Automatic Chord Recognition

Ke Ma
Department of Computer Sciences
University of Wisconsin-Madison
Madison, WI 53706
kma@cs.wisc.edu

Abstract

Automatic chord recognition is the first step towards complex analyses of music. It has became an active research topic and is of importance for both scientific research and commercial applications. Many methods have been proposed to attack this problem, most of which are based on the Pitch Class Profile. I propose a novel automatic chord recognition method that improves the traditional methods by incorporating some new ideas including the Soft Thresholding denoising, the Improved Pitch Class Profile, and the circular shift and weighted sum based Template Matching. Experiments show that my method is both efficient and accurate.

1 Introduction

Human brain is so highly developed that it is capable of processing and understanding very complex audio signals like music. However, it remains an active research topic for computers to extract important information from music. On the one hand, simple tasks such as beat detection and pitch recognition have been well studied, and lead to many successful applications including BPM (beats per minute) counters and instrument tuners. On the other hand, more comprehensive analyses on more complex signals usually result in limited success.

Automatic chord recognition can be regarded as the first step towards such complex tasks. A chord in music is defined as any harmonic set of three or more notes that is heard as if sounding simultaneously. Given an audio recording of a chord, how can we label it with its chord name, or in other words, recognize the chord? This task is challenging even for music professionals, because many chords can hear very similar. If it is possible to accurately transcribe audio recordings into chord sequences, people can do some further analyses on music structures, or characterize audio recordings based on their chord sequences. That's why automatic chord recognition is of interest to Digital Music and Information Retrieval communities as well.

In this paper, I present a novel automatic chord recognition method. I start by constructing the amplitude spectrum of the input audio recording with the Discrete Fourier Transform. The characteristics of the noise are thoroughly analyzed, and the noise is removed via Soft Thresholding. Then I convert the feature representation to the Improved Pitch Class Profile with the help of the Harmonic Product Spectrum. And finally I label the audio recording with a chord name using a circular shift and weighted sum based Template Matching method with a refined type of weight vector. Here I focus on recognizing guitar chords, but the general procedure is not limited to guitars.

2 Related Work

The traditional approach to recognizing a chord is to first identify individual notes that constitute the chord [1], followed by a rule-based reasoning process to infer the chord. This kind of approach usually fails because of its error-prone first step.

Fujishima [2] introduced a feature representation called the Pitch Class Profile, which avoids identifying individual notes. This representation soon became the mainstream, and various methods were proposed based on that. Some took a static approach by using a template matching algorithm [4,7]; others considered the context of the chord and employed a dynamic probabilisitic model such as the HMM [3,5,6].

My method follows the static approach above. In a framework similar to the previous work such as [2,4,7], I look carefully into the key components of the pipeline and improves them with some new ideas. The main contribution of this paper lies in that: 1) I formalize every step of the chord recognition procedure, and discuss some important implementation details; 2) I improve the chord recognition procedure.

3 My Method

3.1 DFT and Amplitude Spectrum

Audio recordings come in various forms, including different durations and sample rates. There is a need for unifying different audio recordings so that we can develop a procedure applicable to any audio recording regardless of its format. In order to recognize a chord, eventually we have to characterize the notes that construct the chord. That is to say, we care more about the frequency information in audio recordings. These motivations lead us to first construct the amplitude spectrum for the input audio recording.

The method of choice is the Discrete Fourier Transform (DFT) [8]. It converts a discrete signal into coefficients of a finite combination of complex sinusoids. The DFT of a length-N signal can be expressed as

$$\tilde{x_k} = \sum_{n=0}^{N-1} x_n e^{-2\pi i k n/N},\ k \in \{0, 1, ..., N-1\},$$

or alternatively, the product of a $N \times N$ matrix denoted as U^T and the original signal $\boldsymbol{x}$

$$\tilde{\boldsymbol{x}} = U^T \boldsymbol{x}.$$

This is equivalent to performing the Fast Fourier Transform (FFT) algorithm on the signal $\boldsymbol{x}$, which has a time complexity of $O(N \log N)$. An important property of the DFT matrix is

$$U^T U = N I_c,$$

where I_c is the $N \times N$ complex identity matrix.

