Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Master's Thesis

An Analysis of Automatic Chord Recognition Procedures for Music Recordings

submitted by
Nanzhu Jiang

submitted
February 4, 2011

Supervisor / Advisor
Priv.-Doz. Dr. Meinard Müller

Reviewers
Priv.-Doz. Dr. Meinard Müller
Prof. Dr. Michael Clausen

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement under Oath

I confirm under oath that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, _____ _____

(Datum / Date) (Unterschrift / Signature)

Acknowledgements

First of all, I am thankful to my supervisor from the bottom of my heart, Meinard Müller, whose excellent guidance, support and patience have always inspired me to work on this thesis. Furthermore, it is him who opens the door of music processing to me and let me find my real interest during my master studies.

I would like to express my deepest gratitude to Peter Grosche. He has made valuable supports in a number of ways such as the suggestions and discussions about problems, patiently helping me to arrange figures and tables of the thesis. His passionate attitude towards research also encouraged me when I get frustrated at the difficulties.

This thesis would not have been possible without the help of Verena Konz. She has precisely labeled the pieces of music which are of great importance for me. I am grateful for all her supports from the beginning of my thesis to the end. I would like to express the gratitude to her for the many music knowledge she explained as well as many many useful information about daily life.

I am indebted to many of my friends who support me during the writing of my thesis. Thomas Prätzlich, Philipp von StypRekowsky, Zhe Zuo, Haichao Guan, Jing Cui, Yuan Gao, Yecheng Gu and Qian Ma have all helped to proofread my thesis. They have given me the detailed and helpful feedback. I am grateful for their help. In particular, some of them were quite busy with their own work, however they still took their time for correcting my errors and offered suggestions.

I would like to specially thank Shuyan Liu and Shujie Li, for always supporting me during the whole period of my thesis. I can still remember the depressed time when I was frustrated by so many things messed up together, it is them who always stay beside me and cheer me up.

Finally, many thanks to my dear parents. Thanks for their unlimited love and care. Thanks very much for their understanding and let me pursue my dream so long and so far away from home.

Abstract

Chord recognition task is to split up a piece of music into segments and assign each of them a chord label according to the analysis of the harmonic content. Making the chord recognition task process audio recordings automatically will be of great help for music information retrieval. Most of the existing chord recognition systems proceed as follows. In the first step, a given audio recording is converted into a sequence of chroma features. In the second step, the feature sequence is passed into chord recognition module in which the features are assigned with chord labels. However, although much research has been done, there is little understanding of the effect of the different processing stages and of the various parameter settings on the final recognition result. In this thesis we analyze the different stages of typical automated chord recognition systems. In particular, we consider several types of chroma features as well as several chord recognition methods based on simple templates and more statistical involved pattern matching. As the main contribution, we conduct extensive experiments to evaluate the impact of different modules. In particular, we investigate the role of the parameters from both the feature side and the recognizer side systematically to reveal how they influence the overall performance. Our aim is to better understand the effect of different stages and their interaction as well as to indicate directions towards potential improvements. Finally, we add some additional processing techniques such as detuning compensation, harmonic-percussive source separation and beat-synchronization, and briefly examine their influence on chord recognition results.

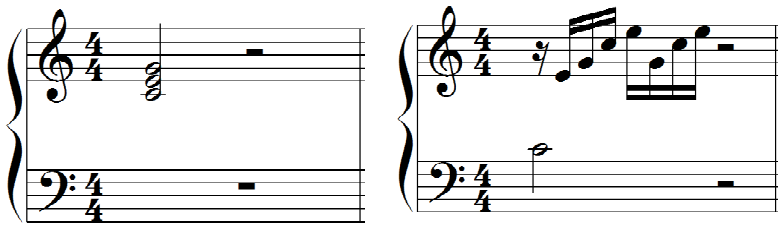
Contents

Chapter 1

Introduction

1.1 Music Background

A chord is defined as the simultaneous sounding of two or more different notes [21]. Normally the notes which form the chord are played together just like a musician presses the corresponding keys at the same time. In some special cases, these notes are played separately yet successively as a musician press the key one by one. Such cases include arpeggios and broken chords. Figure 1.1 illustrates a chord with the notes played simultaneously and Figure 1.1(b) illustrates the arpeggio, where the notes are played separately. Chords themselves and their progression are very important because they compose the harmonic content of a music piece. The analysis of harmonic content is essential in the Western tonal music, thus chords as the fundamental component play a crucial role for the understanding of such music [29]. Furthermore, being a higher-level representation of a music piece compared to the note-level representation, chords bring more insight to the analysis of the music structure and therefore assist many music information retrieval applications such as cover song identification and music segmentation.



(a) (b)

Figure 1.1. (a) C major triad played simultaneously, (b) C major triad in arpeggio.

By looking at the number of distinct notes[1] which compose a chord, we categorize the chords as "triad", "seventh", "ninth", etc. Figure 1.2 illustrates a triad, seventh and ninth chord based on the root note C. The most commonly used chords in music pieces

[1]Here, distinct notes means the notes which are in different pitch classes. For the definition of pitch class, see chapter 2

are "triad" chords. The name "triad" indicates that such chord is composed by three distinct notes. However, since notes which in different octave yet in the same pitch class may be considered as one note when people analyze the chord, it is better to count the number of distinct "pitch classes" or "chromas" instead of distinct notes. Figure 1.3 illustrates how notes are mapped to chroma[2]. To avoid confusion, in this thesis when we mention a component note of a chord, we actually treat all the notes in a pitch class as one note disregarding their octave information.

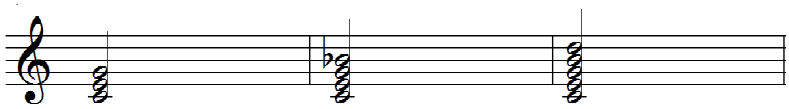
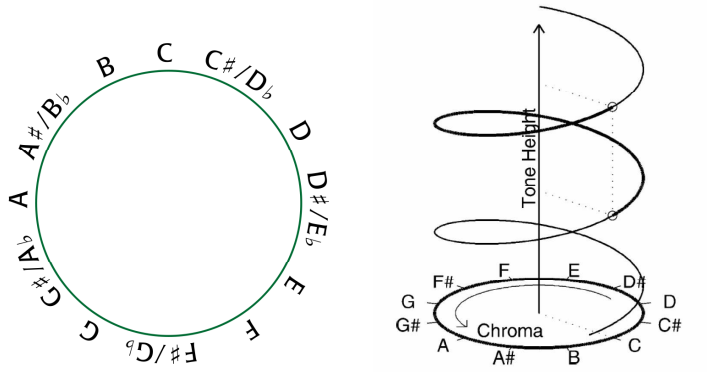


Figure 1.2. Three types of chords with different number of notes based on root note C. From left to right: C major triad, C dominant seventh and C major ninth.



(a) (b)

Figure 1.3. (a) Chromatic circle. (b) Shepards helix of pitch perception [2].

The component notes of a chord are not randomly selected but chosen according to the *musical interval*. It is the musical distance between the pitches of simultaneously sounding notes. The chords can be classified in this way as "major", "minor", "augmented", "diminished" indicating the component notes of these chords have different intervals. Figure 1.4 illustrate these four kind of chords with different intervals. Moreover, every component note has its own name closely related to the intervals. Taking the major triad[3] as an example, the lowest note is called *root note*, and the musical intervals of all other component notes are based on this note; the highest note is called fifth meaning it has a fifth interval from the root note; the middle note is called the third meaning it has a major third interval from the root. For the case of a minor triad, the lowest and highest note are the same for the major triad, the only difference lies in the middle note: this time it is the note which has a minor third interval.

Another essential concept about chords is the *inversions*. The concept of a chord inversions is introduced due to such cases that the root note of a chord is not at the lowest position, but somewhere higher. Figure 1.5 shows the three variations of C major triad, with the

[2]Figure 1.3 (a) is reproduced from `http://en.wikipedia.org/wiki/Chromatic_circle`

[3]Here we mean the major triad is in root position.

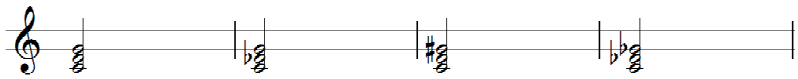


Figure 1.4. Chords with different musical intervals based on the root note C. From left to right: C major triad, C minor triad, C augmented triad and C diminished triad.

left one being the normal C major, the middle one the first inversion and the right one being the second inversion. The lowest note of a chord is named as *bass note*. In the first inversion of C major, the bass note is the third (note E), and the fifth(note G) and the root(note C) are stacked above it. In this case, the root note is shifted one octave higher. In the second inversion, the bass is the fifth(note G) with the root (note C) and the third above it, and both of them are shifted one octave higher.

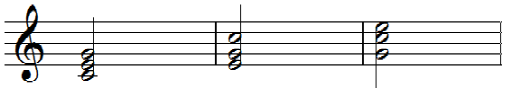


Figure 1.5. C major triads in root-position and inversions. From left to right: root-position, first inversion and second inversion.

As discussed above, the chords are classified by the number of component notes and also by the musical intervals between the notes. Moreover, for each type of chord, we have 12 distinct instances because we have 12 pitch classes which can be used as root note. For example, for the case of a major triad, we have C major triad, $C^{\sharp}$ major triad, D major triad, and so on. Figure 1.6 shows all 12 major triads with the red notes indicating the respective root note. In this thesis, if not specially declared, we will assume that all the chords we consider are triad chords, and we notated them by indicating the root note and type of intervals. For example, `C` is the short form to denote the C major triad and `Cm` to denote C minor triad.

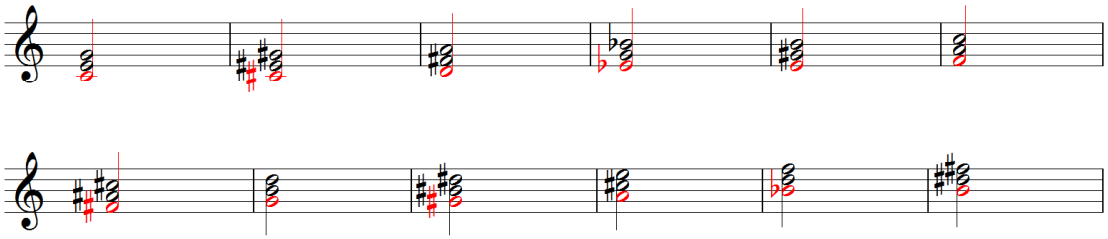


Figure 1.6. All 12 major triads, the note with red color indicates the respective root note.

1.2 Chord Recognition Task

In this thesis, chord recognition refers to the task of analyzing the harmonic content of music recording where the audio representation is first split up into segments and then each segment is assigned a chord label. The segmentation specifies the start time and end time of a chord, and the chord label specifies which chord is played during this time periods.

The motivation to automate the process of chord recognition consists of three aspects. Firstly, although the task is not difficult for trained musicians, it is kind of time-consuming and tedious repeated work. In the recent years, the music audio files are digitized and millions of them are saved in the databases or on the internet, it is impossible for musicians to label the pieces one by one. Secondly, chord labels retrieved from audio files as mid-level feature representation will largely help the high level music information retrieval tasks such as music structure analysis or segmentation, cover song identification, genre classification and other content-based retrieval tasks. Thirdly, an automated chord recognition system can assist musicians to transcribe an audio file more quickly and precisely. There is large demand of chord transcription from music fans who want to re-interpreted pop music and jazz music by guitar or piano, therefore making chord recognition automatic will be of great help to both professional musicians and music fans.

1.3 System Framework

A typical chord recognition system consists of four stages: feature extraction, pre-filtering, pattern matching and post-filtering [4]. The last step is optional. The stage *feature extraction* transforms an audio file into some musically meaningful representations. The stage *pre-filtering* performs smoothing on features which blend a single feature with the nearby context. The stage *pattern matching* proceeds in two steps. Firstly, one defines or learns the patterns of certain chords; such pattern can be a template of feature or a statistical model. Secondly, by comparing similarity between a feature with all the pre-defined chord patterns, we select the pattern which fit the feature the most to be the predicted chord label. This label is the final output. Alternatively, one can add a further *post-processing* stage which smooth the predicted label candidates over time.

In our implementation of the system, we merge the previous referred feature extraction and pre-filtering together as one stage, since our features already integrate internal parameters which control the smoothing effect. Also we treat pattern matching and post-filtering together as one stage, which we called chord recognition stage. We will discuss several pattern matching methods, but only one of them is combined with post-filtering. Figure 1.7 shows the general stages of our system[4].

In the recent research, chord recognition system is designed to process both the real digitized audio and its symbolic form such as MIDI. We mainly focus on real audio files in this thesis. An example of chord recognition results is shown in Figure 1.8, and the

[4]The figure is reproduced from Thomas Prätzlich's slides of Chord Recognition, seminar talk of Music Processing 2010.

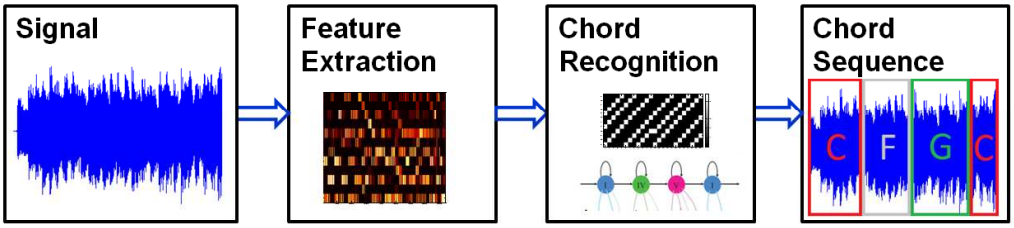


Figure 1.7. Overview of our chord recognition system.

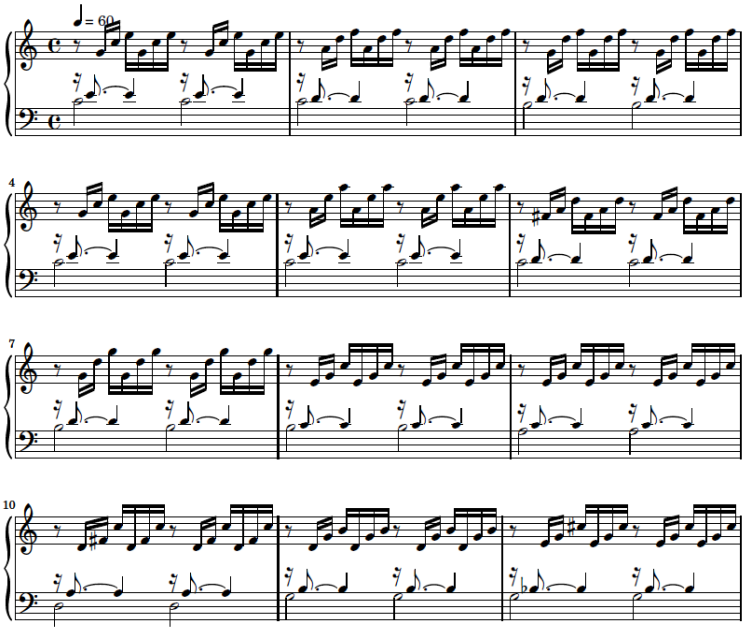
corresponding ground truth labels are shown in Table 1.1. In Figure 1.8(b), we show the output of a chord recognition system for the classical piece of music *Bach BWV846*. The results of four different versions are visualized in our *Interpretation Switcher* demo, with the top one being the ground truth for MIDI version, and the others being the computed result for three different interpretations played by various performers on different instruments. Each colored interval represents a certain chord and its length corresponds to the duration of that chord. By comparing the difference between any of the three computed results with the ground truth, we can easily find recognition errors such as the 2nd measure of the Koopman's interpretation, which wrongly recognize `Dm` represent by pink color as `F` represent by green color.

start time (s)	end time (s)	chord name	chord color
0.100	4.000	`C`	yellow
4.100	8.000	`Dm`	pink
8.100	12.000	`G`	blue
12.100	16.000	`C`	yellow
16.100	20.000	`Am`	red
20.100	24.000	`D`	cyan
24.100	28.000	`G`	blue
28.100	32.000	`C`	yellow
32.100	36.000	`Am`	red
36.100	40.000	`D`	cyan
40.100	44.000	`G`	blue

Table 1.1. Ground truth chord labels of *Bach BWV846*.

It needs to be mentioned that the chord recognition task is somehow ill-defined. Although most of the chords in a piece of music can be uniquely determined, there are some cases where even the chord labels written by musicians may differ from each other. Such cases include for example omitted notes of a chord or added notes which do not belong to the chord, and ambiguous chords like C major seventh and E minor with an added minor sixth. All these cases will make the chord labeling process an ambiguous task for musicians.

To avoid such diversity, we take the same dataset that many other groups use, the 180 Beatles songs for which uniquely determined chord labels exist. The chords are written by Christopher Harte [13]. Besides this, we also include four classical pieces – all the chords are precisely labeled by a trained musician, Verena Konz.



(a)

(b)

Figure 1.8. (a) Score of *Bach BWV846*. (b) corresponding Interpretation Switcher visualization of chord recognition output for the first 11 measures, with the top line being the ground truth, and other three being the computed results.

In order to make our results comparable to the results of other people who research in the same area, we take conventions of MIREX[5] chord competition in the implementation of our chord recognition system. Firstly, our evaluation uses the test data of 180 Beatles songs, which is exactly the test data of MIREX. Secondly, the chords we try to recognize are not the whole chord family but a subset consisting of 12 major triads and 12 minor

[5]The Music Information Retrieval Evaluation eXchange (MIREX) is a community-based formal evaluation framework where research groups can submit their system to join the competition in certain fields such as chord recognition, beat tracking, and so on.

triads, all other chords are forced to be mapped to one of these 24 triads. Thirdly, a strict rule is defined to describe how the original chord labels are mapped to one of the 24 triads. Here, we use the interval comparison of the dyad which takes into account only the first two intervals of each chord [16]. Thus, augmented and diminished chords are mapped to major and minor, respectively. This is also one of the mapping conventions used in MIREX.

1.4 Mathematical Formulation

In this section, we give a formal definition of the chord recognition problem.

Definition 1.1 *Suppose a music* audio file *is represented as $\mathcal{A}$ with its duration represented as T ($T > 0, T \in \mathbb{R}$). Then mathematically the audio file is a function $\mathcal{A} : [0, T) \to \mathbb{R}$. $[0, T)$ is called the* temporal domain *or time line of $\mathcal{A}$.*

Definition 1.2 *For a given time line $[0, T)$, we associate a segmentation into frames as follows. Given a frame length parameter $d \in \mathbb{R}$, we define $\mathbf{f}_n := [t_{n-1}, t_n)$ for $n \in \mathbb{Z}$. Then the frames associate to $[0, T)$ are given by the $\mathbf{F} = \{\mathbf{f}_n | n \in [0 : N]\}$, $N := \lceil \frac{T}{d} \rceil$.*

Definition 1.3 *We define a finite set Λ referred to* chord label set. *Furthermore, we define that Λ consists of* chord labels *$\lambda \in \Lambda$ that refer to the twelve major and minor triads, i.e.,*

$$\Lambda = \{\mathtt{C}, \mathtt{C}^\sharp, \ldots, \mathtt{B}, \mathtt{Cm}, \mathtt{C}^\sharp\mathtt{m}, \ldots, \mathtt{Bm}\}. \tag{1.1}$$

Definition 1.4 *For an arbitrary time frame $\mathbf{f}_n$, the chord recognition for a frame is to assign a chord label $\lambda_{\mathbf{f}_n} \in \Lambda$ to frame $\mathbf{f}_n$. Furthermore, suppose the temporal domain of $\mathcal{A}$ is represented by a sequence of frames as $[\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_N]$, then the chord recognition task for $\mathcal{A}$ consists in assigning to each frame $\mathbf{f}_n$, a chord label $\lambda_{\mathbf{f}_n} \in \Lambda$.*

1.5 Contribution

The motivation of this thesis is not aimed at building up a perfect chord recognition system but to analyze the effect of different stages of the system and their interactions. In particular, the influence of different parameter settings are examined by the extensive experiments which we conduct.

1.6 Organization of Thesis

Since the algorithm of our chord recognition procedure is designed in a modular fashion, the structure of this thesis is also arranged corresponding to system implementation. Figure 1.9 shows the flowchart. Our system starts from the very left with audio file as an input, following the successive steps of feature extraction, chord recognition and evaluation. Finally, we obtain some statistical results which reflect the performance of the system.

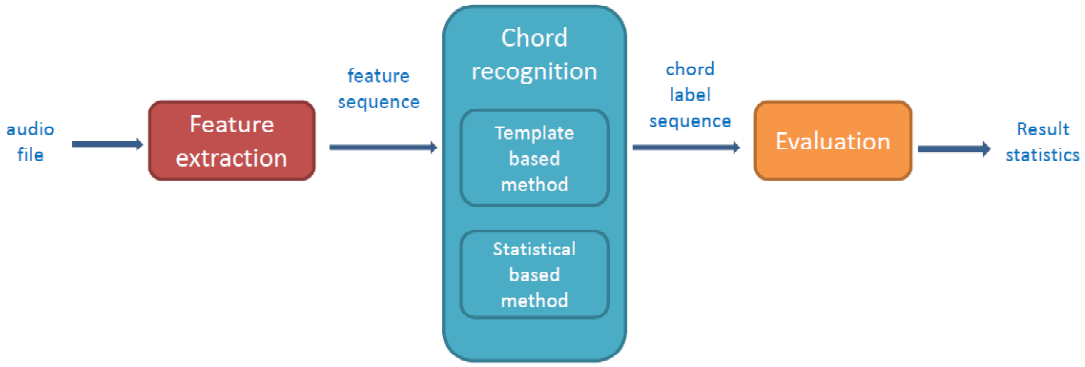


Figure 1.9. Module of thesis structure.

The colored boxes in Figure 1.9 correspond to the chapters in this thesis, and the order of boxes is exactly the same order of chapters. A very important part lies in the chord recognition module, another part of equal importance lies in the evaluation part, where we analyze the recognition results and make the conclusions based on the results. The chord recognition stage involves two different methods, and each method also consists of several variations. Therefore, we split the part of chord recognition into two chapters to describe and emphasize different aspects.

- In Chapter 2, we will describe how we implement *feature extraction*, in other words, how we transform a given audio file into some other representation which carry the music properties important for the chord recognition task and omit or suppress other information which is not relevant for the task.

- In Chapter 3, we discuss some methods of *template-based chord recognition*. These recognition methods measure the distance or similarity between a feature vector and each of the predefined chord templates, and select the chord whose template yield the minimum distance or maximum similarity.

- In Chapter 4, we introduce some methods of *statistical model-based chord recognition*. The methods in this chapter involve a previous training stage to learn a statistical model before the actual recognition, then in the recognition stage the parameters of the model are passed to the pattern matching methods such as Mahalanobis distance or Gaussian probability based methods. Finally, we examine all matches between the chord patterns and the given feature, and select the chord who has the best match as the determined chord for that feature.

- In Chapter 5, we discuss how we conduct the extensive *experiments* to inspect the effect of each stage of the chord recognition system. The parameters settings from both the features and the recognizers are evaluated and important discussions are presented. Furthermore, additional pre-processing techniques such as detuning compensation, harmonic-percussive source separation and beat-synchronization are add

to our chord recognition system, we briefly examine their influence by comparing the results with and without such techniques..

- At last in Chapter 6, we make *conclusions* about the whole thesis and discuss about *future work*.

Chapter 2

Feature Extraction

An audio file contains full of music information which consists of played notes, melody, harmony, beat, tempo, timbre of different instruments, dynamics of the sound, etc. For a specific music processing task, only some of them is relevant and useful. Therefore we need to select the most important information related to the specific task. Usually we name this processing stage as *feature extraction*, being the procedure of transforming an audio file into a musically meaningful representation which keeps the most task-related musical properties while suppressing other unrelated information. At the same time, the form of the representation should be appropriate for the next processing stage.

As the aimed music processing task discussed in this thesis is the chord recognition task, we need to extract audio features which emphasize the musical properties that refer to the aspect of harmony. Such musical properties can be the chord progression, the melody and component notes of a chord and their harmonics. Furthermore, a good feature should not only be able to capture the music properties mentioned above but also suppress or be invariant to some unrelated properties such as timbre and tempo.

As the chord itself is fully determined by its component notes, theoretically if the notes are known, the chord could be identified. Thus the capture of notes plays a crucial role in the feature extraction stage. Moreover, it is better to use pitch class [1] instead of single note to describe the form of the chord since human beings' perception of chords is irrelevant to the octave information of a note. Therefore our designed features should be able to project the information for notes within the same pitch class and distinguish the notes in different pitch classes.

The reason that the features which used in chord recognition being invariant for timbre and tempo is obvious. For example, if two interpretations of a music piece are played by different instruments which yield different timbre, the underlying notes are still the same and therefore the chords formed by these notes are the same as well; if two interpretations are interpreted with different speed, the underlying notes are still the same just their

[1] A pitch class is introduced due to the fact that human's perception for pitch is periodic. According to [23], a pitch can be separated into two components. One is referred to the tone height describing the octave information, the other is the chroma or pitch class. The pitches in the same pitch class sound to have similar "quality" or "color". For example, the pitch class C is a set contains note C0,C1,C2, and so on.

duration change, which means the chords remain the same.

In the following sections, we introduce several features with all of them fulfilling the requirements mentioned above. In Section 2.1, we present pitch features which serve as basis of the other features we use in our experiments. We describe how an audio file is decomposed into spectral bands and then transformed into pitch features with the energy of each band within a short time summing up together. Then based on pitch features, we derive the following features which are in chroma[2] representation in the subsequent sections. First, in Section 2.2, we talk about the common chroma features which summing up the spectral coefficients of the corresponding pitch features in the same chroma class. In Section 2.3 we modify the chroma features by performing logarithm compression on the spectral coefficients before summing up. Then, in Section 2.4, adding a further degree of abstraction by considering short-time statistics over energy distributions within the chroma bands, we obtain CENS (Chroma Energy Normalized Statistics) features, which constitute a family of scalable and robust audio features. To boost the degree of timbre invariance, a novel family of chroma-based CRP audio features has been introduced [25, 24]. We briefly describe CRP features in Section 2.5. Finally, CISP features from Dan Ellis's group will be presented in Section 2.6.

Note that except CISP features, the implementations of MATLAB functions of all other features, respectively the `Pitch`, `CP`, `CLP`, `CENS`, `CRP` we mentioned in this thesis can be found in the Chroma Toolbox, which can be obtained in [22].

2.1 Pitch Features

2.1.1 Pitch Decomposition

In a first step, we decompose a given audio signal into 88 frequency bands corresponding to 88 musical notes from $A0$ to $C8$ which are of equal-tempered scale. Here we introduce some properties of a musical note. Firstly, a musical note can be identified by its MIDI pitch[3] p, e.g., the note $A4$ corresponds to $p = 69$. Secondly, each note is associated with a certain frequency range with fixed center frequency, e.g., the note $A4$ has center frequency 440 Hz. Furthermore, for each of the notes, its MIDI pitch number is related to its center frequency in some logarithm fashion:

Let p denote the pitch number, $p \in [1:120]$, and let f_p denote the center frequency of the pitch, then we have the relation:

$$f_p = 2^{\frac{p-69}{12}} \cdot 440 \tag{2.1}$$

If we plug in pitch number $p = 57$ which is note $A3$, we will get the center frequency $f_{57} = 220$ Hz, which is half of the center frequency of note $A4$. From this we can see that if the pitch of a note is one octave higher than the other note, its center frequency will be twice as the lower note's. Besides, Equation (2.1) also implies that the higher the

[2] chroma has the same meaning as pitch class.

[3] "The property of a sound that correlates to the perceived frequency is commonly referred to as the pitch." [23]

pitch is, the larger the frequency range it occupies. In our case, the decomposition from the audio signal to pitch subbands is realized by a multirate filter bank which consists of an array of suitable bandpass filters. As the pitch gets higher, the bandwidth of the corresponding filter gets wider. Figure 2.1 shows an illustration of the filter bank with different bandwiths.

