gives 3^{10} or 59049 possible crossfade selections. The average rating of the DJ his selection is compared with these other possibilities. This is what is illustrated in Figure 5.10a[6]. The average rating of the selection by the DJ is 3.52 out of 5. More than half of the other possible selections (50.2%) have a better mean score than the DJ its selection, which indicates that the used algorithm does not perform any better than a random selection of crossfades might. However, upon

[5]In reality, 9 candidates are evaluated by the DJ for each crossfade, i.e. 3 new songs and 3 cue points in each new song. However, only three candidates are presented to the user to limit the time needed to perform the user test.

[6]The 'gaps' in these figures are not an error, but a consequence of the fact that some mean quality rating errors cannot be obtained for these ten crossfades given the particular user evaluation scores in this test, regardless of which combination of ten crossfades is selected.

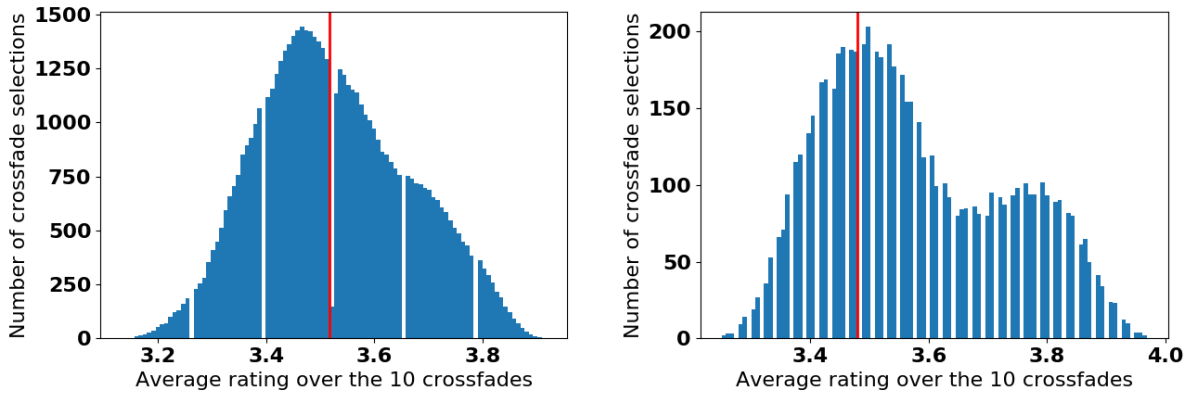


(a) The users their preferred candidate for each of the ten crossfades.



(b) The users their least preferred candidate for each of the ten crossfades.

Figure 5.9: The users their preferred (top) and least preferred (bottom) crossfades for each set of three crossfades. The green bars indicate the number of users that chose the crossfade with the highest DJ quality score. The yellow bars show the number of users that chose the crossfade with the second best score of the automatic DJ as their most preferred (top) resp. least preferred crossfade (bottom). The red bars indicate how many times the crossfade with the lowest automatic automatic DJ quality score was ranked first resp. last by the users.



(a) Histogram of the average quality rating for all possible selections of ten crossfades from the ten evaluated crossfade sets. The red line shows the position in the histogram of the actual selection made by the automatic DJ.

(b) The same histogram as in Figure 5.10a, but for all possible selections of *eight* crossfades from *eight* of the evaluated crossfade sets. Two crossfades are omitted because they are very quiet and non-percussive in nature, in which case the quality model appears to incorrectly estimate the transition quality.

Figure 5.10: User test results: histogram of the average quality rating for all possible selections of ten crossfades from the ten sets of three crossfades.

inspection of the results, there appeared to be two crossfades with an exceptionally large difference in user rating between the DJ his choice and the users their preferred crossfade for those two sets. These are both crossfades where a song transitions into a very quiet song: in these cases, the quality model apparently inadequately estimates the transition quality. Leaving these two crossfades out gives the histogram in Figure 5.10b. Here, the DJ its selection has an average rating that is higher than 70% of the other potential selections. This gives some indication that for transitions that do not include songs that are too quiet, the automatic DJ generally selects crossfades better than a random selection procedure would.