The DFT of a length-N signal $\boldsymbol{x}$ is a length-N complex vector $\tilde{\boldsymbol{x}}$, whose elements are coefficients of the sinusoidal bases of different frequencies. The magnitude of a coefficient is the amplitude of that component, and the angle is the relative phase. In this analysis, the phase information is of no importance. We can construct the 2-sided amplitude spectrum by computing the norms of the complex coefficients normalized by $1/N$ [9],

$$\boldsymbol{S_x^{(2)}} = |\frac{\tilde{\boldsymbol{x}}}{N}|.$$

This amplitude spectrum is called 2-sided because half the energy is displayed at the positive frequencies, and half at the negative frequencies. The spectrum of an audio signal is normally symmetrical around DC, so we can discard the second half of the 2-sided amplitude spectrum and double the rest elements except for DC to construct the 1-sided amplitude spectrum,

$$S_{x_k}^{(1)} = \begin{cases} S_{x_k}^{(2)}, & k = 0, \\ 2S_{x_k}^{(2)}, & k = 1, 2, ..., N/2 - 1. \end{cases}$$

Let's denote $\boldsymbol{S_x} := \boldsymbol{S_x^{(1)}}$ for convenience.

For the following two reasons, we may not want to use all the frequency bands in the 1-sided amplitude spectrum for later analyses. First, a guitar, as a musical instrument, has its own range. The lowest pitch a standard-tuned 6-stringed guitar can play is E_2 (82.407 Hz), and the highest can be $C\#_6$ (1108.7 Hz), D_6 (1174.7 Hz) or E_6 (1318.5 Hz) depending on the number of frets [10]. In other words, if a guitar is not severely out of tune, the sound it produce cannot have components of frequency lower than around 80 Hz. Those components, if any, can be regarded as pure noise. However, the sound can have pretty high frequency compenents due to the overtones. Although the amplitude of the overtones gets lower very fast as the order increases, we do want to keep as much high frequency information in this analysis as possible, so as to accurately estimate the characteristics of the noise. Second, very low and very high frequency components can be misleading because of the compression of the input audio files. As human ears are less sensitive to very low or very high frequencies, audio compressors usually react to different frequencies differently. For example, many MP3 encoders throw away components of frequency higher than 15 kHz [11]. If we are unaware of this fact and use all frequencies for analyses, we may be misled and assume that these high frequency components don't exist in nature.

3.2 Denoising

Noise is involved in every stage from producing a sound, i.e. noise of the instrument, to recording it, i.e. noise of the cable or the recording device. The Central Limit Theorem indicates that the summation of many random processes tends to follow a normal distribution, so it's reasonable to assume the noise in audio recordings is additive white Gaussian noise (AWGN). So the observed audio signal can be represented as a length-N vector $\boldsymbol{x} := \boldsymbol{s} + \boldsymbol{\epsilon}$, where $\boldsymbol{s}$ is the noiseless audio signal and

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 I).$$

In this project, we focus on audio recordings of chords, so by assumption the amplitude spectrum of the noiseless audio signal should be sparse, which only has peaks corresponding to the fundamental frequencies and overtones of the notes that construct the chord. However, because of the presence of the noise, the amplitude spectrum of the observed signal is never sparse. The denoising procedure is to recover the noiseless amplitude spectrum from the noisy observation.

Let's consider how the noise behaves in the frequency domain. The DFT of the noise is $\tilde{\boldsymbol{\epsilon}} := U^T\boldsymbol{\epsilon}$ and follows a complex normal distribution. If we denote $\tilde{\boldsymbol{\epsilon}} := \boldsymbol{a} + \boldsymbol{b}i$, it can be shown that

$$\boldsymbol{a} \sim \mathcal{N}(\mathbf{0}, \frac{\sigma^2 N}{2} I)$$

and

$$\boldsymbol{b} \sim \mathcal{N}(\mathbf{0}, \frac{\sigma^2 N}{2} I).$$

To construct the amplitude spectrum, we compute

$$S_{\epsilon_k} = 2\sqrt{(\frac{a_k}{N})^2 + (\frac{b_k}{N})^2}$$

except for $k = 0$. Obviously, S_{ϵ_k} is the norm of a random complex number whose real and imaginary components are Gaussian with zero mean and equal variance, so it follows a Rayleigh distribution [12],

$$S_{\epsilon_k} \sim Rayleigh(\sqrt{\frac{2\sigma^2}{N}}).$$

Let's denote the parameter as $\beta := \sqrt{2\sigma^2/N}$. Note that the level of the noise decreases in $O(1/\sqrt{N})$, and thus the signal-to-noise ratio (SNR) increases. That's why we want to keep as much information as possible in the first step.