The decomposition from the original signal to 88 pitch related frequency subbands serves as a prerequisite signal processing step. It provides meaningful information about the signal both in the temporal and the frequency domain, indicating at what time which frequency component is present. However, the great contribution of this step only works if a pre-condition is fulfilled: we assumed that the audio file needs to have correct tuning and the instruments are tuned according to the equal-tempered scale. In this way, tuning is a crucial factor which the multirate filter bank is sensitive to. In this thesis, we previously tuned all the audio files before conducting any experiments in order to make sure the pitch features are correct. Besides this, strong onsets or string vibratos will lead to energy spreading in a large frequency range, affecting many subbands and therefore making the decomposition problematic. Without prior detection and smoothing of these frequency fluctuations, one may have imperfect subbands signals when such phenomena occur.

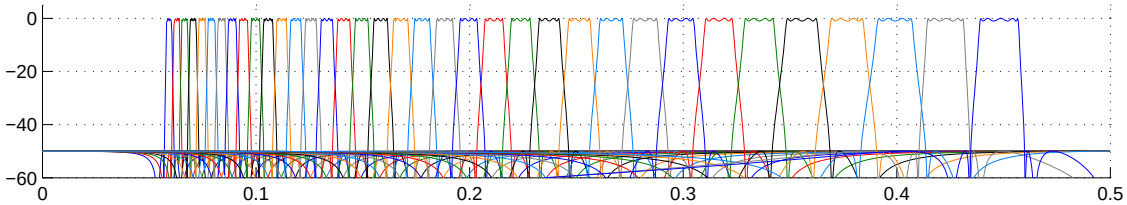


Figure 2.1. A sample array of filters with their respective magnitude responses in dB. (reproduced from [23])

2.1.2 Local Energy (STMSP)

The previous decomposition step allows us to identify the contribution which a certain pitch makes to the overall signal. However, the unit of measurement for the contribution of each pitch is not clear so far [33]. Also, the temporal measurement about the appearance of note is also unclear. To solve this problem, we compute the short-time mean-square power (i. e., the samples of each subband output are squared) using a rectangular window of a fixed length w. Denote a specific subband signal corresponding to pitch p as x_p, k as the index of the sample points inside the window, n as the starting position of the window on the signal, then the STMSP is defined as $\sum_{k\in[n:n+w]} |x_p(k)|^2$. The window is consistently shifted on the signal until the end with each time shifting it by a hop size $h = \frac{w}{2}$, yielding 50% overlap between any of the two neighbor windows. The size of the window is usually chosen as a few milliseconds. Since the window size is closely related to the hop size, and since the hop size is related with how frequent we get a feature from the signal, or in other words, how large the feature rate is, we can directly deduct the feature rate as $\frac{1}{h} \cdot 1000$ which in our case being $\frac{2}{w} \cdot 1000$(the multiplier 1000 is because that the unit of hop size and window size is in millisecond, not second). For example, a window length corresponding to 200 milliseconds leads to a feature rate of 10 Hz.

The result after this processing step is a sequence of features which are referred to as *pitch features*. The pitch features measure the local energy content of each pitch subband and indicate the presence of certain musical notes within the audio signal by selecting the energy which exceed a certain threshold. See [23] for further details. The implementation of pitch features can be found in the MATLAB function `audio_to_pitchSTMSP_via_FB.m` from the Chroma Toolbox[22].

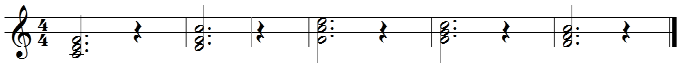
As an example, we extracted pitch features from a synthesized audio file *chordExample1*. The audio file consists of five chords, its score representation is shown in Figure 2.2(a). Figure 2.2(b) shows the spectrogram of the original signal with x-axis indicating the time in seconds and y-axis indicating the frequency in a linear way, with the unit Hertz. Figure 2.2(c) shows the extracted pitch features with y-axis indicating the MIDI pitch number, which associated with frequency in a logarithmic way. The colored intensity represents the STMSP value or local energy. Here, the played notes can be clearly identified in the lower part of Figure 2.2(c). However, as we can observe from the upper part, there are comparably low intensities for some unplayed notes, which are actually caused by the harmonics of the played notes. This phenomenon implies a common challenge in chord recognition task: even with the synthesized example which consists of only one instrument, the presence of harmonics will affect the identification of played notes. In our example, the intensities of harmonics are weak, however in real audio files which involve several instruments, the harmonics caused by different instruments will be blend together with the real played notes. Sometimes the intensities of harmonics are as large as some soft played notes, which make the identification of real played notes more difficult.

2.2 Chroma Features

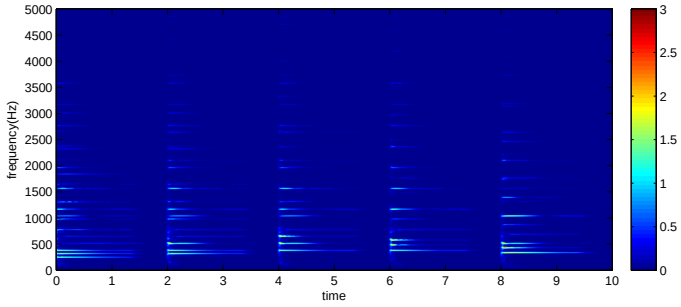
As we discussed in the last section, the energy caused by a certain played note is not only present at the exact frequency that note covers but also at higher frequencies where the harmonics of the note are located. This makes not only the identification of a musical note difficult, but also the identification of component notes of a certain chord difficult. In this section, we introduce a new type of feature named as *chroma features* which can partly solve this problem. Chroma-based audio features are a well-established tool in processing and analyzing music data [2, 9, 23], and the chroma features are particularly suitable for chord recognition.

As we know that human's perception of musical notes has a certain character: if a note is one or more octave higher than another note, then the two notes sound to have the same "tone color" but different "tone height". This phenomenon is referred to as octave equivalence in music theory. Assuming the musical notes are of equal-tempered scale, the chroma correspond to the set $\{\mathrm{C}, \mathrm{C}^\sharp, \mathrm{D}, \ldots, \mathrm{B}\}$ that consists of the twelve pitch spells attributes as used in Western music notation. Note that in the equal-tempered scale, different pitch spellings such as $\mathrm{C}^\sharp$ and $\mathrm{D}^\flat$ refer to the same chroma.

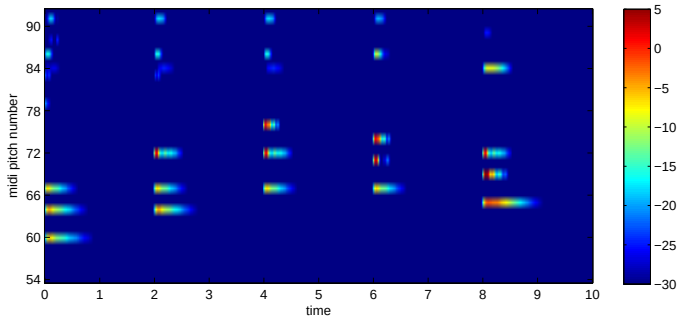
Take MIDI notation for example, note $A4$ denotes the chroma as A and "tone height" as the 4th octave. We can reduce the MIDI notes from 88 pitches to 12 chroma classes by ignoring the octave information of the notes and classifying via chroma information. Chroma features are very suitable for the task of chord recognition. This is because when



(a) score representation



(b) spectrogram



(c) pitch representation

Figure 2.2. Score representation, spectrogram and pitch representation of the synthesized music audio *ChordExample1*.

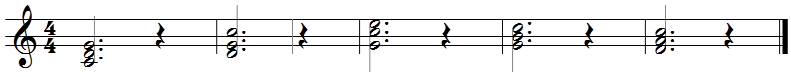
analyzing a musical chord, we are much more interested in the chroma (or pitch class) which the note belongs to other than the absolute pitch of that note. For notes sharing the same chroma but in different octaves, we treat them as identical when considering a component note of the chord. Furthermore, remember that different timbre of instruments will yield different yet distinctive energy distribution at harmonics, and since the energy of a chroma is merged from different pitches corresponding to this chroma together, the difference caused by timbre is well absorbed by chroma features, thus making them robust to the variations of timbre.

The chroma features are in the form of a 12-dimensional vector $x = (x(1), x(2), \ldots, x(12))^T$, where $x(1)$ corresponds to chroma C, $x(2)$ to chroma C$^\sharp$, and so on. Chroma-based features represent the short-time energy of the signal in each of the 12 pitch classes. Often these chroma features are computed by suitably pooling spectral coefficients obtained from a short-time Fourier transform [2, 9]. Similarly, one can start with the pitch representation introduced in Section 2.1. Then, by simply adding up the corresponding values that belong to the same chroma, one obtains a *chroma representation* or *chromagram*. Usually we perform ℓ^2 normalization on the resulting vectors. For the case of near silence or weak noise, the sum of all entries of such chroma vector will be quite small. If the sum falls below a certain threshold, we replace the original chroma vector by the unit vector.

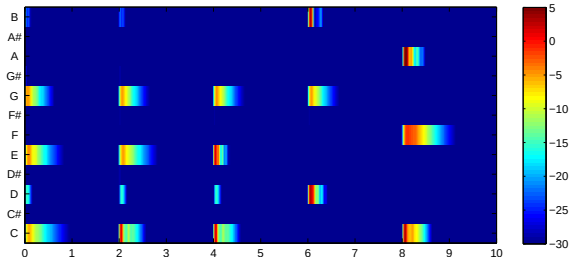
In the following, the resulting features are referred to as *Chroma-Pitch* and we denote them by `CP`, see [23] for more details. The MATLAB function of this part can be found in `pitchSTMSP_to_chroma.m` from our toolbox.

Figure 2.3 illustrates the chromagram of feature `CP` extracted from the audio file *ChordExample1*. The figure in the middle shows the features in the middle step which is after spectral pooling, the figure at bottom shows the final chroma features generated by spectral pooling with ℓ^2 normalization. In Figure 2.3(b), only the onset can be clearly seen in the first 0.5 second of each chord, the intensity of the remaining duration is not visible. In this case we cannot distinguish between the real silence and the remaining sound. However in Figure 2.3(c), the silence can be clearly identified through the unit vector whose energy is everywhere the same for all chromas. Besides, not only the onset but also the remaining duration can be easily seen, especially for the bass note of each chord (red color in the figure).

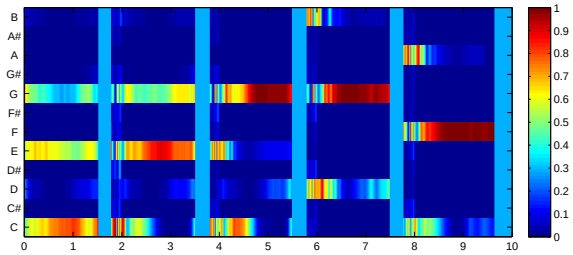
Note that `CP` features still imperfect. For example, from 5.8 to 7.5 seconds in Figure 2.3(c), the chroma G is much stronger than the other two component notes. As the bass note and also the root note of the `G`, its comparably large intensity makes sense. But it should not be so strong that the intensity of other notes are nearly erased after normalization, which make the identification of the other two notes problematic. This is also the case for the chroma F from 5.8 to 7.5 seconds. This imperfection requires further processing step to further balance the difference of intensity.



(a) score representation



(b) chroma features without normalization



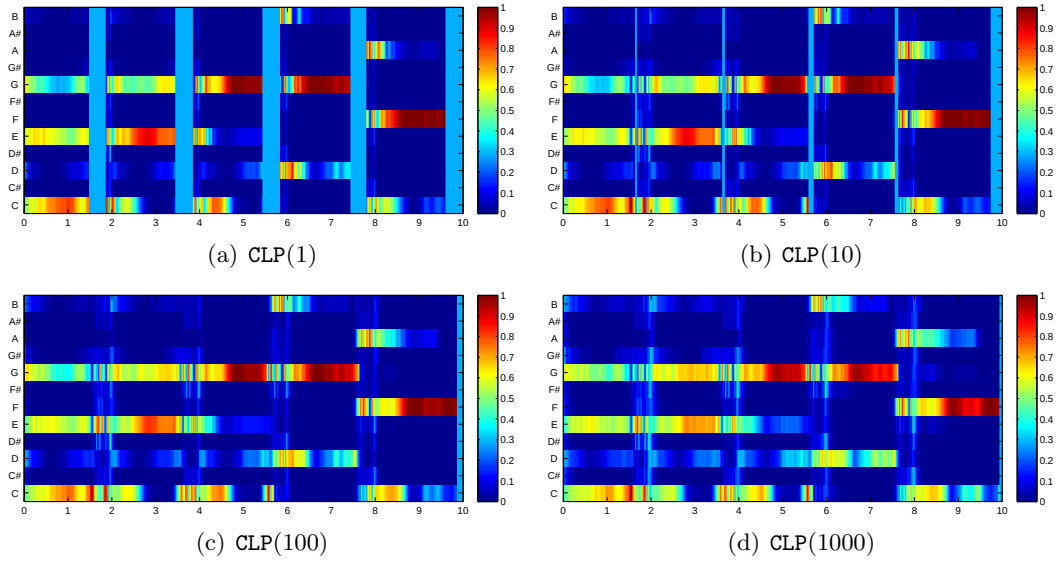
(c) chroma features with normalization

Figure 2.3. Score representation and chromagram of `CP` extracted from *ChordExample1*.

2.3 Chroma Features with Logarithmic Compression

To account for the logarithmic sensation of sound intensity [18, 36], one often applies a logarithmic compression when computing audio features [15]. Besides, this step also works as adjusting the dynamic range of the original signal to enhance the clarity of weaker transients, especially in the high-frequency regions [11]. There are some weak chromas whose intensities are very low. Such chromas are difficult to identify in the chromagram of CP. With the help of logarithmic compression, now it is easy to find their existence.

The procedure of how we derived the chroma features with logarithmic compression is presented as follows. Firstly, we compute pitch features from the audio file as we described in Section 2.1. Secondly, the pitch representation is logarithmized by replacing each entry e by the value $\log(\eta \cdot e + 1)$, where η is a suitable positive constant. Thirdly, we convert the logarithmic pitch features into the chroma representation by spectral pooling and ℓ^2 normalization as we described in Section 2.2. The resulting features are named as *Chroma Features with Logarithmic Compression* and denoted as $\texttt{CLP}(\eta)$ where the parameter η specifies the extent of logarithmic compression. The role of η, which is set to $\eta = 1000$ in most of our experiments, is evaluated in Section 5.3.3.



(a) $\texttt{CLP}(1)$ (b) $\texttt{CLP}(10)$

(c) $\texttt{CLP}(100)$ (d) $\texttt{CLP}(1000)$

Figure 2.4. CLP features extracted from *ChordExample1*.

As an illustrative example, Figure 2.4 provides the chromagrams of $\texttt{CLP}(\eta)$ with $\eta = 1, 10, 100, 1000$ respectively. As we can see from the figure, as η gets larger, the intensity contrast between different chroma gets smaller. For example, for the period from 9.0 to 9.5 seconds, in Figure 2.4(a) the chroma C and A are almost invisible. However in Figure 2.4(d) they can be clearly seen. In particular, the chroma C is more obvious. This helps us to identify the underlying chord F for that period, because all three component notes of F are present. This is much better than the situation where the intensity of only one chroma can be observed while the intensity for other chromas are too low to find. In this situation, it is too hard to decide the underlying chord.

One can observe in Figure 2.4 that the original pause period such as 1.5 to 1.9 seconds

is disappearing as η is increasing. This is a typical evidence that weak intensity will be enhanced by logarithmic compression. In contrast, the chroma C from 0.5 to 1.2 seconds changing the color from red to yellow, which means that the intensity gets less and less as η increases. This is evidence that the original strong intensity will be suppressed to a milder degree. In this way, the logarithmic compression is balancing the difference of intensity, and the η is the controlling parameter of such balance.

2.4 CENS Features

Generally speaking, the *Chroma-Pitch* features have already achieved the goals of feature design aimed for chord recognition since it is able to indicate the behavior of harmonic progression of a music piece. However, it can still be further improved considering variations of musical properties such as dynamics, timbre, articulation, execution of note groups, and temporal micro-deviations. In order to be robust against these variations, we add a further degree of abstraction to *Chroma-Pitch* features by considering short-time statistics over energy distributions within the chroma bands. The features we obtained are named as CENS (Chroma Energy Normalized Statistics) features, and they constitute a family of scalable and robust audio features. CENS features, which have first been introduced in [26], are strongly correlated to the short-time harmonic content of the underlying audio signal while absorbing variations in other parameters. Furthermore, because of their low temporal resolution, CENS features can be processed efficiently, see [26, 23] for details.

In computing `CENS` features, we have five stages with each designed for a different purpose. The computing pipeline is shown in the following:

1. *Normalization.* First, we ℓ^1-normalize the chroma features in order to absorb differences in the sound intensity or dynamics. For the case of very low energy distribution or silence, we replace the chroma vector by a uniformly distributed vector if the norm does not exceed certain threshold.

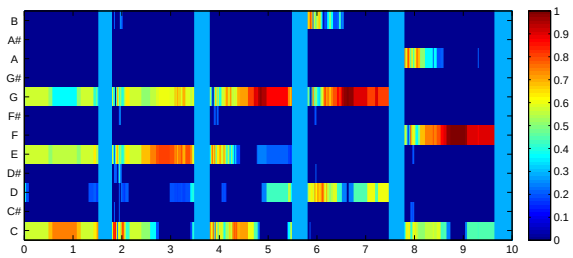
2. *Quantization.* The component of the normalized chroma vector are quantized based on logarithmically chosen thresholds to simulate the humans's perception of loudness. This introduces some kind of logarithmic compression similar to the features `CLP`. The quantization function serves as a mapping function from $[0, 1]$ to $\{0, 1, 2, 3, 4\}$.

3. *Smoothing.* The quantized vectors from last step are now convolved with a Hann window of fixed length w, $w \in \mathbb{N}$. This step works as temporal smoothing to blend in the context information and reduce the influence of local error.

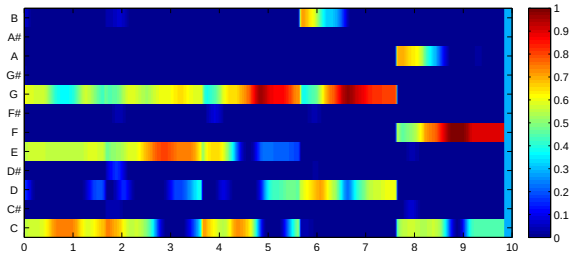
4. *Downsampling.* We downsample the resulting feature vectors by a specific factor d to increase the computation efficiency for the next processing module.

5. *Normalization.* Finally, we perform ℓ^2-normalization to the feature vectors.

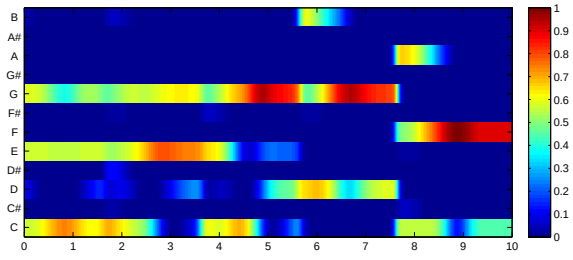
In the following, we denote the resulting CENS features by $\texttt{CENS}(w, d)$ with w indicating the size of convolution window and d indicating the downsampling factor. The MATLAB function of this part can be found in `pitchSTMSP_to_CENS.m` from the Chroma Toolbox [22].



(a) CENS$(1, 1)$



(b) CENS$(21, 1)$



(c) CENS$(41, 1)$

Figure 2.5. CENS features extracted from *ChordExample1*.

The main purpose of using CENS features in our chord recognition system is to take advantage of its internal effect of smoothing at the feature side. This serves as the pre-filtering before the chord recognition module. We also provide chord recognition methods performing smoothing on the recognizer side, which serves as the post-filtering of the final chord decisions. Figure 2.5 illustrate chromagrams with different smoothing window length of CENS features. Having a feature rate of 50 Hz, each feature vector contains music information of 0.02 seconds for the non-smoothing version, which is CENS$(1, 1)$, since $w = 1$ means convolve with the current feature itself. By enlarging w to 21, each feature vector now carries the music information of $0.02 \cdot 21 = 0.42$ second. By comparing Figure 2.5(b) and Figure 2.5(a) we find that the unit vectors corresponding to silence period are smoothed out by the neighbor C. This means that chords tend to be continuous via smoothing. Furthermore by enlarging w to 41, we find that the edge of the chord gets more blurred. For example, at around 7.5 second, the chroma G and D should stop. In CENS$(1, 1)$ it is silence for all chromas at that time point which helps us to make exact decision about the edge of G. However in CENS$(41, 1)$ at that time point, due to the large smoothing window, the presence of G and D coming from the left neighbor features, and A, F, C coming from the right neighbor features mix together. This might leads to errors in chord recognition, and it is hard to decide the edge in the situation when a previous chord finishes or when

a new chord starts.

2.5 CRP Features

To boost the degree of timbre invariance, a novel family of chroma-based CRP audio features has been introduced in [25, 24]. The general idea is to discard timbre-related information similar to that expressed by certain mel-frequency cepstral coefficients (MFCCs). Starting with the pitch representation as introduced in Section 2.1, one first applies a logarithmic compression and transforms the logarithmized pitch representation using a DCT. Note that the logarithmic compression parameter η is set to $\eta = 1000$ in our experiments. Then one only keeps the upper coefficients of the resulting pitch-frequency cepstral coefficients (PFCCs), applies an inverse DCT, and finally projects the resulting pitch vectors onto 12-dimensional chroma vectors. These vectors are referred to as CRP (Chroma DCT-Reduced log Pitch) features. The upper coefficients to be kept are specified by a parameter $n \in [1 : 120]$. In our experiments, we use $n = 55$ if not specified otherwise. While constructing CRP features, just as `CENS` features, we also have the smoothing step which convolve one feature vector with neighbor feature vectors. The window length of the convolution is specified by w. Besides, the downsampling step is contained in CRP features as well. The parameter d is responsible for this. Figure 2.6 shows the chromagram of CRP features extracted from *ChordExample1* with no smoothing effect. Note that different from other features, some of the computed intensities of CRP features are negative values.

CRP features have three specialities. Firstly, as we mentioned above they are designed to be invariant with timbre. Secondly, they integrate logarithmic compression in intensity computing. Thirdly, they include the smoothing technique. All these specialities make CRP features outstanding among all feature types. In the following, similar to `CENS` features we denote CRP features by `CRP`(w, d). The MATLAB function of how we derived CRP features can be found in `pitchSTMSP_to_CRP.m` from the Chroma Toolbox[22].

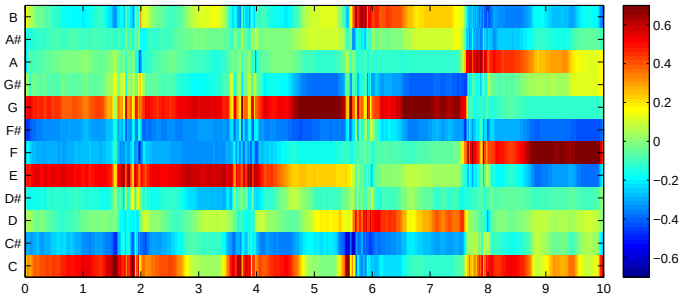


Figure 2.6. `CRP`$(1, 1)$ extracted from *ChordExample1*.

2.6 CISP Features

In this section, we adapt the chroma feature extraction according to Dan Ellis' s instantaneous frequency-based chromagram and we name this feature as CISP features. Using CISP features, we can identify strong tonal components in the spectrum and to get a higher-resolution estimate of the underlying frequency [7]. While generating chroma representation features, using a coarse mapping of FFT bins to the chroma classes which the bins overlap will often yield blurry low frequencies. CISP take advantage of phase-derivative (instantaneous frequency) within each FFT bin, and get a finely estimation of frequency.

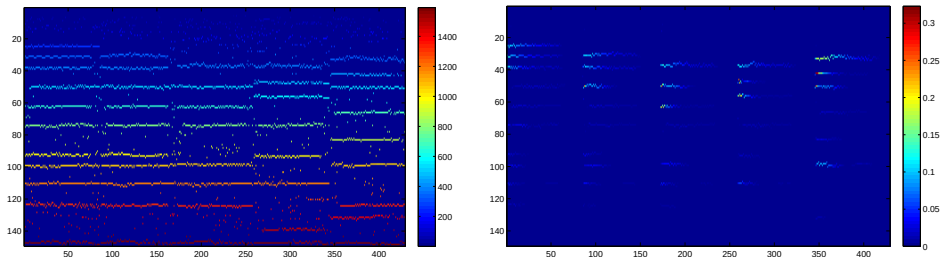
A further motivation of using instantaneous frequency lies in that sinusoidal components of the audio signal contains the most relevant information about the melody [6], which is the tonal component of music. CISP is intended to remove nontonal components and improve frequency resolution beyond FFT bin level [7].

CISP features are constructed as follows. Firstly, a spectrogram is computed using a short time fourier transform. Secondly, for each of the bins of the spectrogram (every bin bounds a range of frequencies), the instantaneous frequency is determined. Thirdly, based on the instantaneous frequency, a noise harmonic component separation is performed. Note that the frequency of a bin is estimated by weighted sum of the frequencies inside the bin with the weights being the corresponding magnitude of those frequencies. Finally, the estimated frequencies are mapped into chroma representation by adding up the magnitude of bins which belong to the same chroma.

CISP feature integrates an automatic tuning step. To avoid problems of tuning, the mapping of frequencies to chroma bins is adjusted by up to ± 0.5 semitones to make the single strongest frequency peak line up exactly with a chroma bin center [7].

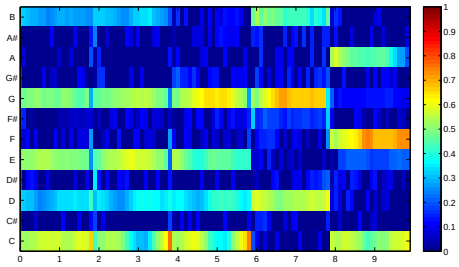
Figure 2.7(a) indicates the color-coded instantaneous frequency values for each bin of a spectrogram. The x-axis indicates the frame number and y-axis indicates the the number of bins. See Figure 2.7(b) for the corresponding magnitude spectrogram. Here, a frequency of zero value (the dark blue in Figure 2.7(a)) indicates that this bin was selected as noise and filtered out. One can observe from the remaining horizontal structures, which are actually the harmonic components. Figure 2.7(c) illustrates the chroma representation of CISP features. We can observe that CISP features are very sensitive to small magnitudes. The computed intensity at B and D are much larger than other features. However, B and D are only harmonics which have much smaller magnitudes compared to the other component notes of `C`. In the other chromagrams which we presented previously, they can hardly be seen.

The MATLAB Function of CISP features can be found in the Intelligent Sound Processing Toolbox [5]. In the following passages, we denote CISP features as `CISP`.



(a) Tracked instantaneous frequency

(b) Spectrum of STFT



(c) Chroma representation

Figure 2.7. Tracked frequencies, tracked magnitude, and chroma representation of `CISP` features extracted from *ChordExample1*.

Chapter 3

Template-based Chord Recognition

After converting the audio file into musically meaningful audio features, we now pass these features into the chord recognition module which automatically classify the feature vectors with respect to given chord labels. In other words, the chord recognition module assigns to each feature vector a chord label.

From this chapter on, we begin to discuss chord recognition methods. In this chapter we focus on template-based methods. In these methods, firstly, some pre-computed feature templates are defined and they served as chord patterns. Here, the templates can be defined from different point of views and therefore have many variations. Secondly, we need to compare how similar a given feature vector is to each of the templates. To this end, we need to find a similarity measure or distance measure between the feature and a template. Thirdly, we assign the chord label by selecting the one which yields the maximum similarity or minimum distance to the given feature.