Unfortunately, only twelve users participated in this test, which is too few to draw conclusions for a generalized population of listeners, especially since the test participants their preferences varied widely. Also the number of crossfades that is evaluated is most likely too small to estimate the DJ its performance in general. Nevertheless, a careful conclusion of this first test is that for most types of crossfades, apart from some specific cases (i.e. quiet, non-percussive transitions), the quality model appears to steer the automatic DJ in the right direction. However, it does not perform an optimal selection yet. Indeed, the model used now only attempts to capture how chaotic and cluttered the output mix is, but more aspects play a role in song selection that this model does not

explicitly consider. For example, songs might need to be in the same musical key, have similar rhythmic patterns, or should not have vocal sections that overlap [1].

5.4 | Conclusion

The automatic DJ system described in this chapter is able to generate a continuous stream of seamlessly mixed Drum and Bass with songs taken from the input library. These songs are preprocessed by the different annotation submodules to detect the beat and downbeat locations and discover the musical structure. The mix is created in a DJing loop. Songs are selected at random, and cue points are chosen using a rule-based framework that considers the high-level structure of the songs. Three types of crossfades are possible, which are selected using a finite state machine. This ensures that the automatic DJ creates a diverse and creative mix. The best crossfade is selected by evaluating the quality of nine crossfade candidates using a machine learning model. A subjective evaluation with 12 participants indicates that this quality model steers the automatic DJ towards selecting higher quality crossfades over lower quality ones for most crossfades. However, this result is obtained after filtering out two of the ten evaluated crossfades, because the algorithm seems to perform very poorly on quiet, non-percussive transitions. Hence, the quality model needs to be improved, and a test with more participants and evaluated crossfades should be conducted to validate its performance. To combine two songs and perform a crossfade, the songs are first time-stretched by means of the WSOLA algorithm. The tempo is held constant at 175 BPM for the entire mix. The songs are overlapped so that they are beat matched, and volume fading and audio equalization is performed to establish a gradual fade-in. Repeating this process for each transition effectively creates a continuous Drum and Bass mix, such that the system fulfills all the requirements defined in Section 1.4.

5.5 | Future work

This chapter has given an overview of how the automatic DJ system combines its different annotation components, a rule-based framework and a music quality model to create a seamless Drum and Bass mix. However, this result could still be improved by making some adaptations or additions, which are discussed below.

The most important drawback of the current system is the absence of an intelligent tracklisting component. A human DJ has a thorough understanding of the music he is playing, and he knows which songs fit together and which do not. However, at this point,

the automatic DJ selects songs at random. Even though the audio quality model attempts to determine the best option out of three candidate songs, it might not consider the appropriate properties to perform this task. It would therefore be interesting to guide the song selection process with an explicit tracklisting algorithm. This is especially important to create a DJ mix with a certain theme or 'flow', i.e. a deliberate compositional progression of songs throughout the mix. This system could take the subgenre of a song into account, the artist, rhythmic patterns occurring in the song, its musical key, the instrumentation and so on.

At this point, the automatic DJ operates at a constant tempo of 175 BPM. While it is often not necessary to change tempo when mixing only Drum and Bass, it should be possible to handle smooth tempo changes of songs, especially if the system would be extended to deal with multiple music genres. The time-stretching module could thus be adapted so that it can perform gradual tempo increases or decreases. Also the song equalization could be augmented. Professional DJ equipment typically has a third 'mid' filter for mid-range frequency components, and adding it would give more options to perform appropriate equalization. However, it was unclear how the mid filter should progress in the fixed filtering settings for the different types of crossfades. Hence, it was not implemented, and now the mid frequency regions are not modified during the crossfade.

The progression of the fader settings during a crossfade is currently fixed for each type of crossfade. Of course, this is not an ideal approach, as the optimal volume and fader settings, fade length and fade speed depend on the songs that are mixed together and the dynamics of the mix. The crossfade could thus be further optimized by determining the optimal equalization settings at each point in the crossfade. As a matter of fact, the initial goal of the developed quality model was to perform such an optimization by tuning the equalizer settings for each downbeat individually, which is also the reason it classifies downbeats instead of longer segments. However, performing a search over the different fader settings proved to be too computationally expensive, even with heavy constraints on the allowed equalization levels.

