Automatic Analysis of Music in Standard MIDI Files

Zheng Jiang

May 2019

Music and Technology
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Roger Dannenberg, Chair
Barnabas Poczos

Submitted in partial fulfillment of the requirements for the degree of Master of Science.

Keywords: MIDI, Melody Analysis, Chord Analysis, Structure Analysis

For my love.

Abstract

The identification of melody, bass, chord and structure are important steps in music analysis. This thesis presents a comprehensive tool to identify the melody and bass in each measure of a Standard MIDI File, and provides chord labels and general structure information. We also share an open dataset of manually labeled music for researchers. We use a Bayesian maximum-likelihood approach and dynamic programming as the basis of our work in the melody identification. We have trained parameters on data sampled from the million song dataset [16, 21] and tested on a dataset including 1703 measures of music from different genres. Our algorithm achieves an overall accuracy of 89% in the test dataset. We compare our results to previous work. For bass identification, since our algorithm is rule-based, we only label the test set. And the bass identification achieves over 95% accuracy. Our chord-labeling algorithm is adopted from Temperley [20] and tested on a manually labeled test set containing 1890 labels. Currently, this algorithm achieves around 78% accuracy for chord root and chord type matching and around 82% accuracy for chord root matching only. We also discovered an optimization: by removing notes in melody channel, we can improve the accuracy for both evaluations by 1.3%. For the structure analysis, we also provide a simple algorithm based on a similarity matrix and give some analysis of the result. By automatically labeling and analyzing MIDI files, a rich source of symbolic music information, we hope to enable new studies of music style, music composition, music structure, and computational music theory.

Acknowledgments

Looking as far into the past as my memory allows, I discern a few milestones that led to the creation of this project. Each milestone is associated with someone who played a vital role in my ability to realize the present work.

Two years ago, I submitted the application for the program of Music and Technology because Music and Computer Science are two essentials in my life. And in this program I am so pleasure to be advised by the pioneer in this field, my advisor, Prof. Roger B. Dannenberg. And I am so honored with a little surprise to become the only student in that year. In addition, I had the opportunity to use the facilities of his research lab. This was a very generous arrangement that few graduate students have the opportunity to enjoy. I am forever grateful to Roger for this.

In the second semester, I took a course named Art and Machine Learning given by Prof. Barnabas Poczos and Prof. Eunsu Kang. In that course we made a Chinese Opera by machine learning algorithms and also present it in the Thirty-second Conference on Neural Information Processing Systems 2018 workshop. And it is my honor that Barnabas becomes my co-advisor to support my research in computer based music analysis.

During the research, our work of melody analysis has been accepted in the Sound and Music Conference 2019. It is such a milestone in my academic life during my graduate.

And great thanks to my parents to support my dreams in both music and computer science!

Thanks to Prof. Richard Randall, Prof. Richard Stern and Prof. Riccardo Schulz for giving me comments on my proposal and thesis.

Also thanks to Shuqi Dai and Junyan Jiang for providing some MIDI files and some of the labels in my dataset.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of Music Information Retrieval began with a focus on audio content-based search and classification. After a decade of intense activity, it became clear that a major problem for machine processing of music is a deep understanding of music structure, patterns, and relationships. Thus, music analysis which identifies melody, bass, chords, and other elements is seen as an important challenge for MIR. Recently, machine learning algorithms have dominated work in MIR. One of the properties of machine learning algorithms is that they require a lot of data. There is a wealth of data in the form of MIDI files, which have pitch, and beat information. This symbolic representation of music is very difficult to extract directly and precisely from audio files. On the other hand, humans do understand the concept of notes, pitches and beats. Working with this level of representation is relevant to human music perception even if we cannot model the transformation of audio into these representations. MIDI gives us the opportunity to extract much useful information for studying music. But so far, there are not any systems that can actually do a good job extracting data from MIDI files in general. Typically, researchers use carefully selected or prepared MIDI files and look for particular information. As far as we know, there is no comprehensive systems for MIDI file analysis. The aim of this thesis is to build a comprehensive system that can extract melody, bass, harmony and structure information, which is the basic information for the music theorist. It is a starting point for any kind of analysis in music theory. The hope is that by building this tool, researchers will be able to get more useful information from MIDI files and use it for various tasks. These tasks include automatic music composition, building models for music listening (the models for music and music expectation when people look at the music notes or listen to music), etc. So having music data to look at is not only important for generating new music but also for studies of music listening. And because there are so many music genres, in this thesis, we decide to focus on pop music.

The thesis consists of the following chapters: Chapter 2 on related work, Chapter 3 on machine-readable music representations, Chapter 4 on melody analysis, Chapter 5 on bass analysis, Chapter 6 on harmony analysis, Chapter 7 on structure analysis and Chapter 8 presents my conclusion.

Chapter 2

Related Work

2.1 Melody and Bass Analysis

In the field of symbolic files, *Skyline* is a very simple algorithm proposed by Uitdenbogerd [22]. In brief, the idea of this method is to pick the highest pitch at any moment as belonging to the melody. Chai and Vercoe offer an enhanced version of this approach [5]. In pop music, we observe that there are often accompaniment notes above the melody line, leading to failure of the Skyline algorithms. Uitdenbogerd presents three more methods [9]: 1) Top Channel (choose the channel with the highest mean pitch), 2) Entropy Channel (choose the channel with the highest entropy), and 3) Entropy Part (segment first, then use Entropy Channel). Shan [18] proposed using greatest volume (MIDI velocity) because melody is typically emphasized through dynamics. Li et al. identify melodies by finding common sequences in multiple MIDI files, but this obviously requires multiple versions of songs [13]. Li, Yang, and Chen [12] use a Neural Network and features such as chord rate, pronunciation rate, average note pitch, instrument, etc., trained on 800 songs to estimate the likelihood that a channel is the melody channel. Velusamy, et al. [23] use a similar approach, but prune notes that do not satisfy certain heuristics and use a hand-crafted linear model for ranking channels [12].

All of these algorithms assume that the melody appears on one and only one channel, so the problem is always to select one of up to 16 channels as the melody channel. Depending on the data, this can be a frequent cause of failure, since the melody can be expressed by different instruments in different channels at different times. An interesting approach is *Tunerank* [25], which groups and labels notes according to harmony and dissonance with other notes, pitch intervals between consecutive notes, and instrumentation, without assuming the melody is in only one channel.

Previous work is hard to evaluate, with accuracy reports ranging from 60% to 97%, no labeled public datasets, and few shared implementations. The properties of music arrangements and orchestrations in MIDI files can cause many problems. The simplest case, often assumed in the literature, is that the melody appears in one and only one channel. At least four more complex conditions are often found: 1) The melody is sometimes played in unison or octaves in another channel, 2) the melody switches from one instrument (channel) to another from one phrase or repetition to another, 3) the melody is fragmented across channels even within a single

measure (this happens but seems to be rare), and 4) there are multiple overlapping melodies as in counterpoint, rounds, etc.

Unlike melody, there is not so many papers that discuss bass extraction in MIDI files. One research area that is related is bass transcription from audio [17]. However, since this thesis will focus on symbolic data, we will not discuss the audio transcription topic here.

There are some applications of detected bass lines. One of them is automatic music genre classification using bass lines [19]. Beside that, the output of our system can also be used in music generation systems.

2.2 Chord Analysis

Temperley [20] proposes an algorithm for harmonic analysis. It generates a representation in which the piece is divided into segments labeled with roots. One of the major innovations of it is that pitches and chords are both represented on a spatial representation known as the "line of fifths".

The dataset is the eternal topic when we consider a computational model for analyzing music. In the audio area, there are some datasets that have been well labeled. For example, Burgoyne [4] proposes such a dataset. Working with a team of trained jazz musicians, they have collected time-aligned transcriptions of the harmony in more than a thousand songs selected randomly from the Billboard "Hot 100" chart in the United States between 1958 and 1991. However, in the symbolic area, such manually labeled datasets are very rare. One method is to convert chord information from leadsheets. Lim et al. [14] publish a CSV Leadsheet Database which contains a lead sheet database provided by Wikifonia.org. It has Western music lead sheets in MusicXML format, including rock, pop, country, jazz, folk, RnB, childrens song, etc. The dataset is split into two sets: a training set of 1802 songs, and a test set of 450 songs. However, because leadsheets do not contain notes other than the melody, it is hard to consider this as a training or testing dataset for the chord analysis task.

Pardo [15] also describes a system for segmenting and labeling tonal music based on the template matching and graph-search techniques. And his idea about evaluation also inspires our evaluation methods in this thesis.

2.3 Structure Analysis

Dannenberg and Goto [6] introduce methods to analyze textures, repetition of phrases or entire sections, and the algorithms to leverage the similarity matrix are quite inspiring. Our algorithm will take some inspiration from this work, although the analysis space is changed from audio to MIDI. And Klapuri et al. [10] gives an overview of state-of-the art methods for computational music structure analysis, where the general goal is to divide an audio recording into temporal segments corresponding to musical parts and to group these segments into musically meaningful categories. For example, Lerdahl et al. [11] models music understanding from the perspective of cognitive science and a grammar of music with the aid of generative linguistics, which is also a

kind of structure analysis. Bruderer et al. [2] investigate the perception of structural boundaries to Western popular music and examine the musical cues responsible for their perception.

Goto and Dannenberg [8] introduce some applications and interfaces that are enabled by advances in automatic music structure analysis. One of them is the SmartMusicKIOSK interface [7]. A user can actively listen to various parts of a song, guided by the visualized music structure ("music map") in the window.

Chapter 3

Machine Readable Music Representation

3.1 Introduction to MIDI

Currently, MIDI is one of the most popular formats for storing the music information. In this section, we will introduce some properties of MIDI to explain the parts that will be used in our research and the information that we ignore.

