University of Passau
Faculty of Computer Science and Mathematics

Lehrstuhl für Mathematik mit Schwerpunkt Digitale Bildverarbeitung

Prof. Dr. Tomas Sauer

Bachelor Thesis

Design and Evaluation of a Simple Chord Detection Algorithm

Christoph Hausner

Date: 08/18/2014

Supervisor : Prof. Dr. Tomas Sauer
tomas.sauer@uni-passau.de
http://www.fim.uni-passau.de/?id=443

Contents

1. Introduction

Chord detection (also referred to as *chord recognition* or *chord estimation*) is the automated process of assigning musical chords to segments of a music piece. It finds application in a wide range of domains: as a tool for transcribing music to lead sheets, as an automated way to populate databases of music metadata which can then be searched by users, for building audio visualizations that react to harmony changes, or as part of other music information retrieval tasks such as song identification, cover identification, genre/mood classification or score following (the determination of the current position in a given sheet of music).

Most research work targets mainstream rock/pop songs because of their popularity and since they are usually based on a very clear sequence of relatively simple chords. On the other hand, highly polyphonic baroque music, for example, may not have an underlying chord progression at all, and jazz pieces can be composed of a wide variety of complex jazz chords whose detection is deemed to be not possible yet with current methods.

Still, much progress has been made in automatic chord recognition over the last decade. While newer and more sophisticated algorithms perform increasingly better at the task, the proposed algorithms also start to get considerably complex and computationally expensive. Even with improved algorithms there hasn't been a key breakthrough in recent years and the estimation quality of today's best performing algorithms still heavily depends on the type of input.

There are different reasons why recognizing chords is everything but trivial. The human brain processes sounds in complex ways that can be difficult to model in algorithms (the field of *psychoacoustics*). For instance, the perceived pitch of a single tone may not be represented in the frequency spectrum at all but result from the interference of multiple other frequencies (partials). [Loy, 2006, p. 157] Different subjects may also hear different chords in the same audio sample. [Burgoyne et al., 2011] The fact that multiple chords must be considered correct poses questions on how to train and evaluate algorithms.

This thesis aims to illustrate how mathematics, signal processing and music theory can all be leveraged and applied together to tackle the chord recognition problem. To that end, we will expand on the theory of Fourier analysis, how it can be used to extract important frequency information from audio signals, how frequency relates to pitch in music, and lastly how to infer chords from pitch information.

For the latter, a simple yet reasonably accurate chord detection algorithm is proposed, implemented and evaluated. The focus for the algorithm was put on a simple design that does not build on sophisticated machine-learning techniques or statistical processing and therefore does not require a separate learning phase.[1] Still, an effort was made to improve detection quality as far as possible while keeping the complexity of the algorithm low.

The rest of this work is structured as follows. In Chapter 2, an introduction to the Fourier transform, the Discrete Fourier Transform (DFT), the Short-Time Fourier Transform (STFT) and their applications in spectral analysis is given. Chapter 3 provides an overview of relevant music theory whose comprehension is beneficial for understanding the rest of this work. Chapter 4 contains an in-depth description of the various steps involved in our algorithm. In Chapter 5, details on how our algorithm fares against competing algorithms on a popular test set of songs are given, common detection issues are examined and the influence of different parameter choices on detection quality is analyzed. Lastly, Chapter 6 provides a summary of the methods and findings presented in this work. Appendix A contains additional information on our algorithm implementation and the contents of the supplied CD-ROM.

[1] While our algorithm does have a set of parameters that can be tuned to a specific training set, we provide default values that already yield good results in most cases.

2. Fourier analysis

Subject of this chapter is the mathematical background of the time-frequency transform that we employ in our algorithm and its application in spectrogram calculation.