Lastly, the constructed quality model is rather simple. For example, it works on individual measures, and it has no notion of the structural and sequential nature of the music it is working with. Given the fact that many Drum and Bass mixes are available online, it might be interesting to use these huge amounts of data to construct e.g. a recurrent deep network that models what good Drum and Bass sounds like.

Chapter 6

Conclusion

The automatic DJ system developed in this thesis is able to create a continuous, seamless mix of Drum and Bass music. It uses several music information retrieval techniques to discover the structure of the input music and mixes songs together using a rule-based framework. The song and cue point selection is optimized by means of a machine learning model that evaluates the crossfade quality, resulting in an enjoyable mix.

Understanding the music structure is of absolute importance for a DJ. To do this, the automatic DJ has three annotation modules, i.e. a beat tracker, a downbeat tracker and a structural segmentation module, that discover the musical structure in a hierarchical way. The beat tracker extracts the tempo and phase of the beats by means of a two-step algorithm using an onset detection function. The beat tracker correctly annotated 159 of the 160 songs in a corpus of Drum and Bass songs, i.e. 99.4%. The downbeat tracker uses these beat annotations to determine which beats are downbeats. This is done for each individual beat by means of a machine learning classifier, after which the algorithm aggregates the beat-wise predictions to obtain a prediction for the entire song. Here, an accuracy of 95.3% is obtained on 43 held-out test songs. Finally, the high-level structure of the music is discovered using a novelty-based approach that uses two self-similarity matrices of the audio. The predictions made using these matrices are post-processed using musically inspired rules. For 153 out of 160 songs, i.e. 95%, the annotations are correctly aligned. Combining these results, 90% of the Drum and Bass songs in the user his music library should be annotated correctly by the system. When evaluating the annotation modules on a second test set of 220 songs, the beat tracker, downbeat tracker and segmentation module achieved accuracies of respectively 98.2%, 98.1% and 94.3%, therefore providing 90.9% of the songs with *structurally correct* annotations, i.e. they align with the correct structural boundaries.

The DJ system then combines this knowledge to mix songs together using a rule-based algorithm. These rules are inspired by DJing best practices and mimic the way a professional DJ performs transitions between songs. A machine learning model determines the quality of the performed crossfades and is used to optimize the selection of songs and cue points. A subjective evaluation with 12 participants indicates that this quality model steers the automatic DJ towards selecting higher quality crossfades over lower quality ones, but unfortunately this does not appear to be the case for all types of crossfades: on quiet, non-percussive transitions, it appears to perform not as well. Hence, the quality model needs to be improved, and a test with more participants and evaluated crossfades should be conducted to validate its performance.

The DJ system developed in this thesis can still be improved on several aspects. Firstly, adding a tracklisting component would enable the automatic DJ system to create mixes with a deliberate composition and progression of songs and only mix song combinations that fit together, which could significantly improve the quality of the resulting mix. Secondly, the crossfading process can be improved by letting the automatic DJ optimize the volume and equalizer settings throughout the crossfade, instead of using preprogrammed transition profiles. For this, it might need a more robust model of what a good Drum and Bass mix sounds like compared to a badly executed crossfade, which could be created by learning from the abundance of professional DJ mixes that are available online. Finally, this system is designed for Drum and Bass only: hence, it could be extended to more music genres in order to reach a larger audience.

To conclude, it is important to realize the potential of an automatic DJing application. For example, integrating this with existing audio streaming services would allow a user to create an infinite DJ mix on the fly from a huge collection of music. A robust automatic DJ is also extremely valuable for bars and nightclubs, as such a system could potentially replace the human DJ, reducing operational expenses. It is furthermore an interesting exploration tool for professional DJs: they could use it to quickly explore lots of new music and discover interesting song combinations and transitions with a minimal amount of work. Finally, an automatic DJ is not constrained by the limitations of a human DJ: it can mix together an arbitrary number of songs with fine-grained control over all the equalizer settings, perform live cutting and pasting of music fragments, explore a myriad of song combinations before choosing the optimal one, and even modify the music or generate completely new music on the fly during the DJ mix. It remains an open question whether computers could eventually surpass professional DJs. After all, what characterizes good

DJs is their strong feeling with the crowd, their deep insight in the music they are working with and their ability to consistently surprise in so many creative ways, aspects that might be very difficult or even impossible to fully automate. Nevertheless, automatic DJ systems have a great creative potential, and open up many interesting options for both listeners and DJs.