We can consider a MIDI file to be a list of events of different categories. In general, there are three major categories: channel event, system event and meta event. A channel event contains a **channel number** (in the specification of MIDI, there are 16 channels, each channel can be assigned an instrument) and is used to represent note on/off, control change, program (instrument) change, pitch wheel, etc. The system event contains start, stop and reset, etc. commands. The meta event contains time signature, tempo, etc. MIDI files do not necessary tell what is the beat number or measure number for each note. The meta event will be used in our system to identify beats and measures. If the meta events of time signature and tempo are missing, that piece will be considered as invalid input to our system. In our research, we are mainly concerned with pitches, durations, and instruments. So we ignore many events associated with synthesizes control, such as volume and pitch wheel events.

Because the MIDI file is stored in a binary format, it is not very convenient for programmer to read/write MIDI information. In our system, we use Serpent[1] to read binary MIDI files and then write the content that we need to an ASCII text file. Our algorithms only consider the information in the text files instead of the original MIDI file.

3.2 Music Representation

How to store music information in a digital computer is an essential concern when we design a music processing system. In this section, we will discuss the music representation for the output of our system. Since there are mainly four parts in our output: melody, bass, harmony, and structure, we will list the specification below:

[1] https://www.cs.cmu.edu/ music/serpent/serpent.htm

Considering that there are many researchers using Matlab, our music representation will be saved in a CSV format, which has at least two advantages: 1) it is easy to check the correctness by eye; 2) it is easy to read CSV data in a variety of languages such as Matlab or Python. This design may introduce an extra cost of storage, but nowadays memory is much cheaper than decades ago when the Standard MIDI File format was designed. So we think this convenience and space trade-off is reasonable.

- **Melody**
  In our output, we use the quantization to restrict the data to integers. We use $\frac{1}{24}$ beat as the minimal unit for duration because it can represent 32^{nd} notes and 64^{th} triplets, which is fine enough for almost any rhythm. So the melody note in the first measure (**m**) second beat (**b**) which has the duration(**b**) of one beat, pitch (**p**) of 60 (the C4 on the keyboard), velocity (**v**) of 127, channel (**c**) of 0, instrument (**i**) number of 1 (piano) is arranged as Table 3.1

Table 3.1: Melody Representation

m	**b**	**d**	**p**	**v**	**c**	**i**
0	24	24	60	127	0	1

- **Bass**
  The bass representation is similar to melody. For example, the bass note in the second measure (**m**) first beat (**b**) which has the duration (**b**) of four beats, pitch (**p**) of 36 (the C2 on the keyboard), velocity (**v**) of 77, channel (**c**) of 1, instrument (**i**) number of 32 (acoustic bass guitar) is arranged as Table 3.2

Table 3.2: Bass Representation

m	**b**	**d**	**p**	**v**	**c**	**i**
1	0	96	48	77	1	33

- **Harmony**
  Unlike melody and bass, in harmony parts, the pitch, velocity, channel, and instrument are ignored. Instead, we encode the root, type and pitch class set. The root is an integer from 0 to 11 (chromatic scale, [C, C#/Db, D, ..., B]). The type is the classification such as major or minor, represented as an integer. The mapping from number to detail meaning can be found in Table 3.3. The pitch class set is a number to represent a pitch class vector as a decimal integer. For example, the C major chord has the pitch class of C, E, and G. The pitch class vector is Table 3.4. In binary, with C as the low order bit, we can encode the set {C, E, G} as number {0, 4, 7} or the binary number 000010010001_B, which is the decimal number 145. We can use the pitch class set to represent rare chords without conventional types. Also, we do not distinguish, say, Major from Major-Seventh chords, so the pitch class set gives additional information on chords. Note also that the pitch class set alone is not sufficient information. For example, C-major-6^{th} has the same pitch class

set as A-minor-7th. The pitch class set is simply the set of all pitches that are observed within the segment of music labeled by the chord. For example, suppose music experts consider the true chord type to be a G dominant 7 chord, but while the chord is playing, the melody plays not only chord tones but non-chord passing tones of Ab and F#. Our chord label will be G Major (we do not consider dominant seventh chords to be a different category. The pitch class set will contain all the observed pitch classes: {D, F, F#, G, Ab, B}. Alternatively, if the true chord type is G dominant 7 and we have the same melody, but the fifth (D) is not played, then the pitch class set will not contain the fifth (D): {F, F#, G, Ab, B}.

Table 3.3: Harmony Representation

#	**Meaning**
0	Major
1	Minor
2	Diminished
3	Augmented
4	Suspended 4
-1	Others

Table 3.4: C Major Pitch Class Vector

B	**A#**	**A**	**G#**	**G**	**F#**	**F**	**E**	**D#**	**D**	**C#**	**C**
0	0	0	0	1	0	0	1	0	0	0	1

So a C major chord in the first measure (**m**), first beat (**b**), with the duration (**d**) of 4 beats, root (**r**) of C, type (**t**) of Major chord and pitch class vector (**s**) of 145 is arranged as Table 3.5

Table 3.5: Harmony Representation

m	**b**	**d**	**r**	**t**	**s**
0	0	96	0	0	145

- **Structure**
  To represent structure, we only consider the measure, duration and section information. As Table 3.6 shows, the first section starts from the first measure and lasts for 8 measures. The first section repeats once (9^{th}-16^{th} measures). The second section starts from the 17^{th} measure and lasts for 8 measures. Then, the first section repeats again. So that is an AABA structure. And in this thesis, we only consider the very high-level structure so the very detailed structures will not be shown. Also notice here the unit of duration is not 24^{th} but the measure, and numbering is zero-based, i.e. 0 denotes the 1^{st} measure, etc.

Table 3.6: Example Structure Representation

m	**d**	**s**
0	8	0
8	8	0
16	8	1
24	8	0

Chapter 4

Melody Analysis

Melody identification is an important early step in music analysis. In this chapter, we present a tool to identify the melody in each measure of a Standard MIDI File. We also share an open dataset of manually labeled music for researchers. We use a Bayesian maximum-likelihood approach and dynamic programming as the basis of our work. We have trained parameters on data sampled from the million song dataset [16, 21] and tested on a dataset including 1703 measures of music from different genres. Our algorithm achieves an overall accuracy of 89% in the test dataset. We compare our results to previous work.

4.1 Introduction

When we listen to a piece of music, the melody is usually the first thing that catches our attention. Therefore, the identification of melody is one of the most important elements of music analysis. Melody is commonly understood to be a prominent linear sequence of pitches, usually higher than harmonizing and bass pitches. The concept of melody resists formalization, making melody identification an interesting music analysis task. Melody is used to identify songs. Often, other elements such as harmony and rhythm are best understood in relation to melody.

Many music applications depend on melody, including Query-by-Humming systems, music cover song identification, emotion detection [24], and expressive performance rendering. Many efforts in automatic composition could benefit from training data consisting of isolated melodies.

There has been a lot of research on extracting melody from audio [9]. The problem is generally easier for MIDI than audio because at least notes are already identified and separated. However, compared to audio, there seems to be less research on melody extraction. Most of the research on MIDI melody is on channel-level identification. This paper will propose an algorithm combining Bayesian probability models and dynamic programming to extract melody at the measure level.

4.2 Dataset

In most related work, published links to datasets have expired, so we collected and manually labeled a new dataset which contains the training data of 5823 measures in 51 songs and test

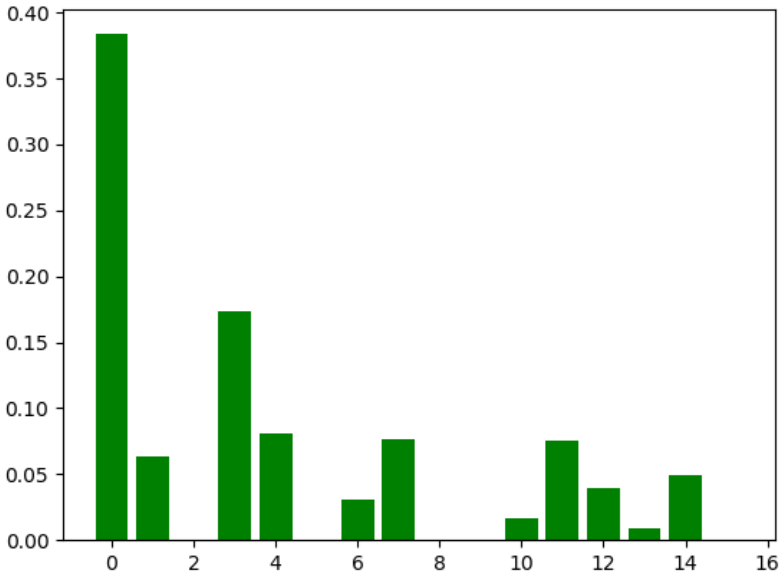

Figure 4.1: Measure Level Melody Channel Distribution

data of 1703 measures in 22 songs. For each song in the training data, the melody is mostly all on one channel, which is labeled as such. (This made labeling much easier, but we had to reject files where the melody appeared significantly in multiple channels or there is no melody at all.) In the test data, the melody is not constrained to a single channel, and each measure of the song is labeled with the channel that contains the melody. Measure boundaries are based on tempo and time signature information in the MIDI file. The MIDI files are drawn from songs in the Lakh dataset [16]. This collection contains MIDI files that are matched to a subset of files in the million song dataset [21]. We used tags there to limit our selection to pop songs.

The test data is collecting from Chinese, Japanese, and American pop songs. We specifically chose popular music because melody is usually present and there is usually a single melody. In addition, we hope to use this research in learning about melody structure in popular music. Figure 4.1 is the distribution of measure level channel number in test data. Channel 0 is most popular melody channel with around 38%. Some channels in our dataset has no existing data, such as channel 2, channel 5, channel 8, channel 9 and channel 15.

It might be noted that there are many high quality MIDI files of piano music. Since all piano notes are typically on one channel, this can make the melody identification or separation a more challenging problem, and different techniques are required. We assume that in our data, once the channel containing the melody is identified, it is fairly easy to obtain the melody. Either the melody is the only thing present in the channel, or the melody is harmonized, and the melody is obtained by removing the lower notes using the Skyline algorithm.

