UNIVERSITY OF CALIFORNIA, SAN DIEGO

An Ensemble Framework for Real-Time Audio Beat Tracking

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Music

by

Michelle L. Daniels

Committee in charge:

Professor Shlomo Dubnov, Chair
Professor Miller Puckette
Professor Bhaskar Rao
Professor Lawrence Saul
Professor Tamara Smyth

2015

The dissertation of Michelle L. Daniels is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2015

DEDICATION

For Nathan and Aunt Ruth

TABLE OF CONTENTS

Signature Page iii

Dedication iv

Table of Contents v

List of Figures ix

List of Tables x

List of Abbreviations xi

List of Symbols xiii

Acknowledgements xv

Vita xvi

Abstract of the Dissertation xvii

Part I Introduction 1

Chapter 1 Introduction 2
1.1 Introduction 2
1.2 Novel Contribution 3
1.3 Dissertation Organization 5

Part II Background and Related Work 7

Chapter 2 Features 8
2.1 Introduction 8
2.2 Onset Detection Functions 8
2.2.1 Spectral Magnitude ODFs 10
2.2.2 Complex Domain ODF 10
2.2.3 Beyond Traditional ODFs 12
2.2.4 Sub-Band Features 13
2.3 Beyond Onsets 15
2.3.1 Low-Level Features 15
2.3.2 Chord Changes 16
2.4 Conclusion 17

Chapter 3 Periodicity Estimation 18
3.1 Introduction 18
3.2 DFT and ACF 19
3.3 Comb Filters 20
3.4 Beat Spectrum 21
3.5 Beat Histogram 22
3.6 Autocorrelation Phase Matrix 23
3.7 Metrical Templates 23
3.8 Matching Periodicity Functions 25
3.9 Conclusion 26

Chapter 4 Beat Detection and Tracking 27
4.1 Introduction 27
4.2 Agent-Based Systems 28
4.2.1 Goto and Muraoka 28
4.2.2 Dixon 30
4.2.3 Oliveira et al. 31
4.2.4 Comparison of Agents 33
4.3 Dynamic Programming 33
4.3.1 Laroche 34
4.3.2 Ellis 34
4.3.3 Eck 34
4.3.4 Klapuri et al. 35
4.3.5 Degara et al. 36
4.3.6 Peeters and Papadopoulos 37
4.4 Other Tracking Methods 38
4.5 Conclusion 40

Chapter 5 Ensemble Learning 41
5.1 Introduction 41
5.2 Ensemble Learning 41
5.2.1 Ensemble Diversity 42
5.2.2 Combining Models 44
5.3 Ensemble Learning and Beat Tracking 45
5.3.1 Non-Causal Beat Tracking Ensembles 46
5.3.2 Real-Time Ensembles 47
5.4 Conclusion 48

Part III The Beaker Beat Tracking Framework 50

Chapter 6 The Ensemble Framework 51
6.1 Introduction 51
6.2 Clustering Hypotheses 53
6.2.1 Joint Clustering 55
6.2.2 Separate Clustering 56
6.3 Scoring and Voting 59
6.3.1 Scoring Trackers 59
6.3.2 Scoring Clusters - Joint Clustering 59
6.3.3 Scoring Clusters - Separate Clustering 60
6.3.4 Voting 61
6.4 Updating Confidences 62
6.4.1 Winners and Losers 62
6.4.2 Tempo Continuity 62
6.4.3 Beat Continuity 63
6.4.4 Combining Confidences 63
6.5 Beat Phase and Detecting Beat Locations 64
6.6 Conclusion 65

Chapter 7 Ensemble Test Trackers 66
7.1 Introduction 66
7.2 Feature Extraction 67
7.2.1 Magnitude-Based ODFs 68
7.2.2 High Frequency Content ODFs 69
7.2.3 Complex Domain ODF 70
7.2.4 Phase-Based ODFs 70
7.3 Tempo Estimation 71
7.3.1 Comb Filters 72
7.3.2 Autocorrelation 73
7.3.3 Discrete Fourier Transform 73
7.4 Beat Phase Estimation 74
7.5 TestTracker Confidence 75
7.5.1 Tempo Confidence 75
7.5.2 Beat Confidence 76
7.6 Conclusion 76

Part IV Evaluation and Results 77

Chapter 8 Evaluation Methodology 78
8.1 Introduction 78
8.2 Datasets 79
8.2.1 Beatles 79

8.2.2 MIREX 2006 80
8.2.3 SMC MIREX 80
8.3 Beat Tracking Evaluation Metrics 81
8.3.1 F-measure 81
8.3.2 Cemgil Accuracy Measure 83
8.3.3 P-score 84
8.3.4 Continuity-Based Methods 86
8.3.5 Information Gain 87
8.4 Evaluating Beaker 88
8.5 Conclusion 89

Chapter 9 Beaker Performance vs. Other Approaches 90
9.1 Introduction 90
9.2 Comparison with Other Systems 90
9.2.1 Beatles Dataset 91
9.2.2 SMC Dataset 92
9.3 Combining Features 93
9.3.1 Combining Features with Beaker 94
9.3.2 Combining Features with Beaker vs. MMA 96
9.3.3 Beaker vs. MMA Varying Other Parameters 100
9.4 Conclusion 103

Chapter 10 Ensemble Diversity 105
10.1 Introduction 105
10.2 Feature Diversity 106
10.3 Predicting Improvement in Combined Ensemble Performance 107
10.4 Combining Periodicity Functions 113
10.5 Conclusion 114

Part V Conclusion 118

Chapter 11 Applications and Conclusion 119
11.1 Introduction 119
11.2 The Beaker UI Application 120
11.2.1 Modes of Operation 121
11.2.2 Visualization 122
11.3 Future Work and Conclusion 128

Bibliography 131

LIST OF FIGURES

Figure 2.1: Example metrical grid (4/4 meter) 8
Figure 2.2: Beatles onset detection functions 11
Figure 2.3: Vivaldi onset detection functions 12
Figure 2.4: Spectral centroid 15

Figure 3.1: Autocorrelation function of an onset detection function 20
Figure 3.2: Expected energy flux templates 24

Figure 4.1: Beat location hypotheses for an agent pair 29

Figure 6.1: Diagram of ensemble framework 51
Figure 6.2: Grid of discrete hypothesis classes 54
Figure 6.3: Hypothesis clusters 55

Figure 7.1: Diagram of TestTracker operation 66

Figure 9.1: Beaker ensemble using only feature F_0 94
Figure 9.2: Beaker ensemble using all features 94
Figure 9.3: Flat MMA 97
Figure 9.4: 2-layer Beaker ensemble 98
Figure 9.5: Beaker + MMA 99

Figure 11.1: Live input mode 121
Figure 11.2: Virtual tapping foot with metronome 123
Figure 11.3: Tempo view with clear metrical structure. Tempo hypotheses are inscribed in a tracker's ellipse and also represented by the vertical position of the ellipse. Lighter-colored ellipses indicate more confident trackers. 124
Figure 11.4: Tempo view with ambiguous metrical structure. Tempo hypotheses are inscribed in a tracker's ellipse and also represented by the vertical position of the ellipse. Lighter-colored ellipses indicate more confident trackers. 126
Figure 11.5: Phase view with one tracker highlighted. Each tracker is represented by a row of ellipses indicating future predicted beat times. Lighter-colored ellipses correspond to more confident trackers. 127

LIST OF TABLES

Table 7.1: Available features 68
Table 7.2: Available periodicity approaches 72

Table 9.1: Beatles dataset algorithm comparison 91
Table 9.2: SMC dataset algorithm comparison 93
Table 9.3: Results for single features vs. all (best single feature for each measure highlighted in bold) 95
Table 9.4: Results for Beaker vs. flat MMA (Beatles dataset) 97
Table 9.5: Results for 2-layer Beaker vs. MMA (Beatles dataset) 99
Table 9.6: Beaker vs. MMA for single-feature ensembles (Beatles dataset) . . 101

Table 10.1: Performance of single-feature ensembles (Beatles dataset) 106
Table 10.2: Performance of two-feature ensembles (Beatles dataset) 108
Table 10.3: Predicting performance of two-feature ensembles (Beatles dataset, items in bold indicate disagreement between predicted and actual performance) 111
Table 10.4: Results for periodicity approach combinations (Beatles dataset) . . 115
Table 10.5: Predicting performance combining multiple periodicity approaches (Beatles dataset) 116

LIST OF ABBREVIATIONS

ACF autocorrelation function

APM autocorrelation phase matrix

BPM beats per minute

dB decibels

DFT discrete Fourier transform

DWT discrete wavelet transform

FFT fast Fourier transform

GACF generalized autocorrelation function

GUI Graphical User Interface

HMM Hidden Markov Model

IAI inter-annotation interval

IBI inter-beat interval

IDFT inverse discrete Fourier transform

IOI inter-onset interval

LDA linear discriminant analysis

MFCC Mel-frequency cepstral coefficient

MIDI Musical Instrument Digital Interface

MIREX Music Information Retrieval Evaluation eXchange

MMA Mean Mutual Agreement

ODF onset detection function

STFT short-time Fourier transform

UI User Interface

LIST OF SYMBOLS

B number of beats in sequence to be evaluated

C_n set of per-song scores for the n^{th} ensemble

E_n n^{th} ensemble

F_n n^{th} feature

$H(x)$ half-wave rectifier function

I binary indicator function

J number of beats in ground-truth annotated sequence

N STFT frame size

P_n n^{th} periodicity estimation approach

S set of songs in a dataset

$X[n, k]$ The STFT of x centered at time n

Δ difference

γ_b time of the b^{th} beat in sequence γ

γ sequence of B beats

AML_c Allowed metrical level continuity required

AML_t Allowed metrical level continuity not required

CML_c Correct metrical level continuity required

CML_t Correct metrical level continuity not required

Cem_acc Cemgil et al. beat tracking accuracy measure

D_g Global information gain

D Information gain

FN false negative

FP false positive

Goto_{acc} Goto beat tracking accuracy measure

Mean8 Mean of eight percentage-based evaluation metrics

TP true positive

ϕ signal phase

τ tempo period

a_j time of the j^{th} beat in sequence a

a sequence of J annotated beats

f_s sample rate

h STFT hop size

i $\sqrt{-1}$

p precision

r recall

t tempo

$w[m]$ time-domain window function

$x[n]$ The n^{th} sample of signal x

x signal x

F-measure F-measure beat tracking evaluation metric

P-score P-score beat tracking evaluation metric

ACKNOWLEDGEMENTS

I would like to thank the members of my committee for their invaluable guidance over the years. My committee chair, Shlomo Dubnov, pointed me in the direction of ensemble learning in the first place, and Miller Puckette somehow always managed to find time to meet with me to talk through ideas and give me advice. My friend and colleague Kevin Larke was also a helpful guide and a wonderful sounding-board for ideas, often helping me to think through challenging problems or see things in a new light. I would also like to thank Peter Otto, Director of Sonic Arts R&D at CalIT2, for all of his efforts that supported my graduate studies through my position as a Graduate Student Researcher. NTT, and Laurin Herr and Natalie Van Osdol of Pacific Interface, were also instrumental in making this possible.

Finally, I would like to thank my parents for their incredible support. My father, Murray Daniels, was always a reassuring force during stressful times, and my mother, Karen Daniels, was always there for me, not just as my mother but also as a computer science professor who was happy to talk to me about clustering algorithms, research, how to do something in LaTeX, or just to guide me through the PhD process as she does with her own students.

Thank you everyone!

VITA

2003	Bachelor of Arts in Music (Music, Science, and Technology) with Distinction, Stanford University
2009	Master of Arts in Music (Computer Music), University of California San Diego
2015	Doctor of Philosophy in Music (Computer Music), University of California San Diego

ABSTRACT OF THE DISSERTATION

An Ensemble Framework for Real-Time Audio Beat Tracking

by

Michelle L. Daniels

Doctor of Philosophy in Music

University of California, San Diego, 2015

Professor Shlomo Dubnov, Chair

Beat tracking is a machine listening task which involves identifying both the time-varying tempo of an acoustic music signal and the location of beats. Ensemble learning methods, where the outputs of multiple classifiers or other models are combined to produce a single more robust output, have become increasingly popular in many machine learning problems but have not been extensively explored for beat tracking applications. Existing ensemble approaches for beat tracking have focused on non-real-time and non-causal beat tracking systems or have been agent-based approaches where agents must interact with the ensemble. However, real-time causal beat tracking is desirable for a number of interesting applications including foot-tapping robots and live musical machine improvisation or accompaniment. This dis-

sertation introduces Beaker, a flexible framework for creating ensembles of arbitrary real-time causal beat trackers which do not require any knowledge of each other or of the ensemble. The Beaker framework uses clustering of ensemble member hypotheses to form discrete hypothesis classes, and reliability measures for each tracker are used to weight hypotheses to determine the ensemble's output. Simple TestTracker beat trackers are introduced as example ensemble members and used to demonstrate the performance of the Beaker ensemble framework. Diverse ensembles of TestTrackers can be created by varying parameters such as the input features and periodicity estimation approaches used by each tracker. Beaker ensembles using TestTrackers are shown to be competitive with a state-of-the-art real-time causal beat tracking system and can even outperform some non-causal beat tracking systems on certain evaluation metrics. Beaker's approach to combining the hypotheses of individual TestTrackers is shown to give superior results compared to recent developments in ensemble methods for non-causal beat tracking. The concept of ensemble diversity is discussed as it relates to Beaker ensembles, and a metric for predicting when the combination of two Beaker ensembles will give improved results over the individual ensembles is introduced. Finally, a real-time implementation and visualization of the Beaker ensemble framework in C++ is presented and areas for future work are discussed.

Part I
Introduction

Chapter 1

Introduction

1.1 Introduction

Beat tracking is a machine listening task which involves identifying both the time-varying tempo (beat period) of an acoustic music signal and the location or alignment of beats (beat phase). As such, it encompasses the task of tempo estimation and is a necessary prerequisite for higher-level tasks such as automatic music transcription. Applications of beat tracking include automatic playlist generation, beat-synchronous effects and processing, beat-synchronous visualization, a foot-tapping or dancing robot, beat-aligned automatic analysis, automatic music transcription, and musical machine improvisation or automatic accompaniment.

When defining the task of beat tracking, it is important to distinguish between *notated* tempo and beat locations and *perceived* tempo and beat locations. For example, if we assume that a quarter note represents a beat, then an eighth note represents half a beat, and consecutive quarter notes played at a tempo of 120 beats per minute (BPM) will sound the same as consecutive eighth notes played at a tempo of 60 BPM. These are two different ways of notating the same rhythmic pattern. Because of the ambiguities of musical notation, most beat tracking systems concentrate not on the task of identifying the composer's notated beats but rather on the task of identifying perceived beats, where the perceived beats are the times a human listener might tap his or her foot in time with music.

The majority of beat tracking algorithms are designed for offline use, meaning that they can be non-causal and are not required to run in real time. Such algorithms can "listen" to an entire song before making decisions about the most likely locations of beats in that song and are useful when the audio to be tracked is known in advance and real-time performance is unnecessary, such as when preprocessing large databases of songs as part of a music recommendation system. This dissertation will focus on the more challenging task of real-time causal beat tracking, which is necessary for applications such as a foot-tapping or dancing robot, musical machine improvisation, and automatic accompaniment. Real-time causal algorithms are capable of processing an unknown incoming audio stream and making decisions in real time about what the current tempo and beat phase are, and whether or not a beat is present. This necessitates causal processing because future input is unknown, and is therefore typically less accurate than non-causal processing. Real-time beat tracking provides additional challenges. For example, consider the case of a foot-tapping robot. When we tap our foot in time with music, we anticipate when the next beat will occur in the future, lifting our foot off the ground and beginning the tapping motion before a beat has happened, so that our foot hits the ground at the same instant that we hear a beat. Reproducing a human listener's predictive ability is one of the challenges in real-time beat tracking.

1.2 Novel Contribution

Beat tracking is an inherently ambiguous process. Ask a group of human listeners to tap their feet in time with a piece of music, and you are likely to find listeners tapping at different tempos (i.e. different metrical levels), and perhaps out of phase with each other (i.e. some on the off-beat). Machine listeners suffer from the same problems. In addition, different beat tracking algorithms might perform better or worse depending on the content of the music being tracked [53] or the evaluation metric used [14]. In most machine listening algorithms, features representing important aspects of the input are extracted and analyzed instead of operating directly on audio samples. In beat tracking, some features work well in music with strong and

sharp onsets, while others are better for softer attacks. Similarly, when estimating tempo, different techniques might emphasize faster or slower tempos. This makes it difficult to design a single beat tracking algorithm which can perform well on all genres of music and various tempo ranges.

Ensemble learning methods from the field of machine learning are intended to address these kinds of situations, where different systems perform well on different inputs and a single model is insufficient to represent all scenarios. In an example of ensemble learning, the output of multiple classifiers is combined to produce results which are better than those of a single classifier. The various classifiers might be trained on different datasets or use different algorithms to determine their results, so that they perform differently depending on the current input. Some algorithms will provide similar or identical results, and agreement among algorithms is an indicator of the correct output.

Along these lines, it has been shown that the task of tempo estimation can benefit from combining the results of multiple tempo estimation algorithms using a voting scheme [37]. Similarly, it should be possible to use an ensemble of beat tracking algorithms to produce a more reliable output than a single algorithm would obtain across a variety of input data.

Past work in ensemble methods for tempo estimation and beat tracking is limited and has focused on combining the output of multiple systems in a non-causal fashion. Results are obtained for each system on each track in a dataset, and then combined after the fact. This is a perfectly valid approach for non-causal systems and non-real-time use. However, for real-time systems, the concept of an ensemble of beat trackers takes on a different meaning. Decisions must be made about how to combine tracker output at each time instant rather than combining beat sequences. For example, rather than discrete beat times, a real-time beat tracking application may require continuously-updating tempo and beat phase values.

This dissertation is focused on the development of a framework for using ensemble methods in real-time causal beat tracking systems. The beat tracking ensemble framework, nicknamed *Beaker*, requires members of the ensemble to output current tempo estimates and next predicted beat locations. The framework was

designed from the beginning for causal real-time use rather than using causal adaptations of non-causal algorithms as some real-time beat tracking systems do [54]. Because it is designed for real-time use, Beaker does not assume that its input will be at a constant tempo. It is designed for adaptability at the expense of some output continuity, and as a result can adapt quickly to tempo changes, including the end of one song and start of another at a completely different tempo. This is in contrast to the many beat tracking systems which either assume constant-tempo input [27], include strong constraints minimizing the likelihood of tempo changes [46], or require a specific state recovery and reset in the system to adapt to large tempo changes [54]. Inside the Beaker framework, the results of ensemble trackers are clustered into discrete classes and combined via methods developed for ensembles of classifiers, using reliability measures at both the ensemble and tracker levels to emphasize the best-performing trackers. The output of the ensemble consists of the current estimated tempo and a current estimated beat phase from which the next beat location can be predicted if desired.

Simple causal beat trackers, called *TestTrackers* were used in the development and testing of the Beaker ensemble framework. A diverse ensemble of TestTrackers can be created by the use of different parameters that include the choice of input feature, approach to periodicity estimation, and allowed tempo range. However, any causal beat tracker could be a member of a Beaker ensemble, provided that it outputs the necessary tempo and beat estimates and reliability measures to the ensemble, and the goal of this work was not to create the best possible ensemble members in the form of TestTrackers but rather to develop a flexible framework for creating ensembles of arbitrary causal beat trackers.

1.3 Dissertation Organization

Related work in beat tracking will be discussed in Chapters 2 through 5. Chapter 2 will discuss the variety of input features that have been used in beat tracking systems, focusing on different kinds of onset detection functions (ODFs). Chapter 3 will discuss a wide range of approaches to tempo estimation, a neces-

sary component of beat tracking systems. Chapter 4 will describe how various beat tracking systems use these input features and tempo estimation approaches to detect beats and track changes in tempo and beat locations over time. Chapter 5 will then introduce ensemble learning techniques as they are traditionally applied in classification systems and provide an overview of prior beat tracking work using ensemble learning.

Following this survey of related work, the Beaker framework for beat tracking using ensemble learning will be introduced in Chapter 6, and the TestTracker beat trackers that were used to develop and test the Beaker framework will be described in Chapter 7. Chapter 8 will then give an overview of the various datasets and metrics used to evaluate the Beaker framework. This will be followed by Chapter 9, which will demonstrate the performance of the Beaker ensemble learning framework in comparison to other beat tracking systems and to another approach to beat tracking using ensemble learning in a non-causal fashion. It will be shown that a Beaker ensemble of TestTrackers performs comparably to a state-of-the-art causal beat tracker and that Beaker's ensemble approach provides better results than a state-of-the-art approach to combining non-causal beat tracker output. Chapter 10 will then discuss the concept of ensemble diversity as it relates to Beaker ensembles, demonstrating that a larger ensemble is not always better. A preliminary method for determining whether the merging of two ensembles will provide improved results over the individual ensembles will be presented.

Finally, Chapter 11 will describe a real-world implementation and application of the Beaker framework and discuss areas for future work.

Part II
Background and Related Work

Chapter 2

Features

2.1 Introduction

Feature extraction for beat tracking is the process of analyzing an acoustic audio signal and converting it to one or more forms where periodicities and beat locations can be more readily observed. Such representations of the signal are referred to as *features*. In this chapter, I will review techniques that have been developed to extract audio signal features that help identify musical events that might correspond to beat locations.

2.2 Onset Detection Functions

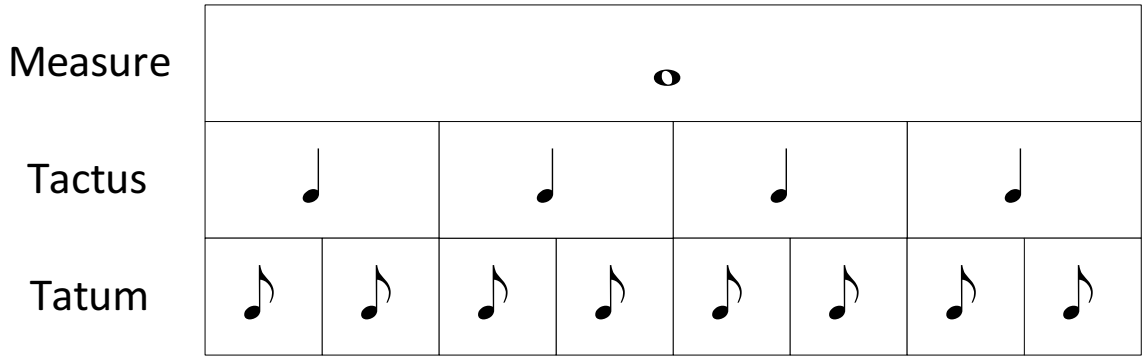

Figure 2.1: Example metrical grid (4/4 meter)

Most existing beat tracking or tempo estimation systems use some sort of note or event onset detection function (ODF) as their feature for input to the periodicity detection stage, where an ODF is a derivation of the original signal with peaks indicating likely onset locations. While not all onsets lie on beats, in most music the majority will lie approximately on a metrical grid such as that shown in Figure 2.1, which is defined by tatum, tactus, and measure durations, where the tatum is the smallest subdivision of a beat, the tactus is the beat, and a measure is a grouping of beats defined by musical meter. For example, Figure 2.1 represents a measure in 4/4 meter, meaning that there are four beats in a measure and the beat is at the quarter note level. In comparison, a measure in 3/8 meter would have three beats per measure with the beat at the eighth note level. Because onsets fall roughly on this metrical grid, one can ideally observe periodic behavior at each of the tatum, tactus, and measure periods by examining the locations of onsets. In early beat tracking systems that used symbolic data (such as Musical Instrument Digital Interface (MIDI) data) as input, onsets were explicitly defined. However, modern systems generally operate directly on acoustic audio signals and therefore must determine likely onset locations automatically using some kind of ODF.

ODFs are typically computed on a frame-by-frame basis, resulting in a signal which is at a lower sampling rate than the original audio signal, where each sample value indicates the likelihood of an onset occurring in a particular analysis frame. Given an ODF, onsets can be estimated explicitly using peak-picking, where peaks in the ODF are identified as likely onset locations. This typically requires some kind of thresholding and can be error-prone and context-sensitive. However, some early audio-based beat trackers used this explicit approach to produce a list of discrete onsets that could be used as input to a beat tracking system that was originally designed for use with symbolic input [21].

ODFs can also be used as implicit indicators of the presence of onsets. Instead of peak-picking and providing a list of discrete note onsets to the periodicity estimation stage of a beat tracker, periodicities can be detected in the ODF directly, by methods such as computing the autocorrelation of the ODF. Because the peak-picking process is imperfect and eliminates possibly meaningful information at an

early stage, most current beat tracking systems use some kind of ODF with this implicit approach, and algorithms for tempo estimation which used ODFs in this way have been shown to outperform those that relied on lists of discrete onsets [37].

The type of ODF that works best for beat tracking depends on the type of content. The remainder of this section will discuss the ODFs most commonly used in beat tracking systems and their performance on different kinds of content.

2.2.1 Spectral Magnitude ODFs

Spectral magnitude-based ODFs can take a variety of forms, but they all begin by computing the difference between magnitude spectra of consecutive analysis frames. Summing these differences across frequency bins indicates the amount of change in the spectrum, with greater change corresponding to a greater likelihood of an onset. Dixon [22] claims that summing the absolute difference between frames gives better results than summing the squared difference, which had been used in some earlier work [24]. Using the half-wave rectified difference between frames, where negative differences are set to zero, emphasizes onsets (where energy increases) compared to sudden decays (where energy decreases) [7]. This type of ODF was used by Dixon [23] and more recently Oliveira et al. [54] in their beat-tracking systems.

Laroche [51] used a variation of this approach to generate his *Energy Flux Function*, introducing a compression function which emphasized higher-frequency bins where onsets were more likely to be noticeable. Alonso et al. [3] also used spectral magnitude to compute an ODF, but instead of computing the difference from frame to frame they used a higher-order differentiating filter, which they found gave significant improvement compared to the first-order difference used in other systems.

2.2.2 Complex Domain ODF

Magnitude-based ODFs typically perform well for percussive sounds, which have sharp onsets that are clearly visible as increases in energy between consecutive magnitude spectra. Unfortunately, magnitude-based ODFs are not as sensitive to

onsets with smoother attacks such as bowed string notes or low-frequency events. For such content, a phase-based ODF that looks for discontinuities in phase evolution between frames, indicative of changing notes, can perform better [7]. However, phase-based ODFs don't work as well on noisy or less tonal signals where the phase evolution is not as predictable to begin with.

To address the issue of content-dependent ODFs, a complex domain ODF was developed, which is sensitive to both magnitude changes and phase discontinuities [6]. Instead of measuring differences in the magnitude spectra of consecutive frames, the complex domain ODF measures the Euclidean distance between complex spectra of consecutive frames, thereby taking into account both magnitude and phase information. Because of its flexibility, the complex domain ODF has been used in a variety of beat tracking systems, including those by Degara et al. [19], Davies and Plumbley [16], and Stark et al. [65]. However, the spectral magnitude-based ODFs are still sometimes preferred because they are computationally more efficient than the complex domain approach [54].

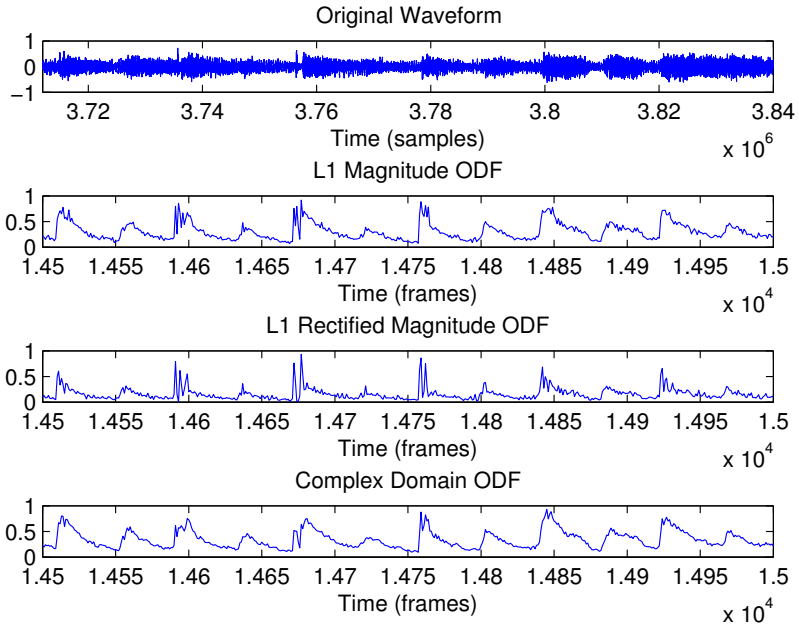

Figure 2.2: Beatles onset detection functions

Figure 2.2 shows, from top to bottom, the original waveform for a segment

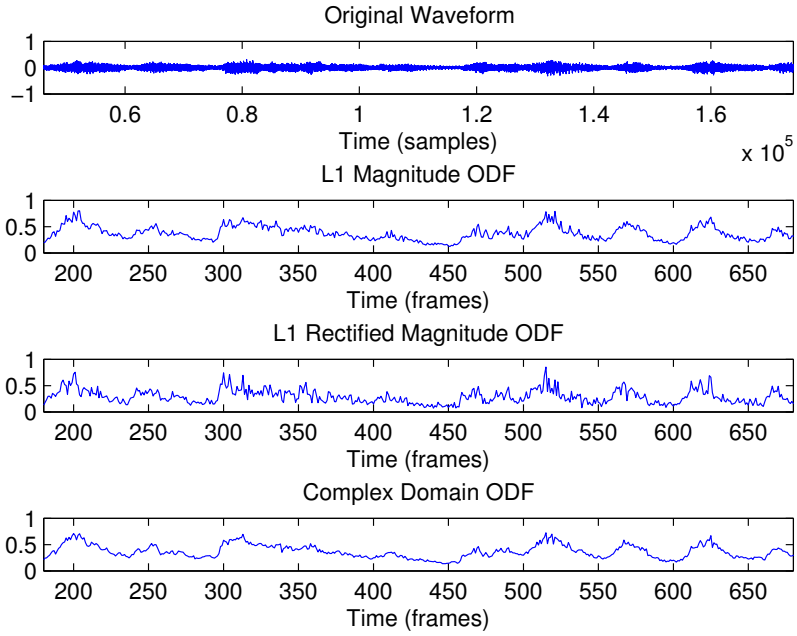

Figure 2.3: Vivaldi onset detection functions

of Beatles music, the ODF based on spectral magnitude absolute (L1) differences, the ODF based on rectified spectral magnitude L1 differences, and the complex domain ODF. Figure 2.3 shows the same ODFs but for an excerpt of Vivaldi string music. None of the three ODFs is significantly better than the others for both examples. However, the L1 rectified magnitude ODF has the sharpest peaks for the Beatles music but the noisiest and least clear peaks in the string music, illustrating the content-dependent nature of ODFs. Interestingly, although string music is supposed to be an area where the complex domain ODF is superior to magnitude-based ODFs, in this particular example the complex domain ODF is very similar to the L1 magnitude ODF and doesn't show a clear advantage.