Putting all together, the amplitude spectrum of the observed noisy audio signal $\boldsymbol{S_x}$ is the summation of a sparse non-negative vector $\boldsymbol{S_s}$ that represents the spectral characteristics of the chord and a noise vector $\boldsymbol{S_\epsilon}$ whose elements follow a Rayleigh distribution with an unknown parameter β.

In order to recover the sparse vector $\boldsymbol{S_s}$ with high confidence, we need to estimate this unknown parameter β. The significant elements of the sparse vector $\boldsymbol{S_s}$ can be regarded as outliers in the obeservation $\boldsymbol{S_x}$, so we should take advantage of a robust estimation approach that is insensitive to outliers. The robust statistic of choice is the median absolute deviation (MAD) [13], which is defined as

$$\begin{aligned} MAD &= \underset{j}{\text{median}}(|S_{\epsilon_j} - \underset{k}{\text{median}}(S_{\epsilon_k})|) \\ &\approx \underset{j}{\text{median}}(|S_{x_j} - \underset{k}{\text{median}}(S_{x_k})|). \end{aligned}$$

The MAD measures variability of data, and is commonly used as a consistent estimator of a scale parameter when multiplied with a constant factor. We need to examine the specific distribution to compute the factor. It's not hard to show that the median of the Rayleigh distribution with a scale parameter β is $\beta\sqrt{\ln 4}$. According to the definition of the MAD,

$$\begin{aligned} \frac{1}{2} &= P(|S_{\epsilon_k} - \beta\sqrt{\ln 4}| \le MAD) \\ &= e^{-\frac{(\beta\sqrt{\ln 4} - MAD)^2}{2\beta^2}} - e^{-\frac{(\beta\sqrt{\ln 4} + MAD)^2}{2\beta^2}}. \end{aligned}$$

Solving for β yields

$$\hat{\beta} = 2.2299\ MAD.$$

With the estimate $\hat{\beta}$, we can employ a soft thresholding operator to recover the sparse vector $\boldsymbol{S_s}$. As the observation $\boldsymbol{S_x}$ is an amplitude spectrum, all of its elements are non-negative. The significant elements of the sparse vector $\boldsymbol{S_s}$ are assumed to be much larger than the elements of the noise vector $\boldsymbol{S_\epsilon}$. Given these facts, we consider the one-sided version of the soft thresholding operator, which is defined as

$$\hat{S_{s_k}} = \max(S_{x_k} - \tau, 0).$$

The threshold τ is chosen to be

$$\tau = \sqrt{2\hat{\beta}^2 \log N}.$$

This choice is motivated by the equivalent multiple testing problem. Imagine the observation is just noise, and we want to rule out all the elements via thresholding. So we have

$$\begin{aligned} P(\bigcup_{k=0}^{N-1} S_{\epsilon_k} \ge t) &\le \sum_{k=0}^{N-1} P(S_{\epsilon_k} \ge t) \\ &= N e^{-\frac{t^2}{2\beta^2}}, \end{aligned}$$

or,

$$P(\bigcup_{k=0}^{N-1} S_{\epsilon_k} \ge \sqrt{2\beta^2 \log \frac{N}{\delta}}) \le \delta.$$

3.3 Improved Pitch Class Profile

The next step is to design the feature vector that can be used for chord recognition from the amplitude spectrum. There are at lease two kinds of complexities we need to handle. The first complexity is related to octaves. A chord is usually defined as a set of pitch classes; it retains its identity if the notes are in different octaves, or are stacked in a different way vertically. For example, a C chord can be composed of $\{C_3, E_3, G_3\}$, or $\{C_4, E_4, G_4\}$, or $\{E_4, G_4, C_5\}$, or even $\{C_3, E_3, G_3, C_4, E_4\}$.

The second complexity is related to overtones. When we play a note, the musical instrument never produces a simple signal of its fundamental frequency; rather, the signal is a combination of signals of integer multiples of its fundamental frequency which are called overtones. The presence of overtones may confuse chord recognition systems because it appears to the systems that some extra notes are involved in the chord. Take the C chord $\{C_3, E_3, G_3\}$ as an example. The 3rd overtone of E_3 (164.81 Hz) is 494.43 Hz, which is very close to the fundamental frequency of B_4 (493.88 Hz), but B_4 is not a note in the chord.