Bibliography

[1] J. Steventon, *DJing for dummies*. John Wiley & Sons, 2014.

[2] London IMO Records, "A History of Drum and Bass and the Key Drum and Bass Artists," 2011. [Online]. Available: https://web.archive.org/web/20120112042953/http://www.imorecords.co.uk/drum-bass-articles/history-drum-bass-key-drum-and-bass-artists/

[3] Wikipedia, "Amen break — Wikipedia, The Free Encyclopedia," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Amen{_}break

[4] D. Cliff, "Hang the DJ: Automatic sequencing and seamless mixing of dance-music tracks," *Hp Laboratories Technical Report Hpl*, vol. 104, pp. 1–11, 2000.

[5] ——, "hpDJ: An automated DJ with floorshow feedback," *Consuming Music Together*, vol. 35, pp. 241–264, 2006.

[6] T. Jehan, "Creating music by listening," Ph.D. dissertation, Massachusetts Institute of Technology, 2005.

[7] D. Bouckenhove and J. Martens, "Automatisch Mixen Van Muzieknummers Op Basis Van Tempo, Zang, Energie En Akkoordinformatie," 2007.

[8] H.-Y. Lin, Y.-T. Lin, M.-C. Tien, and J.-L. Wu, "Music Paste: Concatenating Music Clips Based on Chroma and Rhythm Features," *ISMIR*, pp. 213–218, 2009.

[9] H. Ishizaki, K. Hoashi, and Y. Takishima, "Full-Automatic DJ Mixing System with Optimal Tempo Adjustment based on Measurement Function of User Discomfort." *ISMIR*, pp. 135–140, 2009.

[10] T. Hirai, "MusicMixer : Computer-Aided DJ System based on an Automatic Song Mixing," in *Proceedings of the 12th International Conference on Advances in Computer Entertainment Technology*, 2015.

[11] M. E. P. Davies, P. Hamel, K. Yoshii, and M. Goto, "AutoMashUpper: An Automatic Multi-Song Mashup System." *Proceedings of the 14th International Society for Music Information Retrieval Conference, ISMIR 2013*, pp. 575—-580, 2013.

[12] ——, "AutoMashUpper: Automatic creation of multi-song music mashups," *IEEE/ACM Transactions on Speech and Language Processing (TASLP)*, vol. 22, no. 12, pp. 1726–1737, 2014.

[13] J. Foote, "Automatic audio segmentation using a measure of audio novelty," *International Conference on Multimedia and Expo*, vol. 1, no. C, pp. 452–455, 2000.

[14] M. E. P. Davies, A. M. Stark, F. Gouyon, and M. Goto, "Improvasher : a real-time mashup system for live musical input," *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 541–544, 2014.

[15] S. Jun, J. Rew, and E. Hwam, "Runner's Jukebox: A Music Player for Running Using Pace Recognition and Music Mixing," *International Conferences on Advances in Multimedia*, no. 7, p. 26, 2015.

[16] C.-l. Lee, Y.-t. Lin, Z.-r. Yao, and F.-y. Lee, "Automatic Mashup Creation by Considering both Vertical and Horizontal Mashabilities," *ISMIR*, pp. 399–405, 2015.

[17] Y.-t. Lin, I.-t. Liu, J.-s. R. Jang, and J.-l. Wu, "Audio Musical Dice Game : A User-Preference-Aware Medley Generating System," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 11, no. 4, 2015.

[18] I.-t. Liu and Y.-t. Lin, "Music Cut and Paste: A Personalized Musical Medley Generating System," *ISMIR*, pp. 463–468, 2013.