2.2.3 Beyond Traditional ODFs

Ellis [27] expanded on the traditional magnitude spectrum-based ODF to create an *onset strength envelope*. He computed the Mel spectrum[1] in decibels (dB)

[1]The Mel spectrum uses a non-linear frequency scale meant to represent human perception of pitch.

for each analysis frame and derived the half-wave rectified difference between adjacent frames. The differences were summed across Mel bands as they would be summed over frequency bins in more traditional ODFs, but in this case the differences were in dB, so it was as if linear differences were being multiplied rather than summed. The results were smoothed over time to create an onset strength envelope which, aside from being in dB rather than a linear scale, could be used like any other ODF. Similarly, Eck [25] also used magnitude in dB, with frequency bands spaced on a log scale, which he found outperformed linear magnitude ODFs, including Scheirer's [63]. In contrast, Dixon [22] reports that log (dB) magnitude did not perform as well as linear magnitude for magnitude-based ODFs in his own empirical tests, so there is no clear consensus about the use of linear vs. log magnitude. However, linear vs. Mel or log frequency scale could also be a factor affecting performance in those cases.

2.2.4 Sub-Band Features

In their beat tracking system, Klapuri et al. [46] attempt to identify the degree of *musical accent* in a signal. They first compute the power in each of 36 critical-band-based frequency bands[2]. The difference in power between consecutive analysis frames is computed and then normalized using the current frame's power in order to approximate the change in loudness in each band over time. As with the ODFs in Sections 2.2.1 and 2.2.2, the difference is half-wave rectified to emphasize onsets rather than sudden decays.

Before passing these band-wise accent signals to the periodicity estimation stage of their system, Klapuri et al. combine some number of adjacent bands to produce a smaller number of higher-bandwidth accent signals. They discuss the trade-offs between the number of sub-bands used for feature extraction and the number of sub-bands used for periodicity estimation. Scheirer [63] had used amplitude envelopes rather than Klapuri et al.'s power envelopes, and computed them in six frequency sub-bands, each approximately one octave in bandwidth. Periodicity estimation was then performed by passing each sub-band signal through a bank of

[2]Critical bands are narrow at low frequencies and wider at higher frequencies, and are intended to reflect the way the human ear responds to sound.

comb filter resonators representing different tempo periods, and the output energy of each comb filter was summed across all six bands before the maximum energy filter was chosen as the optimal tempo. However, the limited number of sub-bands in Scheirer's system meant that tonal or harmonic changes were not very noticeable. In contrast, fast Fourier transform (FFT)-based ODFs typically have more than enough frequency resolution for harmonic changes to be noticeable.

Goto and Muraoka [32] used multiple FFT-based ODFs to detect onsets, with each ODF based on spectral changes in different ranges of consecutive FFT bins. Inspired by this approach, in order to make harmonic changes visible in their band-wise accent signals, Klapuri et al. use a larger number of sub-bands in initial computations and later, after computing power differences, combine them to make fewer bands which could then each be used to detect periodicities separately. They set both the initial number of sub-bands and combined number of sub-bands as parameters to their system and tested various combinations of these parameters. They found that their system performed best when many narrow-band power difference signals (which could show tonal changes) were combined to make three or four wider-bandwidth "accent bands" on which periodicity analysis would be performed, rather than when the sub-bands were not combined at all, since onsets were not very visible in the individual narrower bands. In the opposite extreme, three or four sub-bands outperformed the scenario when all sub-bands were combined to produce one single full-bandwidth accent function the way the ODFs in Sections 2.2.1 and 2.2.2 were computed. Klapuri's results suggest that systems using a single ODF computed across the entire spectrum could possibly be improved by performing periodicity analysis on multiple sub-band ODFs.

Another example of sub-band ODF computation is the use of multiple sub-bands by Tzanetakis and Cook during feature extraction to create their beat histograms [67]. Their octave-spaced sub-bands were derived using the discrete wavelet transform (DWT), but were essentially time domain envelopes for each frequency band. Because of the octave spacing like Scheirer's bands, these sub-bands were too wide to indicate the presence of tonal changes. Also, in contrast to Klapuri and Scheirer's sub-band approaches, Tzanetakis and Cook added all of the sub-bands

before periodicity analysis and beat histogram generation.

2.3 Beyond Onsets

2.3.1 Low-Level Features

High-level features such as ODFs based on spectral magnitude or power changes and phase discontinuities are not the only features that can help identify onsets or beats. Ideally, even more features could be considered. For example, Figure 2.4 shows the spectral centroid of the same Beatles segment from Figure 2.2. The spectral centroid can be thought of as the "center of mass" of the magnitude spectrum and can indicate how much a spectrum is dominated by low vs. high frequencies. While it is a well-known low-level feature for machine listening tasks in general, spectral centroid is not a typical onset detection feature. However, at least for this example, it shows clear periodic behavior that could help determine tempo.

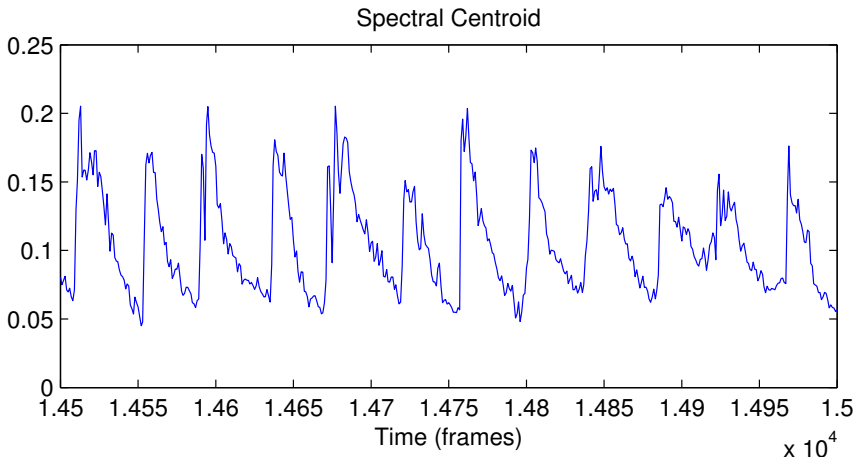

Figure 2.4: Spectral centroid

Gouyon et al. [36] tested 172 different possible features for both beat classification and beat tracking tasks, including spectral centroid, and found that timbral features such as Mel-frequency cepstral coefficient (MFCC) deltas, which are commonly used in speech recognition, were most valuable for classification, while features that identified onsets were the most valuable for beat tracking. However, their results

may not generalize beyond the specific beat tracking system used. Dixon's BeatRoot beat tracker [23] was used to evaluate the suitability of each feature for input to a beat tracker, but it requires discrete onsets as input, and therefore the evaluation required peak-picking on each feature to detect onsets. In other words, each feature was treated as an onset detection function, so it is no surprise that the authors concluded that onset detection functions were the best features for beat tracking.

2.3.2 Chord Changes

Some other possible alternatives to traditional ODFs are features which indicate the presence of harmonic or chord changes. Such changes are more likely to occur on a beat than between beats and more likely to occur at the start of a measure than elsewhere, so features which detect these changes regardless of the presence of strong onsets can be valuable. While the ODFs described in Sections 2.2.1 and 2.2.2 can detect changes in the frequency content of a signal, they do not specifically determine whether those changes reflect a chord change or a melodic figure.

Goto and Muraoka [35] devised a method intended to determine when a chord change had occurred without having to solve the more difficult task of identifying each chord, and they used this as a feature for their beat tracking system. Using provisional beat location estimates based on a more traditional ODF, they looked for peaks in the spectrum indicative of tonal components and, comparing the strength and frequency of peaks between beat time estimates, they were able to estimate the likelihood of a chord change occurring at each time and use this to evaluate the reliability of different beat hypotheses. However, Goto and Muraoka's approach was still quite primitive and does not seem to actually distinguish between melodic changes and harmonic (chord) changes.

Eronen and Klapuri [28] used chroma features to generate a musical accent function which was a stronger indicator of chord changes than the musical accent function used in their earlier work. Chroma features are a representation of the strength of different pitch classes[3] in a musical signal. Therefore, chord changes

[3]Musical notes that occur an octave apart in frequency are considered to be part of the same

should result in large changes in chroma features, but melodic variations within or around the same chord should result in only small changes to chroma features. Eronen and Klapuri compute their chroma features in 36 bands (12 semitones per octave, each divided in thirds), and, treating the 36 chroma "bins" like the 36 sub-band power signals in their earlier work [46], they compute the half-wave rectified difference in each bin, from which a single musical accent function is derived. The performance of the chroma features in a tempo estimation system was compared to the performance of the musical accent function proposed in their earlier work but using the same tempo estimation system, and the differences in performance between the two were not statistically significant. However, the authors did note that the two feature approaches performed well on different subsets of the test data, implying that an approach combining the two feature extraction methods could outperform either one individually.

2.4 Conclusion

This chapter has presented different approaches to feature extraction for beat tracking systems, discussing both low-level features such as spectral centroid and high-level features such as onset detection functions or chroma features. The dependence of ODF performance on signal content was discussed, with the more versatile complex domain ODF mentioned as a possible way to better handle sounds like bowed strings which are not well-represented with spectral magnitude ODFs that are better suited to percussive attacks. The benefits of computing features for periodicity estimation in multiple frequency bands were described, along with the tradeoffs between narrow sub-bands (which are better for detecting tonal changes but some sub-bands may not show any sign of an onset) and wider sub-bands (where onsets are likely visible in all bands to some extent, but tonal changes can be missed). The next chapter will describe ways in which the beat tracking features introduced here can be used to identify patterns and estimate tempo and other periodicities.

---

pitch class. For example, middle C and the C an octave above it are both in the "C" pitch class.

Chapter 3

Periodicity Estimation

3.1 Introduction

Techniques for estimating the rhythm and meter of music, given only the acoustic signal, combine feature detection with higher-level pattern matching algorithms which look for repeated patterns or periodicities in the chosen features. Chapter 2 discussed possible approaches to beat tracking feature extraction in detail. This chapter will review the various ways in which those features can be used to find higher-level patterns indicative of rhythmic structure, typically producing estimates of possible tempo candidates or beat locations.

Early beat tracking and tempo estimation systems operated primarily on symbolic data such as Musical Instrument Digital Interface (MIDI) files, where lists of discrete note event onset times were available, so various techniques were developed for finding patterns in onset times to estimate likely tempos. In more recent tempo and beat tracking systems, symbolic data is no longer used as input. Instead, acoustic audio data is used. As mentioned in Chapter 2, the peak-picking required to produce a list of onsets from an acoustic audio signal is an unreliable and error-prone process, so newer beat tracking systems prefer to detect periodicities by examining frame features such as an onset detection function (ODF) or other input feature directly. Gouyon et al. [37] compared a number of tempo estimation algorithms and concluded that the ones that used frame-level features like those described in chapter

2 were more robust than those that relied on discrete lists of onsets. This chapter will therefore focus on methods for estimating periodicities using such frame-level features.

3.2 DFT and ACF

Given an ODF, accent function, or other frame-level feature, two of the simplest ways of computing periodicities are the discrete Fourier transform (DFT) and autocorrelation. The DFT or autocorrelation function (ACF) of the input feature is computed to obtain a periodicity function, upon which peak-picking is performed to identify the dominant periodicities. In the case of the DFT, peaks in the periodicity function identify likely tempo frequencies, while peaks in the ACF represent likely tempo periods. For example, Figure 3.1 shows the ACF of an ODF with peaks at possible tatum, tactus, and measure periods labeled. In both approaches, octave ambiguity is a problem: for the DFT, peaks will be present at both the actual tempo frequency and its integer multiples, while for the ACF, peaks will be present at both the actual tempo period and its integer multiples. Ellis [27] weighted his ACF before peak-picking in order to emphasize the most common range of tempos. While this helped reduce octave ambiguity for common tempos, it also increased the likelihood of errors for less common tempos. In addition to octave ambiguity, both the DFT and ACF also suffer from the classic time/frequency resolution trade-off: computing over a larger time range provides better tempo resolution but makes the system less responsive to sudden changes.

Besides Ellis, both Alonso et al. [3] and Oliveira et al. [54] used an ACF to provide tempo estimates for their beat tracking systems. Davies and Plumbley [16] and Stark et al. [65] also computed the ACF of their ODF. However, they introduced an additional periodicity-finding stage, where the ACF was passed through a shift-invariant comb filterbank that matched the ACF against possible "comb templates". Each comb template contained pulses at integer multiples of a fundamental period (lag) from the ACF, so this process emphasized tempo periods that were supported by the presence of other metrical levels.

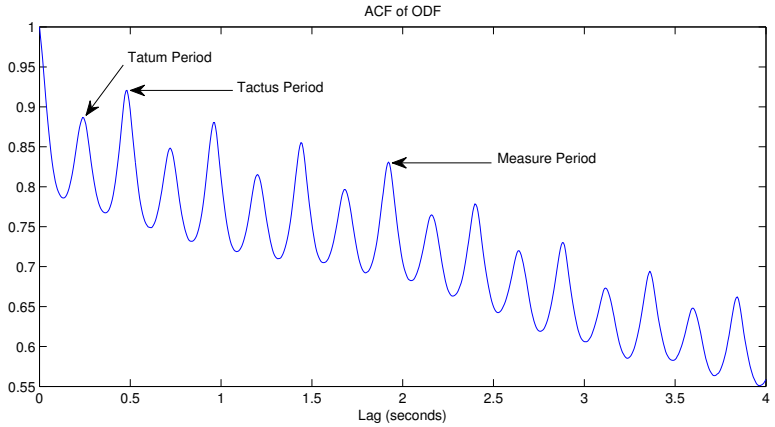

Figure 3.1: Autocorrelation function of an onset detection function

Other methods are combinations of the DFT and ACF approaches, including a method by Peeters [57], which attempts to alleviate the octave ambiguity of both approaches. Peeters computes both the DFT and ACF and maps the values of the ACF into the bins of the DFT, multiplying the two functions. When the values of the ACF are mapped to DFT bins, peaks at multiples of the tempo period become peaks at subdivisions of the tempo frequency. In theory, only the true tempo will have a dominant peak in both the DFT and ACF and will therefore be emphasized when the two are multiplied.

3.3 Comb Filters

Comb filtering has been another popular approach to periodicity estimation in tempo and beat tracking systems, based on work by Scheirer [63]. A filterbank of comb filter resonators is created, where filter coefficients are chosen such that each filter output indicates the degree of resonance at a different tempo period and has the same half-energy decay time. By observing the energy output of each filter, the filter with the strongest resonance can be identified, indicating the most likely tempo. Klapuri et al. [46] expanded on this approach by normalizing the filter output energy to account for skew due to the different tempo periods, providing improved performance compared to Scheirer's system.

Both Scheirer and Klapuri perform the comb filtering on multiple sub-band accent signals and then sum the outputs across sub-bands to create one single periodicity function. Klapuri et al. also suggested the use of the DFT of the periodicity function to help determine the tatum pulse level, since by definition all events in the signal should occur at integer multiples of the tatum period, and the DFT would identify periodicities in the periodicity vector itself. While the DFT and ACF each suffer from octave errors at either multiples or subdivisions of the actual tempo as explained in Section 3.2, Klapuri et al. note that comb filtering suffers from both kinds of octave errors. However, for both Klapuri et al. and Scheirer this is seen as a strength rather than a weakness, because it makes it easier to observe the entire metrical hierarchy.

3.4 Beat Spectrum

Foote and Uchihashi [29] introduced the concept of a *beat spectrum*, which is somewhat unique in that it makes no attempt to identify onsets either explicitly or implicitly. The frame-level feature used is simply the log-magnitude of the spectrum, but instead of computing the difference from frame-to-frame in this feature to emphasize onsets, they instead compute a similarity matrix, where the entry at the ith row and jth column of the matrix is the distance between the feature at frame i and the feature at frame j. They use the cosine of the angle between spectra as their distance measure, since it is independent of frame energy. This is in sharp contrast to ODF-based approaches, where the difference in energy is itself a significant part of the feature. The beat spectrum is derived from the similarity matrix either by summing along the diagonals or computing the ACF of the similarity over either the rows or the columns. A beat spectrogram can be obtained by computing successive beat spectra and can be used to visualize variations in metrical structure over time.

Like peaks in the ACF of an ODF, peaks in the beat spectrum indicate dominant periodicities in the input audio signal. Unlike the ACF of an ODF however, because the beat spectrum incorporates distance measures between spectra of non-adjacent frames, scenarios such as repetition of particular melodic or harmonic events

(even if repeated at a different energy level) will effect the results. The ACF of an ODF will recognize repetitions in rhythmic patterns of onsets, but melodic and harmonic patterns are lost when the ODF is computed.

3.5 Beat Histogram

Tzanetakis and Cook [67] proposed the *beat histogram* for observing metrical hierarchies. They first derive an onset detection function based on the discrete wavelet transform (DWT). The *enhanced autocorrelation function* of the ODF is then computed, by first computing the traditional ACF and then subtracting from the ACF versions of itself time-scaled by different integer factors. The benefit of the enhanced ACF in this context is initially unclear, since the authors claim it removes "repetitive peaks", which would seemingly have the effect of reducing the visibility of different metric levels. However, part of the goal of the beat histogram is to help visualize those different levels. Once the enhanced ACF is computed for each frame over time though, only the first three ACF peaks in the range of valid tempo periods are added to a beat histogram, where each histogram bin corresponds to a different tempo. In this way, the beat histogram accumulates large values in bins corresponding to dominant periodicities in the input audio signal while less dominant periodicities are ignored with the help of the enhanced ACF.

Initially, the beat histogram was used as a feature for genre classification, but in later work, Tzanetakis used the beat histogram for tempo estimation [66]. The two largest peaks in the beat histogram were chosen as the most likely tempo estimates, but there was no attempt to track tempo changes over time. However, the beat histogram accumulated over an entire song has been used to provide information about how variable the song's tempo is, since a tempo which varies greatly will produce less pronounced, more smeared histogram peaks. Compared to averaging periodicity functions over multiple frames, it seems that the value of the beat histogram lies in the fact that only dominant peaks in each frame are added to the histogram, eliminating some of the noise that would be present in a simple average.

3.6 Autocorrelation Phase Matrix

The autocorrelation phase matrix (APM) was introduced by Eck [25] in an attempt to improve upon traditional ACF-based methods. Typically, the ACF is used to provide tempo estimates, but because the ACF does not retain any information about beat phase, beat locations must be estimated separately. The APM aims to allow joint beat period and phase estimation by preserving the intermediate information used to compute a traditional ACF in a matrix, indexed by lag (rows) and phase (columns). The traditional ACF can be obtained by summing across the rows of the APM. Eck compared tempo estimation using a traditional ACF against two methods using the APM. Like Ellis [27], Eck used a "tempo preference window" to give preference to tempos which are more likely to occur in human performances, which can help reduce octave errors.

Both APM-based methods outperformed the traditional ACF in tests using constant-tempo excerpts. In the case of one of the models, APM-Phase, the benefit came from estimating meter by summing metrically-related lags and searching for high-magnitude [phase, lag] pairs. In later work [26], Eck extended his APM approach to estimate changing tempos and beat phases over time by tracking high-magnitude [phase, lag] values in APMs computed on successive frames. Following on Eck's work, Robertson et al. [61] later introduced a comb filter matrix based on the APM, which was supposed to improve performance for syncopated rhythms where lag times corresponding to the beat period would not show high correlation.

3.7 Metrical Templates

Laroche [51] uses what he calls an *energy flux function*, which is a type of ODF, as his input feature. His approach to tempo estimation involves matching the energy flux function to a set of metrical templates. Making the assumption of duple meter[1], he constructs an expected energy flux signal for each candidate tempo. The

[1]In a duple meter, events occur on a metrical grid where relationships between metric levels are factors of two.

expected energy flux signal consists of discrete pulses spaced at the specified tempo period. Additional pulses are added one quarter, one half, and three quarters of the way between beats, but at lower amplitude than the beat pulses, reflecting the expectation that energy flux signals derived from music signals will be stronger on beats than on divisions of beats. For example, expected energy flux signals for three different tempos are illustrated in Figure 3.2, with the template for the fastest tempo at the top and the slowest at the bottom.

Given these expected energy flux signals, Laroche then computes the cross-correlation of each template with the actual energy flux signal derived from the input audio. The tempos whose templates result in the highest cross-correlation values at their tempo periods are chosen as the best candidate tempos to consider in the beat-tracking stage of his algorithm. This is an interesting approach for rock/pop music, which often meets Laroche's simple duple meter assumption. However, to generalize to other musical meters, templates would have to be constructed for each possible tempo and for each possible metrical structure, and this quickly becomes computationally impractical.

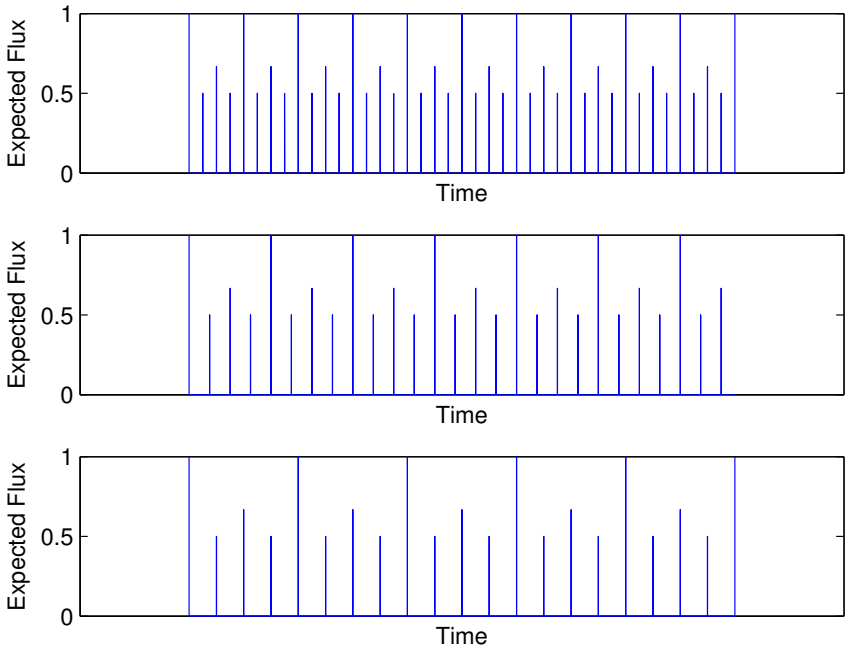

Figure 3.2: Expected energy flux templates

Oliveira et al. [54] used a similar template matching approach to determine the initial beat phase hypotheses used to initialize their beat tracking system. However, their templates considered beat locations only and not the metrical structure as Laroche's did. While that means that their approach is not meter-specific like Laroche's, it also means that peaks in their ODF which occur at multiple of the tatum period but not on beats are not able to contribute to a match.

Peeters and Papadopoulos [58] also used beat template matching in their beat and downbeat[2] tracking system. In one approach, they used meter-dependent templates very similar to Laroche's. However, they also introduced a method based on templates learned from data with annotated beat times using linear discriminant analysis (LDA), rather than hard-coded templates. The LDA-trained templates gave better performance than the default templates, although the authors suggest that the results of training the templates are somewhat genre-specific and therefore even better performance could be obtained by training separate templates for different genres.

3.8 Matching Periodicity Functions

Eronen and Klapuri used the generalized autocorrelation function (GACF) [28] to create their periodicity function, where the GACF is defined using the inverse discrete Fourier transform (IDFT) as

$$\mathrm{GACF}(\tau) = \mathrm{IDFT}(|\mathrm{DFT}(\vec{a}_m)|^p) \tag{3.1}$$

where τ is the tempo period (autocorrelation lag), and $\vec{a}_m$ is the musical accent function for frame m as described in Section 2.3.2. When the parameter $p = 2$, Equation 3.1 reduces to the traditional ACF. However, Eronen and Klapuri experimented with different values for p for each of the different features they tested and found that for the chroma features they used, $p = 0.65$ was optimal in the context of their system. Assuming a constant tempo across all analysis frames, they computed the median of periodicity functions across time to provide one single periodicity vector. Treating

[2]The downbeat is the first beat in a musical measure.

tempo estimation as a true pattern-matching problem, instead of peak-picking on this vector to find the tempo period, they then used k-Nearest Neighbor regression to find matches to periodicity vectors with known (annotated) tempos that were used to train the system.

To compensate for a limited training set and cover a wider range of possible tempos, periodicity vectors were stretched or shrunk by different factors, and matches were also sought for those "resampled" periodicity vectors. The correct tempo was chosen to be the weighted median of the tempos for the closest k training periodicity vectors, with the closest matches given greatest weight. This approach gave better results than traditional methods requiring peak-picking on the periodicity vectors and allowed the metrical structure to be implicitly included as part of the tempo analysis without having to explicitly define rules governing the expected ratios between peaks in the periodicity function. Providing a training set with adequate representation of all possible tempos and metrical structures is a challenge with this approach, but the periodicity vector resampling during the search process helps to improve results when using a limited training set.

3.9 Conclusion

This chapter discussed different ways of identifying patterns in acoustic signals that are indicative of periodicities such as tempo and meter. Unfortunately, comparing the performance of different approaches to rhythmic pattern matching and different choices of input features is difficult at best. Choices of features and periodicity estimation approaches are typically evaluated within the context of a complete beat tracking system, and since different tracking systems behave well with different features or periodicity estimation methods, it is typically not possible to generalize results to all other systems. However, having different periodicity estimation approaches that work best with particular features or inputs suggests that a combination of approaches in an ensemble context may be ideal for beat tracking.

Chapter 4

Beat Detection and Tracking

4.1 Introduction

In this chapter, I will discuss the beat detection and tracking stage of beat tracking systems, which typically occurs once frame-level features such as onset detection functions have been extracted from an acoustic audio signal as described in Chapter 2 and initial analysis of feature periodicities has been performed as discussed in Chapter 3. The output of the beat tracking stage is usually a series of estimated beat locations and tempos. Beat locations can also be described in terms of beat phase, where the beat phase describes the location of the next beat in terms of the current tempo period. Tracking other aspects of the metrical structure of the music beyond the beat, such as tatum period and meter, can sometimes strengthen beat-tracking estimates but is less common. I will describe different approaches for tracking beat locations along with tempo and sometimes meter over time for both real-time and non-real-time applications. Just as humans may change where they tap their feet as a piece of music evolves over time, a beat tracking system must adapt to new information which may cause it to change its hypothesis about tempo and beat location. As a result, the majority of beat tracking systems use either a multiple-agent architecture or dynamic programming-based approaches to consider multiple concurrent hypotheses. I will describe a variety of such systems and discuss other tracking methods that do not fit one of these two categories.

4.2 Agent-Based Systems

Multiple-agent architectures have been used in several different beat tracking systems as a way of maintaining multiple hypotheses in parallel. Each agent maintains its own hypothesis about tempo and next expected beat time, and if one agent loses track of the beat, one of the other agents should have maintained a valid hypothesis and be able to take over. In some systems, a consensus among agents determines predicted beats, and in others a single best agent is chosen to make estimates.

4.2.1 Goto and Muraoka

Motivated by the need to maintain multiple hypotheses in a real-time causal implementation, Goto and Muraoka [32] were the first to use a multi-agent architecture in the context of beat tracking. While many different definitions of the term "agent" have been proposed in computer science [30], Goto and Muraoka define an agent to be a software component that satisfies the following three criteria:

1. the agent interacts with other agents to perform a given task.
2. the agent evaluates its own behavior on the basis of the input.
3. the agent adapts to the input by adjusting its own behavior.