4.3 Algorithm

Our problem consists of labeling each measure of a song with the channel that contains the melody. (It would be useful also to allow non-labels, or *nil*, indicating there is no melody, but our study ignores this option.) The algorithm begins with a Bayesian model to estimate $\boldsymbol{M}_{m,c}$, the probability that channel c in measure m contains the melody. The estimation uses features that are assumed to be jointly normally distributed and independent. Features are calculated from the content of each channel, considering the measure itself and N previous and subsequent measures, with $N \in 0, 1, ...$ In all of our experiments, we assume that melody *never* appears on channel 10, which is used for drums in General MIDI.

Although we could stop there and report the most likely channel in each measure,

$$c_m = \underset{c}{\operatorname{argmax}}\, \boldsymbol{M}_{m,c} \tag{4.1}$$

this does not work well in practice. There are many cases where a measure of accompaniment, counter-melody or bass appears to be more "melody-like" than the true melody. (For example, the melody could simply be a whole note in some measures.) However, it is rare for the melody to switch from one channel to another because typically the melody is played by one instrument on one channel. Channel switches are only likely to occur when the melody is repeated or on major phrase boundaries.

We can consider the melody channel for each measure, c_m, as a sequence of hidden states and per-measure probabilities as observations. We wish to find the most likely overall sequence c_m according to the per-measure probabilities, and taking into account a penalty for switching channels from one measure to the next. We model the probability of the hidden state sequence c_m as:

$$P(c_m) = \prod_m \boldsymbol{M}_{m,c_m} S_{c_{m-1},c_m} \tag{4.2}$$

where $S_{c_{m-1},c_m} = 1$ if there is no change in the channel ($c_{m-1} = c_m$), and S_{c_{m-1},c_m} is some penalty less than one if there is a channel change ($c_{m-1} \neq c_m$). Thus, channel switches are allowed from any measure to the next, but channel switches are considered unlikely, and any labeling that switches channels frequently is considered highly unlikely.

The parameters of this model must be learned, including: statistics for features used to estimate $\boldsymbol{M}_{m,c}$, the best feature set, the number of neighboring measures N to use in computing features, and the penalty S for changing channels. We select the feature set and compute feature statistics using our training dataset, and we evaluate their performance and sensitivity to N and S using the test dataset.

4.3.1 Bayesian Probability Model

The probability of melody given a set of features is represented by Equation 4.3, where C_0 is the condition that the melody is present, C_1 indicates the melody is not present, x_i are feature values, and n is the number of real-valued features. The details of features will be discussed in a later paragraph.

$$P(C_0|x_1, \ldots, x_n) \tag{4.3}$$

By Bayes' theorem, this conditional probability can be rewritten as Equation 4.4.

$$P(C_0|x_1, \ldots, x_n) = \frac{P(C_0)P(x_1, \ldots, x_n|C_0)}{P(x_1, \ldots, x_n)} \tag{4.4}$$

With the assumption of independence for each feature, we can rewrite this as Equation 4.5:

$$P(C_0|x_1, \ldots, x_n) = \frac{1}{Z}P(C_0)\prod_{i=1}^{n} P(x_i|C_0) \tag{4.5}$$

where Z is:

$$Z = P(x_1, \ldots, x_n) = \sum_{k=0}^{1}(P(C_k)\prod_{i=1}^{n} P(x_i|C_k)) \tag{4.6}$$

Our features x_i are continuous values, and we assume they are distributed according to a Gaussian distribution as in Equation 4.7. Under this assumption, we can simply collect feature statistics $\mu_{i,k}$ and $\sigma_{i,k}$ from training data to estimate the probability model.

$$P(x_i = v|C_k) = \frac{1}{\sqrt{2\pi\sigma_{i,k}^2}}e^{-\frac{(v-\mu_{i,k})^2}{2\sigma_{i,k}^2}} \tag{4.7}$$

We now describe the details of features, which are $note\_density$, vel_mean, vel_std, $pitch\_mean$, $pitch\_std$, IOI_mean, and IOI_std:

Note Density

The note density is the sum of all note durations divided by the total length of the music (Equation 4.8). A melody without rests has a note density of 1, a rest has note density of 0, a sequence of triads without rests has a note density of 3, etc.

$$note\_density = \frac{\Sigma_{note} note.\text{dur}}{total\_length} \tag{4.8}$$

Velocity

We take the mean and standard deviation of velocity (Equations 4.9 and 4.10).

$$vel\_mean = \frac{\Sigma_{i=1}^{N} note_i.\text{vel}}{N} \tag{4.9}$$

$$vel\_std = \sqrt{\frac{1}{N-1}\Sigma_{i=1}^{N}(note_i.\text{vel} - vel\_mean)^2} \tag{4.10}$$

Pitch

We take the mean and standard deviation of pitch (Equations 4.11 and 4.12).

$$pitch\_mean = \frac{\Sigma_{i=1}^{N} note_i.\text{pitch}}{N} \tag{4.11}$$

$$pitch\_std = \sqrt{\frac{1}{N-1}\Sigma_{i=1}^{N}(pitch_i.\text{vel} - pitch\_mean)^2} \tag{4.12}$$

Inter-Onset Interval

The Inter-Onset Interval, or IOI, means the interval between onsets of successive notes. Considering that ornaments and chords may introduce a very short IOIs, we set a window of 75 ms, and when two note onsets are within that window, we treat them as a single onset [1]. IOI calculation is described in detail in Algorithm 1.

Result: The mean and standard deviation of a list of notes in onset-time order
stats is an object that implements the calculation of mean and standard deviation;
note[i] is the i^{th} note;
N is the number of notes;
$i \leftarrow 0$;
while $i < N$ **do**
  $j \leftarrow i + 1$;
  while $(j < N) \wedge (note[j].\text{on\_time} - note[j-1].\text{on\_time}) < 0.075$ **do**
    $j \leftarrow j + 1$;
  end
  if $j < N$ **then**
    $IOI \leftarrow note[j].\text{on\_time} - note[i].\text{on\_time}$;
    $stats.\text{add\_point}(IOI)$;
  end
  $i \leftarrow j$;
end
$IOI\_mean \leftarrow stats.\text{get\_mean}()$;
$IOI\_std \leftarrow stats.\text{get\_std}()$;

Algorithm 1: IOI Feature Calculation

4.3.2 Training data

We compute features for each measure and channel of the training data. For feature selection, we use cross-validation, dividing the training songs into 5 groups, holding out each group and estimating $\mu_{i,k}$ and $\sigma_{i,k}$ from the remaining training data, and evaluating the resulting model by counting the number of measures where the melody channel is judged most likely by the model. We take the average result over all five groups.

After doing this for every combination of features (7 features, thus 127 combinations), for various values of N (the maximum distance to neighboring measures to use in feature calculation), we determine the features that produce the best result for each value of N.

We then re-estimate the probability model using *all* of the training data. In principle, we should also use the training data to learn the best window size N and the best penalty S, but in our training data, melodies are all in one channel, so the ideal value of N for this data should be large, and the ideal S should be zero (highest penalty) to prevent the melody from changing channels. Instead, we will determine N and S from our test data, where every measure is labeled as melody or not, and we will report how these parameters effect accuracy using the test dataset.

4.3.3 Melodic probability

To prepare for the dynamic programming step, we compute $\boldsymbol{M}_{m,c}$, which is the natural log of the probability of melody in channel c at measure m. (The feature values are different for each combination of m and c.) In the next step, note that if we find the labels with the greatest sum of log probabilities, it is equivalent to finding the labels with the greatest product of probabilities. Logarithms are used to avoid numerical underflow.

4.3.4 Dynamic Programming

We use dynamic programming to select the channel containing the melody in each measure. Algorithm 2 shows how the assignment of channels maximizes the sum of $\boldsymbol{M}_{m,c}$ values adjusted by subtracting $SP = -\log(S)$ each time the melody changes channels. The backtracking step is not shown since it is standard.[1]

4.4 Experiment and Result

4.4.1 Training

We tried different combinations of features, and the results are shown in Table 4.1 for windows with 5 measures ($N = 2$). The top 5 feature sets are shown along with the mean and standard deviation of accuracy across 5-fold cross-validation. Differences among the top feature sets are minimal. We use all features except velocity standard deviation.

Table 4.2 shows the results using each feature individually for 5-measure windows. This shows that all features offer some information (random guessing would be 1/15 or less than 7% correct), but no single feature works nearly as well as the best combination.

If we assume the melody appears in only one channel, which is mostly the case for this training dataset, we can consider the measure-by-measure melody channel results as votes, picking the channel with the majority of votes as the melody channel. Our best feature set (all but velocity standard deviation) gives an accuracy of 96% (2 errors out of 51 songs), using 5-fold cross-validation. In the next section, we relax the assumption that the melody appears in only one channel.

[1] https://en.wikipedia.org/wiki/Viterbi_algorithm

Result: For each measure, determine the channel containing the melody.
N is the number of measures, indexed from 0 to $N-1$;
C is the number of channels, indexed from 0 to $C-1$;
SP is channel switch penalty, a parameter; SP = -ln(S);
$\boldsymbol{A}_{m,c}$ is the accumulated score for measure m and channel c;
$\boldsymbol{B}_{m,c}$ stores the optimal channel number of previous measure;
$\boldsymbol{M}_{m,c}$ tells how melodic is channel c in measure m;

```
for i in [0 ... C) do
    A_{0,i} ← M_{0,i};
end
for m in [0 ... N) do
    for c in [0 ... C) do
        x ← A_{m-1,c} + M_{m,c};
        B_{m,c} = c;
        for i in [0 ... C) do
            y ← A_{m-1,i} + M_{m,c} − SP;
            if c ≠ i ∧ y > x then
                x ← y;
                B_{m,c} ← i;
            end
        end
        A_{m,c} ← x;
    end
end
```

Algorithm 2: Dynammic Programming

Table 4.1: Mean and standard deviation of accuracy in 5-fold cross validation using the top 5 feature sets, window size = 5. Here, nd means $note\_density$, pm means $pitch\_mean$, ps means $pitch\_std$, im means IOI_mean, is means IOI_std, vm means vel_mean, and vs means vel_std.

nd	pm	ps	im	is	vm	vs	mean	std
1	1	1	1	1	1	0	72.40%	7.20%
1	1	0	1	1	1	0	71.60%	7.06%
1	1	1	1	1	1	1	71.40%	8.26%
1	1	1	1	0	1	0	71.40%	7.70%
1	1	1	1	0	1	1	71.00%	8.83%

Table 4.2: Mean and standard deviation of accuracy in 5-fold cross validation using individual features, window size = 5

nd	pm	ps	im	is	vm	vs	mean	std
1	0	0	0	0	0	0	45.80%	7.22%
0	1	0	0	0	0	0	44.60%	8.11%
0	0	1	0	0	0	0	35.80%	6.50%
0	0	0	1	0	0	0	31.00%	2.55%
0	0	0	0	1	0	0	30.40%	5.64%
0	0	0	0	0	1	0	32.00%	5.87%
0	0	0	0	0	0	1	20.60%	4.10%

4.4.2 Testing

Our test dataset labels each measure with a set of channels containing melody. In measures with no melody, this is the empty set. In some measures, the melody is duplicated in different channels, so the label can can contain more than one channel. Note that if we can identify one channel containing the melody, it is simple to search for copies in the other melodies. Our algorithm labels *every* measure with exactly one melody channel. We consider the output to be correct either if it is in the set of true melody channels according to our manual labels, or if the label is the empty set. Typically, the empty set (no melody label) appears in introductions, endings, and measures where the melody channel rests. In these cases (approximately 12% of all measures), there is no clearly correct answer, so any output is counted as correct.