The remainder of this chapter is organized as follows. We start by introducing the general procedure of template-based recognition methods in Section 3.1. Then three different specifications of template sets, namely the binary templates, the harmonically enriched templates and the averaged templates, will be described in Section 3.2. After that we present the setting of distance measure. In particular, the specification of cosine distance is introduced in Section 3.3. Finally, in Section 3.4, we summarize the advantages and disadvantages of using template-based methods.

3.1 Template-based Chord Recognition

In the previous stage of feature extraction, we transform a given audio recording into a chroma-based feature sequence $X := (x_1, x_2, \ldots, x_N)$, $x_n \in \mathcal{F} := \mathbb{R}^{12}$, $n \in [1 : N]$. Here in the stage of chord recognition, we use template-based recognizers to assign chord labels to the feature sequence. In this section, we will describe the general procedure of template-based chord recognition. The procedure can be described as follows.

- Firstly, we define our set of templates. Given a chord label set Λ, the template set $\mathcal{T}$ is a subset of the feature set $\mathcal{F}$. $\mathcal{T}$ consists of 24 chroma-based chord patterns corresponding to 12 major and 12 minor triads. The elements of $\mathcal{T}$ are indexed by the element of chord label set Λ. we denote a template as $\mathbf{t}_\lambda \in \mathcal{T}$,$(\lambda \in \Lambda)$.

- Secondly, in order to compute the distance between a feature vector and a chord template, we need a distance measure. Since a template is a feature vector with special meaning, we fix the distance measure $d : \mathcal{F} \times \mathcal{F} \to \mathbb{R}$. This distance measures how different a feature vector compared to a chord template.

- Finally, for a given feature vector, we compute its distance with each of the chord templates. Now the template-based chord recognition procedures simply consists in assigning the chord label that minimizes the distance between the corresponding template and the given feature vector $x = x_n$:

$$\lambda_x := \underset{\lambda \in \Lambda}{\operatorname{argmin}}\ d(\mathbf{t}_\lambda, x). \tag{3.1}$$

Both the template set and the distance measure are not fixed in the procedure. In the following passages, we complete the procedure with three template settings defined from different aspects of chords and use cosine distance as measurement. By changing the templates or distance, one can check the different recognition results and further inspect whether a certain template setting is meaningful and whether it is suitable for the given type of features. Besides, the template-based methods work in a purely framewise fashion and no temporal context is considered. We are not the first one to use template-based methods as chord recognizers, similar previous work can be found in [12, 17, 28].

3.2 Specification of Chord Template Sets

In this section, we consider three variations of template setting. Remember that the template set $\mathcal{T}$ is a subset of the feature set $\mathcal{F} = \mathbb{R}^{12}$, that is to say, a template is also a feature vector. However, a template has a special meaning compared to the normal feature vectors which extracted from audio files, because it describes a certain chord pattern in the representation of a feature vector. It can be set variously from different point of views. The three variations of template setting which we introduce in this section only differ in how the weights are setted to the entries of the template vector. The weights can either be manually set considering the theoretical characteristic of a chord, or they can be set by learning their general characteristics from the real data in practice.

Here are the three variation of templates:

- Set of binary templates: $\mathcal{T}^b$ with elements $\mathbf{t}^b_\lambda$
- Set of harmonically enriched templates: $\mathcal{T}^h$ with elements $\mathbf{t}^h_\lambda$
- Set of average templates: $\mathcal{T}^a$ with elements $\mathbf{t}^a_\lambda$

The details of how we set the templates will be described in the following subsections. There is an a general trick which is named as "cyclically shift", and it is used in all the template setting. We first introduce it here.

3.2.1 Cyclically Shift

For each of the template sets, we only set two templates instead of setting all templates. We set one for `C` and the other for `Cm`, and denote them as $\mathbf{t}_{\texttt{C}}$ and $\mathbf{t}_{\texttt{Cm}}$. The templates for other major triads are computed by cyclically shifting $\mathbf{t}_{\texttt{C}}$, and other minor triads are computed by cyclically shifting $\mathbf{t}_{\texttt{Cm}}$. The reason of involving cyclically shift is to utilize the characteristics of the chords. Since the musical interval between the third and root, fifth and root are always fixed for the same type of chord, one can derive the same type of the chords by first changing the root note and then make sure the third and fifth note from the musical interval. Therefore, we can derive the templates for the same type of chords by cyclically shifting the position of the notes.

Thus for later usage, we define an operation that allows for cyclically shifting the components of a feature vector $x := (x(1), \ldots, x(D))^{\mathrm{T}} \in \mathcal{F}$. To this end, we introduce the shift operator $\sigma : \mathcal{F} \to \mathcal{F}$ defined by

$$\sigma((x(1), \ldots, x(D))^{\mathrm{T}}) := (x(D), x(1) \ldots, x(D-1))^{\mathrm{T}}. \tag{3.2}$$

Iteratively applying the shift operator, one obtains

$$\sigma^i(x) = \sigma(\sigma^{i-1}(x)). \tag{3.3}$$

for $i \in \mathbb{Z}$. Obviously, $\sigma^D = \sigma^0$ is the identity on $\mathcal{F}$. Therefore, in the following, we only consider the shift index i modulo D. We extend the shift operator to the space of sequences $\mathcal{F}^N$ in a canonical way and denote the resulting operator again by $\sigma : \mathcal{F}^N \to \mathcal{F}^N$:

$$\sigma(X) := (\sigma(x_1), \sigma(x_2), \ldots, \sigma(x_N)), \tag{3.4}$$

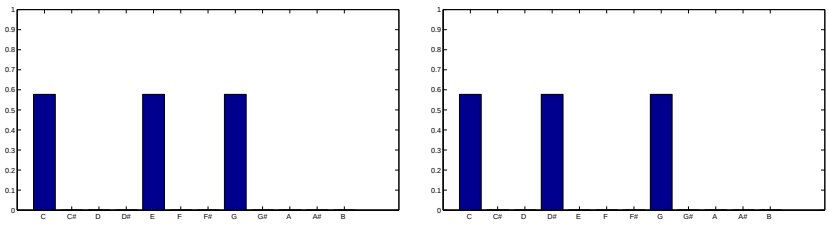
for a given sequence $X := (x_1, x_2, \ldots, x_N) \in \mathcal{F}^N$.

3.2.2 Binary Templates

The first template setting introduced here is designed to be the simplest one among all the settings. The motivation of introducing binary templates is to simulate the fact that a chord is formed by its component notes[1]. For example, `C` is composed by the note C, E and G; `Cm` is composed by C, E♭ and G. While designing the templates in this method, for every given chord, we only consider the component notes of chord and totally discard other non-component notes. Thus it is reasonable to involve a binary setting since a note is either a component or a non-component one.

We set the binary templates as follows. Each template in the set $\mathcal{T}^b$ is a 12-dimensional binary vectors with three entries equal to one and other entries equal to zero. The three non-zero entries correspond to the three component notes of a chord.

[1]Here the notes we mentioned are as chromas, which come from distinctive pitch classes



(a) $\mathbf{t}^b_{\texttt{C}}$ (b) $\mathbf{t}^b_{\texttt{Cm}}$

Figure 3.1. Binary templates for `C` and `Cm`.

For example, the binary template corresponding to `C` `C` $= \{\mathrm{C}, \mathrm{E}, \mathrm{G}\}$ is given by

$$\mathbf{t}^b_{\texttt{C}} = (1,0,0,0,1,0,0,1,0,0,0,0)^T.$$

The binary template corresponding to `Cm` `Cm` $= \{\mathrm{C}, \mathrm{E}\flat, \mathrm{G}\}$ is given by

$$\mathbf{t}^b_{\texttt{Cm}} = (1,0,0,1,0,0,0,1,0,0,0,0)^T.$$

Figure 3.1 illustrates the setting of binary templates for `C` and `Cm`. We denote a binary template by $\mathbf{t}^b$. Note that we perform the ℓ^2-normalization on the values shown above in order to fit the features which are also with ℓ^2-normalization. The advantage of the binary setting is its simplicity and efficiency. The same weights on the component notes is fair to count the contribution from each of the component notes. In contrast, the simplicity also leads to a limitation: it considers only the very ideal instance of a chord which works theoretically. However in practice, the intensity of the three component notes may not be exactly the same but very different. Also, it ignores too much information, for instance, it totally disregard the non-component notes, which may contribute to form the pattern of a chord.

3.2.3 Harmonically Enriched Templates

For the recognition method using harmonically enriched templates, the weights for the entries of a template considers not only the component notes of a chord but also their harmonics. The motivation of this method is to simulate a phenomenon in practice that when a single note is played on instrument, the sound is not a simple pure tone with a well-defined frequency [23]. According to [23], "the sound of a musical tone can be regarded as a superposition of harmonics or overtones - whose frequencies differ by an integer multiple from a certain fundamental frequency.". Here, the fundamental frequency is the frequency of the lowest harmonics. If the fundamental frequency is f, the harmonics have frequencies $2f$, $3f$, $4f$, etc. For example, on piano, if one presses the key which has the MIDI pitch `C4`, one can only recognize the sound as `C4` by human's perception. However, actually the sound the piano generates contains not only `C4`, but also its harmonics. These harmonics include `C5` which has double times of frequency of `C4`, `G5` which has triple times, `C6` which has four times, and so on.

The reason why humans can not perceive the harmonics lies in the intensity or loudness of the harmonics. Take piano for instance, the intensity of the harmonics are much smaller

than the intensity of the note which has the fundamental frequency. Especially, the higher the harmonics, the smaller the intensity. The note at fundamental frequency contributes the most to the overall intensity. This is the usual case for piano, but not universal for any of the instruments. For some instruments, it might be the case that the first harmonic contributes the most of intensity and what humans perceive is the sound of 1st harmonic, while the sound of fundamental frequency can hardly be perceived.

In this thesis, we set up the simulation of harmonics in the way that Emilia Gómez suggested in [10]. She modeled the harmonics of notes using simulating the spectral envelope. To this end, we set the weights to the notes of a chord as follows. Firstly, we assign the theoretical amplitude to all component notes of a chord. Secondly, for each of the component notes, we consider its harmonics. For each harmonic, we assign a theoretical amplitude, and this amplitude is related to which overtone the harmonic is for the component note. The value is assigned by an empirical decay factor s multiplied by the amplitude of component notes. The decay factor is to model the amplitude of different harmonics such that the contribution decrease as the frequency increase [29]. Gómez set s as 0.6.

The contribution for the first 6 harmonics of a note is given in Table 3.1. In this way, for each component note of a chord, its intensity contribution to the chord consists of the intensity from the component note itself, and the intensity from all its harmonics. Since the 1st harmonic is just the component note itself, the added intensities are coming from the other 5 higher harmonics. We include all these values in our templates. In order to avoid zero weight for any of the notes, we initially set the weights of all notes to a very small value $\epsilon = 0.005$ instead of zero.

Table 3.2 present the information about the frequency and index of the harmonics and the corresponding decayed factor for the component note `C4`, `E4` and `G4` of chord `C`, assuming the notes are played in the fourth octave. The resulting harmonic template considers all the contributions from the notes in these tables and project these contributions into 12-dimensional chroma representation. The resulting template are as follows:

The harmonic template correspond to `C` `C` $= \{\mathrm{C}, \mathrm{E}, \mathrm{G}\}$ with decay factor $s = 0.6$ is given by

$$\mathbf{t}^h_{\mathtt{C}} = (0.254, 0.005, 0.061, 0.005, 0.272, 0.005, 0.005, 0.315, 0.018, 0.005, 0.005, 0.079)^T.$$

The harmonic template correspond to `Cm` `Cm` $= \{\mathrm{C}, \mathrm{E}\flat, \mathrm{G}\}$ with decay factor $s = 0.6$ is given by

$$\mathbf{t}^h_{\mathtt{Cm}} = (0.254, 0.005, 0.612, 0.254, 0.018, 0.005, 0.005, 0.333, 0.005, 0.005, 0.061, 0.018)^T.$$

Figure 3.2 illustrates the final harmonic templates for `C` and `Cm`. This is also the version with ℓ^2-normalization. We denote a harmonically enriched template by $\mathbf{t}^h$.

3.2.4 Averaged Templates

In this setting, the template vector is not composed of manually set values anymore, but values derived from averaging of practical training data. This also means that the

index	frequency	factor
1	f	1
2	$2f$	s
3	$3f$	s^2
4	$4f$	s^3
5	$5f$	s^4
6	$6f$	s^5

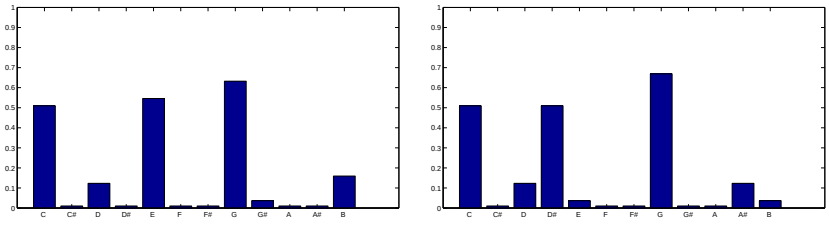
Table 3.1. Contributions for the first six harmonics of a note.

index	note	frequency	factor
1	C4	$f_{\texttt{C4}}$	1
2	C5	$2f_{\texttt{C4}}$	s
3	G5	$3f_{\texttt{C4}}$	s^2
4	C6	$4f_{\texttt{C4}}$	s^3
5	E6	$5f_{\texttt{C4}}$	s^4
6	G6	$6f_{\texttt{C4}}$	s^5

index	note	frequency	factor
1	E4	$f_{\texttt{E4}}$	1
2	E5	$2f_{\texttt{E4}}$	s
3	B5	$3f_{\texttt{E4}}$	s^2
4	E6	$4f_{\texttt{E4}}$	s^3
5	G♯6	$5f_{\texttt{E4}}$	s^4
6	B6	$6f_{\texttt{E4}}$	s^5

index	note	frequency	factor
1	G4	$f_{\texttt{G4}}$	1
2	G5	$2f_{\texttt{G4}}$	s
3	D6	$3f_{\texttt{G4}}$	s^2
4	G6	$4f_{\texttt{G4}}$	s^3
5	B6	$5f_{\texttt{G4}}$	s^4
6	D7	$6f_{\texttt{G4}}$	s^5

Table 3.2. Contributions for the first six harmonics of component notes of C. Assuming the chords is played in the 4th octave. From left to right corresponding to the notes C4, E4 and G4.



(a) $\mathbf{t}^h_{\texttt{C}}$ (b) $\mathbf{t}^h_{\texttt{Cm}}$

Figure 3.2. Harmonically enriched templates for C and Cm.

templates we describe in this set are not fixed as the previous binary or harmonically enriched setting, but vary according to different training datasets. These are the most significant differences compared to the previous two settings.

Figure 3.3 illustrates the procedure of generating the averaged templates. The typical training data basically consists of some music audio files and corresponding chord annotation labels. We divide all the training data we have into several partitions, and call each of the partition a *training dataset*. Since our system is evaluated in a framewise fashion, we need to divide the training data into the form of frames, meaning that we segment the audio files into feature frames and parse the annotation files into label frames. The audio files are transformed into the feature vectors in the stage of feature extraction, and the chord annotation labels are parsed and aligned with the feature vectors in this stage. For example, if the feature rate is 10Hz, and we have an annotation of C from 1.1 to 2.0 seconds, then there will be 10 feature vectors with each of them occupying 0.1 second of this period and labeled with C. The number of label frames is exactly corresponding to the number of feature frames. In case the annotation file is not completely annotated, for the intervals without annotation, we set it to "N" indicating non-annotation.

After that we have many feature vectors with the ground truth chord labels in hand. Usually we do not know whether we could cover all chords in the training data so that

we have at least one instance for each of the chord. Even if we could cover all the chords, some chords might have too few instances. To avoid this problem, we shift all feature frames and their chord labels to `C` or `Cm`. To achieve this, we perform cyclically shifting on all non `C`-labeled or `Cm`-labeled features. After that, all feature vectors are cyclicly shifted from different chords to `C` or `Cm`. This procedure works as follows.

1. First we need to compute how many semitones are needed to shift from an arbitrary chord with label λ_1 to the objective chord with label λ_2. To achieve this, we define the following two functions.

   Suppose we have a mapping function $M : \Lambda \to \mathbb{Z}$, which maps a chord to a positive integer i, $i \in [1 : 24]$, starting from $i = 1$ indicating `C`, $i = 2$ indicating $\texttt{C}^\sharp$, $i = 13$ indicating `Cm`, $i = 14$ indicating $\texttt{C}^\sharp\texttt{m}$ and so on. Table 3.3 shows the mapping from all major and minor triads to corresponding numbers. Furthermore we define the function $d_{chord} : \Lambda \times \Lambda \to \mathbb{Z}$ to compute the semitone distance between the two chords λ_1 and λ_2 :

$$d_{chord}(\lambda_1, \lambda_2) := |M(\lambda_1) - M(\lambda_2)| . \tag{3.5}$$

2. Denote the feature frame corresponding to the label λ_1 as x_1, which is the one to be shifted. Also denote the result feature frame x_2 corresponding to the label λ_2, which is the objective feature we want. Then x_2 is computed as :

$$x_2 = \sigma^{(d_{chord}(\lambda_1,\lambda_2))}(x_1). \tag{3.6}$$

   Note that we shift major chords to `C` and minor chords to `Cm` .

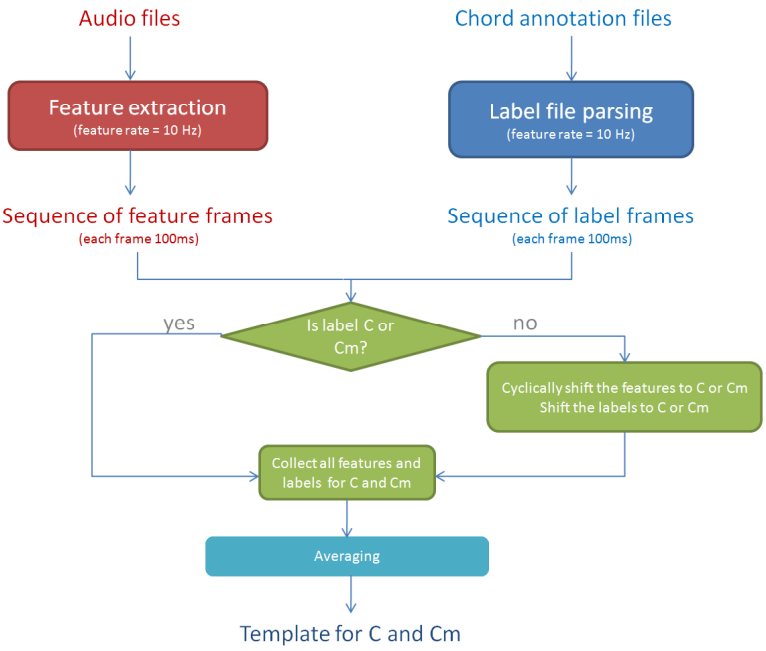


Figure 3.3. Procedure of generating the averaged template $\mathcal{T}^a$.

chord name	mapped value
C	1
C♯	2
D	3
D♯	4
E	5
F	6
F♯	7
G	8
G♯	9
A	10
A♯	11
B	12

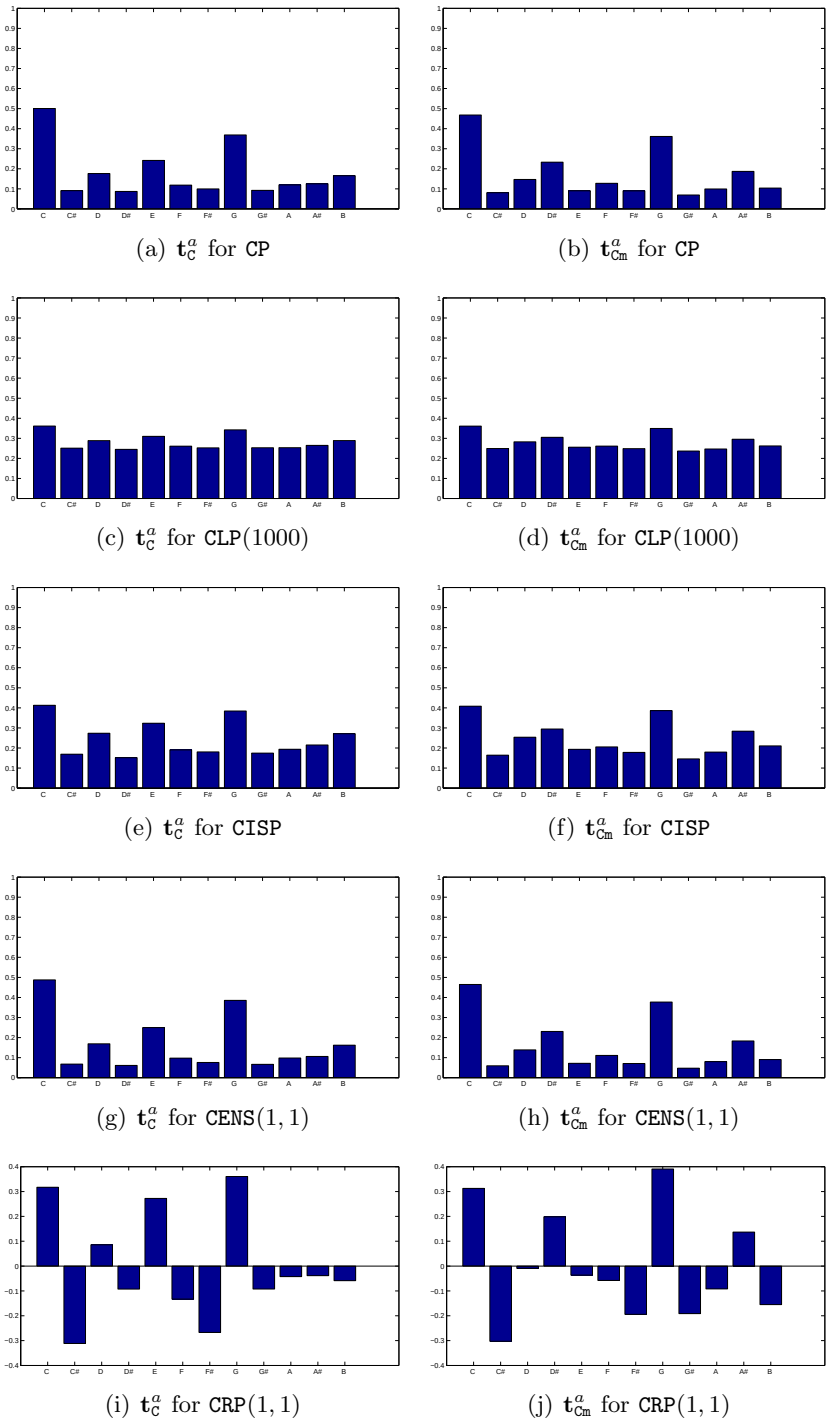
chord name	mapped value
Cm	13
C♯m	14
Dm	15
D♯m	16
Em	17
Fm	18
F♯m	19
Gm	20
G♯m	21
Am	22
A♯m	23
Bm	24

Table 3.3. Mapping function d_{chord} for major and minor triads.

Here we take a feature vector $x_{\mathtt{G}} = ((x(1), \ldots, x(12))$ for example, since we want to cyclically shift it to C, the semitone distance $d_{chord}(\lambda_C, \lambda_G) = |M(\lambda_G) - M(\lambda_C)| = 8-1 = 7$, thus $x_{\mathtt{C}} = \sigma^{(d_{chord}(\lambda_G, \lambda_C))}(x_{\mathtt{G}}) = \sigma^7(x_{\mathtt{G}}) = ((x(8), \ldots, x(12), x(1), \ldots, x(7))$.

After the step cyclically shifting, all features vectors are either C or Cm. The huge amount of instances allows us to estimate the average value as template more convincible. It should be much better than estimating the average value for each chord while relying on a small amount of instances, which is not capable to reveal the real pattern of the chords. Now we consider the averaged feature vector of C as the template for C, and the averaged feature vector of Cm as the template for Cm.

Note that the binary templates or harmonically enriched templates are fixed for all feature types. However the averaged templates are varying not only for different feature types but also for different training dataset. Figure 5.4 illustrate the average templates of C and Cm for different features using training dataset $\mathcal{D}_1^{\text{Beatles}} \cup \mathcal{D}_2^{\text{Beatles}}$. We denote an average template by $\mathbf{t}^a$.



(a) $\mathbf{t}^a_{\mathtt{C}}$ for `CP` (b) $\mathbf{t}^a_{\mathtt{Cm}}$ for `CP`

(c) $\mathbf{t}^a_{\mathtt{C}}$ for `CLP`(1000) (d) $\mathbf{t}^a_{\mathtt{Cm}}$ for `CLP`(1000)

(e) $\mathbf{t}^a_{\mathtt{C}}$ for `CISP` (f) $\mathbf{t}^a_{\mathtt{Cm}}$ for `CISP`

(g) $\mathbf{t}^a_{\mathtt{C}}$ for `CENS`$(1,1)$ (h) $\mathbf{t}^a_{\mathtt{Cm}}$ for `CENS`$(1,1)$

(i) $\mathbf{t}^a_{\mathtt{C}}$ for `CRP`$(1,1)$ (j) $\mathbf{t}^a_{\mathtt{Cm}}$ for `CRP`$(1,1)$

Figure 3.4. Averaged templates of different features for chord `C` and `Cm`.

3.3 Specification of Distance Measures

Having the templates as the patterns of chords, in this section we compare how similar a given feature vector is to these templates. To this end, we need to define the distance in order to measure the extent of pattern matching.

If not specified otherwise, we use the cosine measure $d = d_C$ defined by

$$d_C(x, y) = 1 - \frac{\langle x|y\rangle}{\|x\| \cdot \|y\|}, \tag{3.7}$$

for $x, y \in \mathcal{F} \setminus \{0\}$. In the case $x = 0$ or $y = 0$, we set $d_C(x, y) = 1$. Here, $\|\cdot\|$ denotes the Euclidean norm (also referred to as ℓ^2-norm). Note that for ℓ^2-normalized vectors x, y, one obtains

$$d_C(x, y) = 1 - \langle x|y\rangle = \frac{\|x - y\|^2}{2}. \tag{3.8}$$

Remember from the last step of chord recognition procedure, the assigned chord label is the one which minimizes the distance between the corresponding template and the given feature vector x. Plug in the three different template sets in Equation (4.7), we derive three template-based recognition methods as follows.