In Goto and Muraoka's beat tracking system, agents are grouped in pairs where they work together to cover multiple possible beat phases. This satisfies the first requirement from their definition of agents, that the agent interacts with others to perform a task. While all agents follow the same features, each agent pair bases its hypotheses on different parameters determining how the features are interpreted. For example, each pair might detect onsets in different frequency subbands, estimate tempo over different time intervals, or permit different ranges of valid tempo hypotheses. After estimating the current tempo, the two agents in a pair each choose beat phase hypotheses that are offset by half the chosen tempo period, helping to guard against the common beat-tracking error of tracking halfway between beats. An example of phase-offset beat location hypotheses for a pair

of agents is shown in Figure 4.1. Unlike some other beat-tracking systems, there is no explicit induction stage initializing Goto and Muraoka's system with tempo or beat estimates. Instead, the agents' varying parameters force them to begin with a range of different hypotheses.

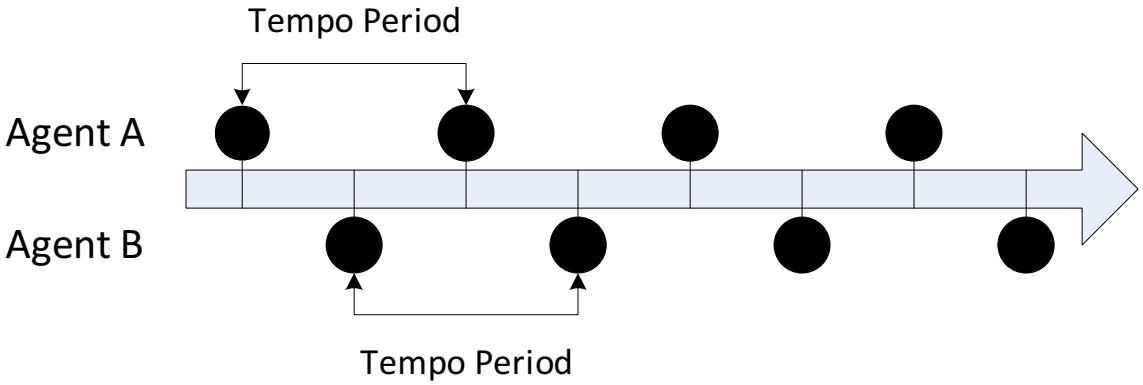

Figure 4.1: Beat location hypotheses for an agent pair

Goto and Muraoka's agents are able to evaluate their own reliability over time and adjust their parameters to focus on values that give high reliability scores, satisfying the second and third requirements from their definition of agents. The reliability of an agent is determined by how well beat times that were predicted in the past correspond to onsets that were later detected as the system progressed in real-time. For music with drum sounds [32], the agent's reliability is also modified by how well estimated drum patterns match pre-registered patterns, and for music without drum sounds [34], reliability is modified based on how well estimated chord change locations coincide with predicted beats. An agent's parameters can be adjusted if both agents in a pair are considered unreliable for a certain minimum time, but only the least reliable of all agent pairs is allowed to adjust its parameters at any given time. The agent-pair's parameters are adjusted within the available parameter space, in the direction of the parameters governing the most reliable agent pair and to a set of parameters that is not already covered by a different agent-pair. A highly-reliable agent's parameters can also be adjusted to reduce the range of valid tempo periods considered by that agent, focusing that agent's attention on a reliable tempo estimate.

The final element of the system consists of a hypothesis manager which makes the final beat location predictions based on the hypothesis of each agent. Agents with similar tempo and beat hypotheses are grouped, and each group is given a reliability score based on the reliability of each agent in the group. The hypothesis of the most reliable agent in the most reliable group is chosen as the output, in an approach reminiscent of weighted majority voting to combine classifier output in ensemble learning methods.

4.2.2 Dixon

Dixon also uses a multiple-agent based approach [21][23], but for a non-causal beat tracking system rather than real-time and causal. He does not provide a definition for his concept of agents, but his agents operate differently than Goto and Muraoka's: each agent is initialized with a different initial tempo and beat phase hypothesis, but they track the same features over time with the same parameters. Unlike Goto and Muraoka's system, Dixon's system includes an induction stage, where initial tempos are estimated by clustering inter-onset intervals (IOIs), and these initial tempos are used to ensure that the agents are initialized with a wide range of hypotheses. Recall that Goto and Muraoka's agents are not initialized with specific tempos, but they do explicitly recompute periodicity estimates over time, while Dixon's agents never repeat the periodicity estimation after the induction period.

At the tracking stage of Dixon's system, features consist simply of a discrete list of detected beat events (onsets) and the salience of each event. The list of beat events is parsed sequentially, and for each event, agents evaluate their own performance by scoring themselves on how well their predicted beats match with the detected event and how salient that event was, so that matching a more salient event gives a higher score than matching a less-salient event. Matching any beat event increases an agent's score, but there is no penalty for missing a beat event or predicting a beat where no event occurs.

Matches are considered within two tolerance windows: an inner window and

an outer window. If an agent matches an event within the inner tolerance window but not exactly, it adjusts its tempo hypothesis based on whether it undershot or overshot the event, based on the assumption that the tempo must be changing (however, this does not seem to consider the possibility that the tempo has not changed but the event was performed slightly before or after the beat for an expressive musical purpose). This is how agent tempo hypotheses get updated even though the agents never explicitly perform any new periodicity estimation. If an agent misses the inner tolerance window but matches within the outer tolerance window, the agent does not change its hypothesis. Instead, it clones itself and one of the pair will proceed under the assumption that the missed onset was a beat while the other agent proceeds under the opposite assumption, that the missed onset was not a beat.

Once agents have been scored across the entire list of detected beat events, the agent with the highest total score is chosen as the winner, and that agent's history of predicted beat locations is taken to be the output of the system. This is similar to the optimal pathfinding of the dynamic programming-based approaches that will be described in Section 4.3, since the agent's score is really a score of the agent's path of beat locations, and agents whose paths converge are merged, keeping the higher-scoring agent's path history. Therefore, just as dynamic programming enables us to find optimal paths, backtracking through the beat path of the agent with the highest final score yields the globally-optimal beat path.

4.2.3 Oliveira et al.

Oliveira et al. [55][54] later implemented a real-time causal version of Dixon's system. Like Dixon's agents, Oliveira et al.'s agents each maintain a tempo and beat phase hypothesis and track the same feature over time. However, their agents are initialized or induced in a way that improves upon Dixon's approach by no longer requiring discrete onset detection from the beginning. First, a spectral magnitude onset detection function (ODF) is derived from the input audio signal and its auto-correlation function (ACF) is computed. Initial tempo hypotheses are chosen from peaks in the ACF, and initial beat phase hypotheses are determined by the quality

of beat template matches to the ODF at each of the candidate tempos. The system is then initialized with an agent representing each candidate [tempo, beat phase] hypothesis, and with an approach similar to Dixon's ranking of IOI clusters, agents are given initial scores based on whether their tempo period estimate has integer relationships to other candidate tempos.

Agent tracking then proceeds similarly to Dixon's system, except that a continuous ODF is used to evaluate each agent's beat predictions instead of Dixon's discrete beat event list. Oliveira et al. keep the system of inner and outer tolerance windows, adjusting an agent's tempo hypothesis if its beat location prediction slightly overshoots or undershoots a local maximum in the ODF. Where Dixon's agents generate a new cloned agent if a beat prediction matches the outer beat tolerance window, Oliveira et al.'s agents produce three new agents to cover a wider range of alternate hypotheses. A monitoring system is also added for state recovery purposes, which looks for sudden decreases in agent scores to identify when a new song has started (i.e. in a streaming radio broadcast) or if the tempo has changed dramatically enough that the system is no longer stable. The induction process can then be repeated using more current data, essentially resetting the system. This is an important addition, since in Dixon's system, unless a tempo was detected during the IOI clustering and induction stage, the only way that tempo could be included as a hypothesis was through gradual shifts away from one of the initial candidate tempos.

Oliveira et al.'s agent scoring is performed by an "agent referee" and is more sophisticated than the self-scoring by Dixon's agents in that it introduces a penalty for bad predictions in addition to rewarding good predictions. A non-causal mode chooses the beat prediction history for the agent with the highest final score, as in Dixon's system. However, in the causal mode, globally-optimal pathfinding is not possible, so the agent with the highest score at each point in time is used to make beat predictions. This is in contrast to Goto's system, where predictions are based on agreement between agents, treating the agents as an ensemble and not relying only on one agent's hypothesis.

4.2.4 Comparison of Agents

This section has described three beat tracking systems based on two very different multiple-agent architectures. Each approach has its strengths and weaknesses. For example, Goto and Muraoka's system allows new tempos to be introduced into the system over time, while in Dixon's approach tempos can only change gradually. However, Dixon's approach has a beat induction stage which results in the agents starting with more accurate tempo estimates in the first place. Goto and Muraoka's system makes certain limiting assumptions about meter (always 4/4) while Dixon's meter assumptions are less restrictive. In general though, their architecture is more flexible than Dixon's. It allows agents to track different features and base hypotheses on different parameters, while including a type of weighted majority voting to determine the best beat predictions. In contrast, Dixon's approach assumes that all agents follow the same features, and agreement between agents' hypotheses results in agents merging, making a majority voting system like that of Goto and Muraoka irrelevant.

4.3 Dynamic Programming

In offline beat tracking systems, where a full song can be analyzed before making any decisions about beat locations, approaches involving dynamic programming, Hidden Markov Models (HMMs), and variations on the Viterbi algorithm [60] to determine optimal "paths" of beats through an excerpt have been popular and have generally performed well. In addition to offline use, dynamic programming approaches can also be modified for use in real-time causal settings, although other techniques such as the agent-based approaches already described may be more appropriate, since the Viterbi algorithm can no longer be used to find a globally optimal path through the entire song. This section will describe several approaches which vary widely in terms of what parameters they choose to optimize in the tracking process as well as how optimality is defined.

4.3.1 Laroche

In his system, Laroche [51] chooses to find the optimal path of [tempo, downbeat location] pairs over time, with initial candidates selected as the best matches of energy flux functions to the metrical templates described in Section 3.7. Paths are scored based on the quality of matches to these pre-defined templates, smoothness of tempo over time, and whether the distances between successive downbeat candidates are consistent with the current tempo.

4.3.2 Ellis

Ellis [27] also uses dynamic programming in his beat tracking system, but unlike Laroche's system where tempo is being tracked with the beat, Ellis assumes a constant tempo throughout a song (estimated once using the autocorrelation of his input ODF feature as described in Section 3.2). Given this tempo, he then maximizes an objective function to find the optimal path of beat locations. The objective function has a high value when there are peaks in the ODF (likely beat locations) and when the distance between successive beat times is consistent with the chosen tempo period.

4.3.3 Eck

Eck's approach to beat tracking, using the autocorrelation phase matrix (APM) described in Section 3.6, uses a variation of the Viterbi algorithm to find an optimal trajectory of high-magnitude [phase, lag] states through a series of APMs computed over the duration of a song or excerpt [26], where *phase* defines the beat phase and *lag* defines the tempo period. Because an APM contains a large number of possible states but changes in tempo and beat phase are typically small, a Gaussian window is used to limit the variation in phase and lag between consecutive APMs, reducing the number of possible transitions that must be considered in the Viterbi decoding although limiting the potential to respond to actual large changes.

4.3.4 Klapuri et al.

Klapuri et al. [46] use a an HMM to determine an optimal path of [tatum, tactus, measure] periods, making certain musically-informed assumptions to simplify the formulation of the HMM parameters. An HMM is defined by hidden variables or states, observations of those states, prior probabilities for each state, the probabilities of transitioning between states, and the probability of an observation given a state. In Klapuri et al.'s HMM, the hidden variables (unknowns) are [tatum period, tactus period, measure period] triplets, and the observations consist of the musical accent function computed as described in Section 2.2.3 and then processed as discussed in Section 3.3 to obtain a periodicity estimation function for each point in time.

Prior probabilities for tatum, tactus, and measure periods were estimated to have a log-normal distribution with parameters estimated from the authors' annotated database. Observation probabilities for a particular [tatum, tactus, measure] period triplet are assumed to be proportional to the product of the periodicity function at tactus and measure periods and the discrete Fourier transform (DFT) of the periodicity function evaluated at the tatum frequency. In other words, we expect the periodicity function to have strong peaks at tactus and measure periods, and we expect the DFT of the periodicity function to have a strong peak at the tatum frequency. Transition probabilities are based on the assumptions that periods are slowly varying, large changes in period are more likely for large periods than for small periods, and period doubling and halving are equally probable.

Assumptions about conditional independence of the hidden variables include that given the tactus period at time $n-1$, the tactus period at time n is conditionally independent of the tatum and measure periods at time n, and given the tactus period at time n, the tatum and measure periods at time n are conditionally independent. Given these assumptions and probabilities, the Viterbi algorithm is used to find the optimal path of [tatum, tactus, measure] periods through each song; in other words, the sequence of [tatum, tactus, measure] periods that is most likely to have occurred given the observations (periodicity functions). To reduce computation time, the candidates considered in the Viterbi algorithm are periodically chosen as

permutations of the 5 most likely tatum, tactus, and measure periods, yielding 125 candidate triplets.

The HMM just described only determines tatum, tactus, and measure periods however, so in Klapuri et al.'s beat tracking system, additional HMMs are needed to determine optimal tactus phase (location) and measure phase given the optimal tempo (the tatum phase is defined by the tactus phase). In both HMMs, the observations are the outputs of the comb filter corresponding to the chosen tactus or measure periods for each sub-band (described in Section 3.3), and the hidden variable is either the tactus phase or measure phase. Prior probabilities are assumed to be uniformly distributed, while transition probabilities are chosen to reward beats which fall within a Gaussian error window of a valid predicted beat location given the chosen period. Tactus phase observation likelihoods are proportional to the weighted sum of tactus period comb filter outputs across frequency sub-bands, similar to the way Scheirer [63] suggested finding beat phase from his comb filters. Observation likelihoods for the measure phase rely on higher-level musical knowledge and are derived based on how well the comb filter output for the chosen measure period matches certain predefined rhythmic patterns that are consistent with a 4/4 meter. Therefore the measure phase estimation does not generalize well to music with other time signatures, although the tactus phase estimation is meter-independent.

Overall, the approach described by Klapuri et al. has been shown to perform well for tempo estimation - it was the winning algorithm in the 2004 tempo induction contest during the ISMIR 2004 conference [37]. It also performed very well at the 2006 Music Information Retrieval Evaluation eXchange (MIREX) evaluation for both the beat tracking and tempo estimation tasks [53].

4.3.5 Degara et al.

Degara et al. [19] also use separate HMMs in their beat tracking system to track tempo and beat phase. An offline version of the HMM-like approach used by Stark et al.'s [65] real-time system to be described in Section 4.4 is first used to derive an optimal tempo (beat period) path. This tempo knowledge is then used by the

second HMM to estimate optimal beat locations (beat phase). In this second HMM, the value of the hidden variable is the number of frames since the last beat, so a value of 0 means that there is a beat event in the current frame. The observations are the values of a complex domain ODF at each frame. To determine beat times, the Viterbi algorithm is used to find the optimal sequence of states, and the beats are said to occur at each time when the state is 0. Observation probabilities for state 0 (beat state) are assumed to be proportional to values of the ODF (o_t), while probabilities for other states (non-beat states) are assumed to be proportional to $1 - o_t$. A uniform distribution for initial probabilities is used, and transition probabilities take the form of a Gaussian distribution centered at each expected beat location (given the already-estimated tempo), where the Gaussian distribution models the probability of deviations from expected beat times.

An important aspect of Degara et al.'s beat tracking system is the use of reliability measures to determine the confidence of tempo and beat estimates. For example, if the structure of the ODF is not clearly periodic, that reduces the reliability of the system.

4.3.6 Peeters and Papadopoulos

Peeters [57] uses an HMM to estimate time-varying tempo without attempting to track the beat. In his HMM, the hidden states are [tempo, meter] pairs, where meter is represented by one of three meter templates. Observations are periodicity estimation functions computed using the combined DFT + ACF approach described in Section 3.2, and observation probabilities are based on values of the periodicity estimation function summed across related tempos defined by the current meter template. As in other approaches, the transition probabilities favor tempo and meter continuity, and prior probabilities are uniform for meter but take the form of a Gaussian probability density function for tempos, favoring a common range of tempos. The optimal path of [tempo, meter] pairs is computed with the Viterbi algorithm as usual.

Later, Peeters and Papadopoulos [58] use a different HMM in a beat and

downbeat tracking system. They assume that the tempo and meter of their content are already known and are given as inputs to the system (tempo is allowed to be time-varying, only a 4/4 meter was tested). In their system, the hidden variables are [beat time, beat-position-in-a-bar] pairs, where beat-position-in-a-bar (bpib) indicates whether the beat falls on the first, second, third, or fourth beat in a bar (measure), and observations are the degree of match to a beat template, chroma variation, and spectral balance (ratio between high-frequency energy and low-frequency energy).

Initial probabilities are uniform for bpib and favor beat times that are close to the beginning of a song. Transition probabilities are based on the assumption that beat times always increase, distances between beats should be consistent with the local tempo within some Gaussian error window, and bpib always transitions in a repeating cycle ($[1, 2, 3, 4, 1, 2, 3, 4, ...]$ for 4/4 meter). Observation probabilities for beat times are based on correlation between an ODF and a beat template generated for the known tempo at each candidate beat time, and bpib observation probabilities are based on the assumption that chords are more likely to change on the downbeat than in other positions in a bar. Also included in bpib observation probabilities is the assumption, as in Goto and Muraoka's work [32], that a particular drum pattern is present, so that it is possible to estimate beat positions 1 and 3 vs. 2 and 4 in a 4/4 measure based on high vs. low frequency content that would indicate the presence of snare drum or kick (bass) drum. In evaluations, this system performed comparably to Klapuri et al.'s system (described in Section 4.3.4) for beat tracking and was significantly better for downbeat tracking.

4.4 Other Tracking Methods

Unlike the majority of systems described here, Robertson et al.'s real-time beat tracking system maintains only one [tempo, beat phase] hypothesis at a time [61]. Their system uses a weighting function to emphasize tempo and beat continuity, where the weighting function is the product of two Gaussians centered at the current [tempo, phase] hypothesis. In order for the system to change its hypothesis, the

strength of the new hypothesis multiplied by the weighting function evaluated at the new hypothesis coordinates must be greater than the strength of the original hypothesis. Depending on the values chosen for the Gaussian spreads, this approach can prohibit the system from making large jumps in [tempo, phase] space. The system is therefore best suited to music with a fairly constant tempo.

Bock et al. [8] use an ensemble of multiple recurrent neural networks for non-causal beat tracking, with the neural network most suited to the current content being tracked is chosen on a per-song basis from the ensemble. A dynamic Bayesian network similar to previously-described dynamic programming approaches is also used to jointly estimate tempo and beat phase.

Stark et al.'s real-time system [65] estimates tempo and beat location separately. Tempo is modeled as slowly-varying and limited to a range of 80 beats per minute (BPM) to 160 BPM. An initial tempo is estimated using Davies and Plumbley's shift-invariant comb filterbank method described in Section 3.2 and tracked over time using a transition matrix consisting of Gaussian columns centered at each possible tempo (row). For each frame, before choosing a new tempo from the comb filterbank output, the output is weighted using the transition matrix and the tempo probabilities from the previous frame to emphasize tempo continuity. The tempo is then used to predict the next beat using a causal version of Ellis's dynamic programming approach mentioned in Section 4.3.2, extrapolating forward from the most likely previous beat location.

The final system that will be described here is the context-dependent two-state system by Davies and Plumbley [16]. In this system, the General State, which considers all possible tempo hypotheses, is first used to estimate a stable tempo. The Context-Dependent State then proceeds to track the tempo and beat but only considers tempos in a small window around its initial tempo, while the General State continues to work in parallel to the Context-Dependent State, examining the full range of tempo hypotheses. If the General State and Context-Dependent State's tempo estimates deviate by too much, with the General State observing a new stable tempo, then a new Context-Dependent State is generated to replace the old obsolete one. Each future beat is predicted based on the current stable tempo estimate.

4.5 Conclusion

This chapter has discussed how beat tracking systems, including multiple-agent based systems and dynamic programming based systems, maintain and/or choose between multiple hypotheses about tempo and beat location over time. Approaches to beat tracking vary widely based on whether the system is causal or non-causal and whether it must operate in real time. Some systems are also limited to specific genres, meters, or limited tempo ranges, so different beat tracking algorithms perform well on different content, making it impossible to label any one approach as the best.

Chapter 5

Ensemble Learning

5.1 Introduction

This chapter will provide some background on the field of ensemble learning as traditionally used in machine learning applications. Then it will discuss prior work that has been done using ensemble learning in conjunction with beat tracking, related machine listening tasks, and real-time applications.

5.2 Ensemble Learning

Ensemble learning methods are approaches to machine learning that involve combining multiple classifiers, regressors, or other models in order to obtain a more accurate result than could be achieved by any single model. The performance of an ensemble depends on both the diversity of the ensemble and the method used to combine the hypotheses of ensemble members, regardless of whether supervised learning or unsupervised learning methods are used. While the Beaker framework is an unsupervised approach and does not involve classifier training, the overview provided in this chapter will include some popular methods for supervised ensemble learning such as bagging and boosting, since they are still relevant to the concept of ensemble diversity. Section 5.2.1 will introduce ensemble diversity and discuss a variety of approaches commonly used to create diversity in ensembles. Then Section

5.2.2 will describe possible methods for combining the output of multiple models to determine the output of an ensemble. Overall, both methods for generating a diverse ensemble and methods for determining the combined output of an ensemble are areas of ongoing research in machine learning.

5.2.1 Ensemble Diversity

The models that make up a successful ensemble are generally complimentary in some way. That is, they perform well on different kinds of input, so that when one model performs poorly another will perform well, allowing it to compensate for variations in the performance of a single model. Such an ensemble is considered to be *diverse*, and diversity is a significant factor in the performance of an ensemble [59]. While the concept of models making errors on different inputs is easy to understand, ensemble diversity is not well defined in the computational sense. For example, numerous measures exist for measuring the diversity of an ensemble of classifiers, and there is no one measure that is considered to be the best in all situations [59][48].

Ensemble diversity is typically achieved in a number of ways. In supervised learning, where models must be trained on a particular dataset, ensemble diversity is often achieved by training different models on different datasets or on different subsets of a dataset. Comprehensive surveys of a variety of such methods can be found in a variety of sources including [20], [59], and [64], and some of the most popular methods will be described briefly here.

One of the earliest methods of this kind was Schapire's *boosting* [62], which involves the consecutive training of three classifiers in order to boost the performance of a weak learner (classifier) to a strong one. In the boosting algorithm, the first classifier is trained on a random subset of the training data. The second is then trained on the most informative subset of the training data, given the first classifier - a dataset where the first classifier correctly classified half of the instances and incorrectly classified the other half. A third classifier is then trained on the subset of the training data where the first and second classifiers disagree. A majority vote between the three classifiers is used to determine the output of the ensemble, so that

the third classifier is only needed when the first two disagree.

Freund and Schapire later introduced *AdaBoost* [31], a generalization of the boosting algorithm, and many variations of AdaBoost have been developed since, due to its popularity [59]. In AdaBoost, classifiers are trained consecutively as in traditional boosting. The first is trained on a random subset of the training data, and subsequent classifiers are trained with increasingly difficult subsets of the training data, representing training instances that were misclassified by previously-trained classifiers. A weighted majority vote is used to determine the ensemble's output, where each classifier's weight is determined by its performance during training such that poorly-performing classifiers are given less weight than those that perform well.

Jacobs et al. introduced the concept of a *mixture-of-experts* [43] using an ensemble of neural networks, each of which is an "expert" on particular kinds of input data or certain subsets of the feature space. If the output of the experts is discrete-valued, a second layer *gating network* chooses which of the experts is best suited to the current input and chooses that network's output as the output of the system. If the output is continuous-valued, the gating network might combine the experts by weighting each output according to how well that expert performed on a particular kind of input during training. Other kinds of classifiers can be used for the experts, but the gating network is typically a neural network.

Breiman's *bagging* algorithm [10] (short for *bootstrap aggregating*) is another method for training classifiers on different subsets of a training dataset. With bagging, unlike boosting, an ensemble of otherwise identical classifiers is trained using random subsets of the training data, rather than basing subsequent classifier training on the performance of other ensemble members. The ensemble's output is determined by majority vote among the classifiers. Bagging can be very useful when only a small training dataset is available, as each classifier can be trained on a relatively large subset of the training data while still creating a diverse ensemble. Unstable models are preferred for this use of bagging, meaning that small variations in the training data subsets can create significantly different classifier performance. One of the more popular examples of this is Breiman's *Random Forests* algorithm [11], which can be viewed as a variation on bagging which uses decision trees for classifiers [59].

Random Forests are perhaps better known in the context of their use of different feature subsets to create a diverse ensemble. This is another way in which diversity can be achieved, and can be applicable to both supervised and unsupervised learning applications. Models in an ensemble can use different input features or subsets of a feature set. Ho introduced the *random subspace method*, training decision trees on random subsets of the available input features [40], while Oza and Tumer introduced an *input decimation* approach [56], which also trains different classifiers on different input features. However, the input decimation approach does not randomly choose feature subsets but rather performs dimensionality reduction by eliminating irrelevant features and instead choosing ones that correlate well with training data.

5.2.2 Combining Models

Once an ensemble has been generated, numerous methods exist for combining the output of ensemble members to produce the output of the ensemble, and overviews of these can be found in sources such as [44] and [59]. The choice of combination methods depends on the output of the classifiers in the ensemble and whether they output discrete class values, rankings of classes, or the probability of each possible class. The best approach to use when combining ensemble member results is also based on many other factors including the nature of the problem being considered, and, in the case of weighted approaches, how accurately the reliability of different ensemble members can be estimated.

In classifiers that output only their selected class, majority voting or weighted majority voting are popular approaches [59]. For example, a classifier might receive greater weight in a vote if it has superior performance on a training dataset, or weights might be adjusted dynamically based on how a classifier performs over time. The class with the greatest number of votes is chosen as the ensemble output. As mentioned earlier, majority voting and weighted majority voting are commonly used with bagging and boosting algorithms.

Some classifiers might output a ranking of the most likely class choices. In

this case, several approaches are possible for combining classifier output [41]. One option is to look at the ranks each class receives by the ensemble of classifiers and to score classes based on the highest ranking each receives from the various classifiers in the ensemble. Using these scores, the classes can then be ranked a second time, with various ways for breaking ties in the ranking. The highest-ranking class is then chosen as the output of the ensemble. Another possible approach is the *Borda count*, which is a generalization of the majority vote. Each class is assigned a score based on how how many classes are ranked below it by each classifier in the ensemble. The class with the highest Borda count is chosen as the output of the ensemble. Finally, a logistic regression approach can be used to address a problem with the Borda count approach, which is that it blindly treats all classifiers equally. With logistic regression, Borda counts are weighted by the relevance or performance of each classifier.

With classifiers that output the likelihood of each class and not simply a single class estimate or ranking, a wider range of output combination approaches is possible [44][47]. One option is to use these likelihoods to rank the classes and use the methods just described for combining ranks, but additional methods are possible. For example, the product of the likelihoods given by each classifier for each class can be multiplied using a product rule, where the class with the highest likelihood product is chosen as the ensemble output. Alternatively, the sum of the likelihoods can be used, and the class with the greatest sum of likelihoods chosen. This gives results equivalent to an unweighted mean of likelihoods, but a weighted mean can also be computed if appropriate weights for each classifier are known, for example from performance on training data. Another option is to choose the class with the greatest minimum likelihood amongst all classifier outputs or the class with the greatest maximum or median likelihood.

5.3 Ensemble Learning and Beat Tracking

The strengths and weaknesses of the onset detection functions (ODFs) and other possible input features to beat tracking systems based on signal content such as

genre or instrumentation are well known [7][22][36]. Therefore it might be desirable to combine multiple input features to create a more robust beat tracking system. One challenge of combining multiple features however lies in the fact that some features are not directly comparable to each other. For example, one could reasonably combine multiple ODFs by averaging them, assuming they were all normalized to the same range and using the same scale (i.e. all linear or all in decibels (dB)). However, even if they were normalized, it would not necessarily be valid to average, for example, an ODF and the spectral centroid of a signal, since they represent completely unrelated quantities.

These sorts of challenges have helped make ensemble learning methods an area of interest in beat tracking tasks and related tasks such as onset detection, particularly in recent years. Gouyon et al. [37] proposed an approach to tempo estimation which involved combining the outputs of multiple tempo estimation algorithms to determine the optimal tempo. Each algorithm received votes depending on how well the other algorithms agreed with their tempo estimate, and the output of the algorithm with the greatest number of votes was chosen as the tempo. Based on this, Dixon proposed combining various onset detection algorithms in a similar fashion as future work, to balance the strengths and weaknesses of the different approaches [22]. Degara et al. later employed a mixture-of-experts approach to note onset detection [18] combining multiple onset detection functions.

5.3.1 Non-Causal Beat Tracking Ensembles

Recently, an ensemble approach to combining beat trackers using different input features was presented by Zapata et al. [68]. A committee of beat trackers is formed where each tracker uses a different input feature. On a per-song basis, each tracker in the committee computes an output beat sequence, and agreement among output sequences is sought using a Mean Mutual Agreement (MMA) approach. This approach measures the similarity or *mutual agreement* of output beat sequences of pairs of trackers as in [42], and the mean of each tracker's agreement with all other trackers is computed. The output of the tracker with the greatest MMA is chosen

as the output of the committee. This is similar to a mixture-of-experts approach, where each beat tracker is an expert, and MMA serves the role of the gating network. Zapata et al. found that this ensemble approach gives better results than the use of a single beat tracker with a single input feature. A benefit of this approach is that because only output beat sequences are required, MMA is tracker-agnostic, meaning that arbitrary beat trackers can participate as members of the committee without knowing about the committee themselves. However, there is a corresponding downside. Because the MMA approach operates only on completed output beat sequences, it cannot be used in a real-time causal system.