We evaluated accuracy on the test dataset with many values of N and SP. For each value of N, we used the best feature set as determined from the training data and then evaluated the system with different values of SP. The results are shown in Figure 4.2.

Since N and SP are optimized on the test dataset to obtain a best accuracy of 89.15%, there is some risk of overfitting parameters to the test data. Given more labeled data, we would have used a different dataset to select N and SP, and then we could evaluate the entire system on the test dataset. Instead, we argue that the system is not very sensitive to N or SP, so overfitting is unlikely. Figure 4.3 shows how accuracy is affected by varying the window size using an optimal value of $SP = 36$ (again, the window includes the measure $\pm N$ measures, so the window size is $2N + 1$). This figure shows that 5-measure windows worked the best, but windows up to about

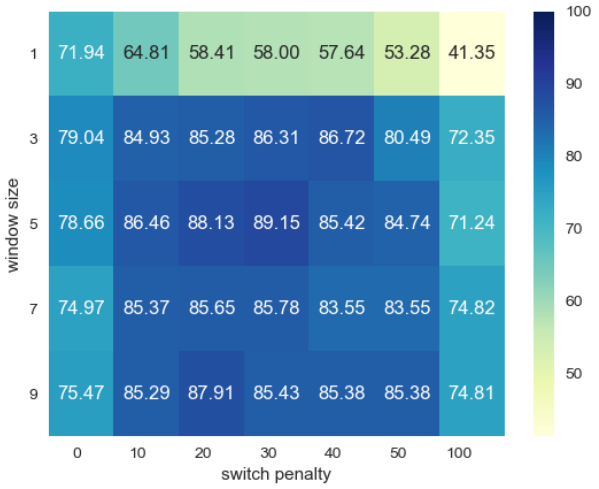

Figure 4.2: Accuracy for different values of window size and switch penalty.

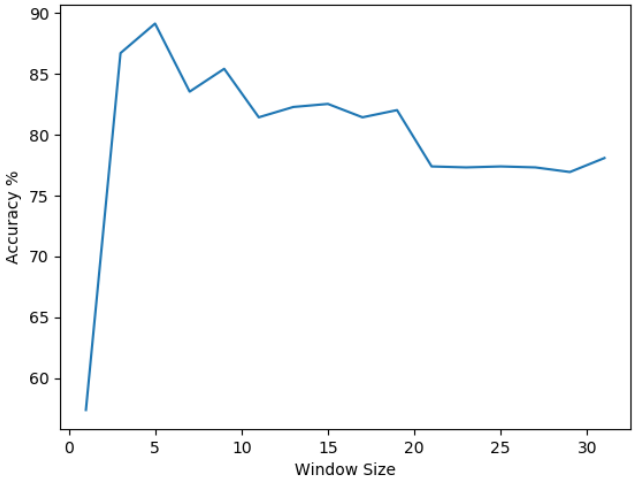

Figure 4.3: Accuracy on the test dataset vs. Window Size ($= 1 + 2N$) for Switch Penalty = 36.

15 measures also work well, with just a few percent variation in accuracy.

Figure 4.4 shows how the accuracy is affected by varying SP, the switch penalty, using the optimal value of $N = 2$. The best performance is obtained with SP between 30 and 38, but any value from 2 to 38 will achieve performance within a few percent of the best. Since both graphs are fairly flat around the best values of N and SP, the exact values of these parameters are not critical for good performance. In fact, we would expect the best values may depend upon style, genre, and other factors.

From the results, we can observe that the increase of window size helps the performance. The highest accuracy goes from 58.00% to 89.15% when the window size grows from 1 to 5.

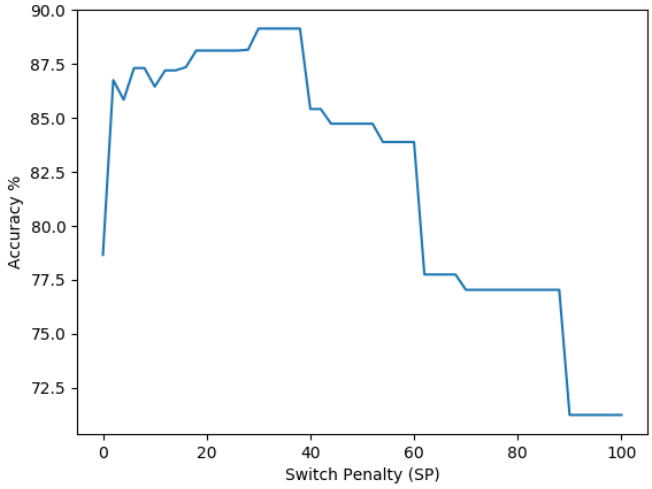

Figure 4.4: Accuracy on the test dataset vs. Switch Penalty with Window Size = 5. The highest accuracy is 89.15% using any Switch Penalty $\in \{30, 32, \ldots 38\}$.

However, accuracy does not continue to increase for even larger windows. We believe the size of 5 measures is large enough to obtain some meaningful statistical features yet small enough to register when the melody has switched channels. In the next section, we analyze some specific examples of success and failure. We also see that the switch penalty matters. When we set the penalty to zero, we get the locally best choice of melody channel at each measure, independent of other measures, but it seems clear that this "locally best, indepenent" policy is not particularly good, and this is why we introduced the Viterbi step to our algorithm. On the other hand, when the penalty is very large, we force all measures to be labeled with the same channel. This is not a good policy either, with at best 71.24% accuracy. The results show that our Viterbi step is effective in using context to improve melody identification.

4.5 Analysis

To better understand our approach, we analyzed some songs in our test dataset.

4.5.1 A Successful Sample

In most of the cases, this algorithm works well. For example, the figure 4.5 shows a clip from the popular song "Hotel California." In the figure, the melody is labeled by our algorithm in red (at the top) and other notes are shown in yellow. Notice that this melody is not particularly "melodic" in that it only uses two pitches and there is a lot of repetition. This is one illustration of the need for multiple features and statistical methods. The features for the melody channel have a higher likelihood according to our learned probabilistic model, and the melody is correctly identified.

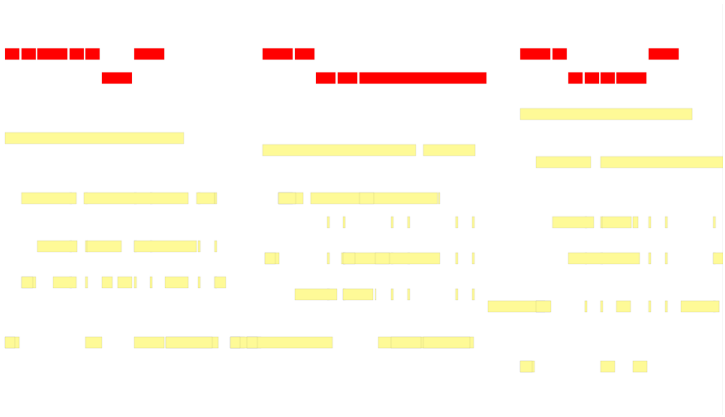

Figure 4.5: The melody detected in the song “Hotel California.”

4.5.2 Failed Samples

A failure case is shown in Figure 4.6. Here, the detected melody is shown in red at the top of the figure. The same (red) channel actually contained the melody in the immediately preceding channels, but at this point the melody switched to another channel, shown below in yellow. (In the figure, the lower melody is visually separated for clarity, but both channels actually occupy the same pitch range.) Evidently, the algorithm continued to label the top (red) channel as melody to avoid the switch penalty that would be required to label the melody correctly. In fact, the true (yellow) melody appeared earlier in the top (red) channel, so perhaps a higher-level analysis of music structure would also be useful for melody identification and disambiguation.

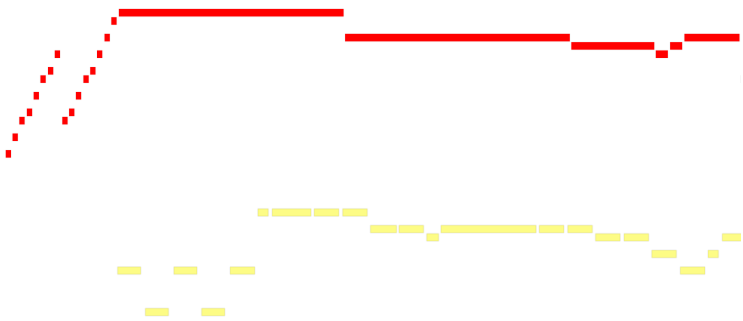

Figure 4.6: The algorithm identified the top (red) channel as melody of “Being,” but the correct melody is shown in yellow at the bottom.