For binary templates we have:

$$\lambda_x := \underset{\lambda\in\Lambda, \mathbf{t}^b_\lambda \in \mathcal{T}^b}{\operatorname{argmin}} \; d_C(\mathbf{t}^b_\lambda, x). \tag{3.9}$$

For harmonically enriched templates we have:

$$\lambda_x := \underset{\lambda\in\Lambda, \mathbf{t}^h_\lambda \in \mathcal{T}^h}{\operatorname{argmin}} \; d_C(\mathbf{t}^h_\lambda, x). \tag{3.10}$$

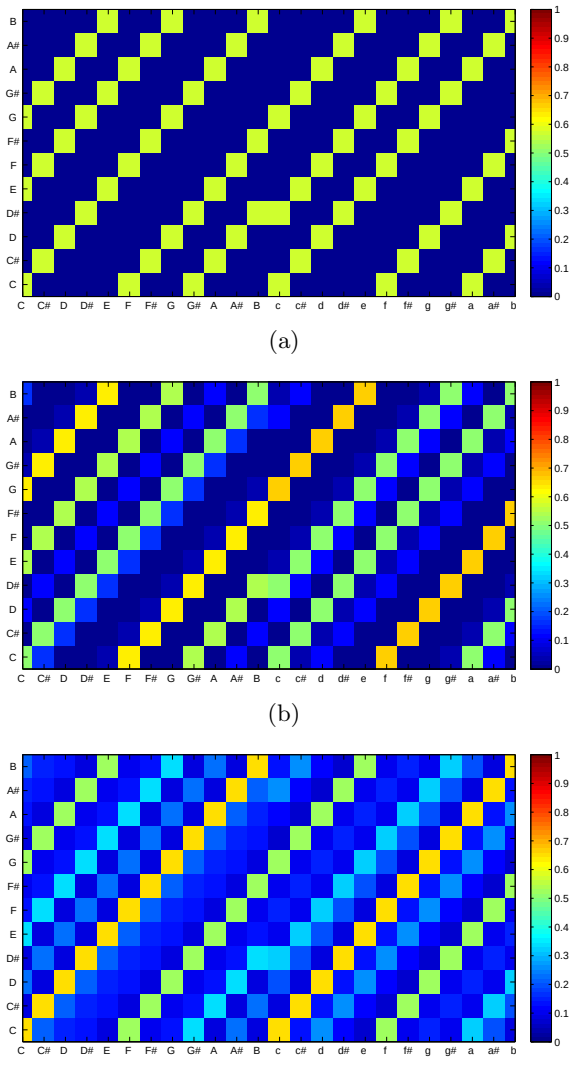
For averaged templates we have:

$$\lambda_x := \underset{\lambda\in\Lambda, \mathbf{t}^a_\lambda \in \mathcal{T}^a}{\operatorname{argmin}} \; d_C(\mathbf{t}^a_\lambda, x). \tag{3.11}$$

3.4 Template Method Summary

All the template methods we discussed previously are straight forward depictions of the chord patterns. They differ mainly on the perspective of describing the chord. Figure 3.5 illustrate the set of all templates of the three different variations.The difference of the settings can be easily seen. The binary templates only give weights to the component notes. The harmonically enriched templates add the consideration of overtones and give the weights to both the component notes and their harmonics. The two set of templates consist of manually set values, and they did not involve any information of the actual data. The averaged templates comes to solve this problem. They are constructed using all the audio files, therefore the averaged value is a profile of how the chord is expressed in practice.

It is hard to judge the quality of these methods by whether they are set with theoretically values or practical values. Sometimes the recognition method using simple binary templates performs the best, and the harmonically enriched templates or averaged templates can also win in some test experiments. However there is a problem with the averaged templates. They are heavily depending on the training data. If the training data contains too many bad instances of chords, the templates we computed might be biased. For example if a training dataset contains more C augmented triad than `C`, since in our system the augmented are mapped to major, the resulting template will have a higher weight both to the note G and G$^\sharp$, which in theory G$^\sharp$ should not occur, and of course such kind of template is not the exact pattern of `C`.



(a)

(b)

(c)

Figure 3.5. Three variation of template sets. (a)$\mathcal{T}^b$. (b)$\mathcal{T}^h$. (c)$\mathcal{T}^a$ using feature `CENS`$(1,1)$.

Chapter 4

Statistical Model-based Chord Recognition

The previously described template-based chord recognition methods define chord patterns in the form of feature vectors. They can be either manually set values for the case of binary templates and harmonically enriched templates, or practically learned values for the case of averaged templates. In this chapter we introduce some statistical model-based methods which define chord patterns not only considering each chroma of feature vectors, but also considering the mutual relation between these chromas. Instead of using feature templates as chord patterns as in the last chapter, we use multivariate Gaussian models to form the chord patterns in this chapter.

This chapter is structured as follows. In Section 4.1, we introduce the multivariate Gaussian model focus on the interpretation of the mean vector and the covariance matrix. In particular, we discuss the covariance matrix in detail. After that we present three chord recognition methods which are based on the multivariate Gaussian model. All these methods consist of both the stage of training and testing. The training stage is the same for each of them. The idea of training is to use some training dataset to learn multivariate Gaussian models as chord patterns, in other words, to estimate the model parameters: mean vectors and the covariance matrices. We describe the specifications of the training process in Section 4.2. Then we describe the testing stage of each recognition method in the following sections. Firstly, in Section 4.3 we introduce the "Mahalanobis distance" based method. This method computes the Mahalanobis distance for a feature and a chord pattern. Secondly, we talk about the "Gaussian probability" in Section 4.4. This method computes the Gaussian probability of a test feature coming from a chord model, or in other words, how closely a test feature match a chord pattern which expressed by Gaussian model. Finally, the recognition method using Hidden Markov Models will be presented in Section 4.5. This method involves a further parameter, the transition matrix, to work as post-smoothing with the consideration of neighbor chords.

4.1 Multivariate Gaussian Distribution

The multivariate Gaussian distribution, or multivariate normal distribution, is one of the most important multidimensional distributions. It is very often to find that the real world data distributed at least approximately multivariate normally. In this thesis, we assume that the feature vector is a 12-dimensional random vector which follows the multivariate Gaussian distribution. In this way, a chord pattern can be described by the 12-dimensional Gaussian distribution. This distribution can be completely determined by the mean vector and the covariance matrix. Some of the content in this section is summarized from the description in [32], [8] and [1].

4.1.1 Mean Vector

Definition 4.1 *Let $x \in \mathbb{R}^D$ be a D-dimensional random vector and all its entries have finite variance.*

The mean vector *of x is a vector μ consisting of the expectation of each element of x, concretely,*

$$\mu = \begin{pmatrix} E(x(1)) \\ E(x(2)) \\ \cdots \\ E(x(p)) \end{pmatrix}$$

The mean vector describes the centroid of a distribution, it tells the center location where the data is distributed in the coordinate system.

4.1.2 Covariance Matrix

Definition 4.2 *Let $x \in \mathbb{R}^D$ be a D-dimensional random vector, and all of its elements have finite variance, then the* covariance matrix $\Sigma \in \mathbb{R}^{D \times D}$ *is the matrix with the value at (i, j)th entry being the covariance:*

$$\Sigma_{ij} = \operatorname{cov}(x(i), x(j)) = \mathrm{E}\left[(x(i) - \mu(i))(X(j) - \mu(j))\right]$$

where:

$$\mu(i) = \mathrm{E}(x(i))$$

is the expected value of the ith entry of the vector x.

The definition above is can be rewrite into the matrix form:

$$\Sigma = E[(x - E[x])(x - E[x])']$$

This matrix form can be seen as a generalization from scalar variance (one-dimensional) to higher dimensions. Remember that in the one-dimensional case the variance of a random variable X^1 is $\sigma^2 = var(X) = E[(X - \mu)^2]$ where $\mu = E(X)$.

4.1.3 Interpretation of Covariance

Since the entries of the covariance matrix are composed by the covariance values, here we discuss the meaning of covariance and which information it models.

As implied by its name, the covariance measures how strong the relationship is between two random variables, say X_1 and X_2. There are two extremes of the covariance. Firstly, the two random variables are totally independent, given the value of the X_1 offers no hint about the value of X_2. In this case, there is no relationship between X_1 and X_2. Secondly, the relationship is quite strong that one variable can directly be determined by the other, for example, $X_2 = f(X_1)$. Given the value of X_1, the value of X_2 is also known without any uncertainty. In practice, the relationship of two random variables is somewhere in between: giving the value of X_1 offers some hints to the value of X_2, thus reduce the uncertainty of the values X_2 would take.

Since the value of the covariance offers the strength of relationship between two random variables, it would be natural to consider the case when they have a strong link: no matter X_1 is positive or negative, it is very likely that X_2 has the same sign. If X_1 is positive, it is very likely that X_2 is positive as well. However, this idea describes the covariance under the assumption that both of the random variables are centered at the mean value 0, in other words, the idea lacks the consideration of translation.

Here is a better description involved with the mean value: if $(X_1 - \mu_1)$ is positive, it is very likely that $(X_2 - \mu_2)$ is also positive, here μ_1 and μ_2 are the mean values of X_1 and X_2, respectively. So are the negative values. We depict this phenomena from another point of view: $(X_1 - \mu_1)(X_2 - \mu_2)$ is likely to be positive, because either both the quantities are simultaneously positive or simultaneously negative.

Note here the scalar product $(X_1 - \mu_1)(X_2 - \mu_2)$ is also a variable which has random value, since we want to have a fixed number to measure the relationship, we take the expectation of $(X_1 - \mu_1)(X_2 - \mu_2)$, and name it as covariance of X_1 and X_2.

Definition 4.3 *The* covariance *between two random variables X_1 and X_2 is:*

$$Cov(X_1, X_2) = E[(X_1 - E[X_1])(X_2 - E[X_2])]$$

Having introduced the formal definition above, here we discuss some interpretations about the value of the covariance:

1. *Large positive value*: A large value indicates that $(X_1 - \mu_1)$ and $(X_2 - \mu_2)$ do have a strong relation between themselves, and it can be shown that relation is then necessarily linear. A positive value indicates that $(X_1 - \mu_1)$ and $(X_2 - u2)$ always

[1] In this thesis, we denote a random vector by x and a random variable by X.

share the same sign, which means they are simultaneously positive or simultaneously negative.

2. *Small positive value*: A small positive value generally gives very few hints about the connection between the two variables. Which means, if X_1 takes a large value, there is little certainty which value X_2 would take, in other words, X_2 can take any value. This also means that, if X_1 changes, it is not likely that X_2 also alters because of X_1.

3. *Very low value* (close to 0): In general, none of the interpretations can be inferred from this value. Either the very low value of covariance is indeed a result of the weak connection between the two variables, or the low value is caused by some nonlinear relationship which the covariance cannot model, and it is the nature of that nonlinear relationship makes the covariance low.

4. *Negative value*: Similarly to the interpretation of positive value, but the negative sign indicates that if $(X_1 - \mu_1)$ takes a large positive value, then $(X_2 - \mu_2)$ is likely to take large negative value. They change in the opposite direction.

5. *Zero*: If the two random variables are independent, then the covariance is 0. But the converse is not true: two variables can have zero covariance, but they are not independent. For example: X_1 is uniform distributed in the range of $[-1, 1]$, and $X_2 = X_1^2$.

4.1.4 Diagonal Entries and Off-diagonal Entries of the Covariance Matrix

Generally speaking, each entry in the covariance matrix is the covariance value of two component variables of the random vector. Note that on the diagonal, the entries are special because they are actually representing the variance of certain component of the random vector, which means that how large the dispersion of that variable is from its mean. The other off-diagonal entries are the regular covariance which measures the linear coupling between the two different components of the random vector. In the later chapter, we conduct experiments on the diagonal entries and off-diagonal entries of the covariance matrix, in order to inspect what the impacts they have on the Gaussian model and how they further influence the performance of the chord recognition system.

4.1.5 Estimation of Covariance Matrix

Basically, there are two methods for estimating the covariance matrix. The first estimation is an unbiased estimator of the covariance matrix, regardless of the what distribution the random variable X follows, provided that the theoretical mean and covariance exist.

Given a set of data samples consisting of n independent observations $x_1, \cdots, x_n$, each of the observation is a p-dimensional random vector $x \in \mathbb{R}^D$, an unbiased estimator of the

$(D \times D)$ covariance matrix $\Sigma = E[(x - E[x])(x - E[x])^T]$ is the sample covariance matrix:

$$\Sigma_{unbiased\_estimate} = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})(x_i - \overline{x})^T \tag{4.1}$$

where x_i is the ith observation, and $\overline{x}$ is the sample mean.

The reason for the divisor being $n-1$ rather than n is the same as the unbiased estimator of sample variance and sample covariance. It happenes because during the estimation, the true mean of the distribution is unknown so that the sample mean is used instead.

The second estimation is for the case which the estimation can be derived on the basis of distribution. When the random vector $x \in \mathbb{R}^D$ is normally distributed, the maximum likelihood estimator is used to estimate the covariance matrix. It turns out that the MLE estimator is slightly different from the unbiased estimator we mentioned above.

$$\Sigma_{MLE\_estimate} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \overline{x})(x_i - \overline{x})^T \tag{4.2}$$

We can find that the two estimations only differs at the divisor. The unbiased estimator is divided by $(n-1)$ whereas the MLE estimator is divided by n. However, when the amount of data samples gets larger, this difference gets smaller. If the amount of samples goes to infinite, then the difference diminishes. In the training stage which we describe later, the covariance matrices of the Gaussian model which we trained are estimated using the unbiased estimator as Equation 4.1.

4.2 Specification of the Chord Models

After introducing the theoretical knowledge, from this section on we discuss the practical methods which use the multivariate Gaussian models as basis. The general idea of these methods is, instead of using templates, one can take multivariate Gaussian models to represent the chord patterns. All these methods involve both the training stage and the testing stage, and the computation of training is the same for each of them.

The aim of the training is to estimate the following parameters from the training dataset: the mean vectors and the covariance matrices of the 24 models corresponding to the 24 chords. The procedure of training is very similar to how we generate the averaged templates in Section 3.2.4, except for the last step. Recall the procedure in Figure 3.3, after cyclically shifting all extracted features and chord labels to `C` and `Cm`, we average all the features for `C` and `Cm` respectively to get the averaged templates. Here in the training of Gaussian models, we replace the last step by the estimation of mean vectors and covariance matrices of `C` and `Cm`. The mean vector is actually equal to the averaged templates, and the covariance matrix is estimated by the unbiased estimator as Equation 4.1.

We denote the estimated covariance matrix, mean vector and the resulting Gaussian density functions for the chord pattern of `C` as $\Sigma_{\mathtt{C}}$, $\mu_{\mathtt{C}}$ and $f_{\mathtt{C}}$, respectively. Similarly, we use the same denotation for the other chords. For each chord label $\lambda \in \Lambda$, we represent

the corresponding parameters as Σ_λ, μ_λ and f_λ. Again as in Section 3.2.4, we compute the Gaussian models for the other chords by cyclically shifting back $\Sigma_{\mathtt{C}}$ and $\mu_{\mathtt{C}}$ for major chords. The minor chords are cyclically shifted from $\Sigma_{\mathtt{Cm}}$ and $\mu_{\mathtt{Cm}}$ as well.

Since we perform the same training algorithm for each of the datasets, the differences of the estimated parameters are only due to the different training dataset we use. Among all the data we have, we previously select some of the data as training data and use them to train the models. Furthermore, we split the training data into different sets and then use any of them or any combination of them in the actual training step. This aims at reducing the risk that one dataset might containing many outliers. We denote a training dataset as $\mathcal{S}$. For specific datasets which we use in our experiments, we name them concretely such as $\mathcal{D}_1^{\text{Beatles}}$, $\mathcal{D}_2^{\text{Beatles}}$ and $\mathcal{D}_3^{\text{Beatles}}$ [2].

Furthermore, we will denote the estimated multivariate distribution as $\mathcal{N}$ with the name of training data as superscript, such as $\mathcal{N}^{\mathcal{D}_1^{\text{Beatles}}}$.

Since the multivariate Gaussian distribution is only determined by its mean vector and covariance matrix, the model will keep both of them in order to indicate a distinct distribution. In the following, we denote the estimated distribution with subscript indicating the used training data and superscript indicating the chord of the distribution. Denoting a multivariate Gaussin distribution as $\mathcal{N}$, we define

$$\mathcal{N}_\lambda^{\mathcal{S}}(\mu, \Sigma) := \mathcal{N}(\mu_\lambda^{\mathcal{S}}, \Sigma_\lambda^{\mathcal{S}})$$

For example, the Gaussian model of $\mathtt{C}$ trained on $\mathcal{D}_1^{\text{Beatles}}$ can be represented as:

$$\mathcal{N}_{\mathtt{C}}^{\mathcal{D}_1^{\text{Beatles}}}(\mu, \Sigma) := \mathcal{N}(\mu_{\mathtt{C}}^{\mathcal{D}_1^{\text{Beatles}}}, \Sigma_{\mathtt{C}}^{\mathcal{D}_1^{\text{Beatles}}})$$

In the end, by plugging in the the estimated μ and Σ, the estimated Gaussian density function which we used in the following methods is:

$$f(x) = (2\pi)^{-D/2} \det(\Sigma)^{-1/2} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right) \tag{4.3}$$

where $f = f_\lambda^{\mathcal{S}}$ with $\mu = \mu_\lambda^{\mathcal{S}}$ and $\Sigma = \Sigma_\lambda^{\mathcal{S}}$.

4.3 Mahalanobis Distance based-Method

The first method we introduce is based on the Mahalanobis Distance[3]. According to [34], "the Mahalanobis distance is a very useful way of determining the similarity of a set of values from an unknown sample to a set of values measured from a collection of known samples". In our scenario, we treat a feature vector which needs to be assign chord label as an unknown sample, and a trained chord distribution representing the the collection of known samples. Since it takes into account the correlations or covariance of the known

[2] These datasets will be described in Section 5.1.1.

[3] The Mahalanobis Distance is first introduced by by P. C. Mahalanobis in 1936.

distribution, it makes itself different from the Euclidean distance which only considers the mean value. Furthermore, Mahalanobis Distance is scale-invariant [19] which means it does not depend on the scale of measurement. This makes it more powerful while analyzing multivariate data.

While applying the Mahalanobis Distance on the chord recognition task, we make two assumptions: the first one is that the distribution of our feature vectors should behave as multivariate normal distribution of a random vector; the second one is that the distributions of the 24 chords should be distinct among each other, otherwise it will yield same distance and make the chord undeterminable. For the first assumption, we check the distribution of practical data and find this assumption holds. For the second assumption, since we use the practical data to train the covariance matrix, and cyclically shift the $\Sigma_{\mathtt{C}}$ and $\Sigma_{\mathtt{Cm}}$ to other major and minor chords, the covariance matrix of the 24 chords are distinctive. Furthermore, the mean vector of each of all chords are different as well. Therefore the second assumption can be fulfilled.

4.3.1 Specification of Distance Measure

Definition 4.4 *Let x be a feature vector, $x \in \mathcal{F} \setminus \{0\}$ and D be the dimension of $\mathcal{F}$, and let $\mu \in \mathbb{R}^D$ be the mean value and $\Sigma \in \mathbb{R}^{D\times D}$ be the covariance matrix of a multivariate normal distribution. Then, we define the Mahalanobis distance $d_M = d_M^{\mu,S} : \mathcal{F} \times \mathcal{F} \to \mathbb{R}$ with respect to μ, Σ by*

$$d_M(x) = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)} \tag{4.4}$$

Typically, the matrix Σ must be semi-positive definite. Furthermore, if the matrix S is the identity matrix, the Mahalanobis distance reduces to the Euclidean distance. If the Σ is diagonal, then the resulting distance measure reduces to the normalized Euclidean distance:

$$d(x) = \sqrt{\sum_{i=1}^{N} \frac{(x(i)-\mu(i))^2}{\sigma(i)^2}}, \tag{4.5}$$

where $\sigma(i)$ is the ith diagonal value of Σ.

4.3.2 Algorithm Procedure

After defining the trained models and distance measure, the chord recognition procedure using Mahalanobis Distance can be described. Similar as in template-based methods, we need to pass the feature sequence $X := (x_1, x_2, \ldots, x_N)$ which extracted from the audio file in the following procedure.

- Firstly, the previously trained models describing the distribution of 24 chords are loaded. We take the mean vector and the covariance matrix from each of the chord models. Here ignoring the concrete training dataset for simplicity, suppose the arbitrary parameters which we load are $\mu_\lambda^{\mathcal{S}}$ and $\Sigma_\lambda^{\mathcal{S}}$

- Secondly, we take the Mahalanobis distance as a measurement to quantify the distance between a feature vector and a chord distribution. By plugging in $\mu_\lambda^{\mathcal{S}}$ and $\Sigma_\lambda^{\mathcal{S}}$, we can measure the distance between x_n and any of the chord distribution $\mathcal{N}_\lambda^{\mathcal{S}}(\mu, \Sigma)$. We denote the distance by $d_M(\mathcal{N}_\lambda^{\mathcal{S}}(\mu, \Sigma), x_n)$.

- Finally, similar to template-based methods, we assign the chord label that minimizes the distance between the corresponding distribution and the given feature vector $x = x_n$:

$$\lambda_{x_n} := \underset{\lambda \in \Lambda}{d_M}(\mathcal{N}_\lambda^{\mathcal{S}}(\mu, \Sigma), x_n). \tag{4.6}$$

4.4 Gaussian Probability based-Method

Instead of having similarity measured by distance function, we define a similarity measurement which allows the comparison of a feature with a Gaussian distribution. The idea is that given a distribution and a feature vector, we treat the feature vector as a sample vector and measure how possible the sample comes from the distribution. In other words, we can measure how possible that a feature vector fit the distribution of a chord pattern. In this way, we can quantify the relationship between a feature vector and a distribution by probability. In this section, we consider the case of multivariate Gaussian distribution and introduce the chord recognition method based on Gaussian probability.

Recall Equation (4.3) that the density function of Gaussian distribution is:

$$f(x) = (2\pi)^{-D/2} \det(\Sigma)^{-1/2} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

For a given chord label $\lambda \in \Lambda$ and a given training dataset $\mathcal{S}$, we denote the density function with $f = f_\lambda^{\mathcal{S}}$ and the parameters as $\mu = \mu_\lambda^{\mathcal{S}}$ and $\Sigma = \Sigma_\lambda^{\mathcal{S}}$.

Based on the formula we see that Gaussian probability is just the weighted exponential form of the Mahalanobis Distance. One can even use the logarithm form so that it is directly related with the Mahalanobis Distance. Both of the methods are based on relative comparison to select the maximum case (or minimum case in Mahalanobis) as the recognition result.

4.4.1 Algorithm Procedure

The chord recognition procedure using Gaussian Probability is the same as the procedure using the Mahalanobis Distance in the first step, i.e., the step of loading the model parameters. We make changes in the following steps, which correspond to the similarity measure and chord label assignment.

- To calculate the similarity, we take the Gaussian Probability as a measurement to quantify the possibility of a feature vector x_n coming from a chord distributions $\mathcal{N}_\lambda^{\mathcal{S}}$

corresponding to the chord label λ and training data set $\mathcal{S}$. By plugging in these parameters into $f_\lambda^{\mathcal{S}}$, we calculate all the probabilities for every $\lambda \in \Lambda$.

- In the end, the chord label is assigned to the feature vector $x = x_n$ by selecting the one whose model gets the maximum probability.

$$\lambda_{x_n} := \underset{\lambda \in \Lambda}{\operatorname{argmin}} \ f_\lambda^{\mathcal{S}}(x_n). \tag{4.7}$$

4.5 Hidden Markov Models-based Method

A Markov model is a stochastic model which fulfilles the Markov property. According to [14], in a hidden Markov model (short form for HMM), "the output for each state corresponds to an output probability distribution." An HMM can be represented by its initial probability, observation probability and transition probability. Adapting these parameters to chord recognition, we consider a chords as a hidden state in HMM, and a feature vector as an observation. The problem can be reformulated as given the sequence of observations of features, what are the underlying states of chords which generate the feature sequence. We use the Gaussian probability which we computed in the last section as observation probability, and train the transition probability from the training data. The initial probability is set to 1/24 indicating each of the chords has a fair chance.

The main idea of HMM-based chord recognition is to introduce temporal context into the chord recognition process. This can be seen as a kind of added post-filtering after the primary pattern matching step. The post-filtering serves as smoothing of the predicted chord labels utilizing the context over time, in order to reduce wrong recognized frames caused by noise or strong onsets, and also to smooth fluctuating frames inside a continued chord duration [4].

4.5.1 Transition Probability

In order to consider the temporal context, one needs a transition probability matrix $A \in \mathbb{R}^{|\Lambda| \times |\Lambda|}$ for a given chord label set Λ. This matrix describes the first-order temporal relationships between the various chords. Each element of the matrix represents the probability of a chord jumping to another. For example, $A(\lambda_1, \lambda_2), \lambda_1 \neq \lambda_2$ represents the probability chord λ_1 jump to λ_2, and $A(\lambda_1, \lambda_1)$ represents the probability of staying in the same chord λ_1.

There are many ways to specify a transition probability matrix. Such a matrix may be trained from a training dataset, we denote the matrix with subscript t represent it is trained and superscript the training data set, for example $A_t^{\mathcal{D}_1^{\text{Beatles}}}$. For the matrix obtained from training, we first parse the original label file into framewise labels with respect to the fixed feature rate of 10 Hz, i.e., 10 labels per second with each of them indicating the chord label for 100 millisecond. To solve the rounding problem of time points which do not start or end sharply at hundred milliseconds, we take the ceiling of the time point when it is the start of a label, while floor of the time point when it is the end of a label. Now we

have the labels for the whole training set with the unit of each hundred millisecond. In the end, the values of the element in the transition matrix is assigned to:

$$A_t(\lambda_1, \lambda_2) = \frac{C(\lambda_1, \lambda_2)}{\sum_{\lambda \in \Lambda} C(\lambda_1, \lambda)} \tag{4.8}$$

where $C(\lambda_1, \lambda_2)$ counts the number of chords jumping where the current label is λ_1 and next label is λ_2, and $\sum C(\lambda_1, \lambda)$ serves as a normalization which counts the jumping from λ_1 to all the labels including itself.

Another way of specifying the transition matrix is to set the value of its elements manually using musical knowledge, we denote such a transition matrix as A_m, with subscript m indicating it is manually set. For example, Bello and Pickens designed a figure of doubly-nested circle fifth to model the chord transition [3].

4.5.2 Viterbi Decoding

The *Viterbi algorithm* is a dynamic programming algorithm to find the most possible sequence of underlying states of hidden Markov models given the sequence of observed events. We use it to decode the hidden states of chords given the sequence of observed feature vectors. In the following, part of the content is summarized from [14] and [31].

The Gaussian probability-based chord recognition as described in Section 4.4 is worked as the observation probability in HMM. The observation probability describes if the current hidden state is i, how likely it will lead to the observed event j. In the discrete form of HMM, the observation probability of the states is usually discrete and fixed. However in the implementation of our system, we need to adapt the HMM in its continuous form: instead of using a fixed value, the observation probability for a certain state is modeled by a specific Gaussian density function. Furthermore, we treat any given feature vector as observed event and plug this vector into the density function. By doing this, as discussed in Section 4.4, we calculate the probability of the vector coming from a certain chord distribution. This probability is then treated as the observation probability which indicates the possibility that a chord state generates such a feature vector as observed event. Certainly, this will lead to different possibilities for different feature vectors and therefore make the observation probability unfixed and vary.

The HMM work out the most possible hidden states by using the Viterbi algorithm. The Viterbi algorithm combines both the observation probabilities and transition probabilities to perform post-filtering, i.e., based on the primary pattern matching result of chord label using Gaussian Probabilities, the Viterbi algorithm will determine the final result considering the context influence specified by transition probabilities.

The idea of the Viterbi algorithm is a kind of dynamic programming which uses the states at time t to deduct the optimal states at $t + 1$. It works recursively to find the optimal hidden states given the observation sequence and the parameters of HMM.

The algorithm runs in two stages. The first stage calculate the objective probability for any state at any time. Suppose we have M hidden states and we denote each state as S_i, $i \in [1 : M]$. We denote the observed events as O_t, $t \in [1 : T]$. Then at an arbitrary

time point t, for each of the states S_i, a partial probability $\delta_t(S_i)$ is defined to indicate the probability of the most probable path ending at the state S_i, given the current observed events $O_1, \cdots, O_t$:

$$\delta_t(S_i) = \max_j(\delta_{t-1}(S_j) \cdot A(S_j, S_i)) \cdot P(O_t|S_i) \tag{4.9}$$

Here, we assume that we already know the probability $\delta_{t-1}(S_j)$ for any of the previous states S_j at time $t-1$, and using the transition probability of $A(S_j, S_i)$ and multiplying it by the current observation probability $P(O_t|S_i)$. We then select the maximum $\delta_t(S_i)$.

After having all the objective probabilities for each state at each time point, the algorithm seeks from the very end backwards to the beginning to find the most probable path of states for the given sequence of observation events.

$$\phi_t(i) = \operatorname*{argmax}_j(\delta_{t-1}(S_j) \cdot A(S_j, S_i)) \tag{4.10}$$

where $\phi_t(i)$ indicates which state is the most optimal state at time t based on the probability computed in the first stage.