To handle the first complexity, I follow the traditional scheme using the Pitch Class Profile (PCP) feature representation [2]. The PCP is just a 12-dimensional vector, each of whose elements represents the power of each semitone pitch class. The procedure of building PCP vectors collapses pitches of the same pitch class into the same bin, considering only the chroma of each pitch rather than the octave. By defining the mapping function

$$p(k) = \text{round}(12\log_2(\frac{k}{N}\frac{f_s}{f_{ref}})) \bmod 12, \quad k = 1, 2, ..., N/2 - 1,$$

where f_s is the sampling rate and f_{ref} is the reference frequency that falls into P_0, we can build the PCP as

$$P_j = \sum_{k:p(k)=j} \hat{S_{s_k}}^2.$$

The mapping function is defined to reflect the fact that human perception of musical intervals is approximately logarithmic with respect to their fundamental frequencies. For instance, people periceive that the distance between A_3 (220 Hz) and A_4 (440 Hz) is the same as that between A_4 and A_5 (880 Hz). The round operator can also accomodate to cases where the instrument is slightly out of tune, i.e. less than 50 cents.

To handle the second complexity, I take advantage of some ideas from an alternative representation called Enhanced Pitch Class Profile (EPCP) [4]. The EPCP improves the PCP by using the Harmonic Product Spectrum (HPS) instead of the amplitude spectrum. The HPS was originally used for pitch detection [14]. The basic idea is that the fundamental frequency can be determined by measuring the overtones and then computing the greatest common divisor. To obtain the HPS, we downsample the amplitude spectrum by integer factors, and multiply all the spectra together. After that, there is a clear peak corresponding to the fundamental frequency because it is the common divisor of the overtones. It turns out that this idea is also applicable to the chord recognition task, as the HPS helps get rid of much of the overtone components, but with a small tweak. The small tweak here is that instead of integer factors, we only downsample the amplitude spectrum by powers of 2, because the other overtones may contribute to other pitch classes than those who comprise chord notes.

My method is different from the EPCP in that I use the M-th root of the HPS to maintain the magnitude of the spectrum if the HPS is computed by multiplying M spectra. This procedure is summarized as

$$\hat{S^*_{s_k}} = \sqrt[M]{\prod_{m=0}^{M-1} \hat{S}_{s_{(2^m k)}}}.$$

Note that the parameter M is related to the number of overtones to be considered and should be carefully set. It should be obvious that we cannot set M to a very small integer, because we want to remove overtone components as completely as possible. But we cannot set M to a very large integer either. As mentioned before, the highest pitch on a guitar is lower than E_6 (1318.5 Hz), and the amplitude spectrum provides information for frequencies no higher than 15 kHz. That means we should be able to consider at least 11 overtones. However, the higher-order overtones have very low amplitudes that are likely to lose after denoising; even if they still exist, their small values tend to cause numeric precision issues. It shows that $M = 4$ is a good choice in practice.

Another difference is that each element in my feature vector represents amplitude instead of power. This is to reduce the influence of several very strong pitch classes in a chord. Therefore, the M-th root of the HPS is converted to a length-12 feature vector by computing

$$P^*_j = \frac{1}{Z}\sqrt{\sum_{k:p(k)=j} \hat{S^*_{s_k}}^2},$$

where Z is a normalizing coefficient so that the largest element in $\boldsymbol{P^*}$ is 1. This vector $\boldsymbol{P^*}$ is called the Improved Pitch Class Profile (IPCP). After obtaining the IPCP, when we talk about notes, they just mean their pitch classes.

3.4 Template Matching

The IPCP roughly tells us how the chord is composed, but we still need to name the chord. A natural way to do this is to define a template for each chord, and try to match the IPCP with the templates [2]. I consider 16 chord types in this project, and each chord type may have 12 different root notes, so there are 192 chords in total. If we list the chords whose root notes

are C, they include: Triad Chords - C Major (C), C Minor (Cm), C Diminished ($Cdim$), C Augmented ($Caug$); Seventh Chords - C Seventh ($C7$), C Major Seventh ($Cmaj7$), C Minor Seventh ($Cm7$), C Minor Major Seventh ($Cmmaj7$), C Diminished Seventh ($Cdim7$), C Half Diminished Seventh ($Chdim7$ or $Cm7b5$); Extended Chords - C Ninth ($C9$), C Major Ninth ($Cmaj9$), C Minor Ninth ($Cm9$); Sixth Chords - C Major Sixth ($C6$), C Minor Sixth ($Cm6$); and Suspended Chords - C Suspended Fourth ($Csus4$).