Another song in our dataset is “Ali Mountain.” In this piece, at measure 12, the melody is split across two different channels, represented in red (darker) and yellow (lighter) in Figure 4.7. Taken together, the combined channels would be judged to be very melodic. However, when we consider the channels separately, it is hard to hear whether either is part of a melody, and our algorithm does not rate either channel highly. Since we assume that the melody will be played by one and only one channel within a measure, the melody is not identified in this test case.

Figure 4.7: Two channels ensemble the melody

4.6 Conclusion

In this paper, we contributed a novel algorithm to detect the melody channel for each measure in a MIDI file. We utilize a Bayesian probability model to estimate the probability that the melody is on a particular channel in each measure. We then use dynamic programming to find the most likely channel for melody in each measure considering that switching channels from measure to measure is unlikely. We obtained an overall accuracy of 89% on our test dataset, which seems to compare favorably to most other results in the literature. The lack of a large shared dataset prohibits a detailed comparison.

Our dataset, including Standard MIDI Files, melody labels, associated software, and documentation are available at the following website:
http://www.cs.cmu.edu/∼music/data/melody-identification.

Chapter 5

Bass Analysis

5.1 Introduction

Beside the melody, the bass part is also very important for music generation, listening and music analysis. The movement of the bass influences the inversion of harmony. Some times, the rhythm of the bass will be the spotlight in a piece of music. Thus, the identification of bass plays one of the key roles in music analysis. Also, harmony and rhythm analysis refer to the information in the bass. A change in the bass often indicates a change of harmony. Many efforts in automatic composition could benefit from training data consisting of isolated bass.

Compared to the melody analysis, it is not so difficult to do the bass analysis. The reason is that melody can appear in the top, middle or even in the bottom pitch range. However, the bass part seldom appears in a middle or high part. Thus, this problem can be solved by a relatively simple algorithm. This thesis will propose a simple algorithm combining to extract bass at the measure level.

5.2 Algorithm

The algorithm for bass analysis is relatively simple compared to melody analysis. Roughly, it consists two parts: 1) pitch mean calculation and 2) argmax selection. We can easily calculate the pitch mean for each measure. The bass channel for each measure is the channel that has the lowest pitch mean, as described by Algorithm 3

5.3 Experiment and Result

Since the algorithm for bass analysis is deterministic, which means there is no training process, we only label the test dataset for this part. As same with melody analysis, the MIDI files are read and transformed to the CSV text file format. Then the system runs the simple algorithm we just discussed. The criterion that we use to evaluate the algorithm is the same as with melody. The overall accuracy is 97.36%

Result: For each measure, determine the channel containing the bass.
N is the number of measures, indexed from 0 to $N-1$;
C is the number of channels, indexed from 0 to $C-1$;
$\boldsymbol{M}_{m,c}$ tells the pitch mean of channel c in measure m if there is no note, assign $+\infty$;
$\boldsymbol{A}_m$ tells the bass channel in measure m;
for m in $[0 \ldots N)$ **do**
  $c' \leftarrow \text{argmin}_c \boldsymbol{M}_{m,c}$;
  if $\boldsymbol{M}_{m,c} == +\infty$ **then**
    $\boldsymbol{A}_m \leftarrow null$;
  end
  else
    $\boldsymbol{A}_m \leftarrow c'$;
  end
end

Algorithm 3: Simple Bass Analysis Algorithm

12 songs (among the 22) get 100% accuracy. 9 other songs have the accuracy range of [90%, 100%). Only 1 out of 22 has the lowest accuracy, which is 80%. The reason is that in our simple algorithm, there is no restriction or penalty for channel switching. So when the pitch mean of the accompaniment is lower than the bass channel, the result will switch immediately. One method to alleviate this is to using an algorithm that has some penalty for the bass channel switch.

5.4 Conclusion

Compared to melody analysis, bass analysis is much easier. We use a bass version of the Skyline algorithm to solve this problem. It is easy to implement and the time complexity is linear of the input size.

Chapter 6

Chord Analysis

6.1 Introduction

Compared to melody and bass which provide a horizontal continuity through time, chords or harmony represent vertical connections across voice in music. Sequences of specific chords are used in analyzing higher levels of music functions. For example, the progress of the Dominant Chord to the Tonic Chord can be considered as evidence of an Authentic Cadence. The different modes of chords also provide different harmonic functions. In the same example, according to the inversion and the highest note, the cadences can be Perfect Authentic Cadence (the chords are in root position, and the tonic is in the highest voice of the final chord), root position Imperfect Authentic Cadence (similar to a perfect authentic cadence, but the highest voice is not the tonic) or inverted Imperfect Authentic Cadence (similar to a Perfect Authentic Cadence, but one or both chords is inverted). In undergraduate music education, Harmony is also one of the most important courses. Harmonic analysis involves human judgement and subjective decision making because music is full of non-chord tones that only suggest chords and harmonic function. Chord labels are ambiguous. However, chord analysis is very important because it provides an explanation of the music. One promising application is that harmonic analysis can be used as input to a music generation system. For example, an algorithm proposed by Brunner [3] generates music by an advanced machine learning model based on generated chords.
The analysis of chords includes several aspects: 1) indicating the starting and ending points of each chord; 2) providing the root of the chord; 3) providing the mode of the chord.

In our representation, the starting and ending points of each chord are indicated by time and duration, which are indicated in beats. The root is given directly. The mode is indicated roughly by the chord type, with additional details encoded as the pitch class set.

6.2 Dataset

One dataset that I used in chord analysis is from David Temperley, a corpus of 45 excerpts of tonal music. The excerpts are from the Kostka and Payne music theory textbook (Kostka and Payne 1984), hereafter referred to as the KP corpus. In this dataset, chords are labeled by the pair: timestamp and chord root, without type information. In this thesis, we contribute a chord

Table 6.1: Chord Notations

Chord Symbol	Chord Name	Example Notes
m	Minor	C Eb G
M	Major	C E G
aug	Augmented	C E G#
dim	Diminished	C Eb Gb
sus4	Suspended Fourth	C F G

analysis dataset with manual labels. There are 20 songs with 1890 chord labels in total. This dataset contains not only root information but also type information for each chord. We use this dataset to test our system.

It is very time-consuming to label the dataset manually. There are over 800 pages music notes for our current dataset and for each page it takes several minutes to label. Comparing to other fields such as Computer Vision, the labels of music is much hard to get.

6.3 Definition of Chords

Chord symbols, names, and examples can be found in Table 6.1. As we introduced in Chapter 3, there are only a subset of chord types (major, minor, augmented, diminished and suspended 4) presented in our output. The detailed pitch class information is also presented to compensate the simplification of chord type.

We decided to use simplified chord types because chord types are often ambiguous. For example, a C13 chord may or may not contain a fifth or ninth or eleventh. The 13th has the same pitch class as the 6th, so the chord might be labeled C6. In some Jazz manuscripts, a chord is labeled as a C6 or C13 when the melody has the sixth, and it is unclear and certainly optional whether the sixth should be played by piano or guitar, adding further confusion. We believe it is just as meaningful to label the chord type as C or C7 and then provide a set of pitches used (which indicate whether the fifth, ninth, eleventh, and thirteenth are included). The basic chord type indicates the chord function (major, dominant, etc.) and the pitch set gives details of how the chord function is realized.

6.4 Algorithm

Our main work was to integrate an algorithm from Temperley [20] into our overall analysis system. Temperley's algorithm is described here. In his algorithm, there are several rules:

1. Pitch Variance Rule: Try to label nearby pitches so that they are close together on the line of fifths.
2. Compatibility Rule: In choosing roots for chord spans, prefer certain TPC(Tonal Pitch Class)-root relationships over others. Prefer them in the following order: 1, 5, 3, b3, b7, ornamental. (An ornamental relationship is any relationship besides these five.)
3. Strong-Beat Rule: Prefer chord spans that start on strong beats of the meter.

4. Harmonic Variance Rule: Prefer roots that are close to the roots of nearby segments on the line of fifths.
5. Ornamental Dissonance Rule: An event is an ornamental dissonance if it does not have a chord-tone relationship to the chosen root. Prefer ornamental dissonances that are closely followed by an event a step or half-step away in pitch height.

However, in Temperley's algorithm, there is no type information. For example, the output will be "A" to indicate the "A chord". But we do not know it is "A minor" or "A Major" or even "A diminished". Thus, in our work, we provide a simple algorithm to predict the type of chord based on the output of Temperley's algorithm and the MIDI file, which is discussed as following steps:

1. In Temperley's output, the timestamp for each chord is given. The starting time (in milliseconds) and finishing time of each chord can be calculated based on that.
2. Given the starting and finishing time, we can get the notes in the corresponding MIDI file. Given the notes, we can get the pitch class set (each element is an integer from 0 to 11).
3. Based on the chord root and pitch class set, the chord type will be determined. For example, if the third ((root + 4) % 12) and fifth ((root + 7) % 12) exist, and the seventh does not, the chord type will be assigned as "Major".

6.4.1 Optimization: Subtracting Melody

For music expression, the melody often contains dissonant notes. Because we have a good melody classifier in Chapter 4, we propose an optimization method, which is to subtract the notes in the melody channel before we apply the chord analysis algorithm. By eliminating dissonant non-chord tones of the melody, we hope to improve the ability to find the correct chord label.

6.5 Evaluation

6.5.1 Evaluation Criterion

In our work, we evaluate the correctness of output by four methods:

1. For each 24^{th} beat, we assign 1 point to it when the corresponding predicted and labeled chords (both root and type) match.
2. For each 24^{th} beat, we assign 1 point when the corresponding predicted and labeled chord roots match.
3. For each chord, we give it 1 point in total, so if there are N 24^{th} beats, each 24^{th} beat gets 1/N points if the predicted chord matches with label (both root and type).
4. The fourth method is like the third, we only give credit when the chord roots match.

Because for each song, the number of chords is different, instead of showing the value of points, we show a percentage value, which is calculated from the assigned point value divided by the number of total potential points.

6.5.2 Result and Analysis

Table 6.5.2 shows the result of Temperley's algorithm. For different music, the accuracy varies a lot. The songs named SanFrancisco and lovelyrose has high accuracy with over 90% for each evaluation. However, the song named lovestory has an accuracy lower than 60% for each evaluation.