In another ensemble approach to beat tracking, Bock et al. [8] use multiple recurrent neural networks, which, unlike most other beat tracking systems, is a supervised learning approach. They create ensemble diversity by training multiple models on different subsets of a dataset, with each subset representing a different musical style. A reference model is then trained on the entire dataset, and for songs where the style matches the model's training data, each model performs slightly better than the reference model. Conversely, when the style of a song does not match the model, each model performs worse than the reference model. Like the system of Zapata et al., Bock et al. treat the multiple models similar to a mixture-of-experts. For each song, they choose the model with the lowest mean squared error compared to the reference model to be used for beat tracking. This simple method for model selection works in this case because the improvement with a matching model over the reference model is typically smaller than the penalty for a mismatched model, making the model closest to the reference more likely to match the style of the current input. In the majority of their results, this multi-model approach performs better than the reference model, and it performs very well in comparison to other state-of-the-art non-causal beat tracking systems.

5.3.2 Real-Time Ensembles

Past work using ensemble learning in real-time applications appears to be focused on the computer vision task of tracking moving objects using binary clas-

sification to distinguish between object and background. Collins et al. performed adaptive feature selection by selecting the most discriminative features to update their model in successive video frames [13]. Grabner and Bischof used an online version of the AdaBoost algorithm for a similar purpose, using a weighted majority vote to obtain the output of their ensemble of classifiers after each successive update [38]. Avidan also uses an online version of AdaBoost and a weighted majority vote but for an arbitrary ensemble of weak classifiers [5].

An approach to real-time beat tracking using arbitrary ensembles has not been introduced prior to this dissertation. Among real-time beat tracking systems, the multiple-agent-based approach of Goto and Muraoka [32] might be viewed as a form of ensemble learning, using an ensemble of agents. However, the dependencies and interactions between agents (which Goto and Muraoka see as strengths of the agent-based approach), mean that their agents are not self-sufficient beat trackers or models, as would typically be the case in traditional ensemble learning applications. As a result, Goto and Muraoka's system does not allow combining arbitrary beat tracking systems as an ensemble - only their agents are supported as ensemble members.

5.4 Conclusion

This chapter has introduced ensemble learning in the context of traditional machine learning models such as classifiers and regressors, focusing on the concepts of ensemble diversity and how to combine the output of multiple models in an ensemble. The use of ensemble learning in past tempo estimation and beat tracking work was reviewed, including a description of the MMA committee-based scheme which will be examined more closely in Chapter 9. Although prior work combining beat tracking and ensemble methods is limited and focused on non-causal processing, the results of Zapata et al. combining features [68] and those of Bock et el. combining musical styles [8] both show benefits from the use of ensemble methods. The remainder of this dissertation will focus on combining the real-time causal benefits of Goto and Muraoka's approach with Zapata et al.'s tracker-agnostic but non-causal MMA

approach, presenting a tracker-agnostic framework that can be used for real-time causal beat tracking.

Part III
The Beaker Beat Tracking Framework

Chapter 6

The Ensemble Framework

6.1 Introduction

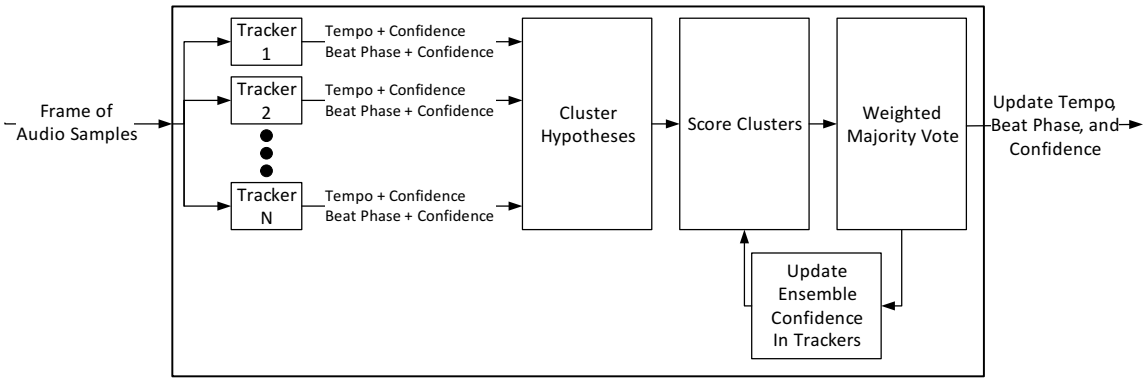

Figure 6.1: Diagram of ensemble framework

This dissertation attempts to expand on the limited previous work combining ensemble learning and beat tracking by creating a framework that allows experimentation using a wide variety of tracker ensembles as well as allowing real-time causal beat tracking. This chapter will describe the Beaker ensemble learning framework in detail, including how it combines the output of an ensemble of trackers to produce a single output. Beat tracking using the Beaker framework begins by generating an ensemble of independent trackers. Unlike agents in a multiple-agent architecture like that described in Section 4.2, beat trackers in a Beaker ensemble are ensemble-

agnostic and context-free. In other words, they do not need to know anything about the other members of the ensemble or the state of the ensemble itself. As a result, a wide variety of beat tracking algorithms could be used as members of a Beaker ensemble, including other Beaker ensembles. Evaluation of this framework has focused on the TestTrackers to be described in Chapter 7, but the framework is general enough to allow any tracker meeting the following criteria to participate in an ensemble:

1. Tracker must be able to estimate the current tempo and provide a confidence level for that hypothesis

2. Tracker must be able to estimate the next beat location and provide a confidence level for that hypothesis

3. Tracker must be able to output these hypotheses on a frame-by-frame basis in a causal fashion

Algorithm 1 Ensemble tracking algorithm

ENSEMBLE-TRACK

```
1:  while audio samples available do
2:      for each tracker in ensemble do
3:          update each tracker with new frame of samples
4:          get new tracker tempo and beat hypotheses and confidences
5:      end for
6:      cluster hypotheses
7:      score clusters
8:      choose winning hypothesis
9:      output current tempo and beat phase estimates
10:     update ensemble's confidence in each tracker
11: end while
```

The ensemble performs beat tracking causally on a frame-by-frame basis. As shown in Figure 6.1 and described in Algorithm 1, a frame of audio samples is provided to each tracker, which is then responsible for reporting to the ensemble its

current tempo hypothesis, confidence in its tempo hypotheses, next predicted beat location, and confidence in its beat prediction. The provided hypotheses are then clustered to create a set of discrete tempo and beat location classes so that the trackers behave as classifiers classifying the input audio data into one of the discrete classes. A weighted majority vote among members of the ensemble (trackers/classifiers) is then performed, with weights given to each tracker based on the tracker's current confidence in its hypotheses, the ensemble's current confidence in the tracker's hypotheses, and a prior probability which can be assigned to each tracker when the ensemble is initialized in the event that a particular ensemble member is known in advance to perform better than others. The hypothesis class winning the weighted majority vote becomes the current output of the ensemble. The process shown in Figure 6.1 is repeated for each successive frame of audio data, and after each vote, the ensemble's confidence in each tracker's hypothesis is updated to affect the weighting of trackers in the next iteration.

Section 6.2 will describe the process of clustering tracker hypotheses into discrete classes. Section 6.3 will then describe the voting process used to determine the winning hypothesis. Section 6.4 will describe how the ensemble determines its confidence in each tracker's hypotheses, and Section 6.5 will discuss how ensemble member tracker beat location hypotheses are converted to an ensemble beat phase output and how this beat phase is used to determine beat locations.

6.2 Clustering Hypotheses

The Beaker ensemble framework is responsible for combining the hypotheses of all ensemble member trackers to obtain an updated global hypothesis at each point in time. To accomplish this, Beaker treats the trackers as classifiers, where the input feature to each classifier is a frame of audio samples, and possible output classes are defined by discrete [tempo, predicted beat location] pairs. There are many ways in which these classes could be defined. In one approach, the range of valid tempo hypotheses could be divided into predefined discrete tempo classes and the range of possible beat locations could be divided into predefined discrete beat location classes.

The set of possible tracker output classes would then be the product of the number of tempo classes and the number of beat location classes. Each square in the grid shown in Figure 6.2 could represent such a class, but this predefined grid-based approach is problematic. To obtain the necessary tempo and beat location output resolution, a large number of possible classes would be required. For example, we might wish to have a resolution of 1 beat per minute (BPM) for tempo classes. However, at the same time, we might want multiple adjacent classes to be treated as one single class, such as combining tempo classes at 119 BPM, 120 BPM, and 121 BPM into one class at 120 BPM.

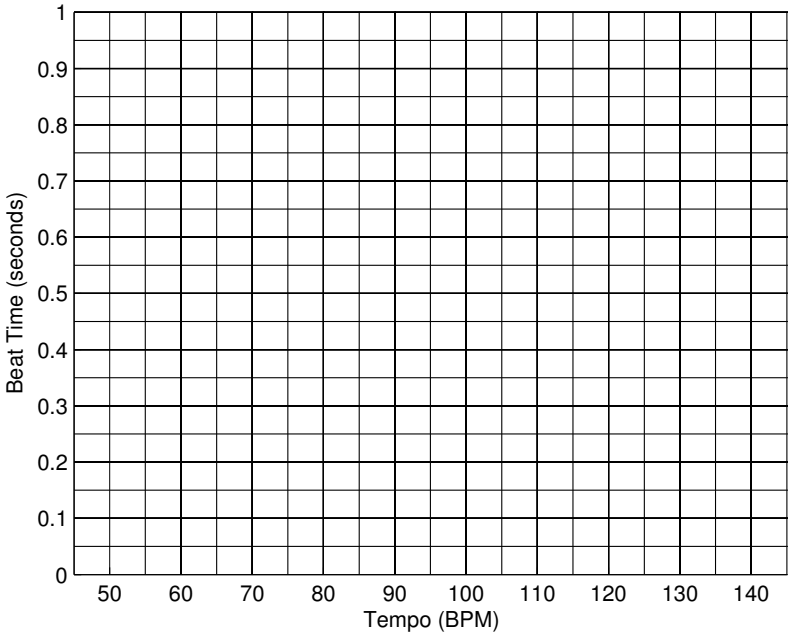

Figure 6.2: Grid of discrete hypothesis classes

Beaker's approach instead uses a clustering algorithm to group tempo and beat location hypotheses into discrete clusters or classes, as illustrated in Figure 6.3. This dynamic approach implicitly provides high resolution output hypotheses while also allowing a wider range of hypotheses to belong to the same class. Beaker has several options for clustering hypotheses. Section 6.2.1 describes how tracker hypotheses can be jointly clustered based on both tempo and beat location. Section 6.2.2 describes possible methods for clustering trackers first by tempo, then

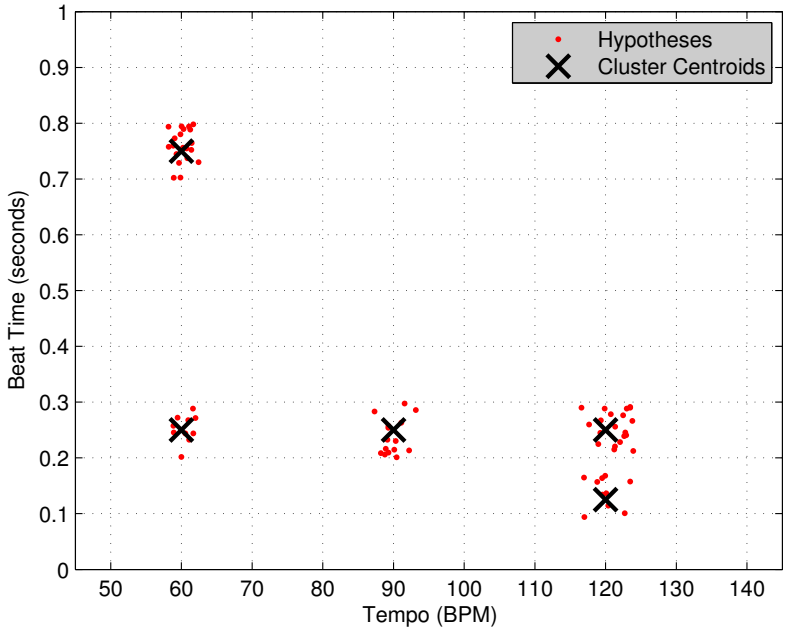

Figure 6.3: Hypothesis clusters

separately by beat location.

6.2.1 Joint Clustering

In the joint clustering approach, pairs of the [tempo, predicted beat location] estimates from each tracker are clustered to determine groupings of similar hypotheses. In this process, Beaker makes no assumptions about the number of clusters that will be needed. It uses a clustering algorithm based on that used by Dixon for tempo clustering [21], with the addition of beat location as a second dimension. The algorithm is quite simple and is outlined in Algorithm 2. For each tracker, the algorithm first checks to see if a cluster exists whose tempo and beat hypotheses are within certain threshold distances of the cluster's centroid. Because tempo and beat location are on different scales, different thresholds are used for each dimension. The tempo threshold is defined as a certain percentage of the cluster's centroid tempo, and the beat threshold is defined as a percentage of the cluster's centroid tempo period. If both hypotheses are within their threshold distances, the tracker

is added to that cluster. Otherwise a new cluster is created for that tracker. If the tracker's hypotheses are within the threshold distances for multiple clusters, the tracker is assigned to the closest cluster. Because tempo and beat location are on different scales, a simple 2-dimensional Euclidean distance measure is not suitable for comparing tempo distance vs. beat distance. To place the hypotheses on similar scales, a scale factor can be specified so that the distance D between two hypotheses is defined as

$$D = w_t \cdot (t_1 - t_2)^2 + w_b \cdot (b_1 - b_2)^2 \tag{6.1}$$

Where $[t_1, b_1]$ is the first [tempo,beat] hypothesis pair, $[t_2, b_2]$ is the second pair, and w_t and w_b are weights controlling the relative importance of tempo vs. beat location differences.

Algorithm 2 Joint tracker clustering algorithm

JOINT-CLUSTER

```
1: while at least one tracker changed clusters and less than max iterations do
2:    for each tracker in ensemble do
3:       find cluster C with closest centroid tempo and beat location
4:       compute tempo and beat radii from centroid tempo and beat of cluster C
5:       if tempo and beat are both within radii then
6:          assign tracker to cluster C
7:          recompute centroid of C
8:       else
9:          assign tracker to new cluster
10:      end if
11:   end for
12: end while
```

6.2.2 Separate Clustering

Each cluster must be assigned a score in order for the ensemble to vote on the "best" output class. In the joint clustering approach described in Section 6.2.1,

clusters must be scored based on both tempo and beat hypotheses simultaneously. However, better performance can be achieved by scoring the hypotheses separately, requiring separate clustering of tempo and beat hypothesis. This approach clusters trackers based on tempo hypothesis, chooses the "best" tempo, and then separately clusters the beat location hypotheses of the trackers with the winning tempo hypothesis. Because this simplifies the problem into two one-dimensional clustering problems rather than a two-dimensional problem, it is easier to explore a variety of clustering approaches this way. In addition, it is not necessary to perform beat clustering for trackers that are not in the winning tempo cluster, because only beat locations for trackers in the winning tempo cluster are considered. Beaker uses a one-dimensional version of the joint clustering approach when clustering beat locations, and that clustering algorithm is also one of the options for tempo clustering. Additionally, Beaker provides the option of using a K-means clustering algorithm for tempo clustering, outlined in Algorithm 3.

Algorithm 3 K-means tracker clustering algorithm

TEMPO-CLUSTER-KMEANS

```
1: pre-define K, the number of clusters
2: initialize cluster centroids
3: while at least one tracker changed clusters and less than max iterations do
4:     for each tracker in ensemble do
5:         assign tracker to cluster with closest tempo centroid
6:     end for
7:     re-compute cluster centroids
8:     merge duplicate clusters
9: end while
```

K-means is a widely-used efficient clustering algorithm. Unlike the clustering algorithm described in Section 6.2.1, K-means requires the user to specify in advance a value for K, the number of clusters. When using K-means in a Beaker ensemble, the value for K is specified as a constant input parameter rather than being dynamically chosen. An optimal value of K can be determined experimentally given a particular

range of allowed tempos. For example, a large range of possible tempos might require a larger K than a more limited tempo range.

Additionally, a decision must be made about how to initialize the cluster centroids. Common approaches include randomly choosing K instances as centroids or uniform initialization over the range of possible values [4]. In a Beaker ensemble, the K-means initialization approach is specified as an input parameter, and several variations on uniform initialization are used since we expect to see tempos over a wide range of possible values. One available option is uniform initialization over the range of possible tempos. Another specifies an exponential scale, where the distance between clusters is smaller for smaller tempos, and becomes larger at higher tempos. The third option is a variation on uniform initialization, where clusters are initialized based on the tempo centroids chosen in the previous analysis frame.

Another consideration when using K-means is the presence of empty clusters. One option when encountering an empty cluster is simply to ignore it, effectively working with fewer than K clusters as a result. Another commonly-used option is to re-initialize the empty cluster with the tempo furthest from any cluster centroid. Both of these approaches are provided as options in a Beaker ensemble and the desired approach can be specified as an input parameter.

Finally, there is also the possibility that the algorithm will produce two clusters with very similar centroids. This scenario is most likely to occur when a tempo cluster is present half-way between two centroids specified during initialization. To prevent a true cluster from being split into two separate clusters, a Beaker ensemble looks for tempo clusters with centroids separated by less than a specified distance. If such a situation is found, the two clusters are merged into one, leaving the other cluster empty.

6.3 Scoring and Voting

6.3.1 Scoring Trackers

After clustering tracker hypotheses, it is necessary to assign scores to each cluster. To do this, Beaker first assigns each tracker in the ensemble a tempo score and a beat score. The tempo score for a tracker, given in Equation 6.2, is the product of three factors: f_t (the tracker's confidence in its own tempo hypothesis), f_e (the ensemble's confidence in the tracker's tempo hypothesis), and a prior confidence weighting p, which is a constant value assigned to the tracker when the ensemble is initialized.

$$\text{tempoScore} = f_t \cdot f_e \cdot p \tag{6.2}$$

Similarly, the beat score for a tracker, given in Equation 6.3, is also the product of three factors: g_t (the tracker's confidence in its own beat hypothesis), g_e (the ensemble's confidence in the tracker's beat hypothesis), and the same prior confidence weighting p used to calculate the tracker's tempo score.

$$\text{beatScore} = g_t \cdot g_e \cdot p \tag{6.3}$$

6.3.2 Scoring Clusters - Joint Clustering

In the joint clustering approach, each cluster is assigned a score based on the sum of the scores of each tracker in that cluster. Let T_i be the set of trackers in the i^{th} cluster. The score for the i^{th} cluster is then given by summing the scores of all trackers in the cluster as in Equation 6.4:

$$\text{clusterScore}_i = \sum_{a \in T_i} (w_t \cdot \text{tempoScore}_a + w_b \cdot \text{beatScore}_\text{a}) \tag{6.4}$$

where tempoScore_a is the tempo score of tracker a, $\text{beatScore}_\text{a}$ is the beat score of tracker a, and w_t and w_b are weights to control the relative importance of tempo and beat scores.

6.3.3 Scoring Clusters - Separate Clustering

Scoring Tempo Clusters

When clustering tempo and beat hypotheses separately, each tempo cluster is assigned a score based on the sum of the tempo confidences of each tracker in that cluster. Let T_i be the set of trackers in the i^{th} tempo cluster. The score for the tempo cluster is then given by summing the tempo scores of all trackers in the cluster as in Equation 6.5:

$$\text{clusterTempoScore}_i = \sum_{a \in T_i} \text{tempoScore}_a \tag{6.5}$$

Tempo cluster scores are then weighted to emphasize tempos that occur in integer relationships to other tempo clusters. The presence of tempos in integer relationships to one another is an indicator of metrical structure so applying this weighting will emphasize tempo clusters that have metrical relationships to other tempo clusters. Candidate tempos with little or no relationship to other tempo clusters are less likely to be the correct tempo. Accordingly, they will receive less weight under this scheme. The weighting $f(d)$ given in Equation 6.6 is based on that used by Dixon to initialize his beat tracking system from tempo clusters [21], where d is a scalar multiplier of the candidate tempos meant to emphasize more common metrical relationships (i.e. ratios of 1, 2, 3, or 4 between metrical levels) over less common relationships (i.e. ratios of 5, 6, 7, or 8).

$$f(d) = \begin{cases} 6 - d & \text{if } 1 \leq d \leq 4 \\ 1 & \text{if } 5 \leq d \leq 8 \\ 0 & \text{otherwise} \end{cases} \tag{6.6}$$

Given the i^{th} tempo cluster centroid t_i and j^{th} tempo cluster centroid t_j, with $1 \leq d \leq 8$, if $(t_i - d \cdot t_j)$ is within a small threshold then the score of cluster i is increased by $\text{clusterTempoScore}_j \cdot f(d)$ and the score of cluster j is increased by $\text{clusterTempoScore}_i \cdot f(d)$

Extending Dixon's scheme, Beaker additionally has an option to weight the two tempos in the integer relationship differently by specifying a weight w. In this

case, the the score of cluster j is increased by $w \cdot \text{clusterTempoScore}_i \cdot f(d)$. A value of $w > 1$ gives greater weight to the cluster with the smaller tempo, and vice-versa.

Scoring Beat Clusters

Each beat cluster is assigned a score based on the sum of the beat confidences of each tracker in that cluster. Let T_i be the set of trackers in the i^{th} beat cluster. The score for the beat cluster is then given by summing the beat scores of all trackers in the cluster as in Equation 6.7:

$$\text{clusterBeatScore}_i = \sum_{a \in T_i} \text{beatScore}_a \tag{6.7}$$

6.3.4 Voting

In all approaches, the tempo, beat, or joint cluster with the maximum score is chosen as representative of the current winning hypothesis. This is equivalent to the weighted majority vote approach to combining the output of an ensemble of classifiers [59], where the weight given to each tracker/classifier is its score. Once winning a winning joint cluster or winning tempo and beat clusters have been chosen, Beaker has the option of computing the actual winning tempo and beat hypotheses in one of two ways. When using joint clustering, the first approach uses the tempo and beat centroids of the winning cluster as the winning hypotheses. Similarly, using separate clustering, that approach uses the centroid of the winning tempo cluster as the winning tempo and the centroid of the winning beat cluster as the winning beat location. The second approach uses the tempo and/or beat hypotheses of the highest-scoring tracker in the winning cluster(s). The latter approach is similar to that used by Goto [32] when combining the hypotheses of multiple agents based on the agents' reliability scores.

The winning beat hypothesis is given in terms of the next predicted beat time during the clustering, scoring, and voting process. Section 6.5 will discuss how this beat hypothesis is converted into the beat phase output from the ensemble and used to determine beat locations.

6.4 Updating Confidences

After Beaker has produced an updated hypothesis, it must update its level of confidence in each tracker based on that tracker's current performance in the ensemble.

6.4.1 Winners and Losers

Trackers with the winning hypothesis are given greater weight than those in losing clusters. This is done by reducing the confidence of losing trackers based on how their tempo cluster scored compared with the winning tempo cluster. The confidence-from-winning (f_w) for a tracker is given by:

$$f_w = \frac{clusterScore_c}{clusterScore_w} \tag{6.8}$$

where c is the index of the tracker's tempo cluster and w is the index of the winning tempo cluster. When separate tempo and beat clustering is used, the cluster score used in this computation is the score of the tracker's tempo cluster. Because only trackers from the winning tempo cluster are clustered by beat, a beat cluster score is not available for all trackers. Therefore, the factor f_w is used to update both the tempo and beat confidences even when only based on tempo scores. Future work will separately consider tracker beat cluster scores when determining how much to penalize the beat confidence of losing trackers.

6.4.2 Tempo Continuity

The confidence factor f_t for a particular tracker is based on the difference between the tracker's tempo estimate $tempo_{tracker}$ and the ensemble's current winning estimate $tempo_{winning}$, scaled by the current winning tempo.

$$f_t = 1.0 - \frac{|tempo_{tracker} - tempo_{winning}|}{tempo_{winning}} \tag{6.9}$$

This gives the ensemble higher confidence in trackers whose tempo estimates agree with the ensemble's output.

6.4.3 Beat Continuity

The confidence factor f_b for a tracker is based on the continuity of the tracker's next predicted beat $beat_{tracker}$ with the ensemble's last recorded beat $beat_{winning}$, similar to how Goto and Muraoka evaluated the reliability of their agents [32] based on agreement with previously predicted beat times.

Equation 6.10 describes the difference between the two beat locations, which are wrapped into a range of $[-\frac{\tau}{2}, \frac{\tau}{2}]$ by subtracting the ensemble's current winning tempo period τ from the difference c times. For example, if a tracker's tempo estimate was half the current winning tempo, it would have twice the winning tempo period. Therefore, if that tracker was in phase with the ensemble's last recorded beat time, the tracker's next predicted beat location would be $beat_{winning} + 2\tau$, resulting in a value of $c = 2$.

$$\text{diff} = |beat_{tracker} - beat_{winning}| - c \cdot \tau \tag{6.10}$$

The beat continuity confidence factor is then computed as shown in equation 6.11, where maxDiff $= \frac{\tau}{2}$.

$$f_b = \frac{\text{maxDiff} - \text{diff}}{\text{maxDiff}} \tag{6.11}$$

6.4.4 Combining Confidences

These continuity and winner-based confidence metrics are combined to produce tempo and beat confidences for each tracker in the ensemble. Equations 6.12 and 6.13 are used to compute the ensemble's confidence in a particular tracker's tempo estimate (f_e) and confidence in the tracker's beat estimates (g_e) at time n:

$$f_e[n] = h \cdot f_e[n-1] + (1.0 - h) \cdot (w_w \cdot f_w + w_c \cdot f_t) \tag{6.12}$$

$$g_e[n] = h \cdot g_e[n-1] + (1.0 - h) \cdot (w_w \cdot f_w + w_c \cdot f_b) \tag{6.13}$$

where w_w and w_c are weights controlling the relative importance of winning vs. continuity, and h is a *confidence history factor* which specifies the level of continuity

enforced between the ensemble's confidence in a given tracker over time. These values are used to determine the next tempo and beat score for that tracker as described in Section 6.3.1.

6.5 Beat Phase and Detecting Beat Locations

In a causal beat tracking system designed for real-time use, it is often more practical to have continuous beat phase estimates rather than discrete beat locations as ensemble output. For example, in the case of a foot-tapping robot application, what is displayed to the user is the beat phase in terms of the current tempo. The foot should be down on a beat, up on the off-beat, and between those positions in between beats and offbeats. Because of this, Beaker ensembles by default provide updated beat phase hypotheses along with tempo estimates. The current winning beat phase is determined from the winning beat cluster time based on the current tempo period, where the beat phase has a range of $[0.0, 1.0]$, where a beat phase of 0.0 means that a beat is occurring in the current frame, a beat phase of 0.5 means that a beat is predicted to occur half a tempo period in the future, and a beat phase of 1.0 means that a beat is predicted to occur one tempo period in the future.

However, there is no guarantee that this phase will advance smoothly in the event that different tempos or beat locations are chosen as winners from frame to frame. To increase continuity in beat phase estimates, a dampening function is applied to new beat phase estimates. A predicted beat phase is computed from the previous beat phase estimate using the current tempo period, and the updated beat phase is computed by interpolating between the winning beat phase and the predicted beat phase. This prevents large unwanted phase discontinuities and essentially causes the ensemble to behave as if it has an internal beat oscillator being perturbed by updated ensemble beat phase estimates. This concept relates to the work of Large and Kolen, who used oscillators to model beat perception [50][49].

When the updated beat phase exceeds 1.0, it is wrapped back into the range of $[0.0, 1.0]$, and the ensemble then knows that a beat has occurred in the current frame. A precise beat location is then derived from the beat phase by comparing the

current and previous beat phase estimates to determine at what time the beat phase hit the 1.0 mark. Actual beat locations are not output in real-time by default but can be saved internally for later query or for writing to a text file when evaluating the system so that recorded beat locations can be compared with annotated datasets as will be described in Chapter 8.

6.6 Conclusion

This chapter introduced the Beaker framework for causal beat tracking using ensembles of beat trackers. Criteria were presented for tracker membership in a Beaker ensemble, and Beaker's approach to combining tracker hypotheses and evaluating their reliability was discussed. The following chapter will describe the TestTrackers used as members of an ensemble during development of this framework.