Defining a template for each of the 192 chords is prohibitively tedious; we can circumvent this by using a property of the chords. For the chords of the same type, the template for a chord is just a shifted version of that of another chord with a different root note. For example, the C chord is composed of $\{C, E, G\}$. And the $C\#$ chord is composed of $\{C\#, F, G\#\}$, each of which is one semitone higher than the corresponding note in the C chord. Therefore, we only need to define templates for the chord types. By shifting the IPCP vector circularly, we can match it with the templates for different types of chords with different root notes. The circular shift operation can be expressed as

$$\text{shift}(\boldsymbol{P^*}, s) = [P^*_{((j+s) \bmod 12)}]_{j=0}^{11},$$

where s is the shift amount and is also associated with a root note.

The template matching method of choice is the weighted sum. Each template is a length-12 weight vector $\boldsymbol{W_t}$, where t means the chord type. The score for a chord is computed as the dot product of the weight vector and the circularly shifted IPCP vector,

$$Score_{s,t} = \boldsymbol{W_t}^T \text{shift}(\boldsymbol{P^*}, s).$$

The pair (s, t) that maximizes the score determines the root note and the type of the chord, and thus we can name the chord by concatenating the root note with the chord type postfix.

The last challenge is to define the templates. Let's assume that the IPCP format we use is $[C, C\#, D, D\#, E, F, F\#, G, G\#, A, A\#, B]$, and we define templates for each chord type with a fixed root note of C. A straightforward way to define a template is to set the elements corresponding to the chord notes to 1, and set the rest to 0. For example, the template for the C chord is $[1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0]$. But this kind of templates is not so satisfying for two reasons. First, this weight vector only encourages the chord notes, but we also want it to discourage the non-chord notes. A large element in the IPCP vector corresponding to a non-chord note should make it less probable to be classfied as this chord; but this weight vector is unable to take care of this aspect. Second, this weight vector cannot distinguish the number of notes in the chord. For example, the $C7$ chord, whose template is $[1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1]$, always has a higher score than the C chord.

Thus I come up with a refined template as follows: the elements corresponding to the chord notes are initially set to 1, and the rest are set to -1; then the weight vector is divided by the number of chord notes. For example, the new template for the C chord is $[\frac{1}{3}, -\frac{1}{3}, -\frac{1}{3}, -\frac{1}{3}, \frac{1}{3}, -\frac{1}{3}, -\frac{1}{3}, \frac{1}{3}, -\frac{1}{3}, -\frac{1}{3}, -\frac{1}{3}, -\frac{1}{3}]$. The negative weights discourage the non-chord notes, while the scaling accounts for the number of chord notes. This kind of refined templates is both easy to interpret and achieves decent classification accuracy.

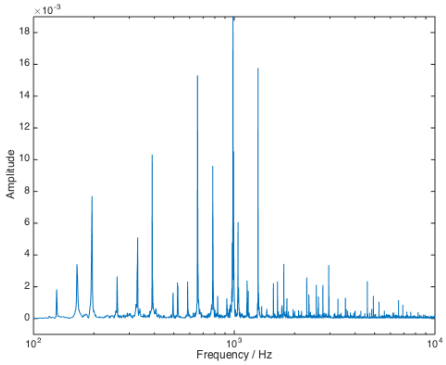
4 Experiments

4.1 Recognizing the C Chord: An Example

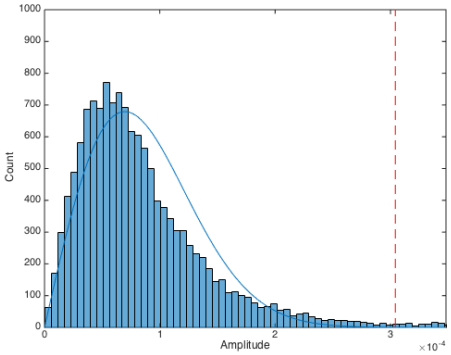
Let's start by demonstrating the automatic chord recognition method applied to an audio recording of the popular C chord. The C chord is played on an electric guitar (unplugged), and is recorded by a mobile phone. The duration of the audio recording is about 2 seconds.

