Music Informatics – Assignment 1: Beat Tracking

# 1. Introduction

This report describes the research and implementation of a beat tracking software written in Python 3. Its aim is to detect beats and downbeats in a given audio file. In this particular case the focus lies on ballroom dancing music. The chosen paper for the implementation is the dynamic programming approach by Ellis [1]. However, due to its substandard performance, a major part of the implementation was changed in order to improve its accuracy.

# Assumptions

The focus on (Western) ballroom dancing music imposes some restrictions on the set of probable rhythms and metres. Styles like slow waltz, tango (ballroom), Viennese waltz, foxtrot and quickstep are predominantly performed to music in 3/4 and 4/4 metres, with the exception of samba and paso doble (2/4 metre)[[1]](#footnote-1). This information informed the design choice of focusing on those three metres in the pre-processing of audio files.

The approach described by Ellis imposes a hard limit on the ability of the algorithm to deal with tempo changes over the course of the analysis of one file. This limit was reduced, but not relieved. The presented implementation is able to deal with slowly increasing and decreasing tempi, but rapid tempo changes are not registered.

# Theory

## Onset detection function

The onset detection function lies at the heart of the pre-processing phase and heavily influences the system’s accuracy. In this case, the onset detection function is given by the *onset strength envelope* (OSE), based on the description given by Ellis. It is based on the spectral flux onset detection function and aims to represent the best times for choosing a beat. Additional post-processing is done to improve its accuracy.

## Tempo period bias

The tempo period bias aims to give a global estimate of the tempi in the current dataset. It is based on prior knowledge of the tempi of songs in the dataset (coming from tapping data or similar annotations) and is subsequently used to determine individual songs’ tempi. In the description given by Ellis it is assumed to be the mean tempo (in beats per minute) of a subset of the total dataset, thus producing a value of around 120 BPM. Applying this approach to the given dataset yielded a value of around 128 BPM. However, testing has shown that assuming the tempo period bias to be 120 bpm produces no significant difference in the accuracy of results.

## Tempo estimation

The tempo estimation of individual songs is based on the prior knowledge given by the tempo period bias. Candidates for the probability of the three metres described above are calculated by autocorrelating the onset detection function and taking weighted averages of the autocorrelation and two resampled versions of it. This method reliably identifies a tempo estimate as well as the represented metre.

## Beat Tracking

### Dynamic programming approach

The approach described by Ellis performs beat tracking in two passes. The forward pass consists of an iteration over the OSE and stores the best candidate for a *past* peak for a given point in time in the objective function *C*: A combination of the value of the OSE at point *t* and the maximum value of a window of weighted previous values of C. This recursive definition results in a value that reflects both the current value of the OSE (i.e. is high for points at which the OSE indicates an onset) and previous values of C itself. That mechanism aims to a counterbalance selecting only high values from the onset detection function, which in theory should suppress selections of high onset values just after a beat was found. During the forward pass both the values of C and the indices of the *previous* best beat candidate for any point in time are stored in a separate list *P\**. The backward pass then consists of selecting the highest value of the objective function from a windowed range at the *end* of C, which gives the position of the last beat of the song. Using the array of indices of beat candidates stored in P\*, the previous beats of the song can then be acquired by reading the values of P\*, which always point to the best candidate for the previous beat. The resulting list is reversed and gives the indices of beats.

### State model approach

Because of the rigidness of the dynamic programming approach in regard to tempo changes and its high sensitivity for the selection of false positives in the *BallroomData* dataset a second approach was implemented, building in part on the techniques described by Ellis. This approach also takes the OSE as onset detection function and uses the same tempo estimation process as described above.

However instead of defining current beat probabilities recursively, here the OSE is interpreted as a search space. Its main difference is that the probabilities for beats are calculated using a combination of peak indices in the OSE and the probability of a peak occurring at a given point, considering the tempo estimate. Through the tempo estimation, the system has a loose concept of the metric context of the piece, which results in fewer selections of false positive. Additionally, a mechanism to detect longer breaks in a piece was added. In some of the examples in the dataset non-rhythmic breaks occur, especially at the beginning of the piece. These can be a voice pre-counting the tempo, or general background noise before the music begins. To address this issue (for example, the a voice counting in is still picked up by the OSE) the system looks for these non-rhythmic breaks, which are naturally longer than the assumption of inter-onset interval time, and resets its metric context.