The *Fourier transform* is named in honor of Jean Baptiste Joseph Fourier (1768–1830) who originally studied the related concept of Fourier series to describe heat transfer in solid objects. [Howard and Angus, 2009, p. 437] The Fourier transform decomposes a function into an infinite sum of sine and cosine waves, each with a specific amplitude, phase and frequency. The expression "time-frequency transform" originates from the fact that the transformed function no longer represents a signal that changes over time but encodes how energy is distributed in frequency space over the entire signal's length.[1] Today, Fourier analysis and its applications play a fundamental role in a wide range of disciplines, above else in signal processing, data compression, solving of partial differential equations and number theory. For this work, we will solely focus on the aspect of (audio) signal processing.

The remaining of this chapter is in large parts based on [Roads et al., 1997, pp. 35–53]. We begin with the Fourier transform defined for absolutely integrable functions and then cover discrete variants and applications thereof.

2.1. Fourier transform

Let f be an absolutely integrable real-valued signal[2], i.e. $f : \mathbb{R} \to \mathbb{R}$ with $\int_{\mathbb{R}} |f(t)| \, \mathrm{d}t < \infty$. The Fourier transform of f is defined as:

$$\widehat{f}(\xi) := \int_{\mathbb{R}} f(t) \cdot e^{-i\xi t} \mathrm{d}t \in \mathbb{C}, \quad \xi \in \mathbb{R}$$

In layman's terms, $\widehat{f}(\xi)$ can be interpreted as the amount of a particular frequency ξ in the signal (see Figures 2.1 and 2.2). This is well illustrated when considering the *inverse Fourier transform* which maps a frequency spectrum back to a time-variant signal:

$$f(t) = \frac{1}{2\pi} \cdot \int_{\mathbb{R}} \widehat{f}(\xi) \cdot e^{i\xi t} \mathrm{d}t, \quad t \in \mathbb{R}$$

[1] Apart from the Fourier transform, there exist many other time-frequency transforms with different properties and application areas. Most notably, wavelets.

[2] While the transform can be defined for complex functions it is usually not needed in audio processing as the input signal is real-valued.

Since $e^{i\xi t} = \cos\xi + i\sin\xi$ it is easily seen that $\widehat{f}(\xi)$ act as weights or intensities[3] of sine and cosine waves with frequency ξ.

Assuming the unit of t is seconds (s), ξ corresponds to $\frac{\xi}{2\pi}$ Hertz (s^{-1}). For $\xi < 0$ it can be shown that $\widehat{f}(\xi) = \overline{\widehat{f}(-\xi)}$, provided that f is real-valued.

It has to be noted that the Fourier transform is only invertible under certain conditions. For a large set of practically relevant functions (e.g. continuous L_1 functions whose Fourier transform is in L_1), however, the Fourier transform can be inverted. [Benson, 2006, p. 76]

While the continuous Fourier transform is powerful and convenient from a mathematical point of view, it is hard to be applied in practice since real-world signals have a finite length and are usually sampled at discrete points. A variant better suited in this regard is the *discrete Fourier transform*.

2.2. Discrete Fourier transform

The discrete Fourier transform (DFT) is a discrete variant of the Fourier transform that handles periodic time-discrete signals.

Let c be the sequence of an N-periodic signal in $\mathbb{R}$. The DFT of c is defined as:

$$\widehat{c}(m) := \sum_{k=0}^{N-1} c(k) \cdot e^{\frac{-2\pi i k m}{N}}, \quad m \in \{0, ..., N-1\}$$

Unlike the continuous Fourier transform, the DFT yields a discrete frequency spectrum of finite size. The frequencies from 0 up to the sampling rate of the input are discretized into a set of N equally spaced points (also called *frequency bins*).[4] If σ is the input sampling rate, $\widehat{c}(m)$ corresponds to frequency $\frac{m}{N} \cdot \sigma$.[5]

Just like the continuous Fourier transform, the DFT can be inverted by the *inverse discrete Fourier transform* (IDFT):

$$c(m) = \frac{1}{N} \cdot \sum_{k=0}^{N-1} \widehat{c}(k) \cdot e^{\frac{2\pi i k m}{N}}, \quad m \in \{0, ..., N-1\}$$

In contrast to the continuous case, the inversion of the DFT is always possible and not dependent on any further conditions.