After reading the audio file, we can construct its amplitude spectrum as shown in Figure 1a. Just as expected, there are a few significant elements that reflect the fundamental frequencies and overtones of the chord notes, and all the other elements are just noise that is small in magnitude. Although not shown here, if we examine the amplitude spectrum carefully at around 15 kHz, we can see a step there clearly. All the elements corresponding to frequecies higher than 15 kHz are 0s due to the audio compression. It's very important to discard this high frequency part of the amplitude spectrum, especially when characterizing the noise.

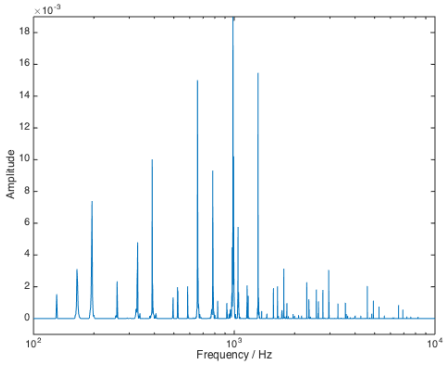
We can then estimate the parameter of the Rayleigh distribution that underlies the noise. The amplitude histogram and the estimated Rayleigh distribution density are plotted in Figure 1b. The histogram almost fits the Rayleigh distribution density, except that it has a larger skewness and a heavier tail. That might mean that the additive white Gaussian noise assumption is not perfectly accurate, but it is still a fairly good approximation. The denoising threshold is determined accordingly, shown as the red dashed line. Elements that are smaller than this threshold are considered pure noise and are suppressed.



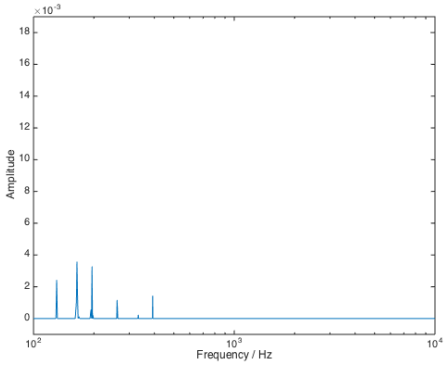
(a) Amplitude Spectrum



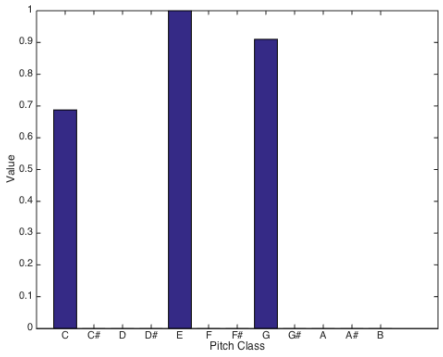
(b) Amplitude Histogram



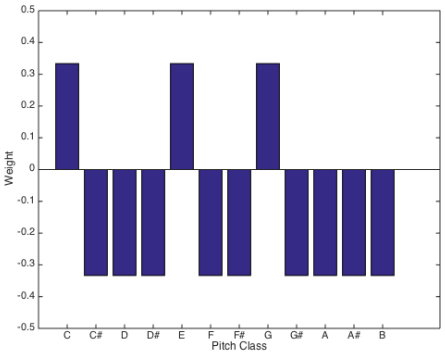
(c) Denoised Amplitude Spectrum



(d) Harmonic Product Spectrum (M-th root of)



(e) Improved Pitch Class Profile



(f) Template for the C Chord

Figure 1: Recognition of the C Chord

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
M	1	2	1	1	1	1	1	1	1	1	1	2
m	1	1	1	1	1	1	1	1	1	1	1	1
dim	1	1	1	2	1	1	1	1	1	7	1	1
aug	1	1	2	1	2	2	2	3	2	1	1	1
7	1	1	2	4	1	2	1	1	1	1	1	2
maj7	1	1	1	3	1	1	1	1	1	2	1	1
m7	1	1	1	5	1	1	1	1	1	1	2	2
mmaj7	1	1	1	4	1	1	1	1	1	2	1	1
dim7	1	1	1	1	1	1	2	1	1	1	1	1
hdim7	1	1	1	2	1	1	1	1	1	13	1	1
6	2	1	1	2	1	1	4	1	2	1	1	1
m6	5	1	1	1	2	1	2	1	1	1	1	1
9	1	1	1	1	1	5	1	2	1	2	1	1
maj9	1	1	1	1	1	1	1	1	1	2	1	1
m9	2	1	1	1	1	1	1	2	1	1	2	1
sus4	1	1	1	1	1	1	1	1	1	1	1	1

Table 1: Recognition of Single Chords

Denoising gives us a much cleaner amplitude spectrum as shown in Figure 1c.