Chapter 5

Experiments

In this chapter we present the extensive experiments which we conducted to inspect the effect of different stages of our chord recognition system. We depict the overall experimental setup in Section 5.1. In particular, the dataset including audio files and chord annotations will be described in detail and the evaluation methodology will be introduced. In Section 5.2 we discuss the evaluation measures based on precision and recall. In Section 5.3, we conduct experiments on extracted features. Here, the optimization techniques such as logarithmic compression, smoothing, pitch range separation will be extensively evaluated. Besides, we discuss the results of experiments and analyze certain effect. In Section 5.4, we do experiments on chord recognizers. Here, the six different chord recognizers as described in Chapter 3 and 4 will be extensively evaluated. We will point out the advantage and disadvantage of a certain recognition methods when combing with different features. From Section 5.5 to 5.8, experiments on other aspects will be introduced. We first present the effect of tuning and using different training data. Then, the impact of combining our existed chord recognition module with other techniques will be analyzed. These aspects include harmonic-percussive source separation and using beat-wise features for evaluation instead of frame-wise features.

5.1 Experiments Setup

5.1.1 Dataset

In this thesis, all the datasets we use consist of both audio files and corresponding chord annotation files which contain ground truth chord labels.

We prepared three different kinds of music data for our experiments. Firstly, we take the Beatles albums as representatives for pop music. We take all 180 songs as a whole dataset, and denote it as $\mathcal{D}^{\text{Beatles}}$. We further partition $\mathcal{D}^{\text{Beatles}}$ into three subset, where all the recordings are ordered alphabetically and we put the first 60 recordings into $\mathcal{D}_1^{\text{Beatles}}$, the second 60 recordings into $\mathcal{D}_2^{\text{Beatles}}$, and the last 60 recordings into $\mathcal{D}_3^{\text{Beatles}}$. The chord annotations for Beatles albums are written by Christopher Harte [13]. Secondly, we take the four classical pieces as representatives for classical music. We put these four pieces

together in a dataset and name it as $\mathcal{D}^4$. See Table 5.1 for details. The chord annotations in this dataset are precisely written by Verena Konz. Furthermore, we particularly choose 16 Beatles recordings and four classical pieces together as a combination of pop and classical music. We name it as $\mathcal{D}^{20}$. See Table 5.2 for detail.

File ID	File name (first 50 characters)	Performer	Length
BachBWV846Fischer	Bach_BWV846_Fischer_22050_mono	Fischer	84
Beet5Bernstein	Beethoven_op067_1_symphony_5_bernstein_22050_mono	Bernstein	519
ChopMazurkaSmith	pid9054b-21	Smith	91
SchumannKonz	Schumann_Op068No4_Konz	Konz	83

Table 5.1. Description of the dataset $\mathcal{D}^4$.

File ID	File name (first 50 characters)	Performer	Length
BeatlesAHardDaysNight	Beatles_AHardDaysNight_Beatles_1964-AHardDaysNigh	Beatles	152
BeatlesAllMyLoving	Beatles_AllMyLoving_Beatles_1963-WithTheBeatles-0	Beatles	129
BeatlesAllYouNeedIsLove	Beatles_AllYouNeedIsLove_Beatles_1967-MagicalMyst	Beatles	228
BeatlesBoys	Beatles_Boys_Beatles_1963-PleasePleaseMe-05	Beatles	147
BeatlesDoYouWantToKnowASecret	Beatles_DoYouWantToKnowASecret_Beatles_1963-Pleas	Beatles	119
BeatlesEightDaysAWeek	Beatles_EightDaysAWeek_Beatles_1964-BeatlesForSal	Beatles	165
BeatlesGotToGetYouIntoMyLife	Beatles_GotToGetYouIntoMyLife_Beatles_1966-Revolv	Beatles	150
BeatlesHelp	Beatles_Help_Beatles_1965-Help-01	Beatles	141
BeatlesHereComesTheSun	Beatles_HereComesTheSun_Beatles_1969-AbbeyRoad-07	Beatles	185
BeatlesLetItBe	Beatles_LetItBe_Beatles_1970-LetItBe-06	Beatles	243
BeatlesLovelyRita	Beatles_LovelyRita_Beatles_1967-SgtPeppersLonelyH	Beatles	162
BeatlesLoveMeDo	Beatles_LoveMeDo_Beatles_1963-PleasePleaseMe-08	Beatles	142
BeatlesTicketToRide	Beatles_TicketToRide_Beatles_1965-Help-07	Beatles	192
BeatlesTwistAndShout	Beatles_TwistAndShout_Beatles_1963-PleasePleaseMe	Beatles	153
BeatlesWhatGoesOn	Beatles_WhatGoesOn_Beatles_1965-RubberSoul-08	Beatles	170
BeatlesYesterday	Beatles_Yesterday_Beatles_1965-Help-13	Beatles	127
BachBWV846Fischer	Bach_BWV846_Fischer_22050_mono	Fischer	84
Beet5Bernstein	Beethoven_op067_1_symphony_5_bernstein_22050_mono	Bernstein	519
ChopMazurkaSmith	pid9054b-21	Smith	91
SchumannKonz	Schumann_Op068No4_Konz	Konz	83

Table 5.2. Description of the dataset $\mathcal{D}^{20}$.

5.1.2 Annotation Convention

In order to have a universal format of chord annotations, a certain rule is introduced by Christoph Harte in [13]. For the four classical pieces which we used, Verena Konz followed exactly the same rule when annotating the chords. Besides, the annotated chords are forced to map into one of the 24 triads when we parse the annotations. Here, we only consider the first two intervals of each chord [16]. Thus, augmented chords are mapped to major and diminished chords are mapped to minor, respectively. In some cases, there are regions where no chord exists. Such regions are annotated with "N" as annotation. In our evaluation, we discard the regions where no chords exist, that is to say, whatever the computed result is in this region, it does not affect the recognition accuracy for the whole piece.

5.2 Evaluation via Precision and Recall

The evaluation measure of our chord recognition system is via precision and recall. In particular, we take F-measure, which is derived from precision and recall, as our recognition accuracy. In the field of information retrieval, precision and recall are defined as follows [20].

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|} \tag{5.1}$$

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|} \tag{5.2}$$

Adapt the concept into chord recognition, one can treat a chord label as a document. Therefore an annotated chord label can be seen as a relevant document and a computed chord label as a retrieved document. By plugging in both the label sequence of an audio file into equation 5.1 and 5.2, one can calculate the F-measure as recognition accuracy [35]:

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{5.3}$$

Note that our chord recognition system is based on *framewise* evaluation. All the chord annotations are segmented into frames with fixed length. The frame length depends on the feature rate which is always 10 in our experimental setting. The number of segmented annotation frames corresponds to the number of computed chord label frames, which equals to the number of feature frames.

5.3 Experiments on Extracted Features

From this section on, we discuss the extensive experiments which we conducted. In Chapter 2 we have introduced several features which transform the original audio files into musically meaningful representations, in particular, all of them are in the chroma representation. These features are then passed into the chord recognition module which make chord label decisions via classification based on some template methods or statistical models. To understand which influence the features make to the whole process, we conduct series of experiments in this section that mainly focus on the feature side. We evaluate the performance variations by using all the features with different setting of parameters, so that we can explore the effect of parameters. From the difference of the performance results, we can clearly discover how these feature settings increase or decrease the chord recognition accuracies. Moreover, we can infer the advantage and the disadvantage of using a certain type of feature when combining it with different chord recognizers.

This section is structured as follows. Section 5.3.1 illustrates the overall performance of the combinations of all features and recognizers. We provide two sets of results with different training datasets so that we can deduct the feature performance more generally than

using a single dataset. Section 5.3.3 illustrates the influence of logarithmic compression on the magnitude of loudness. With different extents of compression, we will see the changes in the chromagram and changes in the recognition result as well. By enlarging the window size when computing the CENS and CRP, we examine how smoothing affects the final performance in the Section 5.3.4. Finally, Section 5.3.5 discuss the pitch range separation in order to observe the different behavior of features extracted from only the melody part or the bass part of a piece of music.

5.3.1 Overall Performance

As the first introductive experiment, we provide the overall recognition accuracies of our chord recognition system with all feature types and all chord recognition methods. The parameters of the features are selected to be the optimal ones in this overall illustration. Here we use $\mathcal{D}^{\text{Beatles}}$ as the test dataset so that we can compare the performance of our system to the ones in other groups. Note that recognition accuracies of the two recognition methods $\mathtt{T}^{\mathtt{b}}$ and $\mathtt{T}^{\mathtt{h}}$ only depend on the test dataset while the recognition accuracies of other other four methods depend on both the test dataset and the training dataset. In order to alleviate the possible effect of unfair training, we provide two tables of results with each trained on different datasets respectively.

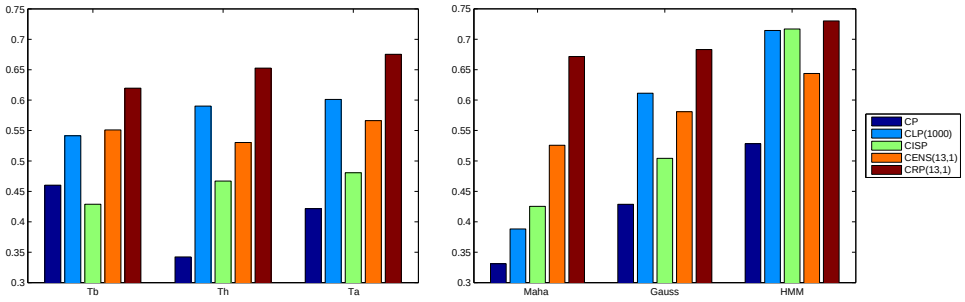
Table 5.3 shows the evaluation result of chord recognition for the five feature types and for six chord recognition methods with some of them previously trained on the dataset $\mathcal{D}_1^{\text{Beatles}}$ and all of them tested on $\mathcal{D}^{\text{Beatles}}$. The rows of the table change the chord recognizer in turn and the columns change the features in turn. In order to have a better visualization of the performance result, we provide the bar figure corresponding to the values of Table 5.3 in Figure 5.1. From both the table and the figure we can find that CP performs the worst with almost all chord recognizers except with Tb. This makes sense since CP is the most basic chroma feature without any additional processing such as smoothing, or logarithmic compression. Feature CLP(1000) performs quite well and can be considered as the second best feature. For all the chord recognizers except Maha, feature CLP(1000) wins the second place. (Here for HMM, CLP(1000) got 0.717 and CISP got 0.714, we ignore this small difference and consider both of them win the second place.) Feature CISP behaves very strange: it performs excellent with HMM but defective with other chord recognizers. It got one time the worst feature with Tb, three times the second worst place with Th Ta GP. This indicates that CISP depends very much on smoothing. Furthermore, its bad performance with Tb indicates that its magnitude computation is not that emphasizing the component note of the chord but gives extra weight to other notes. Feature CENS(13, 1) can be considered as a fair feature and wins the middle place among the five feature types. Note that it already contains quantization on magnitude, which works similar to logarithmic compression at the feature side. Additionally, smoothing is also included when giving the window size 13 which in this case equals to 1.3 seconds. Although it includes both the optimization techniques, it is still worse than CLP(1000) for most of the recognizers. This indicates that the magnitude computation of a single feature is more important than smoothing with context. However, this does not mean that smoothing is not important. Without smoothing, CENS behaves quite unsatisfiable. We will discuss this in Section 5.3.4. At last comes the winner feature CRP(13, 1). It wins the

first place with all the chord recognizers. As we described in Chapter 2, the CRP feature is designed to boost timbre invariance. Besides, it contains the logarithmic compression as CLP and smoothing as CENS. The combination of these techniques makes it prominent and outstanding among all features.

From Table 5.3 we can already tell the huge recognition difference caused by feature side. For all the chord recognizers, the best feature is always the CRP(13, 1) and the worst is CP for most of the time and CISP for one time. And for each of the recognizers, the biggest increased recognition accuracy is 0.20, 0.31, 0.25, 0.34, 0.26 and 0.21 respectively. This implies the importance of computing audio features in a suitable way. Without good features, one can not obtain satisfiable result even using the complex chord recognizers.

CR	CP	CLP(1000)	CISP	CENS(13, 1)	CRP(13, 1)
$\mathtt{T}^{\mathtt{b}}$	0.460	0.541	0.429	0.551	0.620
$\mathtt{T}^{\mathtt{h}}$	0.342	0.590	0.467	0.530	0.653
$\mathtt{T}^{\mathtt{a}}$	0.422	0.601	0.481	0.566	0.675
Maha	0.331	0.388	0.425	0.526	0.672
GP	0.429	0.611	0.504	0.581	0.683
HMM	0.528	0.714	0.717	0.644	**0.730**

Table 5.3. Recognition accuracies of all features and all recognition methods. Train: $\mathcal{D}_1^{\text{Beatles}}$. Test: $\mathcal{D}^{\text{Beatles}}$.



(a) Template-based methods (b) Statistical model-based methods

Figure 5.1. Bar visualization of Table 5.3. Result comparison from feature side.

Table 5.4 shows the evaluation result as well but using training dataset $\mathcal{D}_2^{\text{Beatles}}$. Since $\mathtt{T}^{\mathtt{b}}$ and $\mathtt{T}^{\mathtt{h}}$ do not include training procedure, the results of the both methods stays the same as in Table 5.3. By comparing the two tables, we can find that most of the values which locate at the same position of the two tables have only small differences below 0.030, except for the feature CLP(1000) with Maha surprisingly yield the biggest difference 0.105, and second biggest at CLP(1000) with HMM yield the difference 0.035. The later value is even being the highest recognition accuracies among the two tables. From this we conclude that except feature CLP(1000), all other features are not that sensitive to the variations of training data, and the feature CLP(1000) is comparably sensitive than the other features.

CR	CP	CLP(1000)	CISP	CENS(13, 1)	CRP(13, 1)
T^b	0.460	0.541	0.429	0.551	0.620
T^h	0.342	0.590	0.467	0.530	0.653
T^a	0.419	0.607	0.484	0.568	0.670
Maha	0.387	0.493	0.453	0.556	0.681
GP	0.411	0.621	0.496	0.562	0.660
HMM	0.525	**0.749**	0.717	0.639	0.710

Table 5.4. Recognition accuracies of all features with all recognition methods. Train: $\mathcal{D}_2^{\text{Beatles}}$. Test: $\mathcal{D}^{\text{Beatles}}$.

5.3.2 Feature Comparison

In this section, we compare features by visualizing all of the features extracted from the same audio file. We excerpt the first ten seconds from the song BeatlesLetItBe for illustration. Figure 5.2 shows the intensity difference of chromagrams among five different features on the left side. Then for each of the feature type, we pass the features to the template baseline method and get the corresponding recognition result. The visualization of these results and the recognition accuracy are shown on the right side of Figure 5.2. The image range of the chromagrams for feature CP, CLP(1000), CISP and CENS(13, 1) are set to $[0, 1]$ for easy comparison, while for feature CRP(13, 1) the range is $[-0.7, 0.7]$ because CRP features have negative intensity and the the range $[-0.7, 0.7]$ yields a good visualization. For the result visualizations, the four different colors have a specific meaning: green means annotated ground truth chords, red means computed chords but not coincide with ground truth annotation, blue means computed chords and coincide with ground truth annotation, white means no annotations.

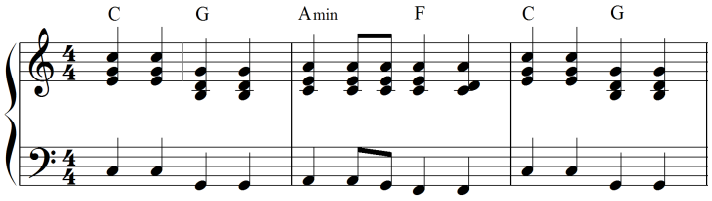
By comparing the color intensity of chromagrams, we can compare the magnitude or energy of different feature types. We find that CLP(1000) and CISP share very similar contrast in intensity, CP and CENS(13, 1) are to some extent similar as well. Although the intensity contrast of CRP(13, 1) is higher than other features, by careful observation we find the red highlight chromas and their time position are very close to the blue or yellow highlight chromas in CLP. Therefore we can roughly divide the five feature types into two groups by similarity of contrast: one is CLP(1000),CISP and CRP(13, 1); the other is CP and CENS(13, 1). Actually this similarity of contrast makes sense because it reflects the different intensity computation of the chromagrams. On one hand, for feature CP and CENS(13, 1), the intensity is in the form of energy, that is to say, the squared power of amplitude of the original signal. Here squared power implies that, the difference of intensity will be much enlarged after squaring the original value. On the other hand, for CISP, the intensity is in the form of magnitude, which derived from fourier transformation, and can be considered as the scaled amplitude of the original signal. For CLP(1000) and CRP(13, 1), the intensity of the both features are in the manner of energy, however they perform logarithmic compression on the energy afterwards while computing the intensity, therefore the effect of squared power is to some extent cancel out. This yields the intensity

of chromagram in a manner between energy or magnitude. This is also the reason why the chromagrams of CLP(1000) and CRP$(13, 1)$ have less contrast than CP and CENS$(13, 1)$.

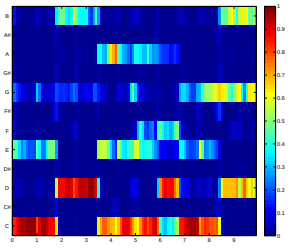
The logarithmic compression in energy actually plays an important role in the chord recognition task. This is because when speaking of a single chord, we give all component notes of this chord the same weight. Given the sheet music, we can decide the chords without any magnitude information from the played notes. Therefore the information of magnitude difference of component notes is not that useful in chord recognition task. For example, from 5.9 to 6.8 seconds of BeatlesLetItBe, the ground truth annotation is "F:maj6" and after mapping to triad chord it becomes F. In Figure 5.2, only CLP and CRP correctly recognize this period while the other features make wrong decision to Dm. From musical point of view, the error makes sense because "F:maj6" consists of the component notes F, A, C and D. They can either be mapped to F which is composed of F, A and C; or be mapped to D which is composed of D, F and A. In Figure 5.2(b) and 5.2(h), we can find that CP and CENS have a high weight for chroma D from 5.9 to 6.8 seconds, this directly leads to the wrong recognition of Dm instead of the correct F.

Another important factor is the smoothing effect. For the features without temporal smoothing, their behavior is comparably unstable than the features with smoothing. They have such disadvantage in particular for the case when the duration of a chord last for a long time. For example, from 4.7 to 4.9 seconds of BeatlesLetItBe, feature CP, CLP and CISP all wrongly recognize the chord as C due to the presence of the additional note G in the chord Am. However with smoothing window length 13, feature CENS and CRP successfully stay in Am because the contribution of note G is reduced when blending with the strong Am context. This is also the main advantage of features with smoothing to gain a higher recognition accuracy than those without smoothing especially for long lasting chords.

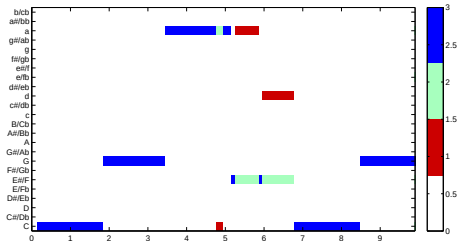
From the comparison among different features we conclude the positive effect of logarithmic compression and smoothing, however these two techniques also have limitations and sometimes yield negative effects. We will analyze this in detail in the following sections.



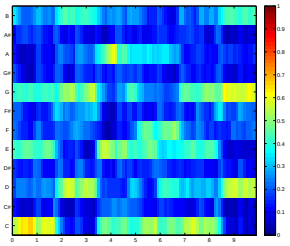
(a) Score of the first 10 seconds (corresponding to the first three measures) of `BeatlesLetItBe`.



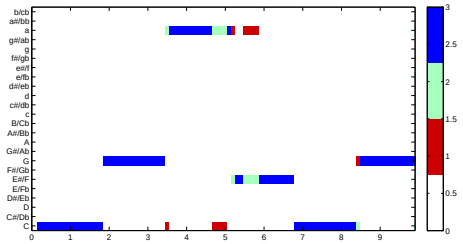
(b) `CP`



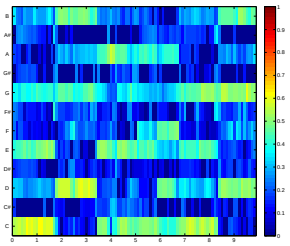
(c) Recognition accuracy 0.820



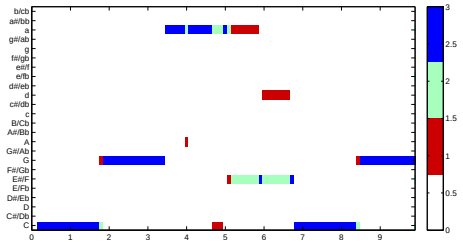
(d) `CLP`(1000)



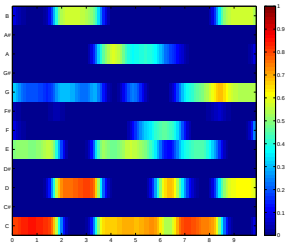
(e) Recognition accuracy 0.870



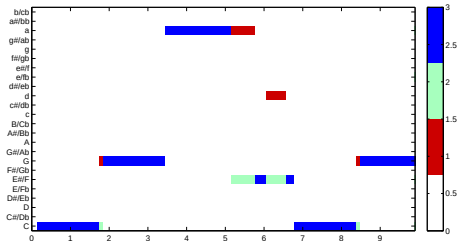
(f) `CISP`



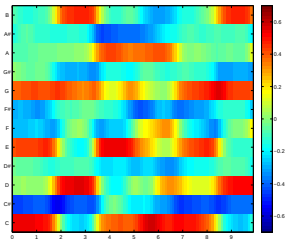
(g) Recognition accuracy 0.770



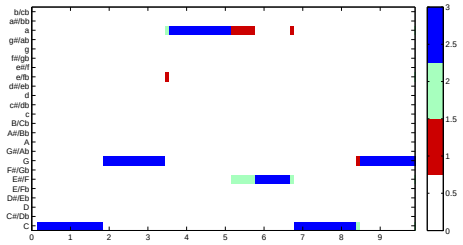
(h) `CENS`$(13, 1)$



(i) Recognition accuracy 0.850



(j) `CRP`$(13, 1)$



(k) Recognition accuracy 0.890

Figure 5.2. Comparisons of features and their recognition accuracies using chord recognizer $\mathtt{T_b}$.

5.3.3 Effect of Logarithmic Compression

As we mentioned in Section 2.3, we apply logarithmic compression to the original pitch features in order to simulate humans' sensation of sound intensity. Besides, this step also works as adjusting the dynamic range of the original signal to enhance the clarity of weaker transients, especially in the high-frequency regions [11]. To this end, we replace each intensity e of pitch features by the value $\log(\eta \cdot e + 1)$, where η in our experiments are set as 1,10,100,1000 and 10000. By changing the value of η in our experiments, we can easily see the influence of logarithmic compression on chord recognition.

Table 5.5 shows the evaluation result using four chord recognition methods and features with and without logarithmic compression. Here, `CP` denotes the original pitch features without logarithmic compression. Other features are `CLP` with different extent of compression specified by η. Besides, Figure 5.3 is generated by using the numbers of Table 5.5 and offers a better visualization.

CR	CP	CLP(1)	CLP(10)	CLP(100)	CLP(1000)	CLP(10000)
$\mathtt{T}^{\mathtt{b}}$	0.460	0.508	0.544	**0.553**	0.541	0.521
$\mathtt{T}^{\mathtt{a}}$	0.422	0.513	0.577	**0.607**	0.601	0.577
GP	0.429	0.525	0.584	0.608	**0.611**	0.606
HMM	0.528	0.640	0.693	0.712	**0.714**	0.563

Table 5.5. Recognition differences of features with various logarithmic compression parameter. Train: $\mathcal{D}_1^{\text{Beatles}}$. Test: $\mathcal{D}^{\text{Beatles}}$.

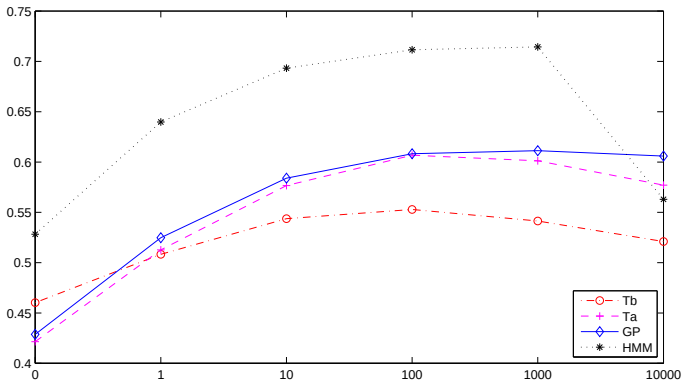


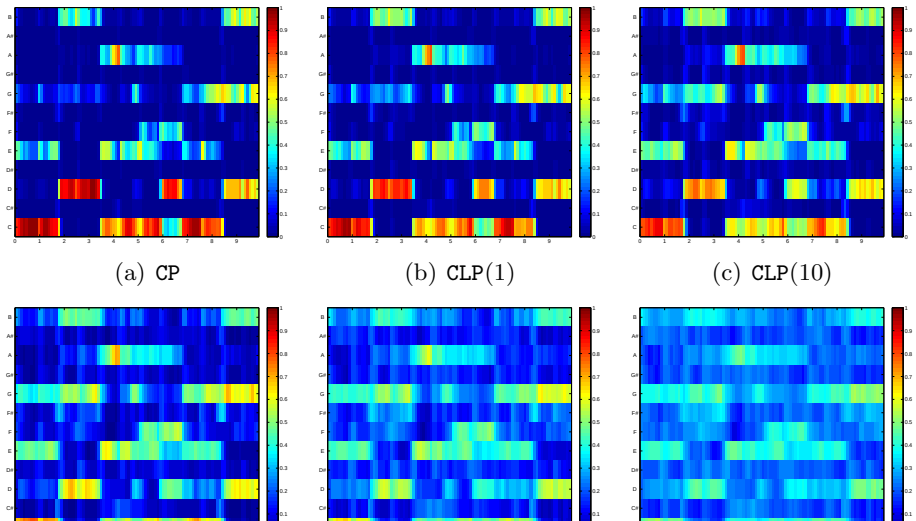
Figure 5.3. Visualization of Table 5.5. Effect of logarithmic compression.

Overall, enlarging the logarithmic compression until $\eta = 1000$ improves performance over the original pitch features by 18% using most of the chord recognizers except for using $\mathtt{T}^{\mathtt{b}}$ which get only 9% increase. From the chord recognizer point of view, the trend of having logarithmic compression behaves very similar among $\mathtt{T}^{\mathtt{a}}$, `GP` and `HMM`. The effect of increasing recognition accuracies is less obvious when using $\mathtt{T}^{\mathtt{b}}$. Note that for the recognizer `HMM`, the smoothing effect is already internally incorporated. This is why the performance of all features with `HMM` is better than the ones with other recognizers until $\eta = 1000$.

However, it seems that just due to this smoothing effect, when $\eta = 10000$ the HMM has a sharper decrease than other recognizers.

The best compression parameters for template-based methods ($\eta = 100$) and for statistical-model-based methods ($\eta = 1000$) are not the same. However, the result difference of using $\eta = 100$ or $\eta = 1000$ is quite small, which is less than 1%. Enlarging the logarithmic compression to 10000, the recognition accuracies of all the chord recognizers decrease. This indicates the optimal compression parameter lies in the range $[100, 1000]$.

In order to further inspect the effect of logarithmic compression on features, the chromagrams of features with or without different logarithmic parameters are shown in Figure 5.4. From 0 to 1.8 seconds, the chroma G gets much more obvious as the enlarging the compression parameter, while at the same time the chroma C gets less obvious yet still clearly visible. This assists the chord recognizers to determine the correct chord C since the weights of both the components notes are the same in $\mathtt{T}^{\mathtt{b}}$. However, for some cases, logarithmic compression also brings a negative effect. For example, at around 3.4 to 3.5 second, the chord changes from G to Am. In the chromagrams of feature CP and CLP(1), the intensity of chroma G is very weak, and the chroma A, E and C are strong in contrast to G. The chord edge is sharp and clear so that the following recognizers will make correct decisions about the time point when chord changes. But in the chromagrams of feature CLP(1000), the chroma G and chroma A nearly have the same intensity. With the presence chroma of C and E, it is hard for the following chord recognizers to decide whether the correct chord should be C which consists of C,E and G, or should be Am which consists of A, C and E. In other words, the edge of chord changes are blurred in this way. This is also the reason that CLP(1000) is wrong at this time position but CP avoids this error in Figure 5.2.