The gap between Eval1 and Eval2 (or Eval3 and Eval4) shows that some errors are introduced by the type prediction instead of root prediction. In some songs, for example, "california", "lovelyrose", "SanFrancisco", the gap is very small. However, for some of them such as "autumn", the gap is over 10%.

Table 6.2: Chord Test Result without Optimization

Music name	Eval1	Eval2	Eval3	Eval4
0adaf64db7fad3f972b4c4b4a7fb77bf	68.21%	74.67%	69.27%	76.01%
alimountain	85.20%	91.37%	80.36%	87.36%
areyouhappy	70.08%	75.71%	70.86%	77.12%
autumn	59.59%	74.64%	59.29%	72.13%
Being	79.97%	87.43%	76.90%	84.72%
california	90.87%	90.87%	91.56%	91.56%
dontwantanything	78.52%	80.59%	75.68%	78.40%
Heaven	85.87%	89.00%	83.35%	87.25%
lovelyrose	91.50%	91.50%	**92.99**%	**92.99**%
lovestory	49.53%	53.83%	50.35%	55.72%
mayflower	78.41%	84.75%	78.36%	86.93%
moonwish	86.00%	86.00%	85.66%	85.66%
onmyown	82.56%	83.12%	80.85%	81.09%
riverside	74.73%	74.73%	77.51%	77.51%
SanFrancisco	**91.96**%	**91.96**%	91.47%	91.47%
SayYouSayMe	70.12%	76.48%	69.85%	75.31%
skyoftaipei	71.33%	74.95%	66.75%	74.15%
smalltown	82.80%	89.36%	84.84%	90.31%
tenderness	84.74%	86.89%	83.43%	85.69%
Venus	87.35%	90.31%	90.17%	93.82%
Total mean	78.47%	82.41%	77.98%	82.26%
Total std	10.78%	9.16%	10.86%	8.98%

Table 6.5.2 shows the results that with the optimization. For each evaluation, the total mean value of accuracy shown an increase of 1.3%

6.6 Conclusion

In this chapter, we described a pop music dataset with thousands of chord labels and we tested the algorithm for chord analysis using this dataset. We also showed that Temperley's algorithm

Table 6.3: Chord Test Result with Optimization

Music name	Eval1	Eval2	Eval3	Eval4
0adaf64db7fad3f972b4c4b4a7fb77bf	69.54%	75.64%	69.93%	75.90%
alimountain	79.06%	84.74%	71.78%	77.82%
areyouhappy	68.28%	75.86%	69.81%	77.10%
autumn	64.61%	78.93%	65.90%	78.54%
Being	79.97%	87.32%	77.43%	85.23%
california	92.13%	92.13%	92.84%	92.84%
dontwantanything	81.30%	83.36%	79.08%	81.80%
Heaven	87.86%	91.18%	85.87%	90.38%
lovelyrose	91.50%	91.50%	93.11%	93.11%
lovestory	53.13%	57.42%	54.10%	59.54%
mayflower	77.42%	84.78%	77.55%	86.73%
moonwish	**95.28**%	**95.28**%	**93.92**%	**93.92**%
onmyown	83.04%	83.60%	81.83%	82.06%
riverside	78.43%	78.43%	80.55%	80.55%
SanFrancisco	93.86%	93.86%	93.82%	93.82%
SayYouSayMe	73.64%	80.57%	72.66%	78.72%
skyoftaipei	68.80%	71.75%	64.15%	69.64%
smalltown	86.33%	93.58%	89.35%	94.95%
tenderness	85.02%	86.24%	83.69%	84.97%
Venus	87.35%	90.31%	90.17%	93.82%
Total mean	79.83%	83.82%	79.38%	83.57%
Total std	10.67%	8.97%	10.99%	9.09%

can be at least slightly improved by using our automated melody identification to remove the melody from the music before performing chord analysis.

The system and experiments in this chapter contribute a number of things to the field of chord analysis by computer. It is easy to extend the work based on this work. The performance of the system is measured using some clear evaluation methods, providing an empirical baseline against which future work may be measured. We also believe the automated chord labeling is good enough as the first step in bootstrap learning, which would allow us to label a much larger database of music using more advanced statistical machine leanring methods.

Chapter 7

Structure Analysis

7.1 Introduction

It is obvious that music has structure. When we talk about a music, it is natural to have ideas such as "This piece's structure is AABA". For humans it is not very hard to detect structure by listening or analysing music notation, which could be either common practice music or a graphical representation. And for the task of music generation, the high level structure are usually produced by random number generator or manually assigned table. In this thesis, we will present a method that to analyze the structure information automatically from MIDI files and output the "AABA" information in the form that was introduced by Chapter 3.

7.2 Algorithm

The structure analysis algorithm can be divided into three parts: 1) calculation of a similarity matrix; 2) repetition section detection; 3) representation transformation.

7.2.1 Similarity Matrix

Similar with the work in audio music structure analysis as Klapuri et al. [10], we define a *Frame* to be all of the notes in a certain time duration. For example, in our work, the time interval is set to 200 milliseconds, which means all of the notes that overlap this time interval will be considered in the frame.

Similarity Matrix The similarity matrix $S_{i,j}$ means the similarity value of $Frame_i$ and $Frame_j$. Different similarity functions are possible. In this thesis, we test two categories of formulas to calculate similarity values. The inputs of the calculation are two frames and the output is a real number to represent how similar they are.

Maximum Sharing Rate In the *Maximum Sharing Rate* calculation method, $C_{i,j}$ is the number of common pitches. $T_{i,j}$ is the minimum of the number of pitches in $Frame_i$ or $Frame_j$. Or $T_{i,j} = min(|Frame_i|, |Frame_j|)$, where $|Frame|$ means the number of pitches in the frame.

Figure 7.1 is the similarity matrix of Alimountain in metric of Maximum Sharing Rate.

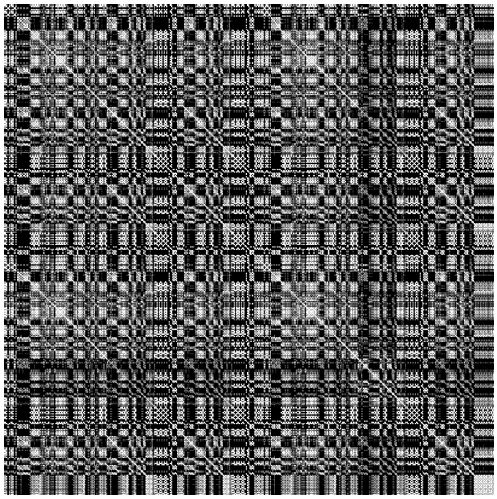

Figure 7.1: Maximum Sharing Rate Similarity Matrix for Alimountain

Chroma Vector Another metric that we will test in our thesis is the *Chroma Vector*. The chroma vector of one frame represents the pitch classes that are present. Suppose that in one frame there are C4 (pitch value is 60) and G4 (pitch value is 67) and C5 (pitch value is 72). In the vector, we have 12 slots represent the pitch classes from 0 to 11. The pitch class of the frame that we have is 0, 7 and 0. So the chroma vector for this frame is [2, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]/$\sqrt{5}$. The counts are divided by $\sqrt{5}$ to normalize the length to 1. The similarity value is the dot product of two chromatic vectors. The meaning of this calculation is that it considers the higher the dot product of two vector, the more similar they are. Figure 7.2 is the similarity matrix of Alimountain in metric of chroma vector.

7.2.2 Repetition Detection

When the similarity matrix is ready, we have the texture of the music. The next step is to detect the repetition fragment from the similarity matrix. It is easy to detect that for the repetition part, there is a long white line. So the repetition detection task is reduced to find the off-diagonal "long white lines" in the picture of similarity matrix.

In our algorithm, we rotate the original similarity matrix picture by 45 degrees, from the square matrix to a diamond matrix. The reason is that in the diamond matrix, each pixel on the same white lines have nearly equal y-coordinate, which is more convenient for computer vision algorithms. Here, by the rule of Occam's Razor, we apply a very simple algorithm to solve this problem. Before introducing our algorithm, we will first present our consideration and intuition.

We can observe from the picture that there are some points that are very white (the frame here is very similar) and some points that are grey (the frames here is not very similar). In music, there are many small variation even the they share the same structure. The white-grey patterns

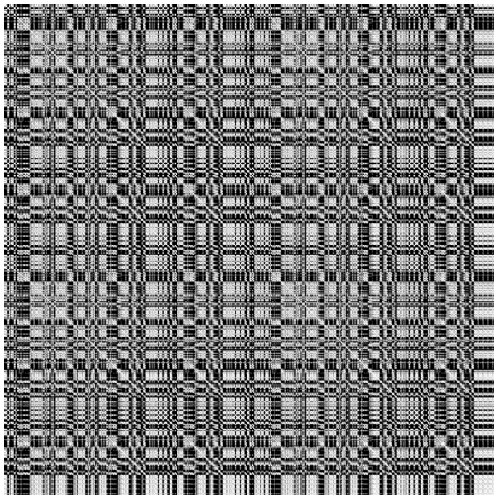

Figure 7.2: Chroma Vector Similarity Matrix for Alimountain

are very common and reasonable. Based on this observation and our intuition, we design an energy based algorithm and clustering method to detect the lines.

Energy Based Algorithm
Here we define the energy to be a real number from 0 to 255. And for convenience, we project the similarity value to a range of [0, 256). For a repetition structure in music, it usually has the same beginning and different variation. With this consideration, we design the **starting threshold**, which means the potential line will start when the similarity value is over a certain value, and we refill the energy to the maximum value. In our algorithm, we manually set it as 200. Then we scan the line from left to right, starting from the starting point. When we scan to a grey block ($0 <$ value ≤ 254), the energy will linearly decay by a factor based on how white the block is. And when we scan to a completely dark block (value = 0), the energy will exponentially decay by a decay factor. Algorithm 4 presents this in detail.