[3] Strictly speaking, amplitudes and phase shifts.

[4] $\widehat{c}(0)$ corresponds to frequency zero, the component in the signal that does not vary over time, sometimes called DC component. It is equivalent to the mean of the signal.

[5] For real-valued signals, half of the DFT coefficients contain redundant information ($\widehat{c}(m) = \overline{\widehat{c}(N-m)}$) and $\frac{N}{2} - 1$ (corresponding to almost the Nyquist frequency) is therefore the highest coefficient of interest in practice.

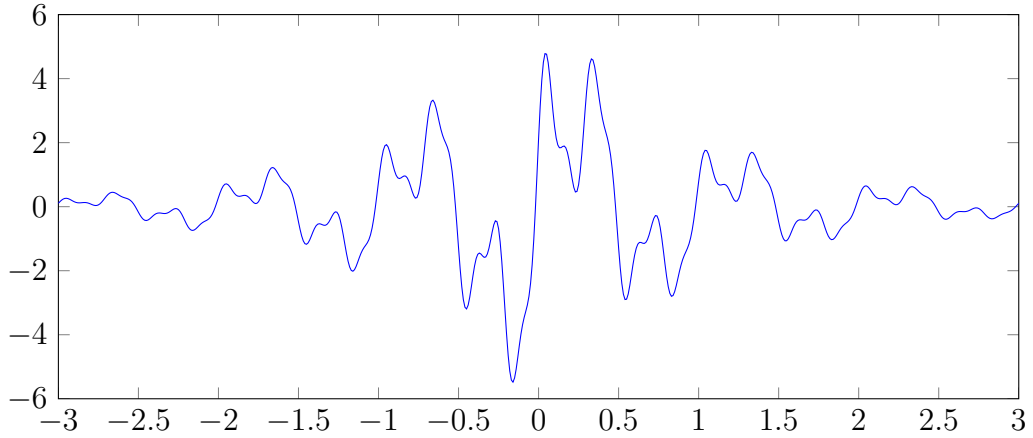


Figure 2.1.: Plot of $f(t) = (4\sin(2\pi t) + 3\sin(3 \cdot 2\pi t + \frac{\pi}{4}) + \sin(7 \cdot 2\pi t)) \cdot e^{-|t|}$. In this example, the sine terms are multiplied by an exponentially falling window function to obtain an absolutely integrable function.