(a) CP (b) CLP(1) (c) CLP(10)

(d) CLP(100) (e) CLP(1000) (f) CLP(10000)

Figure 5.4. Comparisons of different logarithmic compression on chroma features.

5.3.4 Effect of Smoothing

In this section we evaluate the impact of the smoothing at the feature side on the chord recognition performance. Among all the feature types, we only integrated smoothing parameter in CENS and CRP. As we described in Section 2.4 and 2.5, we convolve each single feature vector x with the neighbor feature vectors which are within a Hann window of fixed length w centered on x. Here w specifies the number of the neighbor feature vectors and therefore control the size of smoothing. For the chord recognition methods which require training, the models are retrained using features with the specified smoothing window length w as in the evaluation.

The evaluation results of smoothing using CENS features are shown in Figure 5.5. Recall that for CENS features, CENS(w, d) means we use the smoothing window size w and down-sampling factor d. Here we have no down-sampling at all therefore set $d = 1$ in all the experiments. On the contrary, w is enlarging so that the extent of smoothing is rising. Note that by setting $w = 1$, CENS(1, 1) simply means no smoothing.

In general, the best smoothing result of CENS(w, 1) feature improves performance over the non smoothing CENS(1, 1) by 22%, 16%, 15% and 3% for chord recognizer $\mathtt{T}^{\mathtt{b}}$, $\mathtt{T}^{\mathtt{a}}$, GP and HMM respectively. The best results come from the $w = 25$ for all recognizers except for HMM where $w = 15$. Furthermore, from Figure 5.5 we can see that the overall trend of all recognizers except HMM are quite similar as well. As w is getting larger, the recognition accuracies of all chord recognizers get higher. In particular, increasing w from 1 to 3 gets the most rapid increase in accuracies. For HMM, the reason that it behaves differently from other recognizers is that HMM already integrated post-smoothing due to its internal viterbi decoding algorithm which emphasizes the situation of a chord staying in itself. Therefore, in this experiment it actually contains two stages of smoothing: both on the feature side and on the recognizer side. This makes HMM distinct from other recognizers.

Similar to CENS features, we test the smoothing effect on CRP features as well. Figure 5.6 illustrates the results. The smoothing also improves the recognition accuracies as we increase the window length. However, the extent of increase is not as large as on CENS features. CRP reaches the optimum with $w = 21$ which is smaller than the $w = 25$ for CENS and the increase is 10%, 10%, 11% for the first three recognizers. To our surprise, there is nearly no improvement of using HMM, where the difference between the optimal CRP(13, 1) and CRP(1, 1) is only 0.2%.

Figure 5.7 shows the chromagrams of CENS(w, 1) with different settings of w and the corresponding chord recognition visualization for the first 10 seconds of BeatlesLetItBe. By comparing the recognition accuracy, we find that smoothing works as double-edged sword: on one hand, it smoothes out the error and thus increases the accuracy, on the other hand, it also smoothes out the correct answer and decreases the accuracy. Figure 5.7 reflects this phenomenon obviously: from 4.8 to 5.0 seconds the error occurs for CENS(1, 1) but is smoothed out for CENS(9, 1). This is because that at the time point of 4.8 second in CENS(1, 1), the chroma A has nearly zero intensity in CENS(1, 1), and the chroma G has low intensity yet it is larger then zero. In CENS(9, 1) with the smoothing effect, at that time point since A gets larger intensity while G gets smaller since its neighbor features contain no intensity of G. However, from 5.1 to 5.6 seconds, some correct recognized F

CR	$\mathtt{T}^{\mathtt{b}}$	$\mathtt{T}^{\mathtt{a}}$	GP	HMM
CENS(1, 1)	0.345	0.426	0.442	0.580
CENS(3, 1)	0.428	0.472	0.479	0.591
CENS(5, 1)	0.465	0.499	0.503	0.597
CENS(7, 1)	0.491	0.520	0.522	0.604
CENS(9, 1)	0.511	0.537	0.540	0.610
CENS(11, 1)	0.525	0.551	0.555	0.613
CENS(13, 1)	0.536	0.563	0.566	0.615
CENS(15, 1)	0.546	0.570	0.575	0.616
CENS(17, 1)	0.553	0.576	0.581	0.615
CENS(19, 1)	0.558	0.580	0.585	0.614
CENS(21, 1)	0.562	0.584	0.588	0.613
CENS(23, 1)	0.565	0.586	0.591	0.610
CENS(25, 1)	0.567	0.587	0.592	0.608

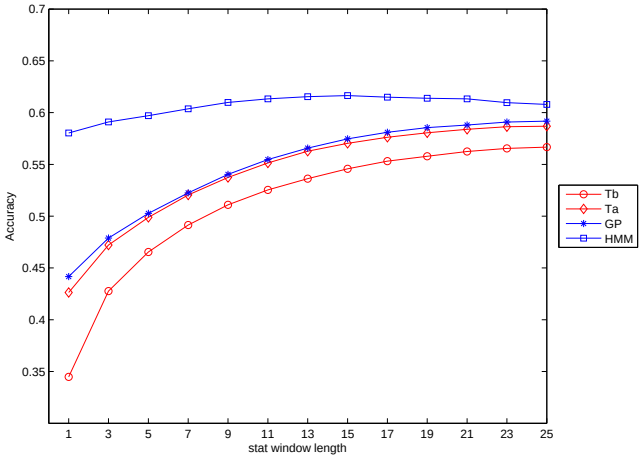


Figure 5.5. Recognition accuracies of using different smoothing window length in CENS features. Train: $\mathcal{D}_1^{\text{Beatles}}$. Test: $\mathcal{D}_3^{\text{Beatles}}$.

CR	$\mathtt{T}^{\mathtt{b}}$	$\mathtt{T}^{\mathtt{a}}$	GP	HMM
CRP(1, 1)	0.523	0.574	0.580	0.713
CRP(3, 1)	0.551	0.604	0.611	0.714
CRP(5, 1)	0.571	0.625	0.632	0.715
CRP(7, 1)	0.586	0.641	0.648	0.715
CRP(9, 1)	0.598	0.652	0.660	0.714
CRP(11, 1)	0.607	0.661	0.671	0.714
CRP(13, 1)	0.613	0.667	0.677	0.715
CRP(15, 1)	0.617	0.671	0.683	0.715
CRP(17, 1)	0.622	0.672	0.686	0.713
CRP(19, 1)	0.624	0.674	0.689	0.712
CRP(21, 1)	0.625	0.674	0.691	0.711
CRP(23, 1)	0.625	0.673	0.691	0.708
CRP(25, 1)	0.624	0.672	0.691	0.703

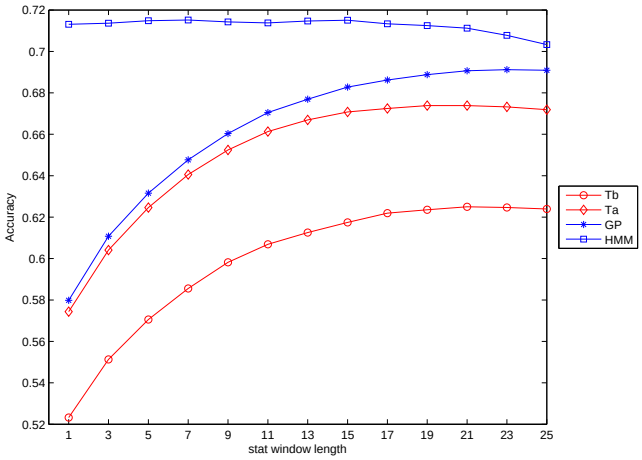
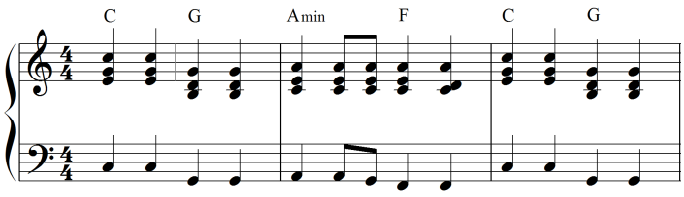


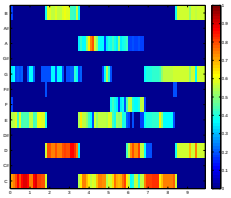
Figure 5.6. Recognition accuracies of using different smoothing window length in CRP features. Train: $\mathcal{D}_1^{\text{Beatles}}$. Test: $\mathcal{D}_3^{\text{Beatles}}$.

was smoothed out to the wrong chord Am. This is because in CENS(1, 1), at that short time, chroma F is originally stronger than A which yields the correct chord F. But after smoothing with the neighbors, chroma F gets weaker than A, because in the neighbor features the intensity of F is less than A.

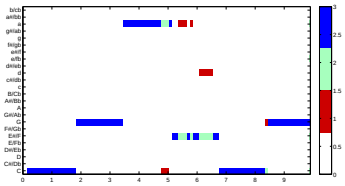
When w is increased to 21, which means each feature is blend with 2.1 seconds of the original audio file, we find that the error which occurs in all other CENS features with smaller w from 6 to 7 second totally disappears in CENS(21, 1). But such large w also blurs the edge of the chord transition. We find that nearly all the start and end of the chord transition is wrong in CENS(21, 1). However when w is less than 9, the edge can be perfectly recognized.



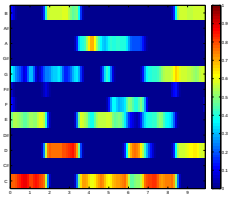
(a) Score of the first 10 seconds (corresponding to the first three measures) of `BeatlesLetItBe`.



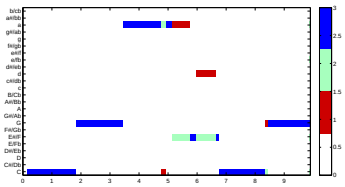
(b) `CENS`$(1, 1)$



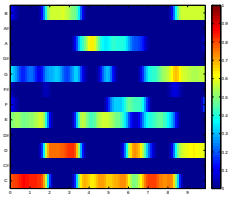
(c) Recognition accuracy 0.850



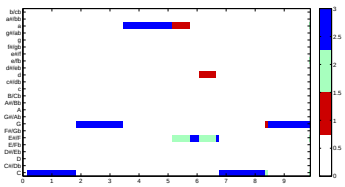
(d) `CENS`$(5, 1)$



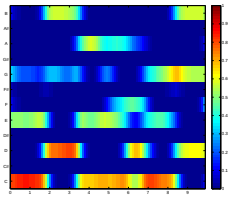
(e) Recognition accuracy 0.820



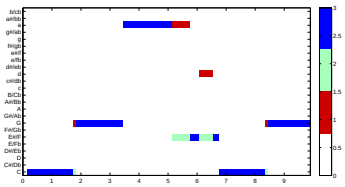
(f) `CENS`$(9, 1)$



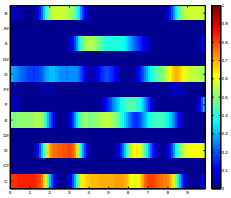
(g) Recognition accuracy 0.850



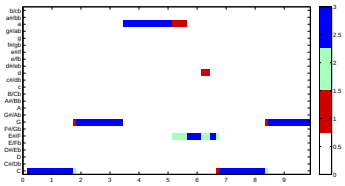
(h) `CENS`$(13, 1)$



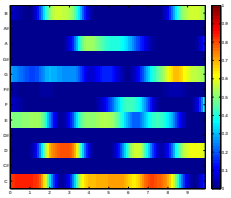
(i) Recognition accuracy 0.850



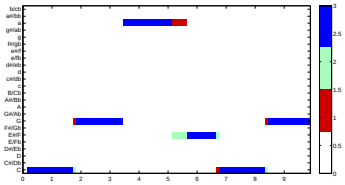
(j) `CENS`$(17, 1)$



(k) Recognition accuracy 0.870



(l) `CENS`$(21, 1)$



(m) Recognition accuracy 0.900

Figure 5.7. Comparisons of different window lengths of smoothing on `CENS` features. Test on recognizer $\mathtt{T}^{\flat}$ and test data : the first 10 seconds of `BeatlesLetItBe`.

The next question is what the optimal value for w is. Although CENS(21, 1) yields the highest recognition accuracy of 0.90 among all the settings, does it really make sense to have such a large smoothing window which blend in the original signal more than two seconds? The answer to that question is, that there is no universal optimal value since the durations of the chord differ for each song. For some of the music pieces which contain Arpeggio such as *Bach BWV846*, it would be good to have a large window than the common pieces of music since the notes of the chords are separately played. While for some of the pop music in which the duration of chords are short and the chords rapidly change, it would be good to have a small window. However, in general, we consider that the window size corresponding to 0.5 second does make sense, since most of the chords last at least 0.5 second.

5.3.5 Effect of Pitch Range Separation

In this section we evaluate the effect of pitch range separation by dividing the whole pitch range $p \in [1 : 120]$ to some smaller range mainly representing melody and bass. We use two types of features in our evaluation. One is CISP, the other is CRP.

Pitch Range Separation for CLP :

In CISP features, the melody part is restricted to $p \in [43 : 92]$ covering musical notes from $G2$ to $G6$, and the bass part is in $p \in [21 : 68]$ covering notes from $A0$ to $A4$. Note there is an overlap range in the two parts. Furthermore, in order to test the effect of using more dimensions other than 12 to represent the chromas, here we combine both the 12 dimensional melody range features and the 12 dimensional bass range features vertically to form 24 dimensional features.

Table 5.6 shows the evaluation results of using CISP features. The first column which is named "12dim all" is the normal CISP we used in other experiments. From the table we see that the general performance of using melody range features is very close to the performance of using whole range features. The biggest recognition difference between the two types of features is only 1.5% at recognizer $\mathtt{T}^{\mathtt{b}}$. However, using bass range features dramatically decreases the performance. The recognition decreases from 7% to 15% compared to the features using the whole range. The biggest difference comes from $\mathtt{T}^{\mathtt{b}}$ as well. This implies that the melody pitch range is more or less able to cover the pitch of most played notes in the dataset, while the bass pitch range captures only several played notes and misses most notes who have higher pitches.

For the comparison of dimensions of features, we find that the results of using 24 dimensions and using 12 dimensions are very close to each other as well. For $\mathtt{T}^{\mathtt{b}}$ and GP, the 24 dimensional features increase the accuracy about 1%. For $\mathtt{T}^{\mathtt{a}}$ and HMM, the accuracy is decreased, but only less than 1%.

Pitch Range Separation for CLP :

Table 5.7 shows the pitch range separation results using CLP features. First we focus on the 12 dimensional features which are in the first three columns. Different from using CISP features, the recognition accuracy of the melody range features is 5% to 7% less than the normal whole range features for all the CLP with $\mathtt{T}^{\mathtt{b}}$. For other recognitions, the difference

CR	12dim all	12dim melody	12dim bass	24dim all
$\mathtt{T}^{\mathtt{b}}$	0.429	0.414	0.279	0.441
$\mathtt{T}^{\mathtt{a}}$	0.481	0.472	0.406	0.475
GP	0.504	0.495	0.399	0.514
HMM	**0.717**	0.703	0.642	0.715

Table 5.6. Recognition differences of pitch range separation of CISP. Train: $\mathcal{D}_1^{\text{Beatles}}$. Test: $\mathcal{D}^{\text{Beatles}}$.

between them is also larger than those tested on CISP features. A common phenomenon occurs in both CLP and sCISP is the bad performance of using bass range features. Again we get about 7% to 20% decrease.

For the dimension comparison of CLP, the result varies with different compression parameters. For CLP(1), the "24dim all" features improves recognition at most 5% and at least 2%. However, this improvement becomes less and less as the log compression parameter η gets larger. Until $\eta = 100$ it still improves the result. But to our surprise, when $\eta = 100$, it decreases the result of GP by 30% and HMM by 35%, while it stays nearly the same for $\mathtt{T}^{\mathtt{b}}$ and $\mathtt{T}^{\mathtt{a}}$. This is because for large compression parameters, the intensity difference between chromas is already quite small and hard to distinguish, now putting such undistinguished chroma features into a complex model, the model pattern we learned will be even more undistinguished among each of the chords. That is why for the template method the performance is still ok but for complex methods it totally fails.

CR	12dim all	12dim melody	12dim bass	24dim all
$\mathtt{T}^{\mathtt{b}}$	0.508	0.455	0.397	0.551
$\mathtt{T}^{\mathtt{a}}$	0.513	0.475	0.353	0.545
GP	0.541	0.490	0.508	0.582
HMM	0.665	0.623	0.599	0.686

CR	12dim all	12dim melody	12dim bass	24dim all
$\mathtt{T}^{\mathtt{b}}$	0.544	0.487	0.396	0.572
$\mathtt{T}^{\mathtt{a}}$	0.577	0.535	0.382	0.585
GP	0.601	0.552	0.535	0.632
HMM	0.710	0.675	0.631	0.727

CR	12dim all	12dim melody	12dim bass	24dim all
$\mathtt{T}^{\mathtt{b}}$	0.553	0.492	0.376	0.569
$\mathtt{T}^{\mathtt{a}}$	0.607	0.560	0.391	0.599
GP	0.622	0.575	0.519	0.642
HMM	0.694	0.672	0.598	0.722

CR	12dim all	12dim melody	12dim bass	24dim all
$\mathtt{T}^{\mathtt{b}}$	0.541	0.474	0.345	0.553
$\mathtt{T}^{\mathtt{a}}$	0.601	0.547	0.362	0.590
GP	0.611	0.545	0.478	0.329
HMM	0.714	0.655	0.572	0.347

Table 5.7. Recognition differences of pitch range separation of CLP. Upper Left: CLP(1), Upper Right: CLP(10). Lower Left: CLP(100). Lower Right: CLP(1000). Train: $\mathcal{D}_1^{\text{Beatles}}$. Test: $\mathcal{D}^{\text{Beatles}}$.

5.4 Experiments on Chord Recognition Methods

As we described previously, after converting a audio file into audio features, we pass the sequence of features into the chord recognition module, and the chord recognition method will classify each of the features to one of the chord classes via pattern matching between the feature and predefined chord patterns. We have introduced three different template based methods in Chapter 3 and three statistical model based methods in Chapter 4. In this section, we conduct several experiments to evaluate the performance of all these methods.

The structure of this section is listed as follows. In Section 5.4.1 we illustrate the overall performance of all features and recognizers. In order to test out the real effect of the recognizers, we adapt the optimal parameter setting to the normal setting on the feature side so that the increased recognition accuracy caused by feature is removed. In Section 5.4.2 we compare all the recognizers and analyze the result and trend of the errors. Since the behavior of the Gaussian probability based method and the HMM based method depends much on the covariance matrix, we analyze the effect of the covariance matrix by changing the diagonal value and off-diagonal value and test their ability of modeling a chord pattern by observing the difference in recognition accuracies. We describe this in Section 5.4.3. Among all the recognition methods, HMM is the only one which integrated post-smoothing after classification. The effect of such post-smoothing is controlled by the transition matrix. Therefore we conduct several experiments which mainly focus on the self-transition to test how the smoothing works and what effect it has on the final recognition result.

5.4.1 Overall Performance

CR	CP	CLP(1000)	CISP	CENS$(1,1)$	CRP$(1,1)$
$\mathtt{T}^{\mathtt{b}}$	0.460	0.541	0.429	0.458	0.528
$\mathtt{T}^{\mathtt{h}}$	0.342	0.590	0.467	0.374	0.563
$\mathtt{T}^{\mathtt{a}}$	0.421	0.601	0.481	0.423	0.583
Maha	0.331	0.388	0.425	0.374	0.577
GP	0.429	0.611	0.504	0.441	0.588
HMM	0.528	0.714	0.717	0.587	**0.723**

Table 5.8. Recognition differences between various recognizers, using features without smoothing. Train: $\mathcal{D}_1^{\text{Beatles}}$. Test: $\mathcal{D}^{\text{Beatles}}$.

In this section we discuss the overall performance of our system focusing on the recognizer side. Table 5.8 shows the recognition accuracies with all six recognizers and five feature types. Figure 5.8 is the corresponding visualization. The parameters of the CENS and CRP are set to the initial value without any smoothing. In this way, we discard the optimization techniques on the feature side so that the real capability of the recognizers can be tested out.

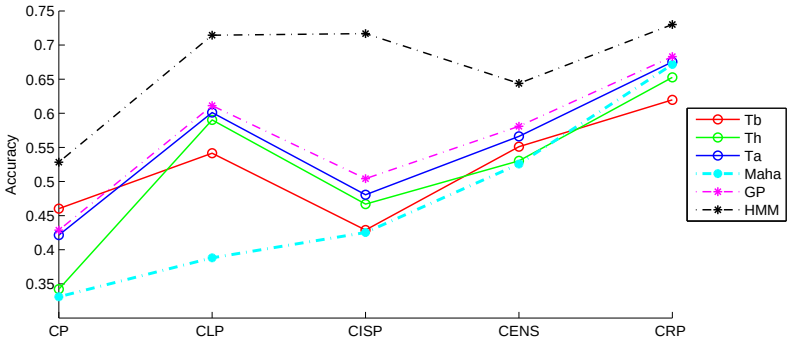


Figure 5.8. Visualization of Table 5.8. Overall performance of the recognizers.

From Figure 5.8 we can see that the overall trends of $\mathtt{T}^{\mathtt{b}}$,$\mathtt{T}^{\mathtt{h}}$,$\mathtt{T}^{\mathtt{a}}$ and GP are very similar, while Maha and HMM behave differently. In comparison to the three template-based recognizers, we previously expected that $\mathtt{T}^{\mathtt{h}}$ should be better than $\mathtt{T}^{\mathtt{b}}$ since it considers the contribution of harmonics, which make its template more close to the real pattern of features than the templates of $\mathtt{T}^{\mathtt{b}}$. However, this expectation is not fulfilled by CP and CENS. This is because for these two features the computed intensity is in the form of energy, which is the squared magnitude; while for the other features they are in the form of magnitude. This implies that template based methods are not suitable for features with a squared magnitude which make the difference of magnitudes larger than their original differences. $\mathtt{T}^{\mathtt{a}}$ is in a similar situation. It learns the pattern of features from training data and therefore should have a better fit with features compared to $\mathtt{T}^{\mathtt{b}}$, but CP and CENS violate the expectation as well. For other features, using $\mathtt{T}^{\mathtt{h}}$ generally increase the accuracy by 4% or 5% compared to $\mathtt{T}^{\mathtt{b}}$, and $\mathtt{T}^{\mathtt{a}}$, which is the best among all template based methods, can have a further increase of 1% or 2% compared to $\mathtt{T}^{\mathtt{h}}$.

The behavior of three statistical model based methods are totally different. So far we cannot interpret the performance of Maha. For GP, previously we supposed it should be much better than $\mathtt{T}^{\mathtt{a}}$, because GP considers both the average value of the feature vectors and the correlation between different chromas of the features, while $\mathtt{T}^{\mathtt{a}}$ takes only the average value. In the table we can see that for all the features there are increases using GP compared to using $\mathtt{T}^{\mathtt{a}}$. However such increases are not that large. The largest increase happens at feature CISP, which gets 0.481 at $\mathtt{T}^{\mathtt{a}}$ and 0.504 at GP and yields the increase of 2.3%. Finally, HMM performs best among all recognizers due to its internal smoothing around the context, and the general increase is 10% to 20%. This again verifies the importance of smoothing. Recall that in Table 5.3, incorporated the smoothing in CENS and CRP from the feature side as well as the smoothing in HMM from recognizer side, we got recognition accuracy 0.644 and 0.730 for the two features respectively. And without smoothing on the feature side, in Table 5.8 we got corresponding 0.587 and 0.723, which are 5.7 and 0.7 less than the previous result. This illustrates the difference between performing smoothing on both feature and recognizer sides and performing only on the feature side. We confirm that each of them can bring huge improvement to a chord recognition system, but when they are combined together, the contribution will largely depend on the quality of the feature. As in our result from both tables, CENS benefits a lot from the combined smoothing but CRP does not.

5.4.2 Error Analysis

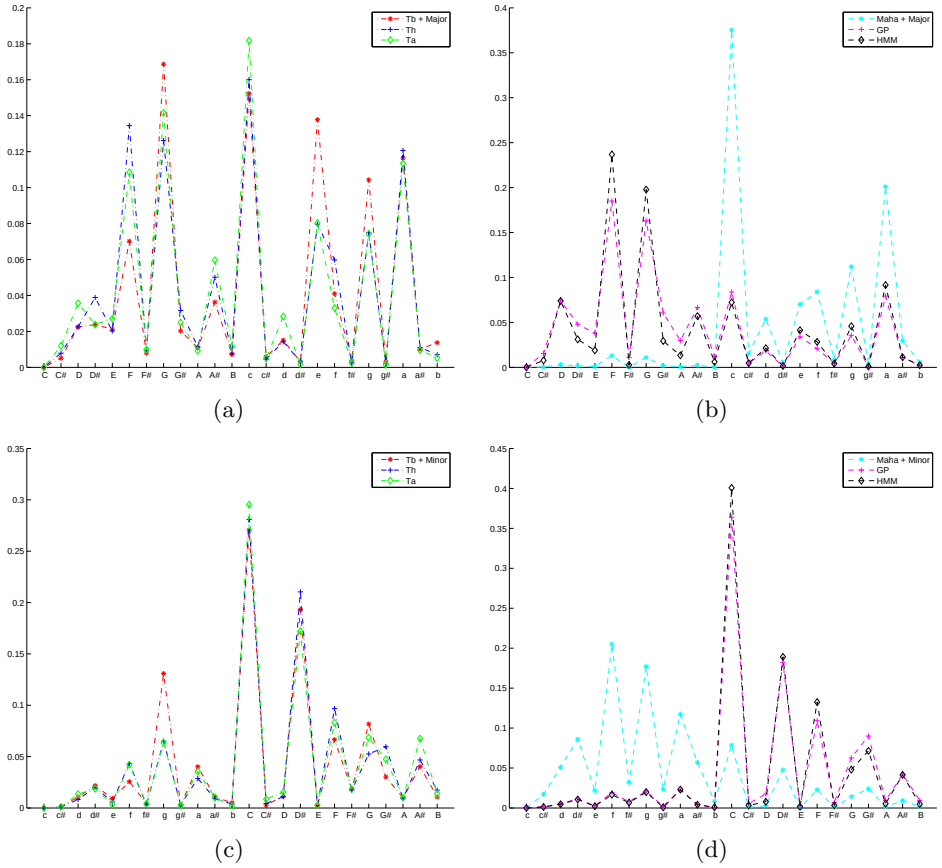


Figure 5.9. Error analysis of chord recognizers. Upper: major chord. Lower: minor chords. Left: on template based methods. Right: on statistical model based methods.

After the analysis of the overall performance of recognizers, in this section we present in which situations the recognizers fail to recognize the chords, that is to say, the analysis of errors. Figure 5.9 illustrates the distribution of errors with the x-axis indicating the chord label, and the y-axis indicating the normalized error ratio. The figure is generated by collecting the wrongly recognized chords of each recognition methods with feature `CLP`(1000) using the same data as in Table 5.8. For convenience of visualizing the errors, all chords with ground truth major are circle shifted to `C` and all chords with ground truth minor are shifted to `Cm`.