Line Clustering
Through the Energy Based Algorithm, we can get a number of short lines. In the Line Clustering step, we will form the short lines into a long line to represent the repetition part. In detail, first we build a graph: each node represent one line, and the edge connects two nodes that are near enough and have some overlap. Here, the definition of near is that the absolute difference of two y-coordinates is less than a threshold, which is 15 in our work. And the overlap means that two lines share a common interval on the x-axis.

7.2.3 Representation Transformation

As described in Chapter 3, the output of our system for structure analysis is a list where each element has the form of "measure", "duration" and "section". In this subsection we will introduce an algorithm to transform the similarity matrix "picture" into the structure representation.

1. Since the picture is symmetric around the diagonal line from upper left to lower right, we

Result: For each block, determine the energy value.
N is the number of rows in the **diamond picture**;
C is the number of columns in the **diamond picture**;
$\boldsymbol{Index}_{i,j}$ is the left most x-coordinate of the line bypass (i,j);
$\boldsymbol{Count}_{i,j}$ is the number of points in the line bypass (i,j);
DARK=0;
START_THRESHOLD=200;
line_index = 0;
for *i* in [*0* . . . *N*) **do**
    last_index = 0;
    last_energy = 0;
    cnt = 0;
    for *j* in [*0* . . . *C*) **do**
        data = diamond[i,j];
        if data $\geq$ START_THRESHOLD **then**
            if *last_index==0* **then**
                line_index += 1;
                cnt += 1;
                last_index = line_index;
                last_energy = MAX_ENERGY;
            end
        end
        if data $\leq$ DARK **then**
            last_energy = exp_decay(last_energy, data);
        end
        else
            last_energy = linear_decay(last_energy, data);
        end
        if last_energy $<$ FINISH_THRESHOLD **then**
            last_index = 0;
            last_energy = 0;
            cnt = 0;
        end
        else
            cnt += 1;
        end
        $Index_{i,j}$ = last_index;
    end
end

Algorithm 4: Energy Based Algorithm

will only consider the lower left triangle in the picture.

2. The red line in the picture with coordinate of (x1, x2, y1, y2) will be transformed as two segments of (x1, y1) and (x2, y2), which means that music from the time of x1 to y1 is similar with that from the time of x2 to y2.
3. We build an abstract graph based on segments. For each segment, we assign a node of it. For each pair of (x1, y1) and (x2, y2) generated from picture, we connect them by an edge. And for each segment pair (a, b) and (c, d), if their overlap is over a threshold (60% in our system), we connect those two nodes by an edge.
4. Then we assign each connected group a color (or a unique group number) to indicate the section number.
5. Sort the line segments increasingly with the key of the first coordinate (x1 if the line is (x1, y1)). In the final output, we quantize the start time and duration to units of measures and truncate the tail of section if it overlaps with the next one.

7.3 Evaluation

Figure 7.3 shows the result after the Energy Based Algorithm, the lines that meet our requirement are colored. In our work, we want to detect the most general structure information. It is easy to observe that in one piece of music, there are many small repetitions. To filter out the small repetitions, we set a threshold of the length of our lines. So we require that the repetition line should be at least $\frac{1}{10}$ of the total length of the original music. The result after the filtering and the clustering algorithm can be found in Figure 7.4. We can find that the general structure for Alimountain is ABBABB or AA in a more high level view.

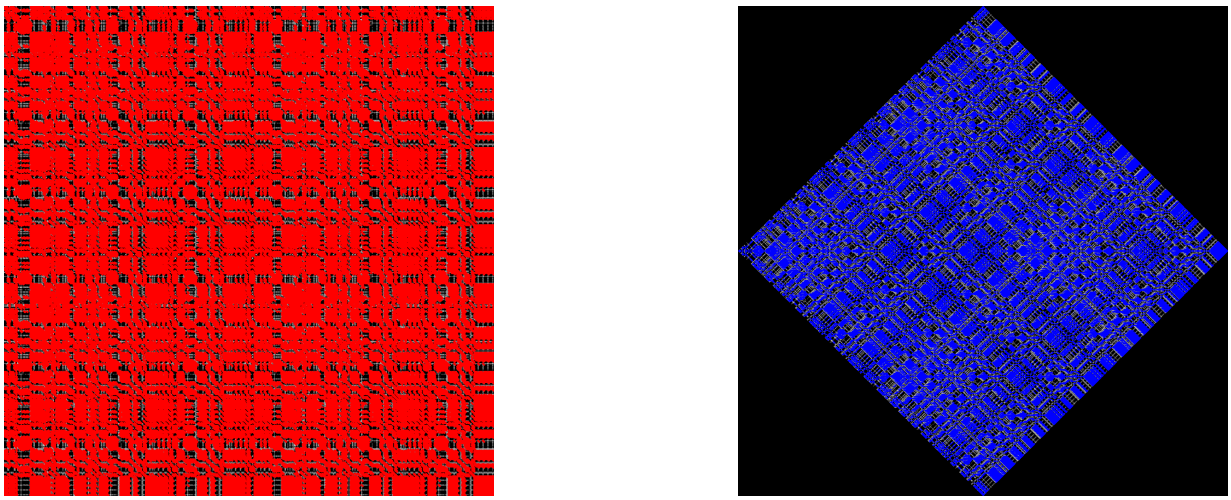

Figure 7.3: After the Energy Based Algorithm for Alimountain

The final output of Alimountain is shown in Table 7.3. The output shows that the structure for the song is BACA, where A's are measures #3 to #47 and measures #55 to #99. When we check the music, we find out that it predicts the main part correctly. However, there is no B or C section in the music. We notice that there is a tempo change around measure #5 from 76 beats

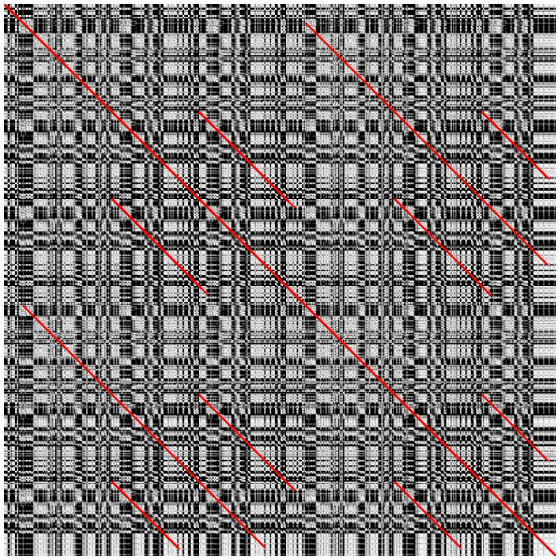

Figure 7.4: After the Filter and Cluster Algorithm for Alimountain

per minutes to 101 beats per minutes. This explains why our algorithm detects there is a separate section before A.

Table 7.1: Final Output of Alimountain

m	**d**	**s**
0	2	2
2	45	1
47	7	3
54	45	1

7.3.1 Evaluation Method for Structure Analysis

Given a reference and an analysis result, we want to evaluate the degree to which the result matches the reference. So in this section we describe such an evaluation method.

Figure 7.5 shows the reference and analysis result for Alimountain. For example, in the reference, the structure category A has two segments: one is measures #1 to #52; another is measures #53 to #101. In the analysis result, there are four different categories. Category A has two segments: one is measures #3 to #47; another is measures #55 to #99. Category B has one segment: measures #1 to #2. Category C and D also has one segment for each: measures #48 to #54 for C and measures #99 to #101.

In structure analysis, the same structure can have different names. For example, a structure ABAB is as same as structure BABA. So finding a quantitative evaluation can be reduced to a matching in a bipartite graph as follows:

1. We form categories in the reference as the left part in the bipartite graph, and categories in the analysis result as the right part in the bipartite graph.

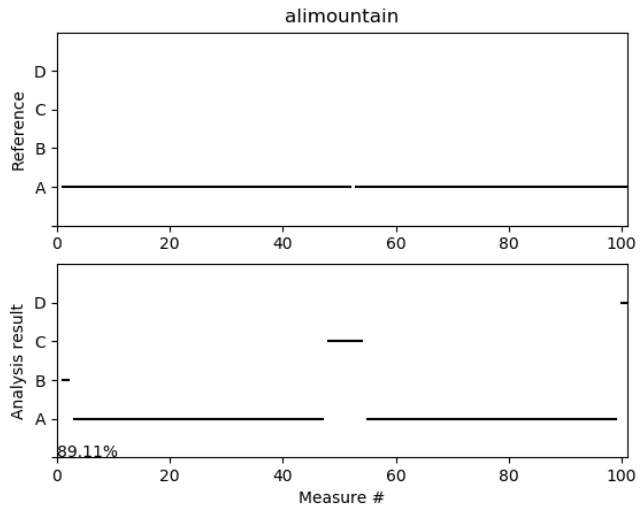

Figure 7.5: Reference and Analysis Result of Alimountain

2. We connect each node in the left part to each node in the right part with a weighted edge. The weight of the edge between X and Y is the overlap score of category X and Y.
3. Overlap scores are calculated as follows: Category X has segments x_1, x_2, ..., x_n. Category Y has segments y_1, y_2, ..., y_m. Let $Xscore_i$ be the maximum overlap between x_i and any one of segments y_j. X_{sum} is the summation of $Xscore_i$. Similarly, let $Yscore_i$ be the maximum overlap between y_i and any one of segments x_j. Y_{sum} is the summation of $Yscore_i$. The overlap score is the minimum of X_{sum} and Y_{sum}.
4. Now we have a weighted bipartite graph. We use the Kuhn-Munkres algorithm to calculate the matching with the maximum weight. The score is the sum of the weights in the matching.

The evaluation score for Alimountain is shown in the lower left on Figure 7.5. In this chapter we analyze 10 songs in total, and the evaluation scores are shown in Table 7.3.1

We also put the results for Moonwish (high evaluation score) and Yesterday (low evaluation score) in Figure 7.6 and Figure 7.7. For Yesterday, the reference structure is AABB. Comparing this to the analysis result, we can see that the repetition of A is detected but has some error in the beginning, and the repetition of B is not detected. As expected, this analysis received a low score. For Moonwish, it is easy to tell from the picture that the analysis result and the reference are nearly the same. Both are clearly AABA and the result receives a high score.