Chapter 7

Ensemble Test Trackers

7.1 Introduction

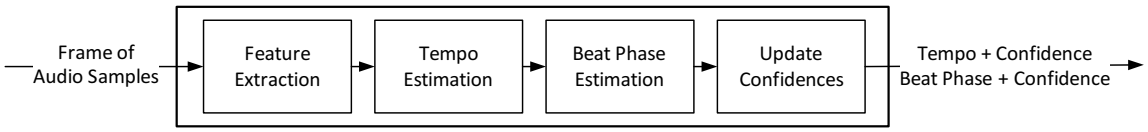

Figure 7.1: Diagram of TestTracker operation

TestTrackers are simple yet flexible real-time causal beat trackers. An ensemble of TestTrackers was used to develop the Beaker ensemble framework described in Chapter 6. As shown in Figure 7.1 and Algorithm 4, each TestTracker first extracts features from its input audio. Available features will be discussed in Section 7.2. A periodicity function is computed from the extracted features, and the current tempo is estimated from the periodicity function. TestTrackers can use one of several methods discussed in Section 7.3 to compute the periodicity function. The tempo estimate is then used to determine the optimal beat phase as discussed in Section 7.4. Once a TestTracker's tempo and beat phase hypotheses have been updated, its confidence level is updated and its hypotheses are given to the ensemble to be combined with those of other trackers. The computation of TestTracker tempo and beat confidences will be discussed in Section 7.5.

Algorithm 4 TestTracker tracking algorithm

TEST-TRACKER-TRACK

1: compute feature from frame of input samples
2: compute periodicity function from feature
3: estimate tempo from periodicity function
4: estimate beat phase
5: update tempo confidence
6: update beat confidence
7: output tempo and beat hypotheses and confidences

7.2 Feature Extraction

Onset detection functions (ODFs), introduced in Chapter 2 and described in detail by Bello et al. [7] and by Dixon [22], are commonly used as input features to tempo estimation and beat tracking systems. As previously discussed, not all onsets occur on beats, and not all beats have an onset occurring on them, but in most music, onsets will occur at points on a metrical grid defined by the time signature of the music. Therefore, by analyzing likely onset locations and observing periodic behavior, we can attempt to discern the tempo and metrical structure of a piece of music and likely locations of beats.

Only ODFs are currently available as input features for the TestTrackers described in this chapter. The nine ODFs used are listed in Table 7.1 and described in the following sections. However, other features, especially ones that might be more indicative of harmonic or chord changes may in some cases be more meaningful than an onset detection function, and such features are planned for future work. When initialized, each TestTracker is assigned one of the 9 available features to use, and this assignment is constant for the life of the TestTracker.

To describe the various ODFs available to TestTrackers, I will use the following notation: Let $x[n]$ be the n^{th} sample of an input audio stream x. The short-time

Table 7.1: Available features

Label	**Feature Name**
F_0	L1 Magnitude
F_1	L1 Magnitude Rectified
F_2	L2 Magnitude
F_3	L2 Magnitude Rectified
F_4	L1 High Frequency Content
F_5	L2 High Frequency Content
F_6	Complex Domain
F_7	L1 Phase Deviation
F_8	L2 Phase Deviation

Fourier transform (STFT) $X[n,k]$ of x is then given as

$$X[n,k] = \sum_{m=-\frac{N}{2}}^{\frac{N}{2}-1} x[hn+m]w[m]e^{-\frac{2i\pi mk}{N}} \tag{7.1}$$

where N is the frame size, h is the hop size, and $w[m]$ is a time-domain windowing function. The magnitude of $X[n,k]$ is given by $|X[n,k]|$ and the phase by $\phi[n,k]$.

In addition, I will use the half-wave rectifier function $H(x)$, given by

$$H(x) = \frac{x+|x|}{2} \tag{7.2}$$

Each TestTracker can be assigned a frequency range over which it will compute its feature. This allows experimentation with sub-band features as introduced in Section 2.2.4. The lowest bin for each TestTracker will be defined as K_1 and the highest as K_2, where $0 \leq K_1 < K_2 < N$.

7.2.1 Magnitude-Based ODFs

TestTrackers can use one of four magnitude-based onset detection functions as their input feature. Each computes the bin-by-bin difference between magnitude spectra of adjacent input audio frames and then sums across the bin magnitudes or

magnitudes squared to get a single value representing the likelihood of an onset in the current frame. Feature F_0 "L1 Magnitude" uses the sum of the magnitude bin differences. Feature F_1 "L1 Magnitude Rectified" does the same but then half-wave rectifies the result to emphasize onsets rather than ends of notes. Feature F_2 "L2 Magnitude" is similar to Feature F_0 but sums the squares of the differences between bin magnitudes. Feature F_3 "L2 Magnitude Rectified" is the half-wave rectified version of Feature F_2.

$$F_0[n] = \sum_{k=K_1}^{K_2} \big|\, |X[n,k]| - |X[n-1,k]| \,\big| \tag{7.3}$$

$$F_1[n] = \sum_{k=K_1}^{K_2} H\left(|X[n,k]| - |X[n-1,k]|\right) \tag{7.4}$$

$$F_2[n] = \sum_{k=K_1}^{K_2} \left(|X[n,k]| - |X[n-1,k]|\right)^2 \tag{7.5}$$

$$F_3[n] = \sum_{k=K_1}^{K_2} \left(H\left(|X[n,k]| - |X[n-1,k]|\right)\right)^2 \tag{7.6}$$

7.2.2 High Frequency Content ODFs

High-frequency content ODFs are also based on frame-by-frame differences in the magnitude spectrum. However, they weight each bin's difference by the bin index, thereby giving greater weight to higher-frequency bins. Feature F_4 "L1 High Frequency Content" uses the sum of weighted magnitude bin differences, while Feature F_5 "L2 High Frequency Content" uses the sum of the squares of the weighted magnitude bin differences.

$$F_4[n] = \frac{1}{K_2 - K_1 + 1} \sum_{k=K_1}^{K_2} k|X[n,k]| \tag{7.7}$$

$$F_5[n] = \frac{1}{K_2 - K_1 + 1} \sum_{k=K_1}^{K_2} k|X[n,k]|^2 \tag{7.8}$$

7.2.3 Complex Domain ODF

Feature F_6 "Complex Domain ODF" incorporates phase information as well as magnitude information by comparing bins of adjacent spectra in the complex domain. The two previous frames' magnitude and phase values are used to predict the current magnitude and phase for a steady-state signal, with the assumption that magnitude stays constant and phase advances linearly. The deviation of the actual magnitude and phase from the predicted values is then indicative of non-steady-state behavior, including onsets.

The predicted value of $X[n,k]$, notated as X_P, is given by:

$$X_P[n,k] = |X[n-1,k]|\, e^{\phi[n-1,k]+\phi'[n-1,k]} \tag{7.9}$$

where $\phi[n,k]$ is the phase of $X[n,k]$ and $\phi'[n,k]$ is the phase advance from frame $n-1$ to frame n, wrapped to the range $(-\pi,\pi]$:

$$\phi'[n,k] = \phi[n,k] - \phi[n-1,k] \tag{7.10}$$

The complex domain ODF is then defined as

$$F_6[n] = \sum_{k=K_1}^{K_2} |X[n,k] - X_P[n,k]| \tag{7.11}$$

7.2.4 Phase-Based ODFs

Phase-based ODFs use only phase information and not the bin magnitudes. Similar to the complex domain ODF, the expected phase advance for each bin is computed and compared with the observed phase advance. Phase-based ODFs are then given by the sum of the differences between expected and observed phases. Feature F_7 "L1 Phase Deviation" sums the differences between observed and expected phase advances, while Feature F_8 "L2 Phase Deviation" sums the squares of the differences.

We define $\phi''[n,k]$ as the change in the phase advance $\phi'[n,k]$:

$$\phi''[n,k] = \phi'[n,k] - \phi'[n-1,k] \tag{7.12}$$

wrapped into the range $(-\pi, \pi]$. Then the phase-based ODFs are defined as:

$$F_7[n] = \frac{1}{(K_2 - K_1 + 1)} \sum_{k=K_1}^{K_2} |\phi''[n,k]| \tag{7.13}$$

$$F_8[n] = \frac{1}{(K_2 - K_1 + 1)} \sum_{k=K_1}^{K_2} |\phi''[n,k]|^2 \tag{7.14}$$

7.3 Tempo Estimation

In order to estimate the current tempo, each TestTracker looks for periodic behavior in its extracted feature. Possible periodicity estimation methods in Beaker currently include a bank of comb filters tuned to candidate tempo periods [46], discrete Fourier transform (DFT) [57], and autocorrelation function (ACF) [27][54], all of which generate a periodicity function from the input feature, some in terms of tempo (DFT) and others in terms of tempo period (ACF, comb filter bank). A list of the available approaches is given in Table 7.2. The time range over which the periodicity function is computed is a parameter to each TestTracker, allowing varying reaction times to tempo changes. TestTrackers are also assigned a range of tempo periods $\tau_1 \leq \tau \leq \tau_2$ from which they will select the maximum of the periodicity function as their current tempo hypothesis. By assigning different tempo ranges, TestTrackers are encouraged to track at different metrical levels, an approach which generates a greater variety of tempo hypotheses and creates a more diverse ensemble. Each of the periodicity estimation approaches described below produces a periodicity function $P[\tau, n]$ at candidate tempo periods τ and time n. The TestTracker's tempo estimate in beats per minute (BPM) for time n is obtained from the maximum of the periodicity function:

$$tempo = \frac{60 \cdot f_s}{\max\limits_{\tau_1 \leq \tau \leq \tau_2} P[n,\tau]} \tag{7.15}$$

where f_s is the sampling rate of the input feature. Additionally, it is possible to set parameters such that additional TestTrackers will output not the maximum but the second or third etc. highest peak in the periodicity function. This also creates a larger number of hypotheses within a Beaker ensemble.

Table 7.2: Available periodicity approaches

Label	**Periodicity Approach**
P_0	Biased Autocorrelation
P_1	Unbiased Autocorrelation
P_2	DFT
P_3	Comb Filter Bank

7.3.1 Comb Filters

The use of banks of comb filters for tempo estimation was discussed in Section 3.3. Beaker TestTrackers using comb filters use the approach of Klapuri et al. [46]. A comb filter is created for each candidate tempo period in the TestTracker's valid tempo range. The delay τ of the comb filter is defined by the tempo period the filter is intended to observe. The output $y[\tau, n]$ of a comb filter with delay τ at time n is given by Equation 7.16, where $F[n]$ is the value of the TestTracker's input feature at time n.

$$y[\tau, n] = \alpha_\tau y[\tau, n - \tau] + (1 - \alpha_\tau)F[n] \tag{7.16}$$

As in [46], the feedback gain α_τ, defined in Equation 7.17, uses a half-time T_0 of 3.0 seconds.

$$\alpha_\tau = 0.5^{\frac{\tau}{T_0}} \tag{7.17}$$

The instantaneous energy $\hat{y}$ of each comb filter at time n is then given by Equation 7.18.

$$\hat{y}[\tau, n] = \frac{1}{\tau} \sum_{i=n-\tau+1}^{n} y[\tau, n]^2 \tag{7.18}$$

The overall power γ_τ of a comb filter with delay τ is computed as in Equation 7.19 and used to normalize the instantaneous energy using Equation 7.20 to give the value of the periodicity function $P_3[\tau, n]$ at tempo period τ and time n.

$$\gamma_\tau = \frac{(1 - \alpha_\tau)^2}{1 - \alpha_\tau^2} \tag{7.19}$$

$$P_3[n,\tau] = \frac{1}{1-\gamma_\tau}\left(\frac{\hat{y}[\tau,n]}{v[n]} - \gamma_\tau\right) \tag{7.20}$$

Finally, the function $v[n]$ given by Equation 7.21 gives the energy of the feature $F[n]$:

$$v[n] = \alpha_1 v[n-1] + (1-\alpha_1)F[n] \tag{7.21}$$

This normalization compensates for the skew seen in earlier comb filter-based periodicity estimation approaches such as that by Scheirer [63], where energy increased as τ increased.

7.3.2 Autocorrelation

Like the comb filter approach to periodicity estimation, autocorrelation produces a periodicity function indexed by tempo period τ, where the tempo period is varied as the autocorrelation lag. The use of autocorrelation for tempo estimation was discussed in Section 3.2. There are many ways of computing the autocorrelation of a finite-length sequence, and TestTrackers have the option of two different approaches. The first is approach P_0, the biased ACF, defined in Equation 7.22:

$$P_0[\tau] = \frac{1}{N}\sum_{n=0}^{N-\tau-1} x[n]x[n+\tau] \tag{7.22}$$

The second is approach P_1, the unbiased ACF, defined in Equation 7.23:

$$P_1[\tau] = \frac{1}{N-\tau}\sum_{n=0}^{N-\tau-1} x[n]x[n+\tau] \tag{7.23}$$

The unbiased ACF attempts to compensate for the fact that in the biased ACF, as τ increases the number of points in the sum decreases, giving extra weight to smaller tempo periods.

7.3.3 Discrete Fourier Transform

The use of the DFT for tempo estimation was discussed in Section 3.2. Unlike the comb filter and autocorrelation approaches, the DFT produces a periodicity function indexed by tempo rather than tempo period. TestTrackers using the DFT

for periodicity estimation use an 8192-sample window, as Peeters did when performing his combined ACF+DFT approach [57]. A Hann windowing function is used, and the number of samples in an analysis frame is padded with zeros if necessary to reach this size of window.

7.4 Beat Phase Estimation

Once a TestTracker has obtained a tempo estimate, it determines the most likely phase alignment of beats that fits the chosen tempo by correlating a simulated beat sequence with its input feature $F[n]$. Similar approaches to determining beat phase have previously been used by others including Goto and Muraoka [34] and Bock and Schedl [9]. The beat sequence offset producing the maximum correlation is chosen as the most likely alignment. An estimated beat phase is then reported in the range $[0.0, 1.0]$, where 0.0 means that a beat is occurring at the current time, and 1.0 represents one tempo period in the future.

When using the comb filter and ACF approaches to tempo estimation, a TestTracker's tempo period is an integer value, and generating a simulated beat sequence with the given tempo period is straightforward using single-sample impulses. However, when using the DFT approach, the tempo period is the inverse of an integer value. Therefore, creating a simulated beat sequence at the correct tempo is not as simple, as simulated beats may lie between samples. To handle this situation, Beaker TestTrackers have several options for determining the optimal beat phase given a particular tempo. Instead of using single-sample impulses, TestTrackers may choose to generate simulated beat sequences using rectangular, triangular, or Gaussian pulses centered at beat locations. The choices of pulse shape and width are parameters of a TestTracker instance. In the case of triangular and Gaussian pulses, more weight is implicitly given to the integer sample closest to the non-integer beat. However, in all cases, only integer numbers of frames are considered as possible phase offsets when determining optimal phase alignment. This means that the accuracy of the beat phase estimation is limited by the choice of feature sampling rate.

7.5 TestTracker Confidence

Each TestTracker is responsible for reporting its level of confidence in its tempo and beat phase hypotheses to the ensemble. This allows the ensemble to give greater weight to more confident trackers when combining tracker results. All confidence values are normalized to be in the range $[0.0, 1.0]$.

7.5.1 Tempo Confidence

TestTracker tempo confidence at a particular time n is a combination of two factors. As discussed in Section 4.3.5, Degara et al. used a measure of the periodic nature of their ODF when determining the reliability of their tracker [19]. Along these same lines, the first factor in TestTracker tempo confidence f_1 is a measure of the "peakiness" of the periodicity function. Peakiness is defined in Equation 7.24 and is based on the relationship between the value of the periodicity function at the chosen tempo period τ_n and the mean μ of the periodicity function P:

$$f_1[n] = \frac{P(\tau_n) - \mu}{P(\tau_n)} \tag{7.24}$$

The mean of the periodicity function for the n^{th} frame is given by Equation 7.25:

$$\mu = \begin{cases} \frac{1}{T}\sum_{\tau=1}^{T} P(\tau) & \text{if } \mu \geq 0.0 \\ 0.0 & \text{otherwise} \end{cases} \tag{7.25}$$

where T is the number of possible tempo periods in the periodicity function.

The second factor f_2, defined in Equation 7.26, is based on continuity with previous tempo estimates made by the same TestTracker, where t_n is the TestTracker's tempo estimate at time n:

$$f_2[n] = \begin{cases} 1.0 - \frac{|t_n - t_{n-1}|}{t_{n-1}}, & \text{if } |t_n - t_{n-1}| < t_{n-1} \\ 0.0, & \text{otherwise} \end{cases} \tag{7.26}$$

At the n^{th} frame, the TestTracker's new tempo confidence, $f[n]$ in Equation 7.27 is given by a weighted sum of factors $f_1[n]$ and $f_2[n]$ with weights w_1 and w_2,

normalized by the sum of the weights to keep the confidence in the range $[0.0, 1.0]$:

$$f[n] = \frac{w_1 \cdot f_1[n] + w_2 \cdot f_2[n]}{w_1 + w_2} \tag{7.27}$$

Finally, the updated confidence for each TestTracker tempoConf$[n]$ is obtained by interpolating between the tracker's previous confidence and new confidence based on a *confidence history factor* parameter w_h to provide some continuity in TestTracker confidence levels over time:

$$\text{tempoConf}[n] = w_h \cdot \text{tempoConf}[n-1] + (1.0 - w_h) \cdot f[n] \tag{7.28}$$

7.5.2 Beat Confidence

The beat confidence value is based on the quality of the correlation between the input feature and estimated beat sequence described in Section 7.4 given the chosen beat phase. In this way, ambiguous beat phase estimates will result in reduced beat confidence.

7.6 Conclusion

This chapter has introduced the simple real-time causal TestTracker beat trackers that were used to develop and test the Beaker ensemble beat tracking framework. Various options for creating diverse ensembles of TestTrackers including different input features, approaches to periodicity estimation, and tempo ranges were discussed, and the methods for determining a TestTracker's confidence in its tempo and beat hypotheses were described.

Part IV
Evaluation and Results

Chapter 8

Evaluation Methodology

8.1 Introduction

This chapter will discuss datasets and methods for evaluating the performance of beat tracking systems. Some beat tracking datasets contain full songs, while others contain song excerpts or combinations of full songs and excerpts. For simplicity, throughout this dissertation the word *song* will be used to describe a member of a dataset, whether it is a full song or only an excerpt. All of the methods described in this chapter are used to evaluate the performance of a system after an entire song has been processed and do not take into consideration whether a system processed the excerpt in question using a causal or non-causal approach.

There are many challenges when evaluating beat tracking systems, particularly since the goal of a beat tracking system is to determine the locations of perceived beats - the places where a human might tap his or her foot in time to music. The most obvious challenges arise from the fact that in the many cases humans themselves cannot agree on one definitive ground-truth tempo or sequence of beat locations. Human listeners might tap their feet at different metrical levels (i.e. twice or half as fast), and some might tap on the off-beat (out of phase). The term *octave error* is commonly used to describe beat tracking errors that occur due to tracking at the wrong metrical level.

Different evaluation metrics, and even different datasets, take these errors into

account in different ways. Starting in 2006, the beat tracking task from the Music Information Retrieval Evaluation eXchange (MIREX) attempted to address the issue of metric level ambiguity by obtaining annotations by multiple listeners and scoring beat tracking systems based on how well their beat locations matched the various annotations [53]. However, obtaining annotations from a large pool of listeners is not practical for many researchers, so most beat tracking datasets still consist of a single set of ground-truth annotated beats for each song or song excerpt. Some evaluation methods attempt to compensate for the single set of annotations by computing beat tracking errors at integer multiples or subdivisions of the annotated tempo [14], but this approach cannot take into consideration the likelihood of a listener agreeing with annotations at each possible metric level.

Another challenge in beat tracking evaluation is the availability of shared datasets. There are often legal (copyright) issues involved with publicly sharing audio files, and this can make it difficult to obtain both audio files and annotations for datasets used by other researchers.

8.2 Datasets

8.2.1 Beatles

The majority of the evaluation of the Beaker beat tracking system was done using the *Beatles dataset*. This is a dataset consisting of twelve Beatles albums annotated by Davies, Degara, and Plumbley [14]. The Beatles were chosen because of the large number of songs available and the variety of tempos and musical styles represented.

Text files containing a single set of ground-truth annotated beat locations are publicly available along with a list of the versions of each Beatles album used to generate the annotations. Users of the dataset are responsible for obtaining their own copies of the appropriate album versions. While this approach was intended to work around the need for sharing copyrighted material, the dataset is not as easy to obtain as the annotators intended, since some album versions were released in the

United Kingdom (where the annotators obtained their copies) differently from in the United States, and these UK versions are not as easy to acquire in other countries. Despite this obstacle, I was able to obtain the correct versions of all twelve albums and complete the dataset.

8.2.2 MIREX 2006

Like most other beat-tracking datasets, the Beatles dataset provides a single sequence of annotated beat locations for each song. However, this fails to take into consideration the different behavior of human listeners and assumes that they would all tap their feet in the same locations. To address this dilemma, McKinney and Moelants recorded human tapping behavior for a collection of 160 constant-tempo song excerpts that have been used in the MIREX audio tempo estimation and audio beat tracking tasks since 2006 [52][53]. In this dataset, 40 separate annotation sequences for each excerpt were generated by different human listeners. Differences in human behavior are therefore captured in the distribution of annotations. While a subset of this dataset consisting of 20 annotated training excerpts has been made publicly available, the complete dataset is reserved for use in MIREX. The relatively small size of the available training dataset, along with the constraint of constant tempo excerpts makes this dataset less suitable for evaluating Beaker than the Beatles dataset.

8.2.3 SMC MIREX

The *SMC MIREX dataset* was added to the MIREX audio beat tracking evaluation beginning in 2012. This dataset consists of 217 song excerpts chosen by Holzapfel et al. using a selective sampling approach [42] to choose excerpts which were particularly challenging for existing beat tracking systems due to characteristics such as expressive performances, tempo or meter changes, or lack of clear onsets. In addition to its use as part of MIREX, this dataset has been made publicly available. It is used in some evaluation of the Beaker beat tracking system along with the Beatles dataset.

8.3 Beat Tracking Evaluation Metrics

This section provides an overview of the beat tracking evaluation methods used in this dissertation. The majority of methods act on a per-song basis within a dataset, with scores averaged across all songs to obtain a single score for the dataset.

The notation used to describe each metric is taken from Davies, Degara, and Plumbley [14]: γ is a sequence of B beats generated by beat tracking algorithm being evaluated and γ_b is the time of the b^{th} beat, while a is the sequence of J ground truth annotations, and a_j is the time of the j^{th} beat. The inter-beat interval (IBI) is $\Delta_b = \gamma_b - \gamma_{b-1}$, while the inter-annotation interval (IAI) is $\Delta_j = a_j - a_{j-1}$.

8.3.1 F-measure

F-measure is an evaluation metric commonly used with binary classifiers. In the case of beat tracking, as described in [14], we define the binary match or non-match of an annotated beat a_j to an estimated beat γ_b using a fixed tolerance window of 70 milliseconds in the indicator function $I[j, b]$ given in equation 8.1.

$$I[j,b] = \begin{cases} 1, & \text{if } |a_j - \gamma_b| < 70 \text{ ms} \\ 0, & \text{otherwise} \end{cases} \tag{8.1}$$

The match $I[j]$ of an annotated beat a_j to any estimated beat is given by equation 8.2.

$$I[j] = \begin{cases} 1, & \text{if } |a_j - \gamma_b| < 70 \text{ ms for any } 1 < b \leq B \\ 0, & \text{otherwise} \end{cases} \tag{8.2}$$

The match of an estimated beat γ_b to any annotated beat is given by equation 8.3.

$$I[b] = \begin{cases} 1, & \text{if } |a_j - \gamma_b| < 70 \text{ ms for any } 1 < j \leq J \\ 0, & \text{otherwise} \end{cases} \tag{8.3}$$

Three quantities are computed for each excerpt. The first is the number of true positives (TP) given by equation 8.4, which is the number of matches between

ground truth annotated beats a and estimates γ. The second is the number of false positives (FP) given by equation 8.5, which is the number of unmatched estimated beats, and the third is the number of false negatives (FN) given by equation 8.6, which is the number of unmatched annotated beats.

$$\mathrm{TP} = \sum_{j=1}^{J} \sum_{b=1}^{B} I[j, b] \tag{8.4}$$

$$\mathrm{FP} = B - \sum_{b=1}^{B} I[b] \tag{8.5}$$

$$\mathrm{FN} = J - \sum_{j=1}^{J} I[j] \tag{8.6}$$

The common statistical measures of precision p and recall r are computed from these quantities as shown in equations 8.7 and 8.8.

$$p = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{8.7}$$

$$r = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{8.8}$$

In this work, as in [14], the traditional definition of F-measure as the harmonic mean of precision and recall is used as in Equation 8.9, although Dixon [21] used a variation of F-measure, shown in Equation 8.10.

$$Fmeasure = \frac{2pr}{p + r} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \times 100\% \tag{8.9}$$

$$Fmeasure = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \times 100\% \tag{8.10}$$

One criticism of F-measure is the fact that it does not take into account true negatives. Also, tracking on the off-beat is heavily penalized, giving a score close to zero, because none of the estimated beats will fall into the the windows around annotations unless the tempo is very fast or the windows very wide.

8.3.2 Cemgil Accuracy Measure

F-measure uses a square window to determine matches between annotated and estimated beats. All matches within that window are given equal weight. However, depending on the precision of the annotations, it may be more informative to give greater weight to the estimated beats which more closely match annotations. To do this, Cemgil et al. use Gaussian windows to evaluate the performance of their tempo tracker [12]. In the Cem_{acc} evaluation metric, the list of annotated beats a_j is compared to the list of estimated beats γ_b. For each a_j, the closest γ_b is found. A Gaussian window is centered at the annotated beat time and used to weight the score of that match, so that estimated beats lying exactly on the annotated beat times give the greatest score. Match scores are summed across all annotated beats and normalized by the number of annotations J and estimated beats B as shown in Equation 8.11, where W is the Gaussian window defined by Equation 8.12 and $\sigma_e = 0.04$ seconds. The results can be interpreted as the percentage of experimental beats that lie within the window of annotated beats.

$$\text{Cem}_{\text{acc}} = \frac{\sum_j \max_b W(a_j - \gamma_b)}{(J+B)/2} \times 100 \tag{8.11}$$

$$W(a_j - \gamma_b) = exp\left(\frac{-(a_j - \gamma_b)^2}{2\sigma_e^2}\right) \tag{8.12}$$

Like F-measure, Cem_{acc} gives a score of almost zero for tracking on the off-beat, because the Gaussian windows are centered at annotated beats and will be close to zero at the location of an off-beat. In general, estimated sequences tracking at the correct tempo but with the wrong beat phase will be penalized based on how far out of phase they are, with the greatest penalty given to off-beat tracking. Unlike F-measure, which considers false positives, Cem_{acc} ignores extra estimated beats, because only the estimated beat closest to each annotation is considered. However, the inclusion of the number of estimated beats B in the denominator of Equation 8.11 means that tracking at multiples of the correct tempo will still incur some penalty. Missing estimated beats (false negatives) are similarly penalized due to the presence of the number of annotated beats J in the denominator in addition to being given a

zero score.

8.3.3 P-score

P-score is a beat tracking evaluation metric used in MIREX since 2006 and based on the cross-correlation of beat sequences [53]. An impulse train T_a is generated with pulses at each annotated beat location a_j as in Equation 8.13, quantized to a sampling rate of 100Hz. Similarly, an impulse train T_γ is generated from estimated beats γ_b as in Equation 8.14. Beat times in the first 5 seconds of an excerpt are ignored in both sequences. The two impulse trains are then cross-correlated as in Equation 8.16, using an error window W defined as 20% of the median IAI across all annotations in a dataset. The normalization factor NP is defined in Equation 8.15 as the number of annotations or the number of estimated beats, whichever is greater. The resulting P-score is in the range $0.0 \leq P \leq 1.0$ but in this work, as in [14], it is multiplied by 100% to be consistent with other measures.

$$T_a(n) = \begin{cases} 1, & \text{if } n = a_j \\ 0, & \text{otherwise} \end{cases} \tag{8.13}$$

$$T_\gamma(n) = \begin{cases} 1, & \text{if } n = \gamma_b \\ 0, & \text{otherwise} \end{cases} \tag{8.14}$$

$$\text{NP} = \max(J, B) \tag{8.15}$$

$$\text{Pscore} = \frac{1}{\text{NP}} \sum_{m=-W}^{W} \sum_{n=1}^{N} T_\gamma[n] \cdot T_a[n-m] \tag{8.16}$$

P-score does not attempt to take into account tracking at different metrical levels or on the off-beat. It was developed for use with the MIREX 2006 dataset described in Section 8.2.2, which contains annotations for each excerpt by 40 human annotators. In that scenario, the P-score was averaged across all annotators, and metrical ambiguity or off-beat tapping were considered implicitly by the fact that different human annotators might tap in different ways. However, this is a weakness

of P-score as an evaluation metric when used with other datasets consisting of a single set of annotations.

Goto Accuracy Measure

Goto and Muraoka introduced a continuity-based beat tracking accuracy measure [33]. Annotated and estimated beats are paired using a criteria where estimated beat γ_b is paired with annotation a_j if γ_b is the closest γ_b within a window of half the IAI Δ_j.

A normalized difference P_j, defined in Equation 8.17 is computed for each pair containing annotated beat a_j and estimated beat γ_b. The normalization factor i_j is defined in Equation 8.18 as half of the local IAI. A pair is considered a match if $P_n < 0.35$.

$$P_j = \begin{cases} \frac{|\gamma_b - a_j|}{i_j}, & \text{if } a_j \text{ is paired} \\ 1, & \text{if } a_j \text{ is unpaired} \end{cases} \tag{8.17}$$

where

$$i_j = \begin{cases} \Delta_{j+1}/2, & \text{if } \gamma_b \geq a_j \\ \Delta_j/2, & \text{if } \gamma_b < a_j \end{cases} \tag{8.18}$$

After pairing, the mean μ, standard deviation σ, and max M of P_n are computed, and the longest correctly-tracked sub-sequence in a sequence is identified. A sub-sequence is correctly-tracked if each annotation has a matching estimated beat according to P_n and no estimated beats in that sub-sequence are unpaired.