There are still more significant elements left in the denoised amplitude spectrum than the number of chord notes. As stated before, that is due to the overtones. We can get rid of many overtone components by computing the Harmonic Product Spectrum as shown in Figure 1d. It can be seen that the high-order overtones are completely removed; only the fundamental frequencies and the 2nd overtones survive. The 2nd overtones don't really affect the overall accuracy because they will eventually be in the same pitch class bin as their fundamental frequencies. By using the M-th root of the HPS, the magnitude of the elements is well preserved. Now we can build the Improved Pitch Class Profile vector from the M-th root of the HPS. The vector is shown in Figure 1e. In this case, the IPCP does a perfect job: there are 3 significant values in the vector, corresponding to the 3 chord notes in the C chord, that is, C, E and G. In other cases, we may not be so lucky, and the non-chord notes have some non-zero values as well. Hopefully, the values of the chord notes are dominant, so we can still get the correct classification.

Finally, we can circularly shift the IPCP vector, and try to match it with the templates. The template for the Major chord can be viewed as the template for the C chord, as shown in Figure 1f. Computation of the score is as simple as doing the dot product of the IPCP vector and the weight vector (i.e. template). It should not be surprising that the score of the C chord (0.865895) is the highest among scores of all 192 chords. The significant values in the IPCP vector are all multiplied with positive weights, and we don't get any punishment for non-chord notes. Therefore, we can conclude that the chord in the input audio recording is the C chord.

4.2 Recognizing Single Chords

I also generate audio recordings of all the 192 chords considered in this project, and test the automatic chord recognition method on them. The audio recordings are all generated by Guitar Pro 6 with Real Sound Engine enabled, so they are very similar to real-world recordings. The durations are about 2 seconds.

The results are shown in Table 1. The number corresponding to each chord represents the rank of the score for the ground truth chord computed with each audio recording. If the rank is 1, that means the ground truth chord has the highest score among all 192 chords, and the chord recognition method classifies the audio recording correctly. If the rank is not exactly 1 but a quite small number, the result is probably acceptable, because the ground truth chord stands out of hundreds of chords, and can be picked out with minor human intervention. If we stick to the highest score, the overall accuracy is 80.21%; if multiple predictions are allowed, say, up to rank 3, the overall accuracy can be as high as 95.83%.

Due to the limited time, I only have the time to generate and test on one audio recording for each chord. Although we can sense the effectiveness of the proposed method to some extent, we cannot say much more about it based on such a small dataset, for example the influence of different chord types. However, I do

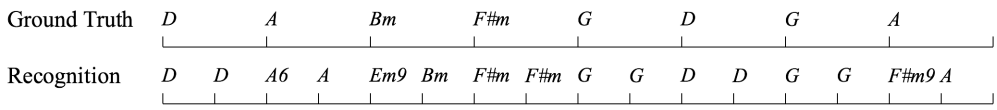


Figure 2: Recognition of a Chord Sequence

notice some problems during this experiment, the most important one of which is that multiple chords may have the same score. This problem gets quite severe if we only allow one prediction and there is a tie for the highest score. A typical example would be the Diminished Seventh chords, whose template (or the template for $Cdim7$ if we don't shift the IPCP vector) is $[\frac{1}{4}, -\frac{1}{4}, -\frac{1}{4}, \frac{1}{4}, -\frac{1}{4}, -\frac{1}{4}, \frac{1}{4}, -\frac{1}{4}, -\frac{1}{4}, \frac{1}{4}, -\frac{1}{4}, -\frac{1}{4}]$. We can easily see that there is a repetitive pattern $\frac{1}{4}, -\frac{1}{4}, -\frac{1}{4}$ in the template. The consequence is that every time we shift the IPCP vector by 3 and match it with this template, we get the same score. Or in other words, $Cdim7$, $D\#dim7$, $F\#dim7$ and $Adim7$ have the same score for any IPCP vector. This is an inherent difficulty in the task, and we can hardly do better without any context information about the chord. If we do have some prior knowledge, for example the key of the music, we might think that some of the 4 chords are more likely than the others, but this is out of the scope of this project.