In contrary to the model described by Ellis, the state model approach is also able to identify downbeats. This is achieved by a dynamic beat counting process, which runs in parallel to the beat detection system. Some constraints are imposed upon the counter, such as resetting it after a break was identified (as beats after longer breaks usually correspond to a downbeat).

# Design

The design guidelines for the system closely adhere to those of common Python libraries. Object oriented programming techniques were avoided for this project, as the level of abstraction and complexity in terms of interdependence of the different parts was sufficiently low. The project consists of seven scripts, which are separated by functionality. All functions are documented using docstrings. In the following part a brief overview of each script is given:

The main portion of the code is included in the Main.py file. It includes the beatTracker(inputFile) method, which returns a list of times of beats and downbeats in seconds. Furthermore, it includes the methods for calculating the OSE and the main algorithm for finding beats and downbeats. The analyse(file) method performs data-loading and pre-processing and is further used in the author’s own evaluation system. The Functions.py file includes common functions used throughout the project, such as tempo estimation, ground truth beat extraction and calculation of the tempo period bias. Ellis\_07\_Search.py contains the algorithm described by Ellis. Evaluation\_mir\_eval.py contains functions that utilise the *mir\_eval* framework for evaluating the performance of the system. Evaluation.py contains the authors own evaluation system, which is similar to the *mir\_eval* framework, however including less functionality. Plot.py contains functions for plotting the OSE and evaluation data (found beats vs. ground truth beats). Globals.py defines system-wide global variables that are used by the other functions.

# High Level Implementation

A high level description of the final algorithm for the analysis of a given audio file is given below:

1. Find the tempo period bias and store it in the set of global variables. NB: Because the tempo period bias only changes if files are removed or added to the dataset, its calculation only occurs once and the result is subsequently stored in the tempo\_period\_bias.txt file. If this file exists, the tempo period bias will not be recalculated *even if audio data is added/removed from the dataset*. To recalculate the tempo period bias this file has to be deleted or cleared of its contents.
2. Load the signal data and the audio file’s sample rate using the *librosa* [2] library
3. Calculate the onset detection function (OSE)
   1. Resample audio to 8kHz
   2. Calculate Mel-spectrogram (40 Mel-bands)
   3. Take first-order difference over time axis
   4. Perform half-wave rectification
   5. Sum remaining values over frequency-band axis (results in a 1D signal)
   6. Apply a high pass filter with a cut-off frequency of 0.4Hz
   7. Smooth signal using a 20ms Gaussian window
   8. Normalise by dividing by its standard deviation
4. Estimate tempo from the OSE
5. Perform beat tracking
   1. Initialise lists of beats and downbeats
   2. Find peaks in the OSE
   3. Select first peak and assume it to be first beat
   4. Look for next beat (i.e. peak) in the range defined by the current position in the OSE, the tempo estimation and a given window size.
      1. If a beat is found, update the current tempo estimation and the beat counter (used for identifying downbeats) and continue with d)
      2. If no beat if found, assume a break, extend search window until a beat is found and reset the beat counter

Steps 1, 3 and 4 are implemented and performed as described in the paper by Ellis.

# Evaluation

The original approach by Ellis performed poorly on the given dataset, with an average F-measure of 0.33. This is mostly due to the algorithm’s sensitivity to false positives. More specifically: It performs better than chance level at detecting correct beats. However, due to its static conception of tempo and inability to detect breaks, beats are “detected” even if they are not currently present. The large number of false positives subsequently produces the low F-measure score.

An example of this behaviour is demonstrated in Figure 1. Here the system tries to adhere to its assumed metrical structure. However, The events happening here are a voice counting from one to four before the actual music starts.