In Figure 5.18(a), the ground truth is `C`, and we visualize the error distribution of three template based chord recognizers. We see that there are high error distributions at `Cm`, `Em` and `Am`. This is due to the musical reasons that all these chords share two components with `C`. What is interesting is the errors at `F` and `G` totally reflect the characteristic of the templates. At `G`, $\mathtt{T}^{\mathtt{b}}$ makes more errors than $\mathtt{T}^{\mathtt{h}}$. This is because the feature of `C` generally contains high intensity at chroma C, E, G, D and B, and the last three notes form `G` which make $\mathtt{T}^{\mathtt{b}}$ confuse between the `C` and `G`. However, $\mathtt{T}^{\mathtt{h}}$ formulates the `G` considering its other two harmonics which do not coincide with the notes or harmonics in `C`. Therefore $\mathtt{T}^{\mathtt{h}}$ successfully beats $\mathtt{T}^{\mathtt{b}}$ at the possible chord confusion between `C` and `G`. However on

the contrary, at F, just because $\mathtt{T}^{\mathtt{h}}$ considers the harmonics which are coincide with the components of C, it was beaten by $\mathtt{T}^{\mathtt{b}}$.

In Figure 5.18(b), the ground truth is still C but we visualize the error of three statistical model based recognizers. The general error distributions of GP and HMM are very similar. At F and G, HMM makes more errors than all other recognizers. This is again due to its internal smoothing effect. Because F and G are subdominant and dominant chords of C, and they are very probably to be the previous or next neighbor of C in the progression of chords. With temporal smoothing, HMM may make errors at the edge between two chords which mistakes the current C to G.

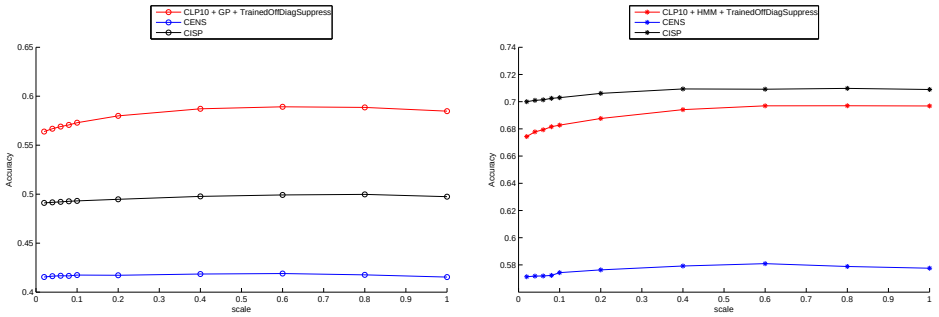
From Figure 5.18(b) and 5.18(f) we see that, GP and HMM tend to classify the features to major chords no matter the ground truth is major or minor. On the contrary, Maha tends to classify features to minor chords. However, we did not find such preferences of templates in Figure 5.18(a) and 5.18(c). Therefore the reason lies in the covariance matrix of the statistical models, which controls the variance of each chroma and also the correlation between different chromas.

Actually, the data ratio of major chords and minor chords is 80% and 20%. Thus the performance of our system is related to the errors coming from major chords. This is why although GP and HMM make more errors in Figure 5.18(f) which is much higher than the error rate of major chords in Figure 5.18(b), they still perform quite well in the overall performance.

5.4.3 Effect of Covariance Matrix

Since the behavior of the Gaussian Probability method and the HMM method depends much on the covariance matrix, we analyze the effect of covariance matrix by changing the diagonal value and off-diagonal value and test their ability of modeling a chord pattern by observing the difference in recognition accuracies.

5.4.3.1 Trained Covariance Matrix



(a) (b)

Figure 5.10. Comparisons of having different suppression parameter on *off-diagonal values* in trained covariance matrix.

Off-diagonal values. First we explore the contributions of the off-diagonal values. To

this end, we first train the covariance matrix using $\mathcal{D}_1^{\text{Beatles}}$ and then suppress the off-diagonal values by multiplying a scale factor s while at the same time keep the diagonal values unchanged. Figure 5.10(a) illustrates the result of using recognizer `GP`, with s varying at the x-axis, and the y-axis showing the corresponding recognition accuracies. The three lines represent three different feature types. By observing the figure, we can hardly find any significant decrease or increase of the recognition accuracy. In fact, the difference between the highest and lowest accuracy is very small, for `CLP`(10) being 2.6%, for `CENS`$(1, 1)$ being 0.8% and for `CISP` being 0.3%. Besides, the highest accuracy comes from $s = 0.6$, and the lowest comes from $s = 0.02$ for all three feature types. Similarly, Figure 5.10(b) illustrates the result using the `HMM` recognizer. And the biggest difference lies in `CENS`$(1, 1)$ which is 2.9%. We can see that `HMM` is not much affected by the suppression of the off-diagonal value as well.

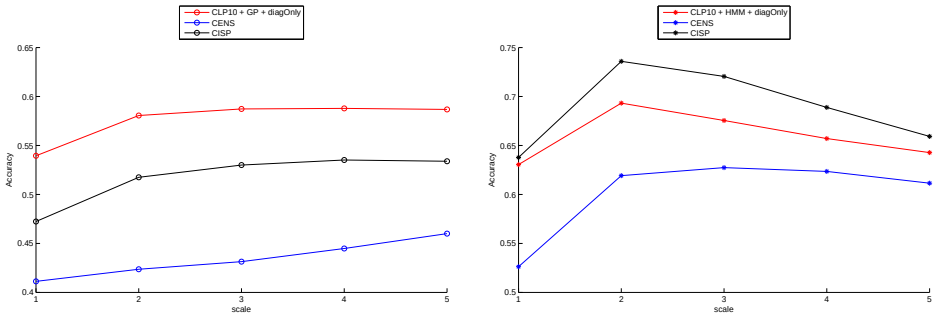
From the observation of the two figures we conclude that using a full covariance matrix is indeed slightly better than using a diagonal covariance matrix, however such improvement is very small and for some of the features it can be negligible. Therefore, one can generally focus on the diagonal value when analyzing the performance of a chord recognition system, which controls the variance of each chroma, other than the off-diagonal value, which controls the correlation between two chromas.

Diagonal values. After the discussion about off-diagonal values, we examine the contributions of diagonal values in Figure 5.11. This time the impact of changing the scale parameter is huge. In Figure 5.11(a) the difference between the highest and lowest recognition accuracies are 4.8%, 6.3% and 5.0% for `CLP`(10), `CISP` and `CENS` respectively. Besides, the overall trend of `CENS` is different from the other two features. `CENS` keeps ascending as we increase the scale while the other two features reach the highest at $s = 3$ and is nearly unchanged for $s = 4$ or $s = 5$.

The behavior of `HMM` is quite different from `GP` when testing the diagonal values. Figure 5.11(b) shows that the original covariance matrix for $s = 1$ performs the worst among all results. Then there is a huge improvement, which is nearly 10% of increase, when setting $s = 2$ for all features. Note that `CLP`(10) and `CISP` even get the highest recognition accuracy at $s = 2$, then the accuracies get around 2% to 3% of decrease as we increase s every time by one.

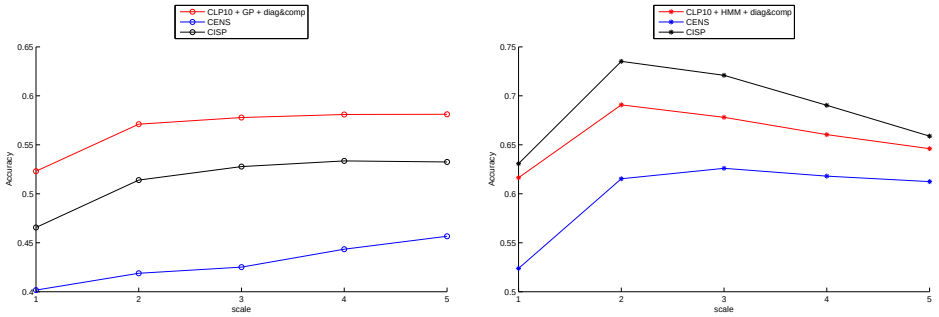
The above discussion indicates that the diagonal values of the covariance matrix are of great importance to the performance of `GP` and `HMM`. Enlarging the scale factor actually means we enlarge the variance of each chroma. Recall the Gaussian density function we derived that such an enlarging effect is exponential with s, thus the original a large variance will become even larger, in other words, less strict with features at outliers. Therefore, the chromas with large variance gets more tolerable for the intensity change than those with a small variance. And now the experiments results show that by enlarging the variance the performance will be improved.

Covariance of component notes. Suppressing all the off-diagonal values as a whole might be too strict in the previous experiment. Since the covariance of the component notes might have impact on the results of experiment, here instead of suppressing all off-diagonal values, we first keep the covariance of the component notes and diagonal values, then change them by multiplying the scale factor. Such covariance entries we



(a) (b)

Figure 5.11. Comparisons of having different scale on *diagonal values* in trained covariance matrix.



(a) (b)

Figure 5.12. Comparisons of having different scale on *all the diagonal values and off-diagonal at component notes* in trained covariance matrix.

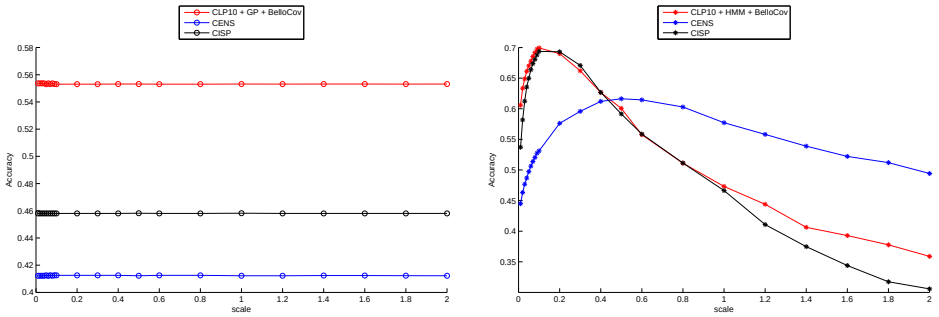
keep include the covariance of C, E,C, G and E, G for chord `C`. In Figure 5.14, we present the result. In comparison to Figure 5.11, we find that the result involving with such covariance of component notes nearly has no difference with the uninvolved result. The largest recognition difference among them is less than 1%.

Conclusion. From all the above discussions we conclude that, for a trained covariance matrix, one can generally focus on the diagonal values which represents the variance of each chroma, and pay attention to the scale of these values; and the off-diagonal values have only little influence on the final performance.

5.4.3.2 Manually Set Covariance Matrix

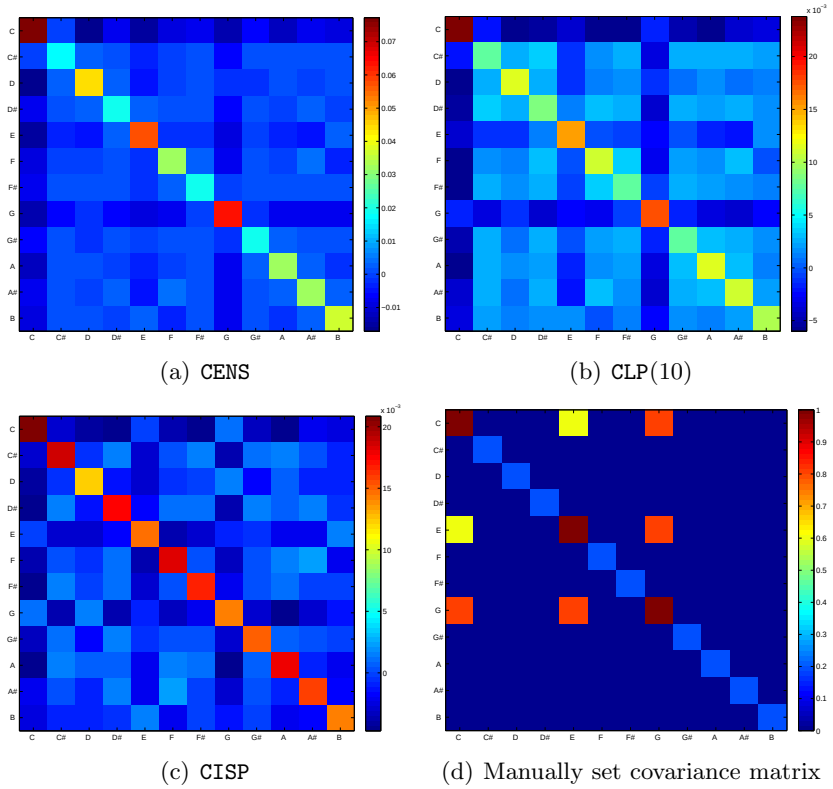
In the previous subsection, we trained the covariance matrix and inspected the impact of diagonal values and off-diagonal values. Here in this subsection, we take the manually set covariance matrix which inspired by musical knowledge. We adapt the setting according to [3], where Bello and Pickens set the values as visualized in Figure 5.14(d). Then we replace the trained covariance matrix with the manually set matrix and circle shift them from `C` or `Cm` to the corresponding chord.

Similar to the previous experiments, we test on the performance by varying the scaling fac-



(a) (b)

Figure 5.13. Comparisons of having different scale on *all the diagonal values and off-diagonal at component notes* in *manually set* covariance matrix.



(a) `CENS` (b) `CLP`(10)

(c) `CISP` (d) Manually set covariance matrix

Figure 5.14. Trained covariance matrix of corresponding feature types and manually set covariance matrix for `C`.

tor on diagonal variances and off-diagonal covariance of component notes. Figure 5.13(a) shows the results for `GP` and Figure 5.13(b) shows the results for `HMM`. We find that `GP` is invariant with the scaling factor, but `HMM` has significant changes with the scaling factor.

Actually the chord labels computed by `GP` with different scaling factor are not exactly the same. After replacing the covariance matrix, although all the computed Gaussian probabilities change, there are only small changes in the chords whose pattern yields maximum probability. The number of such chord differences are so small that using evaluation with precision 0.001 is not enough to find the differences.

However, in HMM we are not selecting the maximum, and the value of Gaussian probabilities severely affects the final performance. This is because the Viterbi algorithm takes both the covariance matrix and transition matrix and then computes the forward probability in an accumulated multiplication manner. In this way, any tiny difference even as small as 0.001 will be exaggerated largely after several times of multiplication. Therefore, if one wants to use the manually set covariance matrix in HMM, one needs to be very carefully to scale the values to fit the intensity of the features. Figure 5.14(a), 5.14(b) and 5.14(c) show the example of covariance feature setting of three features. We can see that the precision of the color bars of CISP and CLP(10) is at 10^{-3} but CENS is at 10^{-2}. This is the reason why the trends of CISP and CLP(10) are close to each other but different with CENS in Figure 5.13(b). Furthermore, by this experiment, we see that using the manually set covariance matrix can also yield a good recognition result as long as the scale of the values are appropriately set.

5.4.4 Effect of Transition Matrix

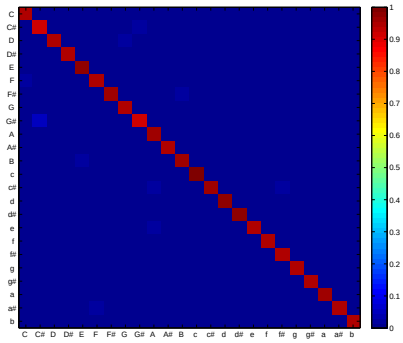
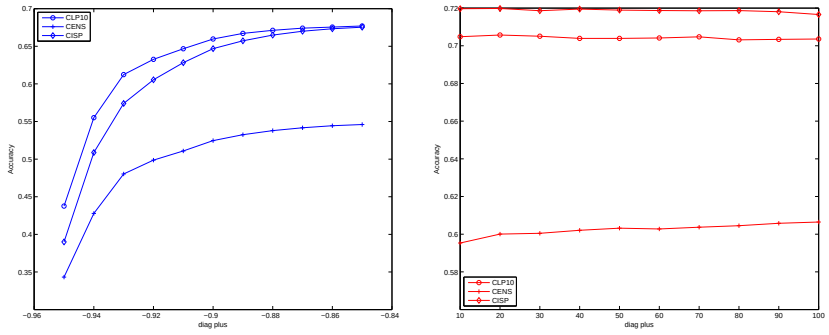


Figure 5.15. Transition Matrix.



(a) (b)

Figure 5.16. Comparisons of different suppression parameters on diagonal values of a transition matrix.

In this subsection, we inspect the diagonal entries of transition matrix. Figure 5.15 shows the transition matrix trained on $\mathcal{D}_2^{\text{Beatles}}$. All the diagonal values are above 0.95. They are much larger than other off-diagonal values, and most of them are below 0.001. A diagonal entry represents the self transition of a chord, and an off-diagonal entry represents the

transition from one chord to another. This is because that we parse the original chord annotations into small frames and we train the transition matrix by absolute counting of the bi-grams. Therefore the transition between the chords will be count once while the self transition which corresponding to long lasting chord will be count many times.

Figure 5.16 shows the effect of changing diagonal entries of the transition matrix. Here we keep all the off-diagonal values unchanged and add an additional value to all the diagonal values. See Equation (5.4). The x-axis of Figure 5.16 indicates the value of a.

$$A'(i,j) = \begin{cases} A(i,j) & \text{if } i \neq j \\ A(i,j) + a & \text{if } i = j \end{cases} \tag{5.4}$$

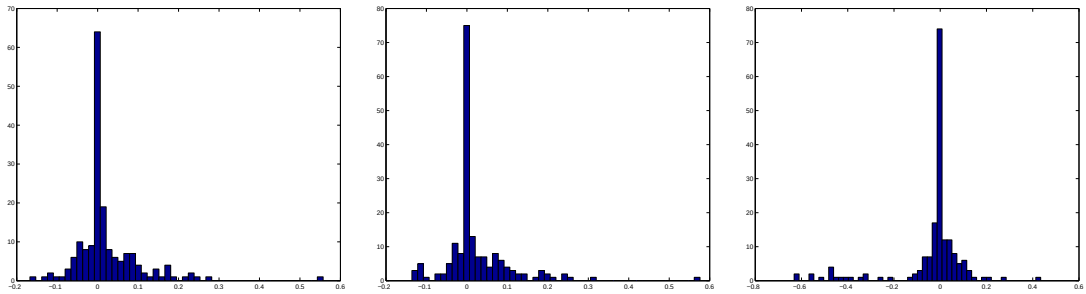
As we can see from Figure 5.16(b), the value of diagonal entries have a large impact on the recognition accuracies. This impact are the same for all the three features we test in this experiment. Since any of the original diagonal entries is above 0.90, after add $a = -0.85$, the diagonal values becomes at least 0.05. Such value is still about 50 times larger than most of the off-diagonal entries. With recognition accuracy 0.67, we consider this does not affect much of the system performance. However when we enlarge the absolute value of a, in particular when a is beyond -0.90, the recognition accuracy decrease rapidly. In Figure 5.16(b), we set a much larger than the original off-diagonal values, the recognition accuracy is higher than in Figure 5.16(a) but changing a has nearly no influence. Both figure together indicate that as long as we emphasize the self transition in a proper range, the performance of the system remains the same. However once we are out of this rage, the system is totally violated.

From now on, we discuss some special aspects of our experiments besides the parameter of feature or chord recognition methods described in the previous sections. We first present the effect of the tuning in section 5.5. We will see the difference of recognition accuracies with and without the tuning of the original audio file. In section 5.6, we conduct our experiments using different training data to observe the influence of the training data on the performance of our system. In section 5.7, we add an optimization technique which called Harmonic Percussive Source Separation to our system. And we discuss how it affects the performance of chord recognition by analyzing the increased and decreased recognition accuracies of having this technique. In section 5.8 we discuss the behavior of using the beat-synchronized features instead of using the frame-wise features. Finally in Section 5.9, we briefly examine the chord recognition on classical dataset.

5.5 Effect of Tuning

CR	tuned	CP	CLP(1000)	CENS(1, 1)	CRP(1, 1)
$\mathtt{T}^{\mathtt{b}}$	−	0.445	0.520	0.445	0.505
$\mathtt{T}^{\mathtt{b}}$	+	0.460	0.541	0.458	0.528
$\mathtt{T}^{\mathtt{h}}$	−	0.332	0.568	0.365	0.538
$\mathtt{T}^{\mathtt{h}}$	+	0.342	0.590	0.375	0.563
$\mathtt{T}^{\mathtt{a}}$	−	0.411	0.590	0.419	0.558
$\mathtt{T}^{\mathtt{a}}$	+	0.419	0.607	0.431	0.581
Maha	−	0.378	0.470	0.402	0.566
Maha	+	0.387	0.493	0.411	0.592
GP	−	0.392	0.602	0.400	0.546
GP	+	0.412	0.621	0.415	0.573
HMM	−	0.512	0.721	0.563	0.689
HMM	+	0.525	0.749	0.583	0.709

Table 5.9. Recognition differences between features with tuning and without tuning. Train: $\mathcal{D}_2^{\text{Beatles}}$. Test: $\mathcal{D}^{\text{Beatles}}$.



(a) CLP(1000) (b) CRP(1, 1) (c) CENS(1, 1)

Figure 5.17. Histogram of recognition with tuning and without tuning, using recognizer $\mathtt{T}^{\mathtt{b}}$.

Although it is asserted by people that the audio files which prepared for chord recognition are in standard tuning, which means for example in a concert the note $A4$ should be exactly 440 Hz, the actual tuning of the music recordings sometimes deviate from the standard tuning. In this section, we check the influence of tuning on the chord recognition. To this

end, we replace the original filter banks which we used to extract pitch features with the six different filter banks which consider tuning deviation from 0, 0.25, 0.33, 0.5, 0.67, 0.75 of the semitones. We select the pitches in a certain range as representative pitches and apply the six filter banks on them and get six versions of the sub bands. Then for each version, we add up the energy of all these sub bands. Finally we select the version which yields maximum energy as the tuning.

Note that such tuning method can not handle the deviation beyond one semitone. Therefore in order to make sure we have the exact tuning, we perform our chord recognition with each time circular shift the features down by one semitone, and with the help of the ground truth chord annotation, we record the number of circular shift with which the chord recognition yields the best recognition accuracies.

Table 5.9 shows the result of recognition accuracies with and without tuning. The plus sign means we integrated tuning while the minus sign means without tuning. We find that most of the integrated recognition rates increase about 1% to 3% compared to the result without tuning, regardless of the feature types and recognizers. Further investigation shows that 50% of the pieces in the $\mathcal{D}^{\text{Beatles}}$ dataset are not exactly tuned, and 10% of the pieces even have deviation beyond one semitone. For most of the untuned pieces, the recognition accuracy in average increased by about 7%. We have even seen an extreme case that with tuning the recognition increased by 57% for the song `BeatlesLovelyRita`. This shows that tuning is a very important pre-processing technique which will largely improve the chord recognition accuracy.

Note that tuning is not always correct and sometimes might decrease the recognition accuracies. Figure 5.17 shows the histogram statistics of the recognition difference between using and not using tuning. In Figure 5.17(c) we find that `CENS` have more decrease than `CLP`(1000) or `CRP`. This is an aspect which is worth future inspection.

5.6 Effect of Using Different Training Datasets

In order to totally examine the influence of using different training data, we test out all potential training dataset which we use in this thesis and evaluate the performance. Table 5.10 shows the recognition of using four different training datasets, with each table evaluated on different features. From the table we find that the various datasets do have influence on the chord recognition result, for example in Table 5.6, evaluated with recognizer `GP` and `CENS`, we got 0.441 when trained on $\mathcal{D}_1^{\text{Beatles}}$, but the results decrease to 0.415 when trained on $\mathcal{D}_2^{\text{Beatles}}$. However, evaluated with `HMM`, the decrease from 0.587 to 0.583 is negligible. This shows that the influence of training data also varies on the recognizers. Furthermore, evaluated with `HMM` and `CRP`, again we find the difference between using $\mathcal{D}_1^{\text{Beatles}}$ and $\mathcal{D}_2^{\text{Beatles}}$ gets 1.4%, which can not be neglected. This implies that we also need to consider the influence of training data together with the feature types.

(a)

	$\mathcal{D}^{\text{Beatles}}$	$\mathcal{D}_1^{\text{Beatles}}$	$\mathcal{D}_2^{\text{Beatles}}$	$\mathcal{D}_3^{\text{Beatles}}$
$\mathtt{T}^{\mathtt{a}}$	0.421	0.422	0.419	0.414
GP	0.426	0.429	0.412	0.410
HMM	0.529	0.528	0.525	0.523

(b)

	$\mathcal{D}^{\text{Beatles}}$	$\mathcal{D}_1^{\text{Beatles}}$	$\mathcal{D}_2^{\text{Beatles}}$	$\mathcal{D}_3^{\text{Beatles}}$
$\mathtt{T}^{\mathtt{a}}$	0.431	0.424	0.431	0.431
GP	0.439	0.441	0.415	0.411
HMM	0.586	0.587	0.583	0.587

(c)

	$\mathcal{D}^{\text{Beatles}}$	$\mathcal{D}_1^{\text{Beatles}}$	$\mathcal{D}_2^{\text{Beatles}}$	$\mathcal{D}_3^{\text{Beatles}}$
$\mathtt{T}^{\mathtt{a}}$	0.582	0.583	0.581	0.583
GP	0.583	0.588	0.573	0.578
HMM	0.718	0.723	0.709	0.721

Table 5.10. Recognition accuracies using different training datasets. Result test on $\mathcal{D}^{\text{Beatles}}$. (a)using feature CP. (b)CENS$(1,1)$. (c)CRP$(1,1)$.

CR	HPSS	CP	CLP(1)	CLP(10)	CLP(100)	CLP(1000)	CENS$(1,1)$	CRP$(1,1)$
$\mathtt{T}^{\mathtt{b}}$	+	0.473	0.521	0.557	0.571	0.565	0.466	0.543
$\mathtt{T}^{\mathtt{b}}$	−	0.460	0.508	0.544	0.553	0.541	0.458	0.528
$\mathtt{T}^{\mathtt{h}}$	+	0.343	0.519	0.587	0.616	0.619	0.374	0.590
$\mathtt{T}^{\mathtt{h}}$	−	0.342	0.497	0.568	0.592	0.590	0.374	0.563
$\mathtt{T}^{\mathtt{a}}$	+	0.427	0.526	0.592	0.624	0.624	0.444	0.608
$\mathtt{T}^{\mathtt{a}}$	−	0.419	0.508	0.577	0.610	0.607	0.431	0.581
Maha	+	0.402	0.488	0.538	0.561	0.568	0.428	0.614
Maha	−	0.387	0.451	0.488	0.495	0.493	0.411	0.592
GP	+	0.417	0.531	0.603	0.635	0.627	0.429	0.603
GP	−	0.412	0.516	0.586	0.619	0.621	0.415	0.573
HMM	+	0.528	0.656	0.722	0.754	**0.757**	0.579	0.724
HMM	−	0.525	0.645	0.706	0.736	0.749	0.583	0.709

Table 5.11. Recognition differences between features with and without HPSS. Train: $\mathcal{D}_2^{\text{Beatles}}$. Test: $\mathcal{D}^{\text{Beatles}}$.

5.7 Harmonic Percussive Source Separation

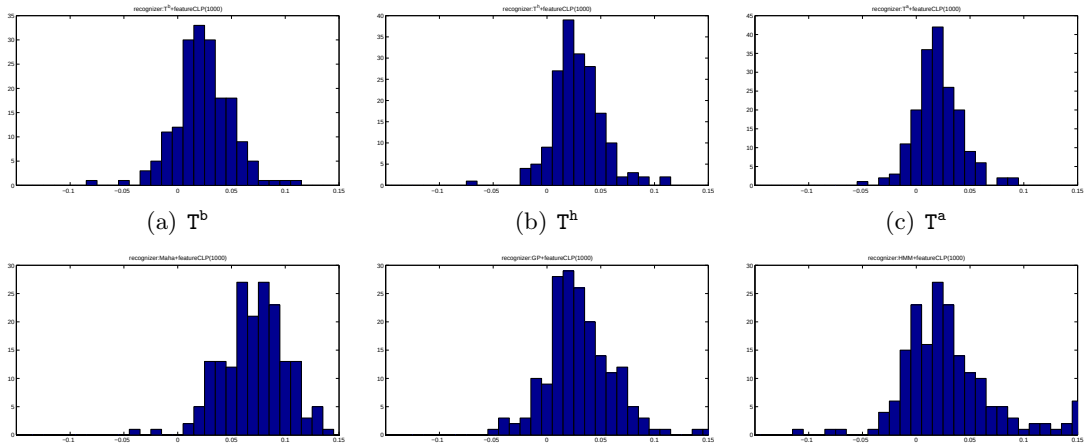
In this experiment, we add the harmonic percussive source separation (HPSS for short) as a pre-processing step for our chord recognition system. This technique is introduced by Ono and Sagayama in [27]. The general idea is to remove the percussive component from the original audio file and to perform a more robust chord recognition on the remaining harmonic component. We recomputed the features from the separated harmonic component of each audio file and then evaluated the recognition accuracies using the same recognizers as before.