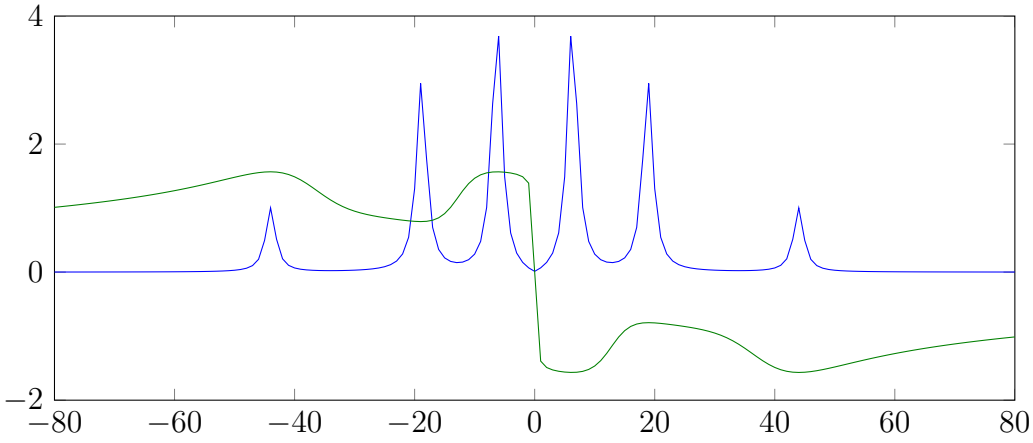


Figure 2.2.: Plot of magnitude (blue) and angle (green) of the complex $\widehat{f}(\xi)$. Note the magnitude peaks at $\xi = 2 \cdot 2\pi, 5 \cdot 2\pi, 11 \cdot 2\pi$ and the angular change near $\xi = 5 \cdot 2\pi$. The reason for the gradual increase and decline in magnitude around the three frequencies is the consequence of an effect called *spectral leakage* caused by the multiplication with $e^{-|t|}$ in f.

The DFT can be efficiently computed with complexity $\mathcal{O}(N \log N)$ using the *Fast Fourier Transform* (FFT) as opposed to a naive implementation requiring $\mathcal{O}(N^2)$. [Benson, 2006, p. 264] In 2000, the FFT made it on a list of the top ten most influential algorithms of the 21th century. [Dongarra and Sullivan, 2000]

2.3. Short-time Fourier transform

The Fourier transform only provides information on the frequency distribution over the entire signal's length. In practice, one is often interested in how the frequency distribution changes over time. One way to overcome this limitation is to not transform the entire signal at once but instead compute the transform for short consecutive time frames independently by means of a sliding window (see Figure 2.3). This approach is called the *short-time Fourier transform* (STFT).

The short-time Fourier transform can be defined as follows[6]:

$$\mathrm{STFT}(r, m) := \sum_{k=0}^{N-1} c(r \cdot I - \frac{N}{2} + k) \cdot w(\frac{2k}{N} - 1) \cdot e^{\frac{-2\pi i k m}{N}}$$

The STFT is a function of time step r and frequency bin m. The step size I denotes the distance between two consecutive time steps and is chosen smaller than the window size (= DFT size) N. It can also be specified in terms of an overlap factor O relative to the window size by $I = \frac{N}{O}$. w is a window function $w : \mathbb{R} \to \mathbb{R}$ that slides over the signal and selects and weights the data before the DFT is calculated.

In the simplest case, w is a rectangular pulse $\chi_{[-1,1]}$. But more sophisticated window functions exist that can provide better results, reducing the severity of artifacts caused by computing the DFT on segments from a non-periodic signal. On the other side, however, windowing itself introduces artifacts. The application of a window function is a simple multiplication of two signals and the Fourier transform of the product of two signals is equivalent to the convolution of the two signal's Fourier transforms. Hence, applying a window to a signal amounts to a convolution of the signal's spectrum with the spectrum of the window function and the artifacts that a window function introduces are a function of the window's frequency response.

Some of the more common types of window functions are rectangular, triangle, Hamming[7], Hann[7], Blackman[7] and Kaiser[8] windows.

Apart from choosing a suitable window function, there is a tradeoff to be made between time and frequency resolution when choosing a window size. Larger windows (and thereby DFT sizes) increase the frequency sampling density but, at the

[6] Analogously, one can define a short-time Fourier transform based on the continuous Fourier transform.