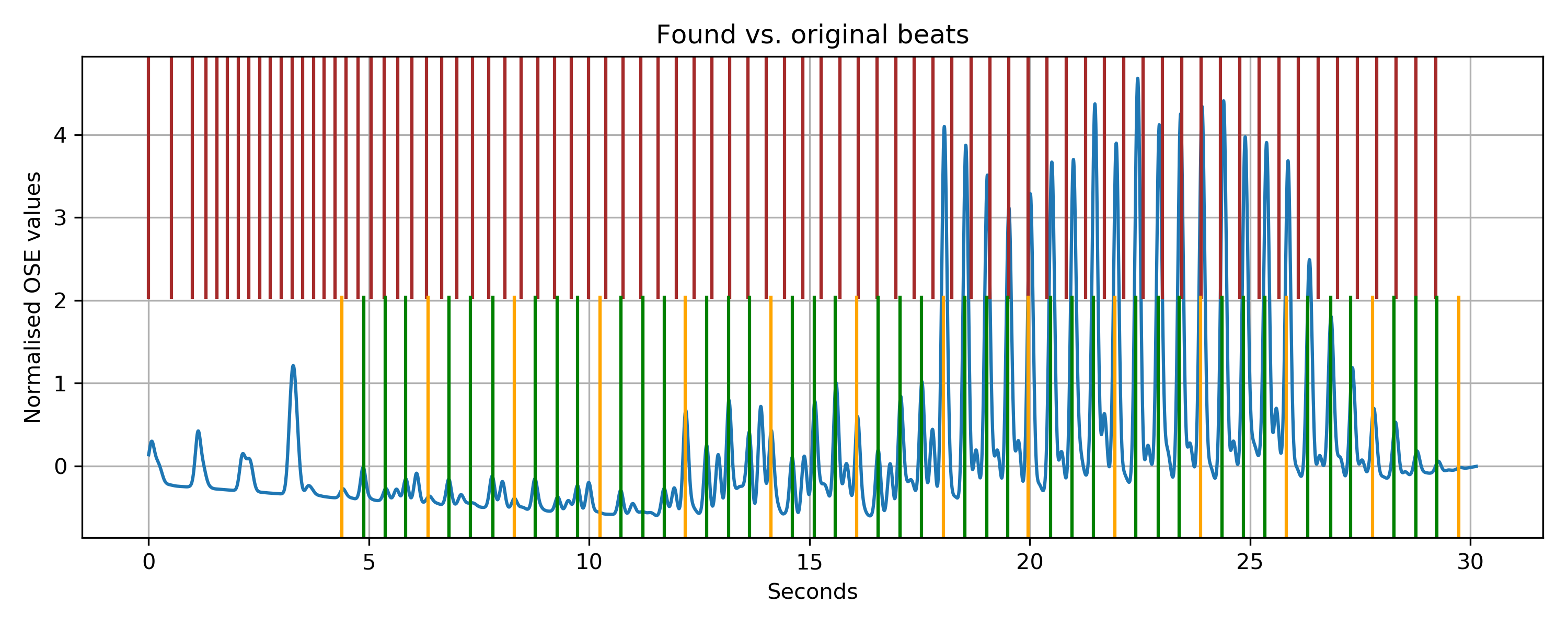


Figure 1: Found (brown) vs. original (green) beats for BallroomData/ChaChaCha/Albums-Cafe\_Paradiso-06.wav (orange lines indicate downbeats)

The use of the state model algorithm and the improvements made to it lead to a significant increase in performance. Average evaluation scores (retrieved using the standard configuration of the *mir\_eval* library) for the whole dataset are given below:

|  |  |
| --- | --- |
| **Measure** | **Score** |
| F-measure (70ms error margin) | 0.69 |
| F-measure Downbeats | 0.16 |
| Cemgil | 0.6 |
| Continuity | 0.37 |

Figure 2 shows the results of the final algorithm for the file “*BallroomData/ChaChaCha/Albums-Cafe\_Paradiso-06.wav*”. Although the model also assumes that the events correspond to music, it identifies the tempo and metrical structure correctly and finds the correct downbeats after three bars of music.