7.4 Conclusion

In this thesis we demonstrated that by utilizing the similarity matrix, a very simple algorithm has the potential to extract the high-level structure information from music given MIDI file. Structure analysis has many applications such as finding the chorus section, producing audio "thumbnails" by eliminating repetition, and making music search faster by reducing long files to main themes.

Table 7.2: Evaluation scores for 10 songs

Song Name	Evaluation Score
Alimountain	89.11%
Being	90.99%
california	**50.00**%
Heaven	78.79%
Lovelyrose	70.00%
Mayflower	60.82%
Moonwish	**97.22**%
Onmyown	66.67%
SanFrancisco	81.58%
Yesterday	65.48%

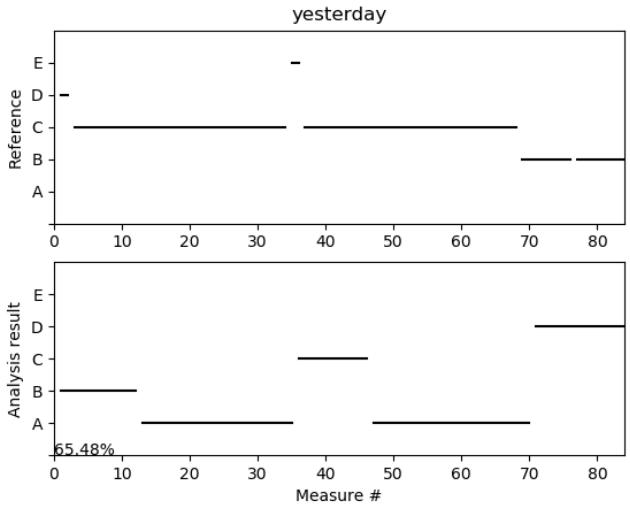

Figure 7.6: Reference and Analysis Result of Yesterday

Current music generation schemes based on sequence learning are not able to learn and generate typical music structures such as a 32-bar AABA form. Automatic structure analysis might be used in future music generation systems to create more typical song forms.

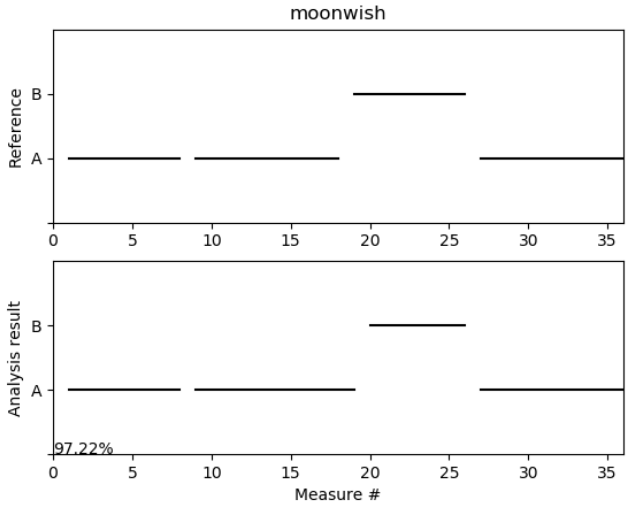

Figure 7.7: Reference and Analysis Result of Moonwish

Chapter 8

Future Work

8.1 Future Work on Melody

We believe further improvements could be made by studying failures. With bootstrapping techniques, it might be possible to obtain much more training data and learn note-by-note melody identification, which would solve the problem of melodic phrases split across two or more channels. Our current dataset is relatively small, so collecting a larger dataset could be beneficial for tuning this algorithm and developing others. With larger datasets, deep learning and other techniques might be enabled. Perhaps bootstrapping (or semi-supervised learning) techniques could be used starting with the present algorithm to label a larger dataset automatically. We also believe that music structure can play an important role in melody identification. Melodies are likely to be longer sequences that are repeated and/or transposed, and these non-local properties might help to distinguish "true" melodies as perceived by human listeners, even when the melodies are not particularly "melodic" in terms of local features.

8.2 Future Work on Chord

In our thesis we adopted Temperley's rule-based algorithm for chord analysis. If we had more labels, it would be promising to invent new data-driven algorithms for this task. Similar with melody, we believe that it is useful to use this algorithm as the bootstrap algorithm to label more data. Another potential direction is to render the music from MIDI and then utilize an audio model as a source of ensemble learning to raise the performance.

8.3 Future Work on Structure

Currently, the structure analysis is focused on very high level abstraction. There are potential details that are meaningful at the phrase level or motif level, which will enrich the music representation. These lower level and hierarchical structure details might be useful as the input to a music generation system. The similarity matrix algorithm has the time complexity of O(N^2) where N is the number of 24^{th} beats in the entire piece. When the MIDI music is very long,

such as more than 20 minutes, this algorithm could be very slow. Thus, an algorithm with better time complexity might be needed. The use of N-grams might be useful, and there is a substantial literature in biological sequence matching that might apply to MIDI data.

Bibliography

[1] Joshua Bloch and Roger B Dannenberg. Real-time accompaniment of polyphonic keyboard performance. In *Proceedings of the 1985 International Computer Music Conference*, pages 279–290, 1985. 4.3.1

[2] Michael J Bruderer, Martin F McKinney, and Armin Kohlrausch. Structural boundary perception in popular music. In *ISMIR*, pages 198–201, 2006. 2.3

[3] Gino Brunner, Yuyi Wang, Roger Wattenhofer, and Jonas Wiesendanger. Jambot: Music theory aware chord based generation of polyphonic music with lstms. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 519–526. IEEE, 2017. 6.1

[4] John Ashley Burgoyne, Jonathan Wild, and Ichiro Fujinaga. An expert ground truth set for audio chord recognition and music analysis. In *ISMIR*, volume 11, pages 633–638, 2011. 2.2

[5] Wei Chai and Barry Vercoe. Melody retrieval on the web. In *Multimedia Computing and Networking 2002*, volume 4673, pages 226–242. International Society for Optics and Photonics, 2001. 2.1

[6] Roger B Dannenberg and Masataka Goto. Music structure analysis from acoustic signals. In *Handbook of Signal Processing in Acoustics*, pages 305–331. Springer, 2008. 2.3

[7] Masataka Goto. A chorus section detection method for musical audio signals and its application to a music listening station. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5):1783–1794, 2006. 2.3

[8] Masataka Goto and Roger B Dannenberg. Music interfaces based on automatic music signal analysis: New ways to create and listen to music. *IEEE Signal Processing Magazine*, 36 (1):74–81, 2019. 2.3

[9] Cihan Isikhan and Giyasettin Ozcan. A survey of melody extraction techniques for music information retrieval. In *Proceedings of 4th Conference on Interdisciplinary Musicology (SIM08), Thessaloniki, Greece*, 2008. 2.1, 4.1

[10] Anssi Klapuri, Jouni Paulus, and Meinard Müller. Audio-based music structure analysis. In *ISMIR, in Proc. of the Int. Society for Music Information Retrieval Conference*, 2010. 2.3, 7.2.1

[11] Fred Lerdahl and Ray S Jackendoff. *A generative theory of tonal music*. MIT press, 1985. 2.3

[12] Jiangtao Li, Xiaohong Yang, and Qingcai Chen. Midi melody extraction based on improved neural network. In *Machine Learning and Cybernetics, 2009 International Conference on*, volume 2, pages 1133–1138. IEEE, 2009. 2.1

[13] Liu Li, Cai Junwei, Wang Lei, and Ma Yan. Melody extraction from polyphonic midi files based on melody similarity. In *Information Science and Engineering, 2008. ISISE'08. International Symposium on*, volume 2, pages 232–235. IEEE, 2008. 2.1

[14] Hyungui Lim, Seungyeon Rhyu, and Kyogu Lee. Chord generation from symbolic melody using blstm networks. *arXiv preprint arXiv:1712.01011*, 2017. 2.2

[15] Bryan Pardo and William P Birmingham. Algorithms for chordal analysis. *Computer Music Journal*, 26(2):27–49, 2002. 2.2

[16] Colin Raffel. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. Columbia University, 2016. (document), 4, 4.2

[17] Matti Ryynanen and Anssi Klapuri. Automatic bass line transcription from streaming polyphonic audio. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 4, pages IV–1437. IEEE, 2007. 2.1

[18] Man-Kwan Shan and Fang-Fei Kuo. Music style mining and classification by melody. *IEICE TRANSACTIONS on Information and Systems*, 86(3):655–659, 2003. 2.1

[19] Umut Simsekli. Automatic music genre classification using bass lines. In *2010 20th International Conference on Pattern Recognition*, pages 4137–4140. IEEE, 2010. 2.1

[20] David Temperley. An algorithm for harmonic analysis. *Music Perception: An Interdisciplinary Journal*, 15(1):31–68, 1997. (document), 2.2, 6.4

[21] Bertin-Mahieux Thierry, PW Ellis Daniel, Whitman Brian, and Paul Lamere. The million song dataset. In *ISMIR 2011: Proc. the 12th InternationalSociety for Music Information Retrieval Conference, October 24–28, 2011, Miami, Florida. University of Miami*, pages 591–596, 2011. (document), 4, 4.2

[22] Alexandra Uitdenbogerd and Justin Zobel. Melodic matching techniques for large music databases. In *Proceedings of the seventh ACM international conference on Multimedia (Part 1)*, pages 57–66. ACM, 1999. 2.1

[23] Sudha Velusamy, Balaji Thoshkahna, and KR Ramakrishnan. A novel melody line identification algorithm for polyphonic midi music. In *International Conference on Multimedia Modeling*, pages 248–257. Springer, 2007. 2.1

[24] Zhao Wei, Li Xiaoli, and Li Yang. Extraction and evaluation model for the basic characteristics of midi file music. In *Control and Decision Conference (2014 CCDC), The 26th Chinese*, pages 2083–2087. IEEE, 2014. 4.1

[25] Hanqing Zhao and Zengchang Qin. Tunerank model for main melody extraction from multi-part musical scores. In *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2014 Sixth International Conference on*, volume 2, pages 176–180. IEEE, 2014. 2.1