[19] Serato, "Serato DJ software," 2017. [Online]. Available: https://serato.com/dj

[20] Native Instruments GmbH, "Traktor Pro 2," 2017. [Online]. Available: https://www.native-instruments.com/en/products/traktor/dj-software/

[21] Atomix Productions, "Virtual DJ," 2017. [Online]. Available: https://www.virtualdj.com/

[22] Mixxx, "Mixxx," 2017. [Online]. Available: https://www.mixxx.org/

[23] T. H. Andersen, "In the {M}ixxx: Novel Digital {DJ} Interfaces," *Proc. of CHI*, pp. 1136–1137, 2005.

[24] M. Müller, *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer, 2015.

[25] J. P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M. B. Sandler, "A Tutorial on Onset Detection in Music Signals," *IEEE Transactions on Speech and Audio Processing*, vol. 13, no. 5, pp. 1035–1047, 2005.

[26] M. Muller, D. P. W. Ellis, A. Klapuri, and G. Richard, "Signal Processing for Music Analysis," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 5, no. 6, pp. 1088–1110, 2011.

[27] P. Masri and A. Bateman, "Improved modelling of attack transients in music analysis-resynthesis." *ICMC*, 1996.

[28] S. Dixon, "Onset detection revisited," *International Conference on Digital Audio Effects (DAFx'06)*, vol. 120, pp. 133–137, 2006.

[29] J. P. Bello, C. Duxbury, M. Davies, and M. Sandler, "On the Use of Phase and Energy for Musical Onset Detection in the Complex Domain," *IEEE Signal Processing Letters*, vol. 11, no. 6, pp. 553–556, 2004.

[30] S. Dixon, "An Interactive Beat Tracking and Visualisation System The Audio-Graphical User Interface," *Proceedings of the International Computer Music Conference ICMC*, pp. 215–218, 2001.

[31] ——, "Evaluation of the Audio Beat Tracking System BeatRoot," *Journal of New Music Research*, vol. 36, no. 1, pp. 39–50, 2007.

[32] C. Cannam and S. Dixon, "BeatRoot," 2013. [Online]. Available: https://code.soundsoftware.ac.uk/projects/beatroot

[33] J. Hockman, M. Davies, and I. Fujinaga, "One in the Jungle: Downbeat Detection in Hardcore, Jungle, and Drum and Bass." *ISMIR*, pp. 169–174, 2012.

[34] M. E. P. Davies and M. D. Plumbley, "Context-dependent beat tracking of musical audio," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 15, no. 3, pp. 1009–1020, 2007.

[35] D. Bogdanov, N. Wack, E. Gómez, S. Gulati, P. Herrera, O. Mayor, G. Roma, J. Salamon, J. R. Zapata, and X. Serra, "Essentia: An Audio Analysis Library for Music Information Retrieval," *ISMIR*, pp. 493–498, 2013.

[36] N. Degara, E. A. Rua, A. Pena, S. Torres-Guijarro, M. E. P. Davies, and M. D. Plumbley, "Reliability-informed beat tracking of musical signals," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 20, no. 1, pp. 278–289, 2012.

[37] J. R. Zapata, M. E. P. Davies, and E. Gómez, "Multi-feature beat tracking," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 22, no. 4, pp. 816–825, 2014.

[38] S. S. Stevens, J. Volkmann, and E. B. Newman, "A Scale for the Measurement of the Psychological Magnitude Pitch," *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937.

[39] M. E. P. Davies and M. D. Plumbley, "A spectral difference approach to downbeat extraction in musical audio," *European Signal Processing Conference*, pp. 5–8, 2006.

[40] M. Pilhofer and H. Day, *Music theory for dummies*. John Wiley & Sons, 2015.

[41] S. Durand, J. P. Bello, B. David, and G. Richard, "Downbeat tracking with multiple features and deep neural networks," *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 409–413, 2015.

[42] ——, "Feature adapted convolutional neural networks for downbeat tracking," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2016-May, 2016.

[43] S. S. Stevens, "The Measurement of Loudness Level," *The Journal of the Acoustical Society of America*, vol. 27, no. 5, pp. 815—-829, 1955.

[44] J. Paulus, M. Müller, and A. Klapuri, "Audio-based music structure analysis," *ISMIR*, pp. 625–636, 2010.

[45] J. Serrà, M. Müller, P. Grosche, and J. L. Arcos, "Unsupervised Detection of Music Boundaries by Time Series Structure Features," *Twenty-Sixth AAAI Conference on Artificial Intelligence*, no. 2009, pp. 1613–1619, 2012.