4.3 Recognizing Chord Sequences

I try to apply the automatic chord recognition method to a real piece of music instead of just single chords. The piece of music I use is a 8 bars' Canon progression in D. The chords are played by strumming, and the strumming pattern is "D-DU-UDU". The BPM is 120, so that each bar takes 2 seconds. The audio recording is also generated by Guitar Pro.

The procedure of recognizing the chord sequence is straightforward. The audio recording is chopped into 1-second chunks, and each chuck is fed into the chord recognition method independently. The chunks should be bar-aligned, that is, every two chunks should correpond exactly to one bar. The result is shown in Figure 2. The overall accuracy is 81.25%, which is consistent with the previous results.

In fact, for this task, we can do much more than analyzing each chunk separately. For example, we can build an N-gram model to take the context of the chord into consideration. There are some very common chord progressions in today's pop music. Some chords are more likely to appear after a specific chord than thousands of other chords. Therefore, the context of the chord is very important when we deal with chord sequences. Other prior knowledge such as the key, the genre and even the author can be helpful as well if we are able to analyze a large corpus of music and build a knowledge base.

5 Conclusion

As an essential component in complex musical analysis systems, automatic chord recognition has gained more and more attention in the last few decades. In this paper, I propose a new automatic chord recognition method, formalize every stage in its pipeline, discuss some important implementation details, and show its effectiveness through experiments. The proposed method is based on a traditional scheme in [2], but enhances it with techniques including the Soft Thresholding denoising, the Improved Pitch Class Profile, and the circular shift and weighted sum based Template Matching.

There are still plenty of opportunities to improve the proposed method. They include refining some of the assumptions so that they are more practical, redesigning the templates so that they can distinguish different chords better, building an N-gram model to incorporate the context information of the chords, and so on. These opportunities can be regarded as the future work.

Acknowledgments

I would like to thank Prof. Robert Nowak and Prof. Rebecca Willett. Their instructions ultimately led to the creation of this work.

References

[1] Chafe, C., & Jaffe, D. (1986, April). Source separation and note identification in polyphonic music. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'86*. (Vol. 11, pp. 1289-1292). IEEE.

[2] Fujishima, T. (1999, October). Realtime chord recognition of musical sound: A system using common lisp music. In *Proc. ICMC* (Vol. 1999, pp. 464-467).

[3] Sheh, A., & Ellis, D. P. (2003). Chord segmentation and recognition using EM-trained hidden Markov models. *ISMIR 2003*, 185-191.

[4] Lee, K. (2006). Automatic chord recognition from audio using enhanced pitch class profile. In *Proc. of the International Computer Music Conference* (p. 26).

[5] Lee, K., & Slaney, M. (2006, October). Automatic Chord Recognition from Audio Using a HMM with Supervised Learning. In *ISMIR* (pp. 133-137).

[6] Cheng, H. T., Yang, Y. H., Lin, Y. C., Liao, I. B., & Chen, H. H. (2008, June). Automatic chord recognition for music classification and retrieval. In *Multimedia and Expo, 2008 IEEE International Conference on* (pp. 1505-1508). IEEE.

[7] Oudre, L., Grenier, Y., & Fvotte, C. (2009, October). Template-based Chord Recognition: Influence of the Chord Types. In *ISMIR* (pp. 153-158).

[8] Brigham, E. Oran (1988). The Fast Fourier Transform and Its Applications. *Prentice-Hall, Inc*.

[9] Cerna, M., & Harvey, A. F. (2000). The fundamentals of FFT-based signal analysis and measurement. *National Instruments, Junho*.

[10] ISO 16:1975 Acoustics – Standard tuning frequency (Standard musical pitch). *International Organization for Standardization*.

[11] Corbett, I. (2012). What data compression does to your music.

[12] Siddiqui, M. M. (1964). Statistical inference for Rayleigh distributions. *Journal of Research of the National Bureau of Standards, Sec. D*, 68(9).

[13] Chave, A. D., Thomson, D. J., & Ander, M. E. (1987). On the robust estimation of power spectra, coherences, and transfer functions. *J. geophys. Res*, 92(B1), 633-648.

[14] Noll, A. M. (1969, April). Pitch determination of human speech by the harmonic product spectrum, the harmonic sum spectrum, and a maximum likelihood estimate. In *Proceedings of the symposium on computer processing communications* (Vol. 779).