Goto and Muraoka obtain a binary answer for each sequence (either correct or not), as they are interested in whether or not their beat tracker converges to the correct tempo and beat phase over time and also in tracking accuracy as defined by μ, σ, M. Their test material consists of 60-second constant-tempo song excerpts, and a sequence is considered correct only if a correctly-tracked sub-sequence starts before 45 seconds into the 60 second excerpt and continues through the end of the excerpt. To be considered correct, each sequence must have $\mu < 0.2$, $\sigma < 0.2$, and $M < 0.35$.

Davies et al. modify Goto and Muraoka's accuracy measure to account for not having 60-second constant-tempo songs, since they are not testing convergence to a single correct tempo [14] but rather overall performance. In the Goto_{acc} accuracy measure, they look for any correctly-tracked period over 25% of the song, which will give a score of 100%. If no correctly-tracked period is found, the song receives a score of 0%. Goto and Muraoka compute scores for three metrical levels: eighth note, quarter note, and half note and also check for off-beat tracking at the quarter note and half note levels. However, with Goto_{acc}, no provision is made for tracking at different metrical levels or on the off-beat, heavily penalizing trackers that give those results even though the modifications of Davies et al. mean that missing one beat towards the end of the song does not result in a score of 0% as it would with Goto and Muraoka's original approach.

8.3.4 Continuity-Based Methods

A family of continuity-based beat tracking evaluation methods was developed by Klapuri [45], inspired by the Goto_{acc} method described in Section 8.3.3. A subset of these methods was used by Hainsworth [39], and this work will use the variations specified by Davies et al. [14]. These methods look for the longest continuous correctly-tracked segment as a percentage of the entire beat sequence, requiring both tempo and beat phase to be within specified tolerances for an estimated beat to be considered correct. Four variations on this approach will be discussed here: CML_{c} (Correct Metrical Level, continuity required), CML_{t} (Correct Metrical Level, continuity not required), AML_{c} (Allowed Metrical Levels, continuity required), and AML_{t} (Allowed Metrical Levels, continuity not required).

Davies et al. [14] define three criteria for an estimated beat γ_b to be considered a correct match to annotated beat a_j. As in the work of Klapuri et al. [46], a tolerance window is defined to be within $\theta = 17.5\%$ of the current IAI Δ_j. The first criteria, in Equation 8.19, is that the estimated beat γ_b must fall within the tolerance window of the annotated beat a_j. The second criteria, in Equation 8.20, is that the previous estimated beat γ_{b-1} must be within the tolerance window of the previous annotated

beat a_{j-1}. Finally, the third criteria, in Equation 8.21, is that the IBI Δ_b must be within the specified tolerance window around the IAI Δ_j. This third criteria forces continuity of tempo as well as beat phase.

$$a_j - \theta\Delta_j < \gamma_b < a_j + \theta\Delta_j \tag{8.19}$$

$$a_{j-1} - \theta\Delta_{j-1} < \gamma_{b-1} < a_{j-1} + \theta\Delta_{j-1} \tag{8.20}$$

$$(1-\theta)\Delta_j < \Delta_b < (1+\theta)\Delta_j \tag{8.21}$$

All continuous correctly-tracked segments of a beat sequence are identified using these criteria. The four metrics $\mathrm{CML_c}$, $\mathrm{CML_t}$, $\mathrm{AML_c}$, and $\mathrm{AML_t}$ are computed from the M identified correct segments, which have length Υ_m. The first metric $\mathrm{CML_c}$ is shown in Equation 8.22 and considers only the longest segment. To reduce the effect of one bad beat in the middle of the sequence (which would result in $\mathrm{CML_c} = 50\%$, the second metric, $\mathrm{CML_t}$ in Equation 8.23, considers the total length of all correctly-tracked segments. Finally the last two metrics $\mathrm{AML_c}$ and $\mathrm{AML_t}$ are computed in the same way but are more general and allow segments to be considered correct if they track at double or half the annotated tempo or on the off-beat.

$$\mathrm{CML_c} = \frac{\max(\Upsilon_m)}{J} \times 100\% \tag{8.22}$$

$$\mathrm{CML_t} = \frac{\sum_{m=1}^{M} \Upsilon_m}{J} \times 100\% \tag{8.23}$$

8.3.5 Information Gain

Davies et al. [17] introduced a beat tracking evaluation metric using the entropy of a beat error histogram. In later work, this was refined to determine the information gain (Kulback-Leibler divergence) between the beat error histogram of annotated vs. estimated sequences and a beat error histogram with a uniform error distribution, where the worst-case scenario is no correlation between annotations and

estimated beats [14] [15]. This information gain is intended to be a measure of the peakiness of the beat error histogram. In a well-performing beat tracker, a very peaky beat error histogram is expected, with a strong peak at 0. In the event that a tracker tracks at a faster metrical level than the annotations, additional peaks would be expected at subdivisions of the beat, and if a tracker tracks on the off-beat or other consistent phase offset, the main peak of the histogram would appear offset from the center. The information gain measure D is computed for a beat error histogram with K bins of height $1/K$ as in Equation 8.24 for each song in a dataset, and the average is used as the score for the dataset. The term $p_\zeta(z_k)$ is the estimated probability mass of the beat error distribution and $H(p_\zeta(z_k))$ is the entropy. Alternatively, the global information gain measure $\mathrm{D_g}$ can be computed from a global beat error histogram where beat errors are across all songs.

$$\mathrm{D} = \log(K) - H(p_\zeta(z_k)) \tag{8.24}$$

Note that unlike all of the other error metrics discussed in this chapter, the information gain measures do not provide scores in the range of 0-100% and therefore cannot be directly compared with the other metrics.

8.4 Evaluating Beaker

Version 0.1 of the BeatEval Toolbox[1] provides a MATLAB implementation of the beat tracking evaluation methods used by Davies et. al. in [14] and described in detail in this chapter. All ten methods provided by the BeatEval Toolbox were used in this dissertation to evaluate the performance of the Beaker beat tracker. These methods are F-measure (Section 8.3.1), $\mathrm{Cem_{acc}}$ (Section 8.3.2), $\mathrm{Goto_{acc}}$ (Section 8.3.3), P-score (Section 8.3.3), $\mathrm{CML_c}$ (Section 8.3.4), $\mathrm{CML_t}$ (Section 8.3.4), $\mathrm{AML_c}$ (Section 8.3.4), $\mathrm{AML_t}$ (Section 8.3.4), D (Section 8.3.5), and $\mathrm{D_g}$ (Section 8.3.5). Because each metric emphasizes different aspects of a beat tracker's performance, it is often difficult to draw conclusions from a single metric. Some systems will perform better

[1] http://c4dm.eecs.qmul.ac.uk/downloads/beateval/

with some metrics and worse with others. To simplify performance comparisons, the following chapters will frequently use the mean of the first eight evaluation metrics mentioned (abbreviated as Mean8). All of these metrics provide scores in a range of 0% (worst) to 100% (best), allowing them to be combined and compared in a meaningful fashion. However, because they do not use a comparable scale, the information gain metrics can not be included in this combined metric.

8.5 Conclusion

This chapter has introduced the beat tracking datasets and metrics that will be used to evaluate Beaker performance in subsequent chapters. The strengths and weaknesses of the various datasets and metrics were discussed, and the Mean8 metric was introduced as a way of combining the eight percentage-based evaluation metrics.

Chapter 9

Beaker Performance vs. Other Approaches

9.1 Introduction

This chapter presents some results of the performance of the Beaker ensemble framework compared to other beat tracking approaches. First, a Beaker ensemble of TestTrackers will be compared with state-of-the art beat tracking systems. Then the performance of Beaker's approach to combining the results of ensemble members using different input features in a causal fashion will be compared with another approach which combines the output of multiple beat trackers after the fact. Beaker ensembles consisting of TestTrackers are used throughout this chapter to evaluate Beaker performance.

9.2 Comparison with Other Systems

This section will show that a Beaker ensemble consisting of TestTrackers is competitive with a state-of-the-art causal beat tracking system and, in some metrics, can even compete with state-of-the-art non-causal beat tracking systems.

9.2.1 Beatles Dataset

Table 9.1: Beatles dataset algorithm comparison

Algorithm	Causal?	F-measure	Cem_{acc}	Goto_{acc}	P-score
Davies	Non-causal	81.4	79.5	76.0	77.0
KEA	Non-causal	80.4	67.0	71.5	77.9
Dixon	Non-causal	83.2	77.0	65.9	79.1
Ellis	Non-causal	75.7	49.7	44.7	69.7
Beaker	Causal	74.9	57.2	46.9	76.8
Determin.	N/A	24.4	17.4	0.0	34.0

Algorithm	CML_{c}	CML_{t}	AML_{c}	AML_{t}	D	D_{g}	Mean8
Davies	63.0	67.6	81.0	87.7	3.46	2.72	76.650
KEA	52.5	68.3	67.3	86.9	2.20	1.46	71.475
Dixon	52.5	64.2	68.8	89.7	2.44	1.70	72.550
Ellis	37.0	46.3	49.2	83.1	2.56	1.51	56.925
Beaker	33.0	61.7	41.7	79.1	1.73	0.73	58.904
Determin.	2.4	15.5	2.8	17.6	0.08	0.01	14.262

In 2009, Davies, Degara, and Plumbley [14] presented an overview of the beat tracking evaluation methods described in Chapter 8. As part of this overview, they provided the results of each of the metrics for four state-of-the-art non-causal beat tracking systems, and compared with the results of a completely deterministic tracker (*Determin.*) that always tracks at 120 beats per minute (BPM). The Beatles dataset described in Section 8.2.1 was used to test the systems. The first tracker included in the comparison was the *Davies* tracker, which was expected to perform very well because it was used in the process of generating the Beatles annotations. The other systems used were those by *KEA* (Klapuri, Eronen, and Astola) [46], *Dixon* [23], and *Ellis* [27].

Davies et al. made the MATLAB code used to generate their results publicly available, so it is possible to use their code to evaluate other systems and compare them to their reported results on the aforementioned trackers. The performance of the Beaker beat tracking system on the Beatles dataset is shown in Table 9.1

in comparison with the other trackers. Because Beaker is a causal system being compared to the other systems which are all non-causal, it is at a disadvantage and we do not necessarily expect to see Beaker outperform the other systems. Despite this, Beaker is able to outperform the Ellis system on the Cem_{acc}, Goto_{acc}, P-score, CML_{t}, and Mean8 metrics. Using the P-score metric, Beaker additionally performs very similarly to the Davies tracker. Where Beaker suffers most compared to the other systems is in the metrics requiring continuity: Goto_{acc}, CML_{c}, and AML_{c}. This behavior is not unexpected, since Beaker, as a real-time causal system, is designed to be more adaptable to changes in tempo and meter at the expense of some continuity.

9.2.2 SMC Dataset

It is also desirable to compare Beaker's performance with a state-of-the-art causal system. The *IBT-C* beat tracker [54], the causal version of Oliveira et al.'s system described in Section 4.2.3, is one such system. It was evaluated in 2012 on the Music Information Retrieval Evaluation eXchange (MIREX) audio beat tracking task, so results for the SMC dataset described in Section 8.2.3 are publicly available for comparison with the performance of Beaker on the same dataset during MIREX 2014. The MIREX results from [2] are shown in Table 9.2. Also shown are results for an improved version of the Beaker beat tracker which represents bug fixes and improvements in the performance since the 2014 MIREX evaluation.

Beaker's results for the 2014 MIREX were verified using the BeatEval toolbox, and all official MIREX results were reproducible except for the information gain metrics D and D_{g}. For this reason, only the official MIREX results of those metrics are included in Table 9.2, and no comparison is available using those metrics with the improved version of Beaker.

The Beaker MIREX 2014 submission only outperformed the IBT-C MIREX 2012 submission on 4 of the 10 metrics, and its Mean8 was lower, but the more recent Beaker implementation shows improvement in all metrics, additionally out-performs the IBT-C MIREX submission in the CML_{c} metric, and has a higher Mean8 score. We can conclude that for the SMC dataset, the performance of the Beaker beat

tracker is comparable to, if not better than, the state-of-the-art causal IBT-C beat tracking system. In addition, one of the most significant weaknesses of the IBT-C beat tracker is that it is designed to track slow changes in tempo and therefore requires explicit state recovery and re-initialization of the system when handling input with sudden tempo changes (such as streaming input of concatenated songs, a scenario not tested in the MIREX evaluation) [54]. The Beaker beat tracker theoretically has an advantage over IBT-C because of its ability to handle such changes without treating them as a special case.

Table 9.2: SMC dataset algorithm comparison

Algorithm	F-measure	Cem_{acc}	Goto_{acc}	P-score
IBT-C MIREX 2012	27.3664	20.9755	4.1475	43.4168
Beaker MIREX 2014	28.9582	21.4169	2.3041	44.0329
Improved Beaker	30.0908	22.6250	3.2258	45.8494

Algorithm	CML_{c}	CML_{t}	AML_{c}	AML_{t}	D	D_{g}	Mean8
IBT-C MIREX 2012	8.0578	13.1900	13.1597	23.1326	0.7115	0.0399	19.1808
Beaker MIREX 2014	5.0919	7.8419	10.7098	20.1576	0.6211	0.0552	17.5642
Improved Beaker	8.2538	12.1517	12.7624	21.7767	N/A	N/A	19.5920

9.3 Combining Features

It has been shown that combining the results of beat trackers using different input features can give improved results over using a beat tracker with a single input feature [68]. A Beaker ensemble can be created using TestTrackers with different input features, and Section 9.3.1 will demonstrate that Beaker ensembles utilizing multiple features out-perform single-feature ensembles. Sections 9.3.2 and 9.3.3 will then compare Beaker's approach to combining trackers with different features to the work of Zapata et al. [68] using non-causal beat trackers and show that Beaker's approach to combining multiple features provides better results than their Mean Mutual Agreement (MMA) approach in addition to being applicable in both causal

and non-causal beat tracking scenarios.

9.3.1 Combining Features with Beaker

An experiment was performed comparing ten different Beaker ensembles. Nine of the ten ensembles consisted of TestTrackers all using the same feature. One ensemble used feature F_0, another feature F_1, etc. as defined in Table 7.1 in Section 7.2. For example, Figure 9.1 shows an ensemble consisting of trackers all using feature F_0. The tenth ensemble was generated by merging the first nine ensembles into one ensemble that included trackers using all of the different available features, as shown in Figure 9.2.

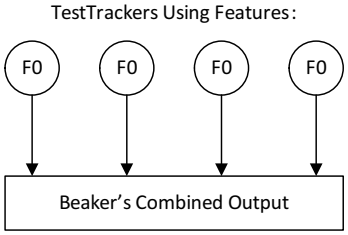

Figure 9.1: Beaker ensemble using only feature F_0

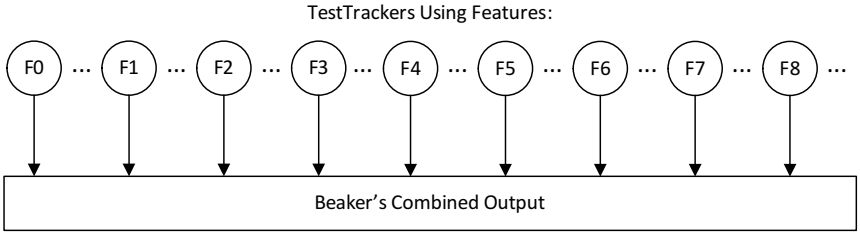

Figure 9.2: Beaker ensemble using all features

As shown in Table 9.3, the multi-feature ensemble out-performed the single-feature ensembles on all metrics. For each metric in Table 9.3, the score for the highest-performing feature is highlighted in bold. Note that while the magnitude-based and complex-domain-based onset detection functions (ODFs) (F_0, F_1, and F_6) consistently outperform the others, there is no one feature that performs the best on all metrics. Because the evaluation metrics emphasize different aspects of

beat tracker performance, this supports the hypothesis that the various features help to create a diverse ensemble. Another important observation is that while on some metrics the improvement is minimal, on the continuity-based metrics including Goto_{acc}, CML_{c}, and AML_{c}, the improvement is more significant. This implies that combining features specifically plays a role in improving beat tracking continuity in a Beaker ensemble, possibly because different features are performing well at different times during a song, helping to maintain continuity at times when a single feature might temporarily perform poorly.

Table 9.3: Results for single features vs. all (best single feature for each measure highlighted in bold)

Feature	F-measure	Cem_{acc}	Goto_{acc}	P-score
F0	74.0	56.3	38.5	76.0
F1	**74.6**	55.7	**43.6**	**76.1**
F2	71.4	54.5	38.0	74.9
F3	71.8	53.9	36.3	74.8
F4	65.1	49.3	27.4	68.5
F5	56.8	43.1	19.0	62.7
F6	73.6	**56.4**	38.5	76.1
F7	32.3	23.5	0	39.9
F8	33.9	24.8	0	41.2
All	74.9	57.2	46.9	76.8

Feature	CML_{c}	CML_{t}	AML_{c}	AML_{t}	D	D_{g}	Mean8
F0	**27.5**	59.1	35.7	**77.9**	1.6465	0.6881	55.6136
F1	26.4	**59.2**	33.5	**77.9**	1.5836	0.6924	**55.8694**
F2	24.4	57.6	30.0	72.1	1.4587	0.6246	52.8432
F3	22.3	57.5	27.4	72.0	1.4123	0.6126	52.0153
F4	20.4	49.7	27.3	65.7	1.2670	0.4796	46.6711
F5	15.0	41.7	18.2	52.7	0.9051	0.3234	38.6383
F6	27.2	58.4	**36.0**	77.0	**1.6625**	**0.6972**	55.3927
F7	2.4	15.8	3.2	19.7	0.2156	0.0394	17.0885
F8	2.6	16.8	3.7	22.0	0.2359	0.0501	18.1258
All	33.0	61.7	41.7	79.1	1.7311	0.7337	58.9035

9.3.2 Combining Features with Beaker vs. MMA

Section 9.3.1 is not the first work to demonstrate the power of combining multiple features for beat tracking. As discussed in Section 5.3, Zapata et al. tested a committee of multiple beat trackers where each tracker used a different input feature [68]. However, they were working with non-causal systems and used a very different approach to combining the trackers. In fact, they were not truly combining tracker outputs but rather were using MMA to determine which tracker's output to use for each song after each tracker's output beat sequence had been generated. This section compares the results of their approach to combining features vs. Beaker's ensemble approach using the Beatles dataset described in Section 8.2.1 and shows that Beaker's ensemble approach gives superior results. Six different algorithms, including Beaker ensembles of TestTrackers, MMA among TestTrackers, MMA among Beaker ensembles, and combinations of these approaches were used in this evaluation. The non-Beaker algorithms were implemented in MATLAB based on the descriptions given in [68].

All of the tests described here begin with the TestTrackers described in Chapter 7. TestTracker parameters varied across 9 possible features, 3 periodicity approaches, 4 tempo ranges, and 2 time ranges, for a total of 216 TestTrackers. The first algorithm tested was *Beaker* where a single Beaker instance was used with all 216 TestTrackers as members of Beaker's ensemble. A diagram of such a system was shown in Figure 9.2.

Next, 216 distinct Beaker instances were created, each representing a single TestTracker. As shown in Figure 9.3, in the *Flat MMA* algorithm, each of the Beaker instances generated an output beat sequence, and MMA was used to determine the optimal sequence for each song in the database. To do this, the information gain score D (mutual agreement) for each sequence compared to each of the other output sequences was computed on a per-song basis. Information gain was chosen out of all of the possible evaluation metrics because of its use in the selective sampling and MMA work of [42] and [68]. The mean of all mutual agreement scores containing a particular output sequence was computed, and the output sequence with the maximum mean

mutual agreement was chosen as the final output sequence for that song. The *Flat Per-Song Max* algorithm behaved similarly, except that the known annotations were used to determine the optimal output sequence for each song. *Flat Per-Song Max* therefore represents the best-case scenario for the *Flat MMA* algorithm when using the information gain measure D as the evaluation metric.

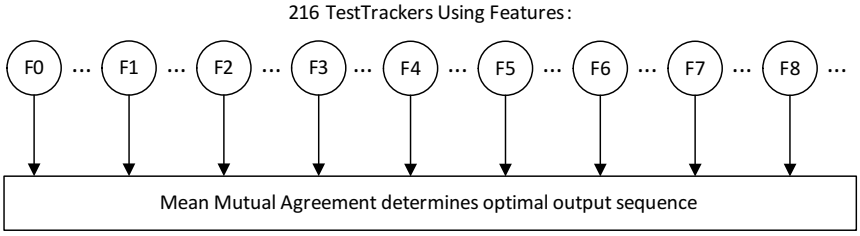

Figure 9.3: Flat MMA

The results for these three approaches are shown in Table 9.4. When combining 216 TestTrackers using *Beaker* vs. *Flat MMA*, Beaker significantly out-performs the MMA approach on all metrics and performs comparably to *Flat Per-Song Max*, the best-case scenario for the MMA approach.

Table 9.4: Results for Beaker vs. flat MMA (Beatles dataset)

Algorithm	F-measure	Cem_{acc}	Goto_{acc}	P-score
Beaker	74.9	57.2	46.9	76.8
Flat MMA	66.0	50.7	39.1	71.0
Flat Per-Song Max	73.9	56.1	48.6	77.7

Algorithm	CML_{c}	CML_{t}	AML_{c}	AML_{t}	D	D_{g}	Mean8
Beaker	33.0	61.7	41.7	79.1	1.7311	0.7337	58.9035
Flat MMA	28.6	54.8	40.3	75.4	1.6345	0.5875	53.2221
Flat Per-Song Max	31.4	65.3	38.6	81.0	1.9092	0.7034	59.0648

Next, additional experiments were performed using a smaller number of trackers as input to the MMA algorithm. These experiments were designed to more closely resemble the tests done in [68], which computed the MMA of committees containing at most 6 trackers, each using a separate input feature, and concluded that feature

diversity gave improved results over the use of a beat tracker using a single feature. These experiments were needed in order to demonstrate that combining features using Beaker rather than MMA gives better results even with smaller ensembles and not only with the larger-sized ensembles/committees used in the tests just described.

For comparison with Beaker in this mode, a *2-Layer Beaker* algorithm was used by treating Beaker trackers as members of an ensemble for a parent Beaker tracker. In this approach, results were generated by first creating 9 separate Beaker ensembles, each using a single feature. Other parameters were identical to those used in *Beaker* (3 periodicity approaches, 4 tempo ranges, and 2 time ranges resulted in 24 trackers per feature). As shown in Figure 9.4, these 9 Beaker instances were then added as ensemble members to a parent Beaker instance (all other parameters were the same in both layers) creating an ensemble where each ensemble member used a single feature. The output of the parent Beaker ensemble was used.

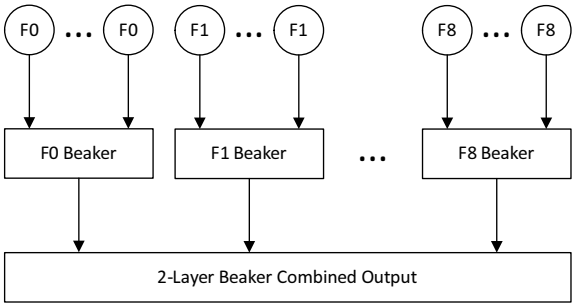

Figure 9.4: 2-layer Beaker ensemble

Results for *Beaker + MMA* were generated in the same way as those for *Flat MMA*, except that, as shown in Figure 9.5, only the 9 Beaker instances used as input to the second Beaker layer in *2-Layer Beaker* were used as committee members to generate output sequences for the MMA algorithm rather than the committee of 216 trackers tested for *Flat MMA*. Similarly, results for *Beaker + Per-Song Max* were generated as in *Flat Per-Song Max*, using the same 9 Beaker instances to generate output sequences. The *Beaker + Per-Song Max* algorithm is therefore the best-case scenario for the *Beaker + MMA* algorithm, based on the use of the information gain D as the evaluation metric.

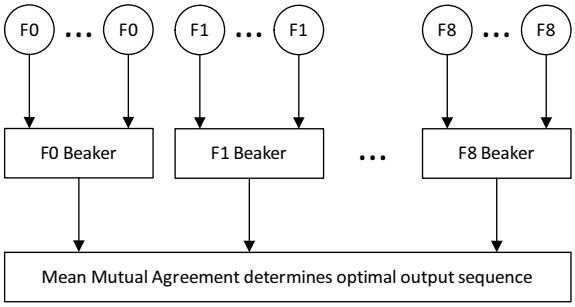

Figure 9.5: Beaker + MMA

Table 9.5: Results for 2-layer Beaker vs. MMA (Beatles dataset)

Algorithm	F-measure	Cem_{acc}	Goto_{acc}	P-score
2-Layer Beaker	75.4	57.5	43.6	77.1
Beaker + MMA	74.8	57.1	40.2	76.8
Beaker + Per-Song Max	75.1	57.1	44.1	77.4

Algorithm	CML_{c}	CML_{t}	AML_{c}	AML_{t}	D	D_{g}	Mean8
2-Layer Beaker	31.0	60.5	39.6	79.1	1.7045	0.7381	57.9698
Beaker + MMA	28.3	59.9	37.0	78.3	1.6684	0.7113	56.5440
Beaker + Per-Song Max	29.4	61.9	37.5	80.2	1.7710	0.7367	57.8268

The results of these experiments are shown in Table 9.5. When combining nine single-feature trackers, *2-Layer Beaker* is able to out-perform *Beaker + MMA* on all metrics, demonstrating again that Beaker's ensemble approach performs better than MMA, although the improvement is less significant than that seen in Table 9.4 for the larger ensembles. On several metrics, *2-Layer Beaker* even outperforms the best-case scenario for *Beaker + Per-Song Max*. It is interesting to note that, as in Section 9.3.1, the superior performance from Beaker is most noticeable in the continuity-based metrics, reinforcing the idea that Beaker's ensemble method is able to compensate for discontinuities in individual tracker output.

In conclusion, for both the large and small ensemble scenarios, Beaker's approach outperforms the MMA approach in quality. It can also be significantly faster when using large ensembles. For example, comparing the output beat sequences of every possible pair out of 216 TestTrackers can be very computationally expensive. Another disadvantage of the MMA approach is that it cannot be used in a causal fashion. This provides Beaker with an additional advantage even when using small ensembles, where the improvement in scores using a Beaker ensemble vs. MMA may be less significant.

9.3.3 Beaker vs. MMA Varying Other Parameters

While Section 9.3.2 compared Beaker's performance to the MMA approach when creating ensembles using multiple features, this section will show that for ensembles of TestTrackers where each ensemble uses a single feature and ensemble diversity is achieved by varying other parameters, combining TestTrackers using Beaker's approach results in significantly better performance than combining TestTrackers using MMA. To do this, for each of the nine possible features F_0 through F_8, an ensemble of 24 TestTrackers was created. Each TestTracker in the ensemble used the same feature but varied other parameters (3 periodicity approaches, 4 tempo ranges, and 2 time ranges). The ensemble's results were combined in a causal fashion by Beaker. This was compared with taking the 24 beat sequences output from the TestTrackers and combining them using MMA. The process was repeated

Table 9.6: Beaker vs. MMA for single-feature ensembles (Beatles dataset)

Algorithm	F-measure	Cem_{acc}	Goto_{acc}	P-score
F0 Beaker	74.0	56.3	38.5	76.0
F0 MMA	67.1	51.0	39.1	71.9
F1 Beaker	74.6	55.7	43.6	76.1
F1 MMA	69.4	52.5	38.5	72.0
F2 Beaker	71.4	54.5	38.0	74.9
F2 MMA	67.0	51.0	35.8	71.7
F3 Beaker	71.8	53.9	36.3	74.8
F3 MMA	69.1	52.3	33.5	72.5
F4 Beaker	65.1	49.3	27.4	68.5
F4 MMA	59.6	44.7	26.8	64.5
F5 Beaker	56.8	43.1	19.0	62.7
F5 MMA	52.0	39.0	14.5	59.2
F6 Beaker	73.6	56.4	38.5	76.1
F6 MMA	65.7	50.0	35.2	71.4
F7 Beaker	32.3	23.5	0.0	39.9
F7 MMA	13.9	10.1	0.6	14.7
F8 Beaker	33.9	24.8	0.0	41.2
F8 MMA	14.6	10.7	0.0	15.1

Continued on next page

Table 9.6: Beaker vs. MMA for single-feature ensembles (Beatles dataset), Continued

Algorithm	CML_c	CML_t	AML_c	AML_t	D	D_g	Mean8
F0 Beaker	27.5	59.1	35.7	77.9	1.6465	0.6881	55.6136
F0 MMA	28.6	56.3	35.9	74.8	1.5628	0.5481	52.8217
F1 Beaker	26.4	59.2	33.5	77.9	1.5836	0.6924	55.8694
F1 MMA	24.8	54.7	35.2	76.8	1.5102	0.5749	52.9873
F2 Beaker	24.4	57.6	30.0	72.1	1.4587	0.6246	52.8432
F2 MMA	22.3	53.9	30.7	73.4	1.4173	0.5371	50.7106
F3 Beaker	22.3	57.5	27.4	72.0	1.4123	0.6126	52.0153
F3 MMA	20.4	54.5	28.9	74.4	1.3837	0.5529	50.6875
F4 Beaker	20.4	49.7	27.3	65.7	1.2670	0.4796	46.6711
F4 MMA	18.8	45.9	27.8	65.6	1.2225	0.3690	44.2146
F5 Beaker	15.0	41.7	18.2	52.7	0.9051	0.3234	38.6383
F5 MMA	13.1	39.1	18.3	55.0	0.9520	0.2490	36.2756
F6 Beaker	27.2	58.4	36.0	77.0	1.6625	0.6972	55.3927
F6 MMA	25.8	54.8	35.6	74.2	1.5918	0.5464	51.5910
F7 Beaker	2.4	15.8	3.2	19.7	0.2156	0.0394	17.0885
F7 MMA	1.3	3.8	2.1	6.1	0.1661	0.0124	6.5848
F8 Beaker	2.6	16.8	3.7	22.0	0.2359	0.0501	18.1258
F8 MMA	1.2	4.0	2.2	6.5	0.1695	0.0128	6.7987

for each of the nine features.