[46] D. Robinson, "ReplayGain specification." 2014. [Online]. Available: http://wiki.hydrogenaud.io/index.php?title=ReplayGain{_}specification

[47] O. Parviainen, "Time and pitch scaling in audio processing." *Software Developer's Journal*, no. 4, 2006. [Online]. Available: https://www.surina.net/article/time-and-pitch-scaling.html

[48] J. Laroche and M. Dolson, "Improved Phase Vocoder Time-Scale Modification of Audio," *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 3, pp. 323–332, 1999.

[49] W. Verhelst and M. Roelands, "An overlap-add technique based on waveform similarity (WSOLA) for high quality time-scale modification of speech," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, no. 1, pp. 2–5, 1993.

[50] Sound on Sound, "Equalisers Explained," 2001. [Online]. Available: https://web.archive.org/web/20131203021159/http://www.soundonsound.com/sos/jul01/articles/equalisers1.asp

[51] R. Bristow-Johnson, "Audio EQ Cookbook." [Online]. Available: http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt

[52] R. Clement, "Yodel 0.3.0," 2014. [Online]. Available: https://pypi.python.org/pypi/yodel

[53] Pioneer, "Pioneer DJM800 Operating Instructions," 2005. [Online]. Available: https://www.strumentimusicali.net/manuali/PIONEER{_}DJM800{_}IT.pdf

[54] T.-F. Wu, C.-J. Lin, and R. C. Weng, "Probability estimates for multi-class classification by pairwise coupling," *Journal of Machine Learning Research*, vol. 5, no. Aug, pp. 975–1005, 2004.

[55] J. C. Brown, "Calculation of a constant-Q spectral transform," *The Journal of the Acoustical Society of America*, vol. 89, no. 1, pp. 425–434, 1991.

Appendix A

Automatic DJ application: User manual

During this master's thesis, a prototype of the automatic DJ system described in this dissertation has been implemented as a Python command line application. It is available on the GitHub repository `https://github.com/lenvdv/auto-dj`. The application has been developed and tested on Ubuntu 16.04 LTS: compatibility with other operating systems is not guaranteed[1].

A.1 | Downloading and installing the application

To install the automatic DJ application, fork off the project from the Git repository. Please ensure that you are working on Ubuntu 16.04 LTS and that Python 2.7.12 is installed. The following dependencies will also need to be installed, for example using the Python package manager `pip` or using `apt-get` on Ubuntu. We refer to the website of the package providers for correct installation instructions. The automatic DJ application uses the following Python packages:

`colorlog (2.10.0)`, `Essentia`[2], `joblib (0.11)`, `librosa (0.5.0)`, `numpy (1.12.1)`, `PyAudio (0.2.8)`, `scikit-learn (0.18.1)`, `scipy (0.19.0)`, `yodel (0.3.0)`

A.2 | Running the application

To run the application, run the `main.py` script in the `auto-dj/Application` directory:

[1] Since this is a Python application, the application might still work on other operating systems. However, not all dependencies might be available for all operating systems.

[2] Please see `http://essentia.upf.edu/documentation/installing.html` for installation instructions.

```
$ auto-dj/Application> python main.py
```

This launches the application in a command window. A screenshot of the application is shown in Figure A.1. The application is controlled using commands. The following commands are available:

loaddir *directory* Add the `.wav` and `.mp3` audio files in the specified directory to the pool of available songs.

annotate Annotate all the files in the pool of available songs that are not annotated yet. Note that this might take a while, and that in the current prototype this can only be interrupted by forcefully exiting the program (using the key combination `Ctrl+C`).

play Start a DJ mix. This command must be called after using the `loaddir` command on at least one directory with some annotated songs. Also used to continue playing after pausing.

pause Pause the DJ mix.

stop Stop the DJ mix.

skip Skip to the next important boundary in the mix. This skips to either the beginning of the next crossfade, the switch point of the current crossfade or the end of the current crossfade, whichever comes first.

s Shorthand for the skip command

showannotated Shows how many of the loaded songs are annotated.

debug Toggle debug information output. This command must be used before starting playback, or it will have no effect.