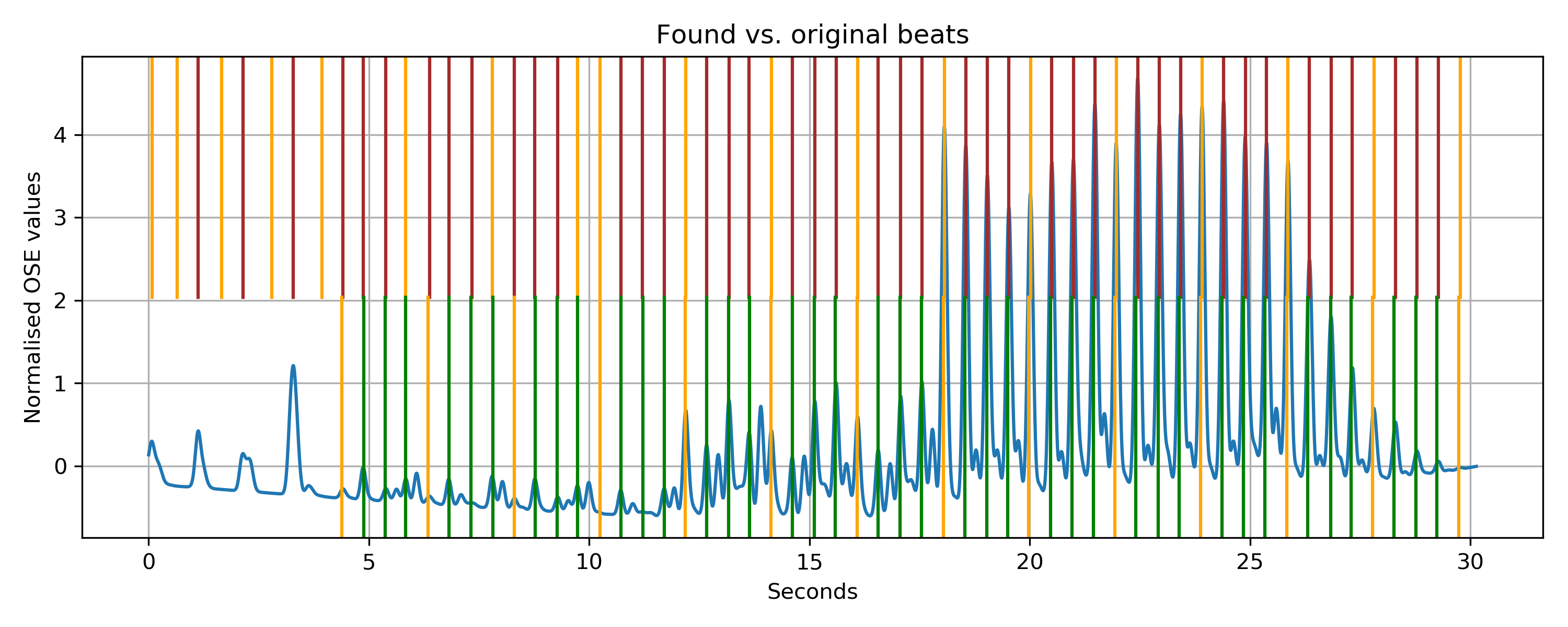


Figure : : Found (brown) vs. original (green) beats for BallroomData/ChaChaCha/Albums-Cafe\_Paradiso-06.wav (orange lines indicate downbeats)

TODO: Evaluation for each style

# Limits

* Only \*.wav files
* Folder limitations

# References

1. Ellis, Daniel. (2007). Beat Tracking by Dynamic Programming. Journal of New Music Research. 36. 51-60. 10.1080/09298210701653344.
2. McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. (2015). librosa: Audio and music signal analysis in python. In Proceedings of the 14th python in science conference, pp. 18-25.

1. <https://www.music4dance.net/dances/ballroom-competition-categories> [↑](#footnote-ref-1)