The results shown in Table 9.6 for the Beatles dataset show that for all features, Beaker ensembles using that constant feature but varying other parameters perform better than using MMA to combine results when using the Mean8 evaluation metric, and that the same holds true for nearly all individual metrics as well. The improvement in Beaker results over MMA are the most dramatic for features F_7 and F_8. These are the two phase-based onset detection functions discussed in Section 7.2.4, and are generally less reliable features when used alone compared to their magnitude or complex domain counterparts, producing lower scores. No single TestTracker's output sequence is particularly reliable using these features, and therefore the MMA approach, which can at best perform as well as the best TestTracker, performs poorly. In contrast, the Beaker approach, which is capable of outperforming all of the individual TestTrackers in its ensemble, is able to perform significantly better than MMA. These results strengthen the conclusion of Section 9.3.2 that Beaker's approach to combining the output of multiple beat trackers is superior to the MMA approach.

9.4 Conclusion

This chapter compared the performance of Beaker ensembles of TestTrackers with other approaches to beat tracking and to using ensemble learning methods with beat tracking. Beaker's performance on the Beatles dataset was shown to outperform certain non-causal beat trackers on some evaluation metrics, and its performance on the SMC dataset was shown to be comparable to that of the state-of-the-art IBT-C causal beat tracker. The merits of creating ensembles using multiple input features were then demonstrated, and Beaker's approach to combining ensembles using TestTrackers with various input features was shown to give better results than MMA, which was previous work in this area using non-causal beat trackers. Compared to MMA and individual features, Beaker's ensemble approach showed superior performance most noticeably in continuity-based metrics, suggesting that the Beaker ensemble framework is able to smooth over discontinuities in individual trackers

- something that a committee-based or mixture-of-experts ensemble approach like MMA cannot do, since its output is always restricted to being the output of one of the committee member trackers.

Chapter 10

Ensemble Diversity

10.1 Introduction

The performance of a Beaker ensemble depends heavily on the composition of its ensemble of trackers. Chapter 9 demonstrated the value of using ensembles of beat trackers using different input features. However, while it is often the case that larger ensembles produce better results, in ensemble learning bigger is not always better. Instead, it is the diversity of the ensemble rather than the size which determines the quality of the ensemble. When dealing with ensembles of classifiers, there is no one definition or measure of ensemble diversity [48], but according to Polikar [59] and Dietterich [20], diversity is achieved when classifiers make different errors on various inputs. Extending this concept to an ensemble of beat trackers, a tracker should add diversity to a Beaker ensemble, thereby improving results, if it performs well when other trackers in the ensemble perform poorly. This chapter will explore the concept of ensemble diversity as it relates to ensembles of trackers with varying parameters, including different features as in Chapter 9 as well as different periodicity estimation methods. It will discuss when and why certain ensembles perform better than others. Finally, a preliminary approach to predicting when the merging of two ensembles containing trackers with different parameters will produce a single stronger ensemble will be presented. All experiments in this chapter were performed using the Beatles dataset with varying-sized ensembles of the TestTrackers described in

Chapter 7 initialized with different parameters.

10.2 Feature Diversity

This section will discuss the results of experiments which created ensembles consisting of TestTrackers using various combinations of two of the nine possible features discussed in Section 7.2. While Section 9.3.1 showed that an ensemble consisting of all nine features always out-performed the corresponding single-feature ensembles, the experiment presented here is intended to determine if the same holds true for ensembles using two features. In other words, does combining two features always improve the performance of an ensemble? If not, when and why does adding trackers to an ensemble improve performance, and can this behavior be predicted?

First, nine single-feature ensembles were created as in the experiment in Section 9.3.1. The performance of the single-feature ensembles using the Mean8 evaluation metric is shown in Table 10.1 (summarized from Table 9.3).

Table 10.1: Performance of single-feature ensembles (Beatles dataset)

Feature	Mean8
F0	55.6136
F1	55.8694
F2	52.8432
F3	52.0153
F4	46.6711
F5	38.6383
F6	55.3927
F7	17.0885
F8	18.1258

Next, 36 ensembles consisting of all possible combinations of two features were created, merging the ensembles of two of the single-feature ensembles *Ensemble 1* (E_1) and *Ensemble 2* (E_2). For example, one ensemble used features F_0 and F_1, another F_0 and F_2 etc. Table 10.2 shows the results for each of the 36 ensembles

compared with the results of the single-feature ensembles that were merged to create each two-feature ensemble.

Based on these results, we can answer the following questions:

1. If I add E_2 to E_1, does this increase the Mean8 score for E_1?

2. If I add E_1 to E_2, does this increase the Mean8 score of E_2?

It is important to note that these scenarios are not symmetrical. If the combined ensemble ($E_1 \cup E_2$) scores greater than E_1 but less than E_2, then the answer to question 1 will be yes, but the answer to question 2 will be no. From Table 10.2, we can see that the answer to question 1 is yes 22 times out of 36, and the answer to question 2 is yes 34 times out of 36. Although the answers to these questions are yes the majority of the time, this is not true for every combination, meaning that a larger ensemble is not always better. What then causes certain combinations of features to not produce improved results? We know that ensemble diversity is a significant factor in the performance of an ensemble, so in the cases where combining features does not improve results, perhaps adding one ensemble to the other does not increase ensemble diversity.

10.3 Predicting Improvement in Combined Ensemble Performance

We want to be able to predict when a combined ensemble will improve results, so we would like to have a measure of ensemble diversity. Unfortunately, no single measure of ensemble diversity for ensembles of classifiers exists. Instead, there are many such measures, and none is ideal for all circumstances [48]. To further complicate things, while a Beaker ensemble of trackers acts like a classifier at each time instant, the results it produces for evaluation (a series of beat times) are not directly the results of the classification. Therefore, measures of diversity in classifiers based on the output of those classifiers cannot be directly applied to the output of a Beaker

Table 10.2: Performance of two-feature ensembles (Beatles dataset)

E_1 Feature	Mean8	E_2 Feature	Mean8	Combined Mean8	Improves E_1	Improves E_2
F0	55.6136	F1	55.8694	57.6623	yes	yes
F0	55.6136	F2	52.8432	56.9987	yes	yes
F0	55.6136	F3	52.0153	56.8787	yes	yes
F0	55.6136	F4	46.6711	54.1770	no	yes
F0	55.6136	F5	38.6383	51.5134	no	yes
F0	55.6136	F6	55.3927	56.5746	yes	yes
F0	55.6136	F7	17.0885	55.2395	no	yes
F0	55.6136	F8	18.1258	55.5003	no	yes
F1	55.8694	F2	52.8432	57.4427	yes	yes
F1	55.8694	F3	52.0153	56.5727	yes	yes
F1	55.8694	F4	46.6711	54.9715	no	yes
F1	55.8694	F5	38.6383	51.8412	no	yes
F1	55.8694	F6	55.3927	57.2992	yes	yes
F1	55.8694	F7	17.0885	55.5795	no	yes
F1	55.8694	F8	18.1258	55.3070	no	yes
F2	52.8432	F3	52.0153	54.0608	yes	yes
F2	52.8432	F4	46.6711	55.1303	yes	yes
F2	52.8432	F5	38.6383	52.1407	no	yes
F2	52.8432	F6	55.3927	57.3548	yes	yes
F2	52.8432	F7	17.0885	53.3604	yes	yes
F2	52.8432	F8	18.1258	53.3557	yes	yes
F3	52.0153	F4	46.6711	54.5636	yes	yes
F3	52.0153	F5	38.6383	50.9496	no	yes
F3	52.0153	F6	55.3927	56.6768	yes	yes
F3	52.0153	F7	17.0885	52.5846	yes	yes
F3	52.0153	F8	18.1258	52.2773	yes	yes
F4	46.6711	F5	38.6383	46.2813	no	yes
F4	46.6711	F6	55.3927	53.7816	yes	no
F4	46.6711	F7	17.0885	46.5345	no	yes
F4	46.6711	F8	18.1258	47.0162	yes	yes
F5	38.6383	F6	55.3927	50.8282	yes	no
F5	38.6383	F7	17.0885	39.2454	yes	yes
F5	38.6383	F8	18.1258	39.2274	yes	yes
F6	55.3927	F7	17.0885	54.6880	no	yes
F6	55.3927	F8	18.1258	54.7023	no	yes
F7	17.0885	F8	18.1258	17.9617	yes	no

ensemble. As a result, a new measure of ensemble diversity specific to ensembles of beat trackers is required.

This section presents a preliminary attempt to create a metric which will use ensemble diversity to predict when the combination of ensembles will produce improved results. The metric is based on the idea that adding E_1 to E_2 will improve the results of E_2 if E_1 performs well when E_2 performs poorly. Because results for the various ensembles are available on a per-song basis within a dataset, the metric acts on a per-song level and not within a song. That means that it cannot take into consideration cases where E_1 performs well on half of a song, while E_2 performs well on the other half, or other such scenarios, but such cases are an area for future work.

To predict whether merging E_1 with E_2 will improve the score of E_2, we first look for songs in a dataset where E_1 out-performs E_2. We want E_1 to perform better than E_2 on songs where E_2 is performing poorly, because if E_2 is already performing well on those songs, adding E_1 is not as necessary. First we will define the set S of m songs in a dataset as $S = \{s_1, s_2, \ldots, s_m\}$, the set C_1 as the per-song scores for E_1: $C_1 = \{c_{11}, c_{12}, \ldots, c_{1m}\}$, and the set C_2 as the per-song scores for E_2: $C_2 = \{c_{21}, c_{22}, \ldots, c_{2m}\}$. We define an ensemble as having poor performance on a song if the score for that song is below the median score for that ensemble across all songs (if the song scores worse than half of all songs). For example, E_2 performs poorly on song s_k if $c_{2k} < \text{median}(C_2)$. Next, we define Y, the set of songs where E_1 performs better than E_2: $Y = \{s_k | c_{1k} > c_{2k}\}$. Then we look at the median score for E_2 on the songs in Y. If that score is lower than the median score for E_2 across all songs in the dataset, then we can say that E_1 performs better than E_2 when E_2 performs poorly.

This criteria alone is not sufficient to predict when adding E_1 to E_2 will improve the score of E_2 however. It is also necessary to determine if the improvement on the songs in Y is significant enough. We can look at the mean of the improvement in scores across songs in Y. If that mean is above a certain threshold ρ_1, we consider the improvement significant. Experimentally, it was also found that even when the median criteria described above was not met (E_1 performed better than E_2 when E_2 was already performing well), a significant enough mean per-song improvement

(greater than a threshold ρ_2) was an indicator that adding E_1 to E_2 would still improve the results of E_2.

We can then define the rules for predicting ensemble improvement as follows. Adding E_1 to E_2 will improve the score of E_2 if either criteria 1 or criteria 2 is met:

1. the median score for E_2 on the set Y is below the median of C_2, and the mean improvement in scores across the set Y is greater than ρ_1, or

2. the mean improvement in scores across the set Y is greater than ρ_2

Initial experiments show that threshold values of $\rho_1 = 6$ and $\rho_2 = 25$ give good results for the Beatles dataset.

Returning to the results presented in Section 10.2, we can check if these rules allow us to successfully predict when certain combinations of ensembles will give improved results. There are 72 possible scenarios for adding E_1 to E_2, and as shown in Table 10.3, these rules correctly predict 57 out of 72 times whether improvement will occur. On the surface, this does not appear to be a high success rate. However, in all but one of the cases where the rules produce an incorrect prediction, the actual improvement (or lack thereof) is small enough to be ambiguous. The one exception is adding the ensemble using feature F_5 (E_1) to the ensemble using feature F_6 (E_2). In this case, as shown in Table 10.3, the score of E_2 decreases by more than 4 percentage points, but the rules predict that the score should have improved. It is likely that this is a case where a finer-grained evaluation, which could compare ensemble performance on segments of a song and not simply the entire song, could give a better prediction, so this is an area for future work.

Table 10.3: Predicting performance of two-feature ensembles (Beatles dataset, items in bold indicate disagreement between predicted and actual performance)

E_1 Feature	Mean8	E_2 Feature	Mean8	Combined Mean8	Improves E_2	Prediction
F1	55.8694	F0	55.6136	57.6623	yes	yes
F2	52.8432	F0	55.6136	56.9987	yes	yes
F3	52.0153	F0	55.6136	56.8787	yes	yes
F4	46.6711	F0	55.6136	54.1770	no	no
F5	38.6383	F0	55.6136	51.5134	no	no
F6	55.3927	F0	55.6136	56.5746	**yes**	**no**
F7	17.0885	F0	55.6136	55.2395	no	no
F8	18.1258	F0	55.6136	55.5003	no	no
F0	55.6136	F1	55.8694	57.6623	**yes**	**no**
F2	52.8432	F1	55.8694	57.4427	yes	yes
F3	52.0153	F1	55.8694	56.5727	yes	yes
F4	46.6711	F1	55.8694	54.9715	**no**	**yes**
F5	38.6383	F1	55.8694	51.8412	no	no
F6	55.3927	F1	55.8694	57.2992	yes	yes
F7	17.0885	F1	55.8694	55.5795	no	no
F8	18.1258	F1	55.8694	55.3070	no	no
F0	55.6136	F2	52.8432	56.9987	yes	yes
F1	55.8694	F2	52.8432	57.4427	yes	yes
F3	52.0153	F2	52.8432	54.0608	**yes**	**no**
F4	46.6711	F2	52.8432	55.1303	yes	yes
F5	38.6383	F2	52.8432	52.1407	no	no
F6	55.3927	F2	52.8432	57.3548	yes	yes
F7	17.0885	F2	52.8432	53.3604	**yes**	**no**
F8	18.1258	F2	52.8432	53.3557	**yes**	**no**

Continued on next page

Table 10.3: Predicting performance of two-feature ensembles (Beatles dataset, items in bold indicate disagreement between predicted and actual performance), Continued

E_1 Feature	Mean8	E_2 Feature	Mean8	Combined Mean8	Improves E_2	Prediction
F0	55.6136	F3	52.0153	56.8787	yes	yes
F1	55.8694	F3	52.0153	56.5727	yes	yes
F2	52.8432	F3	52.0153	54.0608	**yes**	**no**
F4	46.6711	F3	52.0153	54.5636	yes	yes
F5	38.6383	F3	52.0153	50.9496	no	no
F6	55.3927	F3	52.0153	56.6768	yes	yes
F7	17.0885	F3	52.0153	52.5846	**yes**	**no**
F8	18.1258	F3	52.0153	52.2773	**yes**	**no**
F0	55.6136	F4	46.6711	54.1770	yes	yes
F1	55.8694	F4	46.6711	54.9715	yes	yes
F2	52.8432	F4	46.6711	55.1303	yes	yes
F3	52.0153	F4	46.6711	54.5636	yes	yes
F5	38.6383	F4	46.6711	46.2813	**no**	**yes**
F6	55.3927	F4	46.6711	53.7816	yes	yes
F7	17.0885	F4	46.6711	46.5345	no	no
F8	18.1258	F4	46.6711	47.0162	**yes**	**no**
F0	55.6136	F5	38.6383	51.5134	yes	yes
F1	55.8694	F5	38.6383	51.8412	yes	yes
F2	52.8432	F5	38.6383	52.1407	yes	yes
F3	52.0153	F5	38.6383	50.9496	yes	yes
F4	46.6711	F5	38.6383	46.2813	yes	yes
F6	55.3927	F5	38.6383	50.8282	yes	yes
F7	17.0885	F5	38.6383	39.2454	**yes**	**no**
F8	18.1258	F5	38.6383	39.2274	yes	yes

Continued on next page

Table 10.3: Predicting performance of two-feature ensembles (Beatles dataset, items in bold indicate disagreement between predicted and actual performance), Continued

E_1 Feature	Mean8	E_2 Feature	Mean8	Combined Mean8	Improves E_2	Prediction
F0	55.6136	F6	55.3927	56.5746	**yes**	**no**
F1	55.8694	F6	55.3927	57.2992	yes	yes
F2	52.8432	F6	55.3927	57.3548	yes	yes
F3	52.0153	F6	55.3927	56.6768	yes	yes
F4	46.6711	F6	55.3927	53.7816	no	no
F5	38.6383	F6	55.3927	50.8282	**no**	**yes**
F7	17.0885	F6	55.3927	54.6880	no	no
F8	18.1258	F6	55.3927	54.7023	no	no
F0	55.6136	F7	17.0885	55.2395	yes	yes
F1	55.8694	F7	17.0885	55.5795	yes	yes
F2	52.8432	F7	17.0885	53.3604	yes	yes
F3	52.0153	F7	17.0885	52.5846	yes	yes
F4	46.6711	F7	17.0885	46.5345	yes	yes
F5	38.6383	F7	17.0885	39.2454	yes	yes
F6	55.3927	F7	17.0885	54.6880	yes	yes
F8	18.1258	F7	17.0885	17.9617	**yes**	**no**
F0	55.6136	F8	18.1258	55.5003	yes	yes
F1	55.8694	F8	18.1258	55.3070	yes	yes
F2	52.8432	F8	18.1258	53.3557	yes	yes
F3	52.0153	F8	18.1258	52.2773	yes	yes
F4	46.6711	F8	18.1258	47.0162	yes	yes
F5	38.6383	F8	18.1258	39.2274	yes	yes
F6	55.3927	F8	18.1258	54.7023	yes	yes
F7	17.0885	F8	18.1258	17.9617	no	no

10.4 Combining Periodicity Functions

While multiple features can increase ensemble diversity and therefore improve performance, Table 9.3 showed that several of the individual features available to TestTrackers could perform reasonably well by themselves. This suggests that it

would be useful to look at other ways of creating diversity in Beaker ensembles. One such way in which diversity in a Beaker ensemble of TestTrackers can be achieved is by using multiple different periodicity functions to perform tempo estimation. This section examines the results for ensembles using the possible combinations of the four periodicity estimation approaches available to TestTrackers as listed in Table 7.2. Again, all experiments were performed using the Beatles dataset.

The results for ensembles using various combinations of periodicity estimation approaches are shown in Table 10.4. The interesting thing to notice about this data is that the P_0 periodicity estimation function (Biased autocorrelation function (ACF)), while outperforming P_2 and P_3 in ensembles using only one periodicity estimation approach and increasing the scores of those ensembles when added to them, decreases the scores of ensembles using approach P_1 (Unbiased ACF). This occurs because P_0 and P_1 are very similar approaches and therefore tend to perform well or poorly on the same songs. Therefore, adding P_0 to an ensemble that already uses P_1 does not always increase ensemble diversity, and because TestTrackers using P_1 tend to perform better than those using P_0, adding P_0 to such an ensemble can actually decrease performance. As shown in Table 10.5, the method presented in Section 10.3 for predicting when a combination of ensembles will improve results correctly predicts the lack of improvement in three of these scenarios including adding P_0 to an ensemble already using P_1, to an ensemble already using P_1 and P_2, and to an ensemble already using P_1, P_2, and P_3. Interestingly, adding P_0 to an ensemble already containing P_1 does not always decrease results. For example, adding P_0 to an ensemble using P_1 and P_3 slightly increases results, and the method from Section 10.3 correctly predicts this behavior as well.

10.5 Conclusion

This chapter has discussed the concept of ensemble diversity in the context of the Beaker beat tracking ensemble framework and emphasized that a larger ensemble is not always better - diversity is more important than size. Additionally, prelimi-

Table 10.4: Results for periodicity approach combinations (Beatles dataset)

Periodicity Function(s)	F-measure	Cem_{acc}	Goto_{acc}	P-score
P0	66.4	49.4	32.4	70.7
P1	68.1	51.1	38.5	72.7
P2	62.6	47.2	21.2	62.5
P3	62.2	47.1	30.2	67.0
P0, P1	68.5	51.2	40.8	72.9
P0, P2	72.8	54.7	34.1	72.8
P0, P3	69.5	53.0	40.2	73.7
P1, P2	75.3	56.5	40.8	76.5
P1, P3	70.2	53.7	43.0	74.7
P2, P3	70.8	54.2	35.2	72.9
P0, P1, P2	74.2	55.6	43.0	75.6
P0, P1, P3	70.9	53.9	43.6	74.9
P0, P2, P3	72.9	55.8	40.8	74.6
P1, P2, P3	74.9	57.2	46.9	76.8
P0, P1, P2, P3	74.4	56.6	44.7	76.2

Periodicity Function(s)	CML_{c}	CML_{t}	AML_{c}	AML_{t}	D	D_{g}	Mean8
P0	22.3	53.6	26.1	65.6	1.2416	0.5005	48.3102
P1	26.2	58.0	31.3	71.0	1.3837	0.5303	52.1049
P2	17.0	43.0	26.1	61.9	1.2408	0.4667	42.6972
P3	20.6	48.7	23.8	59.9	1.1810	0.4436	44.9636
P0, P1	25.5	57.3	29.9	70.0	1.3535	0.5215	51.9952
P0, P2	24.7	55.2	33.0	73.2	1.5850	0.6656	52.5612
P0, P3	27.9	56.5	32.8	71.1	1.4485	0.5959	53.0855
P1, P2	28.7	60.7	35.9	77.4	1.7091	0.7233	56.4925
P1, P3	29.2	59.5	35.1	74.1	1.5497	0.6323	54.9512
P2, P3	25.3	55.2	33.9	72.9	1.5353	0.6292	52.5494
P0, P1, P2	28.9	60.1	35.6	76.3	1.6577	0.6892	56.1689
P0, P1, P3	29.2	59.9	34.6	74.5	1.5174	0.6211	55.1859
P0, P2, P3	29.6	57.7	38.2	75.5	1.6277	0.6770	55.6242
P1, P2, P3	33.0	61.7	41.7	79.1	1.7311	0.7337	58.9035
P0, P1, P2, P3	30.5	60.8	37.7	78.2	1.6938	0.7203	57.3865

Table 10.5: Predicting performance combining multiple periodicity approaches (Beatles dataset)

E_1 Approach	Mean8	E_2 Approach	Mean8	Combined Mean8	Improves E_2	Prediction
P1	52.1049	P0	48.3102	51.9952	yes	yes
P2	42.6972	P0	48.3102	52.5612	yes	yes
P3	44.9636	P0	48.3102	53.0855	yes	yes
P0	48.3102	P1	52.1049	51.9952	no	no
P2	42.6972	P1	52.1049	56.4925	yes	yes
P3	44.9636	P1	52.1049	54.9512	yes	yes
P0	48.3102	P2	42.6972	52.5612	yes	yes
P1	52.1049	P2	42.6972	56.4925	yes	yes
P3	44.9636	P2	42.6972	52.5494	yes	yes
P0	48.3102	P3	44.9636	53.0855	yes	yes
P1	52.1049	P3	44.9636	54.9512	yes	yes
P2	42.6972	P3	44.9636	52.5494	yes	yes
P0	48.3102	P1, P2	56.4925	56.1689	no	no
P1	52.1049	P0, P2	52.5612	56.1689	yes	yes
P2	42.6972	P0, P1	51.9952	56.1689	yes	yes
P0	48.3102	P1, P3	54.9512	55.1859	yes	yes
P1	52.1049	P0, P3	53.0855	55.1859	yes	yes
P3	44.9636	P0, P1	51.9952	55.1859	yes	yes
P0	48.3102	P2, P3	52.5494	55.6242	yes	yes
P2	42.6972	P0, P3	53.0855	55.6242	yes	yes
P3	44.9636	P0, P2	52.5612	55.6242	yes	yes
P1	52.1049	P2, P3	52.5494	58.9035	yes	yes
P2	42.6972	P1, P3	54.9512	58.9035	yes	yes
P3	44.9636	P1, P2	56.4925	58.9035	yes	yes
P0	48.3102	P1, P2, P3	58.9035	57.3865	no	no
P1	52.1049	P0, P2, P3	55.6242	57.3865	yes	yes
P2	42.6972	P0, P1, P3	55.1859	57.3865	yes	yes
P3	44.9636	P0, P1, P2	56.1689	57.3865	yes	yes

nary work was presented regarding a metric for predicting when the combination of two ensembles would increase ensemble performance, and this metric was shown to perform well when combining ensembles of TestTrackers varying input features and correctly predicted when combining ensembles of TestTrackers varying approaches to periodicity estimation would or would not produce improved results.

Part V
Conclusion

Chapter 11

Applications and Conclusion

11.1 Introduction

This chapter will discuss a real-world implementation and application of the Beaker ensemble framework and provide some conclusions and areas for future work on the framework. The Beaker ensemble framework is implemented as a single C++ class, designed as a library that can be used from a variety of higher-level applications. Possible real-time wrapper applications for Beaker include Pd or Max/MSP externals, VST plugins, and the Beaker UI to be described in Section 11.2. It can also be used in non-real-time scenarios when used as part of a Vamp plugin or a console application for offline processing. Because of the wide range of possible applications, Beaker was designed to be as flexible as possible. It has a large number of parameters that can be optimized for computational efficiency vs. beat tracking performance or for a particular dataset.

In all cases, Beaker works in a causal fashion. The application using a Beaker instance is responsible for providing consecutive blocks of samples which are then analyzed and used to update its hypotheses about the current tempo and beat phase. Updated hypotheses can then be queried by the application, and those hypotheses can be predictive. For example, when playing a wavefile in real-time, an application might provide samples to Beaker at the same time they are played, in which case there is no input latency and no need for predicting a beat in the future. However,

when processing incoming samples from an audio capture interface, there is latency between the time a user hears the samples and when the application receives them. Therefore, to provide results in sync with the user's experience, Beaker must predict future beat locations based on the latency of the capture interface.

Beaker has minimal dependencies on third-party software to simplify integration with other applications. The only dependency of a Beaker ensemble using TestTrackers is FFTW[1], which is used for discrete Fourier transform (DFT) computations in the TestTrackers. The underlying algorithm is implemented in such a way that trackers from the ensemble could be distributed across multiple threads to improve performance, but currently the audio processing all occurs on a single thread. Instead, when using TestTrackers, performance is optimized by eliminating redundant operations that would otherwise arise from trackers with overlapping parameters (i.e. the same input feature).

11.2 The Beaker UI Application

The Beaker UI is a Graphical User Interface (GUI) application wrapper around the Beaker framework. It can track beats in real-time either from a wavefile it plays or live from an audio capture interface. It also provides functionality for offline batch processing and comparison of Beaker results with annotated beat locations and tempos in test datasets. In real-time modes, the User Interface (UI) provides multiple ways of visualizing Beaker's output and internal workings. Screenshots in this section were generated using 112 TestTrackers running in real time. The number of trackers that can be supported in real time for a particular machine depends on both the complexity of the trackers (i.e. feature extraction and approaches to tempo and beat phase estimation) and the parameters chosen for the ensemble, including the type of clustering performed when combining tracker hypotheses. Beaker parameters can be loaded, modified, and saved through a convenient UI to avoid modifying text parameter files by hand. The Beaker UI is a cross-platform application (for Win-

[1] http://www.fftw.org/

dows, Mac OS X, or Linux) implemented using Qt[2] for user interface and graphical elements, supporting both Qt4 and Qt5. Additionally, it uses libsndfile[3] for wavefile reading and PortAudio[4] for audio I/O.