To exit the application, use the `Ctrl+C` key combination.

A.3 | Application structure

The application consists out of several modules, implemented in separate `.py` files. An overview of these files and their role is given below:

audio_quality_model.py The code for the SVM audio quality model to estimate the quality of a crossfade.

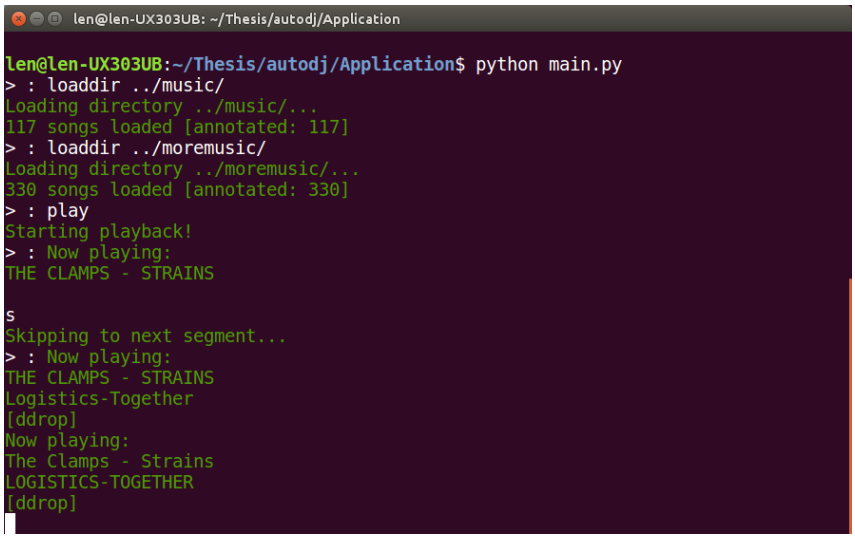


Figure A.1: Screenshot of the automatic DJ application prototype with a typical use case scenario. First, some music directories are loaded using the `loaddir` command. Then, the DJ mix is started using the `play` command. To quickly skip through the mix, the `skip` or `s` command is used. The DJ displays the titles of currently playing song(s) to the user. The title of the current master song is capitalized. Also the type of crossfade is shown.

svm_model.pkl	Pre-trained crossfade quality machine learning model.
feature_scaler.pkl	Pre-trained crossfade quality model feature scaler.
BeatTracker.py	Beat tracking algorithm implementation.
djcontroller.py	Contains the DJing loop and audio playback loop, i.e. the "beating heart" of the application.
main.py	Script that is executed to launch the application. Performs command interpretation and execution.
song.py	Contains the code for the `Song` class, which represents one one audio file and its corresponding annotations.
songcollection.py	Represents the collection of all audio files loaded into the application.
songtransitions.py	Contains code for establishing one crossfade, i.e. applying the volume and equalizer transition profiles.
structuralsegmentation.py	Structural segmentation algorithm implementation.
timestretching.py	Implementation of the WSOLA algorithm.

tracklister.py Contains functions to determine what the next master and slave cue points are using their segmentation annotations and the audio quality model.

util.py Some utility functions used to read and write the annotation files.

DownbeatTracker/downbeatTracker.py Downbeat tracking algorithm implementation.

DownbeatTracker/featureXXX.py Modules that calculate the features for the downbeat tracking task (XXX can be 'Loudness', 'MFCC', 'OnsetIntegral', 'OnsetIntegralCsd', 'OnsetIntegralHfc')

DownbeatTracker/model.pkl Pre-trained downbeat tracking machine learning model.

DownbeatTracker/scaler.pkl Pre-trained downbeat tracking feature scaler.

The Git repository does not only contain the application itself, but also several test scripts, train scripts for the machine learning models, experimental scripts and small demo applications, which were used throughout the development of the application. The application itself can be found in the `Application` directory. For the other scripts it is not guaranteed that they will still work correctly: some scripts were created at the beginning or in the middle of developing the application, and certain changes might have made them deprecated. However, they are still provided in the repository as a future reference. The scripts are provided with comments to explain what they are supposed to do and how they can be invoked.

This page is intentionally left blank