CR	CP	CLP(1)	CLP(10)	CLP(100)	CLP(1000)	CENS(1, 1)	CRP(1, 1)
T^b	0.039	0.048	0.060	0.099	0.111	0.044	0.078
T^h	0.042	0.140	0.079	0.091	0.107	0.036	0.098
T^a	0.037	0.083	0.092	0.074	0.090	0.088	0.105
Maha	0.055	0.128	0.120	0.144	0.142	0.047	0.124
GP	0.027	0.075	0.126	0.134	0.157	0.048	0.127
HMM	0.060	0.160	0.228	0.244	**0.246**	0.090	0.145

Table 5.12. Maximum increased recognition difference with and without HPSS.

CR	CP	CLP(1)	CLP(10)	CLP(100)	CLP(1000)	CENS(1, 1)	CRP(1, 1)
T^b	-0.031	-0.022	-0.045	-0.057	-0.080	-0.034	-0.096
T^h	-0.030	-0.017	-0.012	-0.043	-0.067	-0.029	-0.103
T^a	-0.026	-0.034	-0.036	-0.035	-0.048	-0.038	-0.068
Maha	-0.017	-0.038	-0.027	-0.040	-0.045	-0.027	-0.078
GP	-0.013	-0.053	-0.055	-0.053	-0.053	-0.022	-0.079
HMM	-0.050	-0.092	-0.107	-0.091	-0.109	-0.045	**-0.111**

Table 5.13. Maximum decreased recognition difference with and without HPSS.



(a) T^b (b) T^h (c) T^a

(d) Maha (e) GP (f) HMM

Figure 5.18. Histogram of recognition difference with HPSS, using feature CLP(1000).

Table 5.11 shows the result with plus and minus sign indicating the system with or without HPSS respectively. We can see that with the help of HPSS, almost all the recognition accuracies increase about 1% to 3% for the overall result except CENS(1, 1) with HMM has a slight decrease of 0.4%. Although the averaged increase is not so large, the maximum increase is indeed significant. Table 5.12 shows the maximum increased accuracies and Table 5.13 shows the maximum decreased accuracies. By comparing the two tables we find that using HPSS does help to improve the chord recognition, especially the maximum increase of 24.6% is much larger than the maximum decrease of -11.1%. This indicates that the general trend of adding HPSS on chord recognition will yield better result than the

original, however the ability of HPSS is limited, which sometimes appropriately removes the percussive information and keeps a good harmonic component; and sometimes wrongly separated part of the harmonic component into the percussive component and therefore throws the necessary harmony information away and leads to a bad chord recognition result. In particular, the human voice includes both the harmonic component such as vowels and the percussive component such as consonants. Thus for audio files with human voice it is even harder for HPSS to appropriately separate the two components.

For a better understanding of how HPSS increases or decreases the recognition accuracies, we prepare the histogram statistics in Figure 5.18. The x-axis indicates the extent of increase or decrease, and the y-axis indicates the amount of songs which has that increase or decrease. We can see that for all recognizers except `Maha`, the most increased extent is 0.02 or 0.03. Compared to the statistical model based recognizers, the template based recognizers tend to have less amount of decreases. An interesting fact is that `Maha` was significantly improved when integrated with HPSS, only two songs are decreased. Also in `HMM`, there are more songs with increase 0.15, while the other recognizers did not reach that much. Based on all these interesting facts that we observed, we consider that HPSS is worth further inspection and by setting up the separation parameter appropriately in HPSS, it may largely improve the performance of a chord recognition system.

5.8 Effect of Using Beat Synchronized Feature

BS	CP	CLP(1)	CLP(10)	CLP(100)	CLP(1000)	CENS(1,1)	CRP(1,1)
fw_100	0.530	0.646	0.702	0.727	0.735	0.589	0.727
fw_500	0.566	0.685	0.689	0.676	0.669	0.609	0.695
bw_1.0	0.572	0.716	0.731	0.729	0.714	0.621	0.708

Table 5.14. Recognition accuracies using features with and without beat synchronization, tested with `HMM`. Train: $\mathcal{D}_1^{\text{Beatles}} \cup \mathcal{D}_2^{\text{Beatles}}$.. Test: $\mathcal{D}^{\text{Beatles}}$.

BS	CP	CLP(1)	CLP(10)	CLP(100)	CLP(1000)	CENS(1,1)	CRP(1,1)
fw_100	0.443	0.488	0.523	0.533	0.524	0.324	0.509
fw_500	0.495	0.573	0.584	0.572	0.547	0.386	0.545
bw_1.0	0.506	0.586	0.596	0.584	0.559	0.398	0.555

Table 5.15. Recognition accuracies using features with and without beat synchronization , tested with $\mathtt{T}^{\flat}$. Train: $\mathcal{D}_1^{\text{Beatles}} \cup \mathcal{D}_2^{\text{Beatles}}$.. Test: $\mathcal{D}^{\text{Beatles}}$.

There are many suggestions of integrating beat synchronized features to chord recognition systems in the previous research, for example Papadopoulos and Peeters implemented a system considering beat-chromagrams or Tactus-chromagrams in [30].

In this experiment, we briefly examine the effect of using beat-synchronized features. To this end, firstly we replace the framewise features with the features condensed at

beats. Secondly, the chord labels are computed for each of the beat-synchronized features. Thirdly, in order to have make the result comparable to the previous statistics, at the evaluation step we keep the framewise evaluation by resampling the computed chord labels into frames. Finally, the framewise comparison between computed labels and ground truth labels are performed and the recognition accuracies are calculated.

Table 5.14 shows the result tested with recognizer `HMM` and Table 5.15 shows the result with $\mathtt{T}^{\mathtt{b}}$. The rows change the strategy of features in turn. `fw_100` indicates that each feature is computed using a fixed window with length 100 milliseconds, which is the same as our previous setting where feature rate is 10 Hz. `fw_500` means that the window length is set to 500 milliseconds. Usually the duration of a beat is 500 milliseconds as well. `bw_1.0` means that we take the ground truth beat labels to form the beat windows, and averaging the framewise features inside each beat window to generate the beat-synchronized features. From the tables we can find that using recognizer $\mathtt{T}^{\mathtt{b}}$, for all the features, `fw_500` performs better than `fw_100`, and `bw_1.0` gets the best result. However using recognizer `HMM`, the results varies with the feature types. For example, `CP`, `CLP`(1) and `CLP`(10) get the best result with the beat-synchronized features. But for `CLP`(1000) and `CRP`$(1, 1)$, the best result come from the features with fixed window length 100. We consider that such unstable behavior is due to the combination of beat-synchronized features with the post-smoothing effect of HMM. Averaging the original framewise features inside a beat window can be considered as a kind of smoothing as well. In this case, the results might be over smoothed and therefore we observe a decrease in recognition accuracy compared to the result using framewise features.

5.9 Experiments on Classical Dataset

File ID	`CP`	`CLP`(1)	`CLP`(10)	`CLP`(100)	`CLP`(1000)	`CISP`	`CENS`$(1, 1)$	`CRP`$(1, 1)$
`BachBWV846Fischer`	0.613	0.637	0.657	0.680	0.683	0.602	0.558	0.653
`Beet5Bernstein`	0.648	0.658	0.679	0.700	0.687	0.564	0.653	0.648
`ChopMazurkaSmith`	0.761	0.773	0.795	0.788	0.773	0.746	0.596	0.717
`SchumannKonz`	0.750	0.746	0.766	0.811	0.834	0.814	0.757	0.825

Table 5.16. Recognition accuracies of classical dataset using $\mathtt{T}^{\mathtt{b}}$. Test: $\mathcal{D}^4$.

File ID	`CP`	`CLP`(1)	`CLP`(10)	`CLP`(100)	`CLP`(1000)	`CISP`	`CENS`$(1, 1)$	`CRP`$(1, 1)$
`BachBWV846Fischer`	0.508	0.578	0.624	0.646	0.653	0.694	0.596	0.738
`Beet5Bernstein`	0.572	0.585	0.674	0.729	0.759	0.740	0.700	0.849
`ChopMazurkaSmith`	0.578	0.728	0.712	0.739	0.775	0.811	0.700	0.808
`SchumannKonz`	0.556	0.553	0.562	0.652	0.722	0.846	0.776	0.854

Table 5.17. Recognition accuracies of classical dataset using `HMM`. Train: $\mathcal{D}_1^{\text{Beatles}}$. Test: $\mathcal{D}^4$.

All the datset we used before are from Beatles Album which is the instance of pop music.

In this section, as an additional experiment, we test the chord recognition on the classical dataset $\mathcal{D}^4$.

Table 5.16 shows the recognition accuracies of each of the music piece using recognizer $\mathtt{T}^{\mathtt{b}}$. We find that the chords of all these four classical pieces are recognized for at least 0.613, which can be considered as a good result. Table 5.17 shows the recognition accuracies using `HMM`. By comparison of the two tables, we find that generally $\mathtt{T}^{\mathtt{b}}$ gets better result than `HMM` for the feature type `CP` and `CLP`. In contrast, for feature `CISP`, `CENS` and `CRP`, `HMM` gets better result than $\mathtt{T}^{\mathtt{b}}$. This is an interesting phenomenon which needs further inspection.

We previously assume that the result of using `HMM` might not be satisfied. Because the parameters of `HMM` are trained on the pop music, which may not fit for classical music. However, according to the recognition accuracies of the four pieces, we can consider that the model still work for classical music. In particular, for `CRP`, the result using `HMM` is much larger than using $\mathtt{T}^{\mathtt{b}}$ for the first three pieces.

Chapter 6

Summary

In this thesis, we introduced an automatic chord recognition system and performed systematic evaluations at different stages with a focus on the effect of different parameter settings and their interaction. To this end, we first transformed the audio data to several types of feature representations which captured the harmony-related music content. Then, the feature sequences were passed to the chord recognizer module where several chord recognition methods existed. In template based methods, we previously defined or learned a set of feature templates, and performed pattern matching with cosine distance measure. In statistical model based methods, we learned multivariate Gaussian models to represent the chord patterns and then used these models to perform pattern matching in several methods such as Mahalanobis distance based method, Gaussian probability based method and HMM based method.

Our main contributions can be summarized as follows. Firstly, we developed a modular chord recognition system with various types of features and several chord recognizers. Secondly, we conducted extensive experiments which examined certain parameter settings and their impacts on chord recognition accuracies, and we present detailed analysis and conclusions.

By testing chord recognition performance using various types of features we gained insight of the characteristics of each type of features. We found that factors such as logarithmic compression and smoothing in features had significant impact on recognition accuracies. Instead of 12 dimension, using 24 dimension features also improved the results but such improvement varied with feature types and recognizers. In the recognizer side, we found that the suitability of chord recognizers and features had large impact on recognition accuracies. One needed to consider whether the intensity computation of features such as magnitude or energy, was appropriate for certain recognizers. In the statistical model based recognizers. We found that the main contribution of covariance matrix came from the diagonal values. One can disregard the off-diagonal values for convenience of analysis yet keep nearly the same recognition accuracies by scaling the diagonal values. Instead of using trained covariance matrices, manually set covariance matrices worked effectively as well. However one needs to adjust the scale to fit different types of features. Finally, the post-smoothing introduced by HMM brought a large improvement to recognition accuracies. A crucial factor which made it work was the emphasizing of the self-transition

probability. Furthermore, the performance of HMM depended much on the corporation of covariance matrix and transition matrix. It was very sensitive to the values of the entries of these matrices, and any small difference in value setting might totally destroy the performance.

As future work, one could extend the experiments which we briefly discussed such as using harmonic-percussive source separation as pre-processing step and using beat synchronized features instead of framewise features. Furthermore, an insight gained from all these experiments is that if we could compute a type of feature which successfully emphasize harmony-related content of the music audio while suppress the unnecessary noise, even the simplest recognizer such as binary template-based method will yields a good result. Usually the complex recognizers with many parameters only slightly improve the recognition and the computation efficiency is far less than simple recognizers such as all template based methods. Therefore in the future, it is worth to try to enhance our features. One possibility is to integrate more information such as the bass note of the chord and the key information in the features.

Appendix A

Source Code

In this chapter, the headers of selected MATLAB functions created during the writing of this thesis are reproduced. The headers contain information about the name of the described function and its input/output behavior.

General Routine of Chord Recognition

Here we provide the general routine of chord recognition system. Firstly, the audio file is converted to features. Secondly, the chord labels are computed by one of the chord recognizers. Thirdly, the ground truth labels are prepared. Finally, the computed and ground truth labels are compared to express the recognition accuracy via precision and recall.

```
%-------------------------------------------------------------------------------------------------%
%step 1. Load or compute features for the audio file.
%        The file name and parameter of features are passed as arguments.
[song_features] = audiofile_to_features(filename,parameterFeature);

%step 2. Compute chord labels by one of the chord recognizers.
%        Case 1 corresponds to the template-based method which is introduced in Chapter 3
%        The other cases correspond to the statistical model-based methods which is introduced in Chapter 4
switch chord_recognizer
    case 1
        [computed_chords] = template_based_method(song_features,parameterTemplate);
    case 2
        [computed_chords] = Mahalanobis_distance_based_method(song_features,parameterMaha);
    case 3
        [computed_chords] = Gaussian_probability_based_method(song_features,parameterGP);
    case 4
        [computed_chords] = HMM_based_method(song_features,parameterHMM);
end

%step 3. Prepare the annotated ground truth
annotated_chords = parse_annotation_file(filename,parameterAnno);

%step 4. Evaluate recognition accuracy by comparing the computed and ground truth labels.
         The method of evaluation is introduced in Chapter 5.
[precision,recall,f_measure] = evaluate_result(computed_chords,annotated_chords,parameter);
%-------------------------------------------------------------------------------------------------%
```

Procedure of Training the Chord Models and Transition

There are three parameters in the recognizers which needed to be previous trained: The mean vector and the covariance matrix of a Gaussian model which represents a chord pattern, and the transition matrix which describing the chord transitions. The mean vector can be also considered as the averaged templates of a chord pattern. The function `procedure_training` first takes a trainining dataset as input and extract all the features and corresponding labels, then compute all these parameters and finally save them in a file.

Sample usage:

```
[chord_models,transition_matrix] = procedure_training(trainDataset);
```

```
%-------------------------------------------------------------------------------------------------%
%step 1. extract all the features and corresponding labels from the training dataset
for n=1:size(trainDataset)
    filename = strcat(trainDataset{n});

    %load or compute features for the audio file
    [f_features] = audiofile_to_features(filename,parameterFeature);

    %prepare the annotated ground truth
    [chord_labels] = parse_annotation_file(filename,parameterAnno);

    %Accumulating all features
    [all_features]=[all_features,f_features]; % combine features;

    %Accumulating all labels
    [all_labels] = [all_labels ; chord_labels];

end

%step 2. train the mean vector and covariance matrix for each of the Gaussian model which
%        represent the chord pattern. Note that the mean vector can be used as averaged templates.
[chord_models] = train_MeanVectorAndCovMatrix(all_features,all_labels);

%train the transition matrix which save the transition probabilities between chords
[transition_matrix] = train_TransitionMatrix(L_CP);

%step 3. save all these parameters in a file
save_transition_model(transition_matrix);
save_chord_models(chord_models);
%-------------------------------------------------------------------------------------------------%
```

Feature Extraction

The `audiofile_to_features` function is a wrapper to several feature extraction functions in the Chroma Tool Box [22]. Here we compute a certain type of features, with the type information specified in the parameter.

Sample usage:

```
[f_features] = audiofile_to_features(filename,parameter);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name: audiofile_to_features
% Date of Revision: 19.01.2011
% Programmer: Nanzhu Jiang
```

```
%
% Description:
% extract audio features from the given audio file.
%
% Input:
%         filename;
%         parameter.featureRate;
%         parameter.featureType; %1 : CP
%                                %2 : CLP
%                                %3 : CENS
%                                %4 : CRP
%                                %5 : CISP
% Output:
%         f_features;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Template-based Chord Recognizer

The `template_based_method` function takes audio features as input and computes chord labels for these features by comparing the each of them with the predefined or trained template chord patterns.

Sample usage:

```
[computed_chords] = template_based_method(f_features,parameter);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name: template_based_method
% Date of Revision: 19.01.2011
% Programmer: Nanzhu Jiang
%
% Description:
% Assign chord labels to features via template-based method.
%
% First, load the previous defined or trained templates according to
% templateType. Then, for each of the feature, compute cosine distance
% between the feature and templates. The computed chord for that features
% is the one whose template yields the minimum distance.
%
% Input:
%         f_features
%         parameter.numChroma = 12;
%         parameter.numChords = 24;
%
% Optional parameter:
%         parameter.templateType; % 1: binary templates
%                                 % 2: harmonically enriched templates
%                                 % 3. averaged templates
%         parameter.model; % the model where averaged templates saved
%
% Output:
%         computed_chords
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Mahalanobis Distance-based Chord Recognizer

The `Mahalanobis_distance_based_method` function assigns chord labels to the input features via selecting the minimum Mahalanobis distance between the features and the loaded

chord patterns.

Sample usage:

```
[computed_chords] = Mahalanobis_distance_based_method(f_features,parameter);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name: Mahalanobis_distance_based_method
% Date of Revision: 19.01.2011
% Programmer: Nanzhu Jiang
%
% Description:
% Assign chord labels to features via Mahalanobis distance-based method.
%
% First, for each of the chord, load the mean vector and covariance matrix
% from the Gaussian model which described the chod pattern. Then, compute the
% Mahalanobis Distance for each of the feature and the chord patterns. The computed
% chord for that features is the one whose pattern yields the minimum distance.
%
% Input:
%         f_features;
%         parameter.model; %Gaussian models which specify chord patterns
%                          %For each model, we load the mean vector and the covariance
%                           matrix which determined the model.
%
% Output:
%         computed_chords;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Gaussian Probability-based Chord Recognizer

The `Gaussian_probability_based_method` function represents chord patterns by Gaussian models. Here, the Gaussian probability describes the possibility that a feature belongs to a certain chord distribution. The assigned chord for the feature is determined by selecting the chord pattern whose model gets the maximum Gaussian probability.

Sample usage:

```
[computed_chords] = Gaussian_probability_based_method(f_features,parameter);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name: Gaussian_probability_based_method
% Date of Revision: 19.01.2011
% Programmer: Nanzhu Jiang
%
% Description:
% Assign chord labels to features via Gaussian probability-based method.
%
% First, for each of the chord, load the mean vector and covariance matrix
% from the Gaussian model which described the chod pattern. Then, compute the
% Gaussian probability for each of the feature and the chord patterns. The computed
% chord for that features is the one whose pattern yields the maximum probability.
%
% Input:
%         f_features;
%         parameter.model; %Gaussian models which specify chord patterns
%                          %For each model, we load the mean vector and the covariance
%                           matrix which determined the model.
%
% Output:
%         computed_chords;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

HMM-based Chord Recognizer

The `HMM_based_method` function first computes the Gaussian probabilities for each of the feature, then treat these probabilities as observation probabilities of the Viterbi algorithm. By plug in the observation and transition probabilities as parameters, the Viterbi algorithm computes the chord labels with smoothing of temporal context. The output of Viterbi algorithm is the sequence of final chord labels.

Sample usage:

```
[computed_chords] = HMM_based_method(f_features,parameter);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name: HMM_based_method
% Date of Revision: 19.01.2011
% Programmer: Nanzhu Jiang
%
% Description:
% Assign chord labels to features via HMM-based method.
%
% Firstly, for each of the chord, load the mean vector and covariance matrix
% from the Gaussian model which described the chod pattern. Load the transition
% matrix which describe the chord transition probabilities.
%
% Secondly, compute the Gaussian probability for each of the feature and the
% chord patterns.
%
% Thirdly, the Gaussian probability is treated as observation probability,
% and the transition matrix is treated transition probability. The chord labels
% are computed via Viterbi algorithm, in which the both probability are plugged in
% as parameter.
%
% Input:
%         f_features;
%         parameter.model; %Gaussian models which specify chord patterns
%                          %For each model, we load the mean vector and the covariance
%                           matrix which determined the model.
%
%         parameter.transition_matrix; %24*24 square matrix which saved the transition
%                                       probabilities of the chords.
%
% Output:
%         computed_chords;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Parse Chord Annotations

The `parse_annotation_file` function reads chord annotations from the specified file and map the original chords to one of the 24 triads. The triad chords are further mapped to integers for the convenience of the next evaluation step.

Sample usage:

```
[annotated_chords] = parse_annotation_file(filename,parameter);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name: parse_annotation_file
% Date of Revision: 19.01.2011
% Programmer: Meinard Mueller, Verena Konz, Nanzhu Jiang
```

```
%
% Description:
% parse annotation file, extract chord annotation and segment them into
% frames. The number of annotation frames are equal to the number of
% feature frames specified in the parameter.
%
% Input:
%         filename;
%         parameter.featureFrames;
%         parameter.featureRate;
%
% Output:
%         annotated_chords;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Result Evaluation

The `evaluate_result` function compare the compare the computed chords with the annotated chords, and evaluate correctness via precision and recall. The f-measure is taken as the final recognition accuracy.

Sample usage:

```
[precision,recall,f_measure] = evaluate_result(computed_chords,annotated_chords,parameter);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name: evaluate_result
% Date of Revision: 19.01.2011
% Programmer: Meinard Mueller, Verena Konz, Nanzhu Jiang
%
% Description:
% Result evaluation via precision and recall. Take f-measure as chord recognition
% accuracy.
%
% Input:
%         computed_chords;
%         annotated_chords;
%         parameter;
%
% Optional:
%         parameter.neglectNotAnnotatedChords;
%
% Output:
%         precision;
%         recall;
%         f_measure;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

List of Figures

List of Tables

Bibliography

[1] AI ACCESS. Multivariate normal distribution, mean vector, covariance and covariance matrix. `http://www.aiaccess.net/English/Glossaries/GlosMod/e_gm_multinormal_distri.htm`, Retrieved 12.05.2010.

[2] M. A. Bartsch and G. H. Wakefield. Audio thumbnailing of popular music using chroma-based representations. *IEEE Transactions on Multimedia*, 7(1):96–104, Feb. 2005.

[3] J. P. Bello and J. Pickens. A robust mid-level representation for harmonic content in music signals. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, London, UK, 2005.

[4] T. Cho, R. J. Weiss, and J. P. Bello. Exploring common variations in state of the art chord recognition systems. Barcelona, Spain, 2010.

[5] Dan Ellis and Jesper Højvang Jensen. Isp tool box. `http://kom.aau.dk/project/isound/`, Retrieved 31.12.2010.

[6] K. Dressler. Sinusoidal extraction using an efficient implementation of a multi-resolution FFT. *Proceedings of 9th International Conference on Digital Audio Effects (DAfX-06)*, pages 18–20, 2006.

[7] D. P. W. Ellis and G. E. Poliner. Identifying 'cover songs' with chroma features and dynamic programming beat tracking. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 4, Honolulu, Hawaii, USA, Apr. 2007.

[8] R. Gnanadesikan. *Methods for statistical Data Analysis of Multivariate Observations*. John Wiley & Sons., United States of America, 1977.

[9] E. Gómez. *Tonal Description of Music Audio Signals*. PhD thesis, UPF Barcelona, 2006.

[10] E. Gómez. Tonal description of polyphonic audio for music content processing. *INFORMS Journal on Computing*, 18(3):294–304, 2006.

[11] P. Grosche and M. Müller. Extracting predominant local pulse information from music recordings. *IEEE Transactions on Audio, Speech, and Language Processing*, 2011.

[12] C. Harte and M. Sandler. Automatic chord identification using a quantised chromagram. In *Proceedings of the Audio Enginerring Society*, 2005.

[13] C. Harte, M. Sandler, S. Abdallah, and E. Gómez. Symbolic representation of musical chords: A proposed syntax for text annotations. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, London, GB, 2005.

[14] X. Huang, A. Acero, A. Acero, and H.-W. Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR.

[15] A. P. Klapuri, A. J. Eronen, and J. Astola. Analysis of the meter of acoustic musical signals. *IEEE Transactions on Audio, Speech and Language Processing*, 14(1):342–355, 2006.

[16] V. Konz, M. Müller, and S. Ewert. A multi-perspective evaluation framework for chord recognition. In *Proceedings of the 11th International Conference on Music Information Retrieval (ISMIR)*, pages 9–14, Utrecht, Netherlands, 2010.

[17] K. Lee. Automatic chord recognition from audio using enhanaced pitch class profile. In *Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR)*, 2006.

[18] B. Logan. Mel frequency cepstral coefficients for music modeling. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, Plymouth, Massachusetts, 2000.

[19] G. Madazrov and D. Gjorgjevikj. Evaluation of distance measures for multi-class classification in binary svm decision tree. In *Artificial Intelligence and Soft Computing: 10th International Conference (ICAISC)*, 2010.

[20] J. Makhoul, F. Kubala, R. Schwartz, and R. Weischedel. Performance measures for information extraction. In *In Proceedings of DARPA Broadcast News Workshop*, pages 249–252, 1999.

[21] M. Mauch and S. Dixon. Simultaneous estimation of chords and musical context from audio. *IEEE Trans. Audio, Speech, and Language Processing*, 18(6):1280–1289, 2010.

[22] Meinard Müller and Sebastian Ewert and Peter Grosche. Chroma tool box. `http://www.mpi-inf.mpg.de/~mmueller/chromatoolbox/`, Retrieved 14.12.2010.

[23] M. Müller. *Information Retrieval for Music and Motion*. Springer Verlag, 2007.

[24] M. Müller and S. Ewert. Towards timbre-invariant audio features for harmony-based music. *IEEE Transactions on Audio, Speech, and Language Processing (TASLP)*, 18(3):649–662, 2010.

[25] M. Müller, S. Ewert, and S. Kreuzer. Making chroma features more robust to timbre changes. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1869–1872, Taipei, Taiwan, Apr. 2009.

[26] M. Müller, F. Kurth, and M. Clausen. Audio matching via chroma-based statistical features. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 288–295, 2005.

[27] N. Ono, K. ichi Miyamoto, H. Kameoka, and S. Sagayama. A real-time equalizer of harmonic and percussive components in music signals. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR)*, Philadelphia, Pennsylvania, USA, 2008.

[28] L. Oudre, Y. Grenier, and C. Févotte. Template-based chord recognition: Influence of the chord types. In *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR)*, 2009.

[29] H. Papadopoulos and G. Peeters. Large-scale study of chord estimation algorithms based on chroma representation and hmm. In *Proceedings of the International Workshop on Content-based Multimedia Indexing)*, 2007.

[30] H. Papadopoulos and G. Peeters. Simultaneous estimation of chord progression and downbeats from an audio file. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 121–124, 2008.

[31] L. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition*. Prentice Hall Signal Processing Series, 1993.

[32] A. C. Rencher. *Methods of Multivariate Analysis*. John Wiley & Sons., United States of America, 2002.

[33] A. Schafstein. Determining tempo characteristics of expressive music recordings: An algorithmic approach. *Master Thesis*, 2009.

[34] G. Taguchi and R. Jugulum. *The Mahalanobis-Taguchi Strategy: A Pattern Technology System*. John Wiley & Sons., United States of America, 2002.

[35] Wikipedia. Precision and recall. `http://en.wikipedia.org/wiki/Precision_and_recall`, Retrieved 19.01.2011.

[36] E. Zwicker and H. Fastl. *Psychoacoustics, facts and models*. Springer Verlag, New York, NY, US, 1990.