11.2.1 Modes of Operation

Live Input Mode

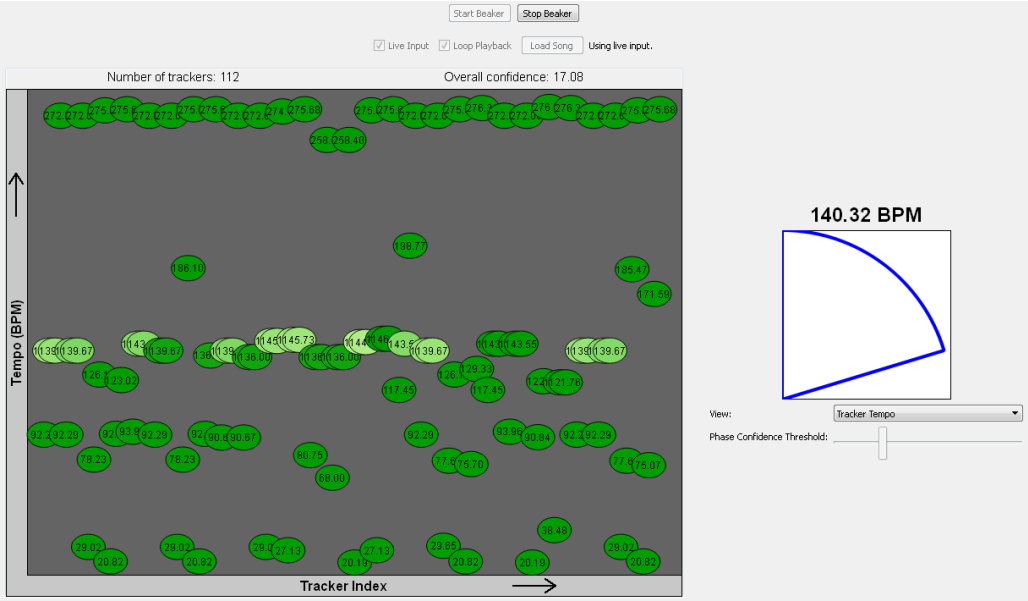

Figure 11.1: Live input mode

In live input mode, shown in Figure 11.1, the Beaker UI captures an incoming audio stream from an audio capture interface. As each block of audio samples is captured, it is given to a Beaker instance which analyzes the new samples and produces an updated tempo and beat phase hypothesis. The updated hypotheses are used to display the current tempo to the user and visualize the current beat phase. These are displayed on the right-hand side of Figure 11.1 along with visualizations of Beaker's internal behavior in the large frame on the left side of Figure 11.1. Because there is

[2]http://qt-project.org/
[3]http://www.mega-nerd.com/libsndfile/
[4]http://www.portaudio.com/

latency between when a user hears the audio and when it is analyzed by Beaker, it is necessary for Beaker to provide predictions for future tempo and beat phase values so that when the UI is updated the user sees visualizations synchronized with the audio they are hearing.

Wavefile Input Mode

Wavefile input mode appears the same as live input mode except that a wavefile is loaded and played from within the Beaker UI rather than capturing an incoming audio stream. At the same time each block of samples is passed to the sound card for playback, the samples are also sent to a Beaker instance for processing, so the visualization and analysis are synchronized (assuming similar audio and video output latencies) without requiring Beaker to predict output beyond the current block.

Batch Processing

Batch processing mode enables offline processing of files for comparison of results with annotated tempos or beat locations in datasets. Beaker is provided audio samples on a block-by-block basis from a wavefile at whatever rate it is able to process them. This rate might be faster or slower than real-time depending on the parameters chosen and the capabilities of the host machine. Once an entire wavefile has been processed, tempos and beat locations can be written to files for later analysis, and if annotation files are provided, various evaluation metrics can be computed to describe Beaker's performance on each wavefile. Using this batch mode, users can generate and save scripts that allow testing of large numbers of wavefiles with many different parameter sets for comparison and view the progress of the scripts while they run.

11.2.2 Visualization

When operating in either live or wavefile input modes, real-time visualizations of the Beaker algorithm are available. The first is always visible: a virtual foot tapping along to the music, along with the current tempo estimate. The remaining

visualizations can be selected to fill the visualization pane shown on the bottom left of Figure 11.1. These include views of the tempo and phase hypotheses generated by the ensemble of trackers and live-updating histograms of the features and periodicity estimation approaches used by the trackers with winning hypotheses. The virtual foot, tracker tempo, and tracker phase views will be described in further detail in this section.

Virtual Tapping Foot

When in real-time mode, Beaker taps its virtual foot in time with the music. For efficient drawing, the virtual foot is represented as a simple pie slice as shown in Figure 11.2. The current beat phase is used to determine the size of the pie slice. A beat phase of 0.0, indicating that a beat is occurring now, results in a pie slice spanning 90 degrees. Half-way between beats, or on the off-beat, the beat phase is 0.5, which maps to a pie slice spanning 0 degrees. Figure 11.2a shows what this view looks like half-way between the off-beat and beat, and Figure 11.2b shows the virtual foot approaching a beat.

(a) Between beat and off-beat (b) Approaching a beat

Figure 11.2: Virtual tapping foot with metronome

Tracker Tempos

The tracker tempo view helps users visualize the tempo hypotheses of the individual trackers that are combined to determine Beaker's tempo estimate. Trackers are evenly spaced horizontally, and each is represented as an ellipse with the

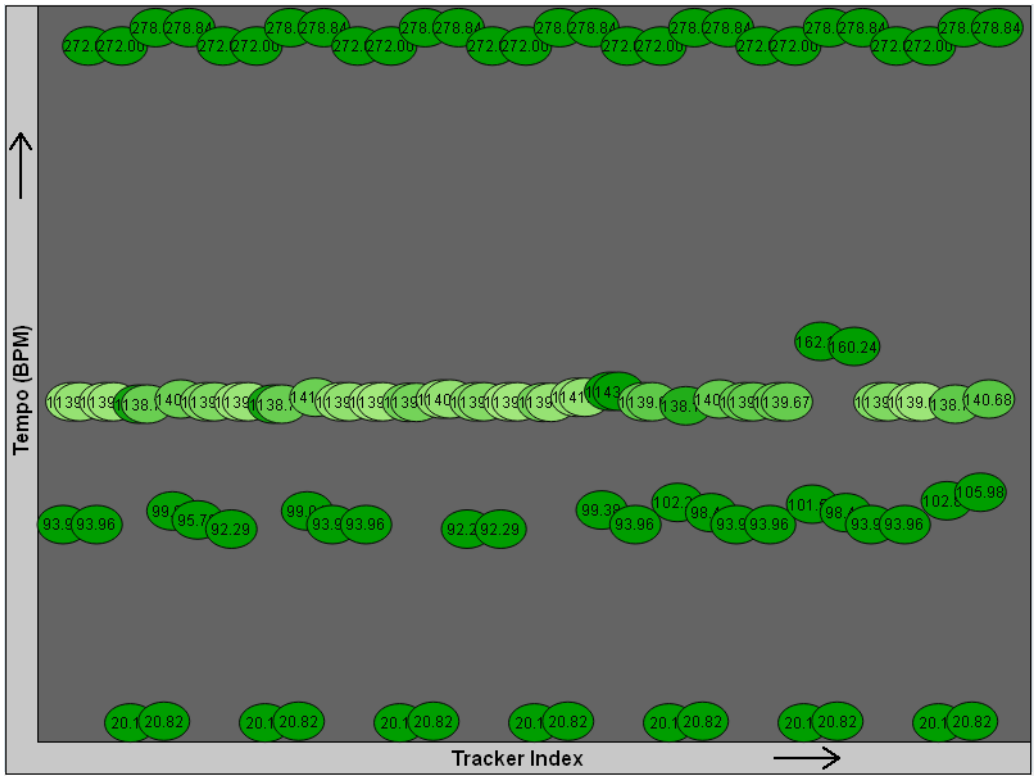

Figure 11.3: Tempo view with clear metrical structure. Tempo hypotheses are inscribed in a tracker's ellipse and also represented by the vertical position of the ellipse. Lighter-colored ellipses indicate more confident trackers.

tracker's tempo in beats per minute (BPM) written inside. The vertical position of a tracker corresponds to the tracker's tempo estimate in BPM, and the brightness of a tracker's ellipse indicates the tracker's confidence in its tempo hypothesis: lighter trackers have higher confidence than darker ones. When "listening" to a song with a strong metrical structure, most trackers will easily converge on one of the possible metrical levels of the song. Figure 11.3 is an example of the tracker tempo view for such a song. Most trackers have agreed that the tempo is around 139 BPM, with some tracking at either twice or two-thirds of that tempo. However, Figure 11.4 shows a less ideal scenario, where the metrical structure of the song is less clear. In this case, while some trackers agree on a tempo of around 239 BPM, there is little agreement between trackers about other possible metrical levels, and most trackers are darkly colored, meaning that they have low confidence in their hypotheses.

Tracker Phases

The tracker phase view shown in Figure 11.5 helps users visualize trackers' hypotheses about beat phase, or the location of the next beat. Trackers are evenly spaced vertically, and an ellipse is drawn to represent a tracker's next predicated beat time along the x-axis, where the current time is at the far left of the view frame, and the far right corresponds to the longest possible tempo period in the future. For trackers with tempo periods smaller than the maximum, additional predicted beat times are extrapolated from the next predicted beat time and ellipses are drawn for these as well. The ellipses corresponding to a single tracker are highlighted in red in Figure 11.5 to illustrate this. As in the tempo view, lighter-colored ellipses correspond to trackers with higher confidence. Because multiple ellipses are drawn for most trackers, refreshing this view is more computationally demanding than refreshing the tempo view. Therefore, the user is able to adjust a threshold which will limit drawing to only the most confident trackers if desired.

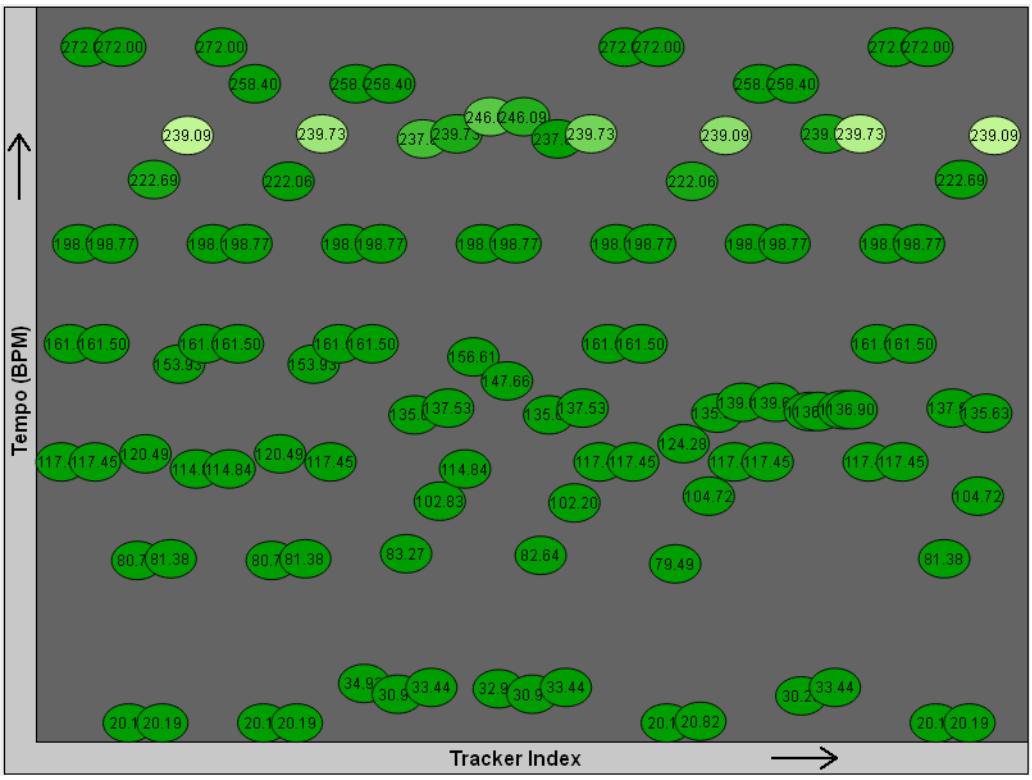

Figure 11.4: Tempo view with ambiguous metrical structure. Tempo hypotheses are inscribed in a tracker's ellipse and also represented by the vertical position of the ellipse. Lighter-colored ellipses indicate more confident trackers.

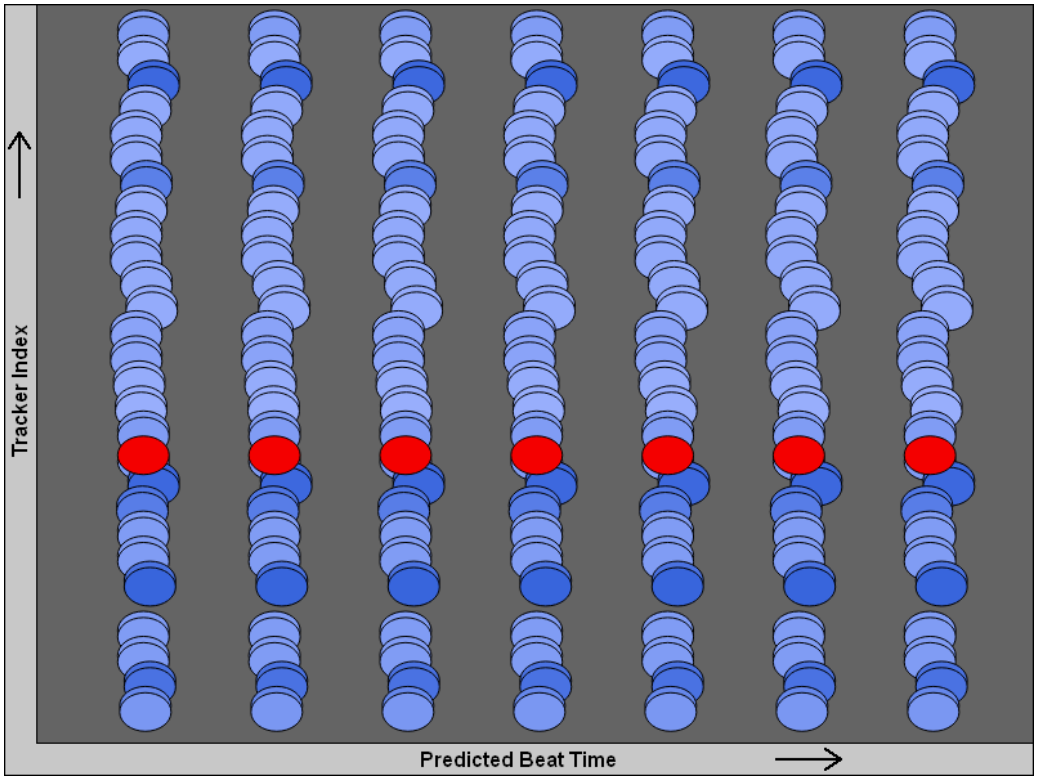

Figure 11.5: Phase view with one tracker highlighted. Each tracker is represented by a row of ellipses indicating future predicted beat times. Lighter-colored ellipses correspond to more confident trackers.

11.3 Future Work and Conclusion

This dissertation has introduced the Beaker framework for using ensembles of beat trackers in real-time causal applications along with the simple TestTracker beat trackers as example ensemble members. Beaker ensembles using TestTrackers were shown to be competitive with a state-of-the-art causal beat tracking system and outperform some non-causal beat tracking systems on certain evaluation metrics. Beaker's approach to combining the hypotheses of individual ensemble members was also shown to give superior results compared to Mean Mutual Agreement (MMA), a recent development in ensemble methods for non-causal beat tracking. The concept of ensemble diversity was discussed as it relates to Beaker ensembles, and a metric for predicting when the combination of two Beaker ensembles will give improved results over the individual ensembles was introduced. Finally, a real-time implementation of the Beaker ensemble framework in C++ and corresponding visualization was presented as a real-world application of this work.

There are many areas for future work on the task of combining ensemble learning methods with beat tracking in the Beaker framework. Some involve improvements and additional experiments related to the TestTrackers used to develop and evaluate Beaker. For example, the list of available input features for TestTrackers is currently limited to onset detection functions (ODFs). However, as discussed in Chapter 2, a number of other features such as chord changes have been used for beat tracking with varying success, and it would be interesting to experiment with these to provide even greater feature diversity to a Beaker ensemble. Similarly, as discussed in Chapters 3 and 4, there are other approaches for both periodicity estimation and beat phase estimation that could potentially increase ensemble diversity and therefore improve performance. Additionally, the method for determining a TestTracker's confidence in its beat phase estimate could be expanded to include continuity with previous beat phase estimates and perhaps a measure of the reliability of the input feature used, similar to how the tempo confidence takes into consideration the peakiness of the periodicity function.

Another way to increase the diversity of ensembles of TestTrackers would

be to allow different TestTrackers to focus on separate channels of multi-channel audio input. Currently all channels are mixed down to one channel for analysis, but because multi-channel recordings often have different balances of instruments in each channel, analyzing those channels separately could create a more diverse ensemble. This approach could apply not only to TestTrackers but to any arbitrary Beaker ensemble member, as each could be provided input audio frames consisting of a subset of available channels.

Other areas for future work are part of the Beaker framework itself. Different approaches to discretizing and clustering or combining tracker hypotheses could be introduced. For example, different clustering algorithms could be added as options, or when using K-means the choice of K could be made dynamically instead of being a hard-coded parameter. Further exploration of the concept of the ensemble's internal beat phase oscillator is also possible.

It would also be interesting to test the Beaker ensemble framework with other causal trackers. For example, if Beaker can perform as well as it does with simple TestTrackers, it should be able to do even better with ensembles of more sophisticated beat trackers such as IBT-C [55] or other existing causal trackers. While the IBT-C code is open source and available as part of the Marsyas framework, in order to use IBT-C as part of a Beaker ensemble, it would be necessary to modify that implementation to output tracker hypotheses and confidence levels on a frame-by-frame basis, as the current implementation only supports writing output text files containing beat and tempo hypotheses [1]. Therefore this remains a task for future work.

There is also significant room for additional work examining the concept of ensemble diversity in a Beaker ensemble. As previously discussed, the metric described in Chapter 10 for predicting when the combination of two ensembles would improve performance operates on a song-by-song basis. However, a finer-grained approach should allow more accurate predictions, enabling us to see cases where two ensembles perform well on different segments of a song and not simply on the entire song.

Overall, this dissertation has demonstrated that ensemble methods can be

applied in a real-time and causal fashion while remaining flexible enough to support arbitrary beat trackers as ensemble members and has laid the foundation for significant future work in the combination of ensemble learning and beat tracking.

Bibliography

[1] ibt - Marsyas User Manual. http://marsyas.info/doc/manual/marsyas-user/ibt.html, Acccessed: 2015-04-16.

[2] MIREX 2012: Audio Beat Tracking - MIREX12 SMC Dataset - Summary. http://nema.lis.illinois.edu/nema_out/mirex2012/results/abt/smc/, Acccessed: 2015-04-26.

[3] Miguel Alonso, Bertrand David, and Gael Richard. Tempo and Beat Estimation of Musical Signals. In *Proc. International Conference on Music Information Retrieval*, 2004.

[4] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, Cambridge, MA, 2004.

[5] Shai Avidan. Ensemble Tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(2):261–271, February 2007.

[6] Juan P. Bello, Chris Duxbury, Mike Davies, and Mark Sandler. On the Use of Phase and Energy for Musical Onset Detection in the Complex Domain. *IEEE Signal Processing Letters*, 11(6):553–556, June 2004.

[7] 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, September 2005.

[8] Sebastian Böck, Florian Krebs, and Gerhard Widmer. A Multi-Model Approach to Beat Tracking Considering Heterogenous Music Styles. In *15th International Society for Music Information Retrieval Conference*, 2014.

[9] Sebastian Böck and Markus Schedl. Enhanced Beat Tracking With Context-Aware Neural Networks. In *Proc. Int. Conf. Digital Audio Effects (DAFx-11)*, pages 135–139, Paris, France, September 2011.

[10] Leo Breiman. Bagging Predictors. *Machine Learning*, 24:123–140, 1996.

[11] Leo Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001.

[12] Ali Taylan Cemgil, Bert Kappen, Peter Desain, and Henkjan Honing. On tempo tracking: Tempogram Representation and Kalman filtering. *Journal of New Music Research*, 29(4):259–273, December 2000.

[13] Robert T. Collins, Yanxi Liu, and Marius Leordeanu. On-Line Selection of Discriminative Tracking Features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1631–1643, 2005.

[14] Matthew E. P. Davies, Norberto Degara, and Mark D. Plumbley. Evaluation Methods for Musical Audio Beat Tracking Algorithms. Technical report, Queen Mary University of London, 2009.

[15] Matthew E. P. Davies, Norberto Degara, and Mark D. Plumbley. Measuring the Performance of Beat Tracking Algorithms Using a Beat Error Histogram. *IEEE Signal Processing Letters*, 18(3):157–160, March 2011.

[16] 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, March 2007.

[17] Matthew E. P. Davies and Mark D. Plumbley. On the Use of Entropy for Beat Tracking Evaluation. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume IV, pages 1305–1308, Hawaii, USA, April 2007.

[18] Norberto Degara, Antonio Pena, Manuel Sobreira-Seoane, and Soledad Torres-Guijarro. A Mixture-of-Experts Approach for Note Onset Detection. In *Proc. of the 126th AES Convention*, May 2009.

[19] Norberto Degara, Enrique Argones Rua, Antonio Pena, Soledad Torres-Guijarro, Matthew E. P. Davies, and Mark D. Plumbley. Reliability-Informed Beat Tracking of Musical Signals. *IEEE Transactions on Audio, Speech and Language Processing*, 20(1):290–301, January 2012.

[20] Thomas G. Dietterich. Ensemble Methods in Machine Learning. In *Multiple Classifier Systems*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, Cagliari, Italy, 2000.

[21] Simon Dixon. Automatic Extraction of Tempo and Beat From Expressive Performances. *Journal of New Music Research*, 30(1):39–58, March 2001.

[22] Simon Dixon. Onset Detection Revisited. In *Proc. of the 9th Int. Conference on Digital Audio Effects, DAFx-06*, pages 133–137, Montreal, Canada, September 2006.

[23] Simon Dixon. Evaluation of the Audio Beat Tracking System BeatRoot. *Journal of New Music Research*, 36(1):39–50, March 2007.

[24] Chris Duxbury, Mark Sandler, and Mike Davies. A Hybrid Approach to Musical Note Onset Detection. In *Prod. of the 5th Int. Conference on Digital Audio Effects (DAFx-02)*, pages 33–38, Hamburg, Germany, September 2002.

[25] Douglas Eck. Identifying Metrical and Temporal Structure With an Autocorrelation Phase Matrix. *Music Perception: An Interdisciplinary Journal*, 24(2):167–176, December 2006.

[26] Douglas Eck. Beat Tracking Using an Autocorrelation Phase Matrix. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2007)*, pages 1313–1316, Honolulu, HI, April 2007.

[27] Daniel P. W. Ellis. Beat Tracking by Dynamic Programming. *Journal of New Music Research*, 36(1):51–60, March 2007.

[28] Antti J. Eronen and Anssi P. Klapuri. Music Tempo Estimation With k-NN Regression. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(1):50–57, January 2010.

[29] Jonathan Foote and Shingo Uchihashi. The Beat Spectrum: A New Approach to Rhythm Analysis. In *Proc. Int. Conference on Multimedia and Expo (ICME)*, Tokyo, Japan, August 2001. IEEE.

[30] Stan Franklin and Art Graesser. Is It an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. In *Proc. of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, Lecture Notes in Computer Science, pages 21–35, August 1996.

[31] Yoav Freund and Robert E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[32] Masataka Goto and Yoichi Muraoka. Music Understanding At The Beat Level - Real-time Beat Tracking For Audio Signals. In *Working Notes of the IJCAI-95 Workshop on Computational Auditory Scene Analysis*, pages 68–75, 1995.

[33] Masataka Goto and Yoichi Muraoka. Issues in Evaluating Beat Tracking Systems. In *Working Notes of the IJCAI-97 Workshop on Issues in AI and Music - Evaluation and Assessment*, pages 9–16, 1997.

[34] Masataka Goto and Yoichi Muraoka. Real-time Rhythm Tracking for Drumless Audio Signals - Chord Change Detection for Musical Decisions. In *Proc. Int.*

Joint. Conf. in Artificial Intelligence: Workshop on Computational Auditory Scene Analysis, pages 135–144, 1997.

[35] Masataka Goto and Yoichi Muraoka. Real-time beat tracking for drumless audio signals: Chord change detection for musical decisions. *Speech Communication*, 27(3-4):311–335, April 1999.

[36] Fabien Gouyon, Simon Dixon, and Gerhard Widmer. Evaluating Low-Level Features for Beat Classification and Tracking. In *Proc. of the 2007 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1309–1312, 2007.

[37] Fabien Gouyon, Anssi P. Klapuri, Simon Dixon, Miguel Alonso, George Tzanetakis, Christian Uhle, and Pedro Cano. An Experimental Comparison of Audio Tempo Induction Algorithms. *IEEE Transactions on Audio, Speech and Language Processing*, 14(5):1832–1844, September 2006.

[38] Helmut Grabner and Horst Bischof. On-line Boosting and Vision. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 260–267, 2006.

[39] Stephen W Hainsworth. *Techniques for the Automated Analysis of Musical Audio*. Doctor of philosophy, University of Cambridge, 2004.

[40] Tin Kam Ho. The Random Subspace Method for Constructing Decision Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[41] Tin Kam Ho, Jonathan J. Hull, and Sargur N. Srihari. Decision Combination in Multiple Classifier Systems. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 16(1):66–75, 1994.

[42] Andre Holzapfel, Matthew E. P. Davies, Jose R. Zapata, Joao Lobato Oliveira, and Fabien Gouyon. Selective Sampling for Beat Tracking Evaluation. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(9):2539–2548, 2012.

[43] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive Mixtures of Local Experts. *Neural Computation*, 3(1):79–87, 1991.

[44] Josef Kittler, Mohamad Hatef, Robert P. W. Duin, and Jiri Matas. On Combining Classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239, March 1998.

[45] Anssi P. Klapuri. Musical meter estimation and music transcription. In *Proc. Cambridge Music Processing Colloquium*, pages 40–45, 2003.

[46] Anssi P. Klapuri, Antti J. Eronen, and Jaakko T. Astola. Analysis of the Meter of Acoustic Musical Signals. *IEEE Transactions on Audio, Speech and Language Processing*, 14(1):342–355, January 2006.

[47] Ludmila I. Kuncheva. A Theoretical Study on Six Classifier Fusion Strategies. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):281–286, 2002.

[48] Ludmila I. Kuncheva and Christopher J. Whitaker. Measures of Diversity in Classifier Ensembles. *Machine Learning*, 51:181–207, 2003.

[49] Edward W. Large. Modeling Beat Perception with a Nonlinear Oscillator. In *Proc. of the 18th Annual Conference of the Cognitive Science Society*, San Diego, CA, July 1996.

[50] Edward W. Large and John F. Kolen. Resonance and the Perception of Musical Meter. *Connection Science*, 6(2):177–208, 1994.

[51] Jean Laroche. Efficient Tempo and Beat Tracking in Audio Recordings. *J. Audio Engineering Society*, 51(4):226–233, April 2003.

[52] Martin F. McKinney and Dirk Moelants. Ambiguity in tempo perception: What draws listeners to different metrical levels? *Music Perception: An Interdisciplinary Journal*, 24(2):155–166, December 2006.

[53] Martin F. McKinney, Dirk Moelants, Matthew E. P. Davies, and Anssi P. Klapuri. Evaluation of Audio Beat Tracking and Music Tempo Extraction Algorithms. *Journal of New Music Research*, 36(1):1–16, March 2007.

[54] João Lobato Oliveira, Matthew E. P. Davies, Fabien Gouyon, and Luis Paulo Reis. Beat Tracking for Multiple Applications: A Multi-Agent System Architecture With State Recovery. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(10):2696–2706, December 2012.

[55] João Lobato Oliveira, Fabien Gouyon, Luis Gustavo Martins, and Luis Paulo Reis. IBT : A Real-Time Tempo and Beat Tracking System. In *Proc. of the 11th International Society for Music Information Retrieval Conference*, pages 291–296, Utrecht, Netherlands, August 2010.

[56] Nikunj C. Oza and Kagan Tumer. Input Decimation Ensembles: Decorrelation through Dimensionality Reduction. In *Multiple Classifier Systems*, volume 2096 of *Lecture Notes in Computer Science*, pages 238–247. 2001.

[57] Geoffroy Peeters. Template-Based Estimation of Time-Varying Tempo. *EURASIP Journal on Advances in Signal Processing*, 2007:1–15, 2007.

[58] Geoffroy Peeters and Helene Papadopoulos. Simultaneous Beat and Downbeat-Tracking Using a Probabilistic Framework: Theory and Large-Scale Evaluation. *IEEE Transactions on Audio, Speech and Language Processing*, 19(6):1754–1769, August 2011.

[59] Robi Polikar. Ensemble Based Systems in Decision Making. *IEEE Circuits and Systems Magazine*, (Third Quarter):21–45, 2006.

[60] Lawrence R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[61] Andrew Robertson, Adam M. Stark, and Mark D. Plumbley. Real-Time Visual Beat Tracking Using a Comb Filter Matrix. In *Proc. International Computer Music Conference*, 2011.

[62] Robert E. Schapire. The Strength of Weak Learnability. *Machine Learning*, 5(2):197–227, June 1990.

[63] Eric D. Scheirer. Tempo and beat analysis of acoustic musical signals. *The Journal of the Acoustical Society of America*, 103(1):588–601, January 1998.

[64] Giovanni Seni and John F. Elder. *Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions*, volume 2 of *Synthesis Lectures on Data Mining and Knowledge Discovery*. Morgan & Claypool, January 2010.

[65] Adam M. Stark, Matthew E. P. Davies, and Mark D. Plumbley. Real-Time Beat-Synchronous Analysis of Musical Audio. In *Proc. of the 12th International Conference on Digital Audio Effects, DAFx-09*, pages 1–6, Como, Italy, September 2009.

[66] George Tzanetakis. Tempo Extraction using Beat Histograms. In *Proceedings of the 1st Music Information Retrieval Evaluation eXchange (MIREX 2005)*, 2005.

[67] George Tzanetakis and Perry Cook. Musical Genre Classification of Audio Signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, July 2002.

[68] José R. Zapata, Matthew E. P. Davies, and Emilia Gómez. Multi-Feature Beat Tracking. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(4):816–825, April 2014.