[7] constructions of one or more cosine terms

[8] constructed using Bessel functions

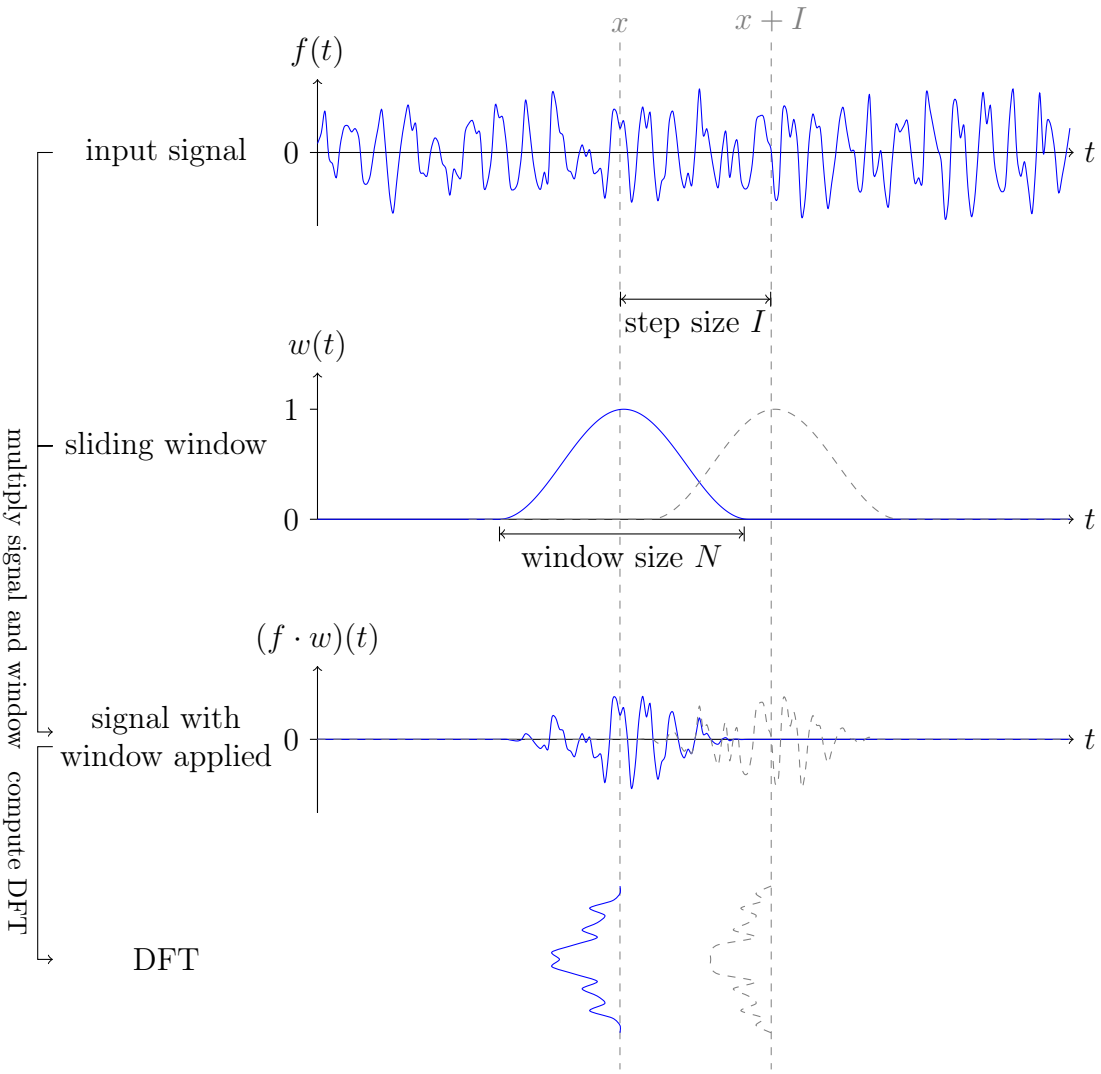


Figure 2.3.: Schematic illustration of the short-time Fourier transform evaluated for two points in time, x (blue) and $x + I$ (gray).

same time, entail a sparser sampling in the time-domain. Conversely, smaller window sizes can give good temporal accuracy but may have poor resolution in the frequency-domain. It bears resemblance to the Heisenberg uncertainty relation in this regard. [Benson, 2006, p. 73] In practice, one has to find a good balance between time and frequency resolution, depending on the specific task.

2.4. Magnitude and power spectrum

The spectrum calculated by the Fourier transform, DFT or STFT consists of complex numbers which, when written in polar coordinates $\widehat{f}(\xi) = \left|\widehat{f}(\xi)\right| \cdot e^{i \cdot \arg(\widehat{f}(\xi))}$, can

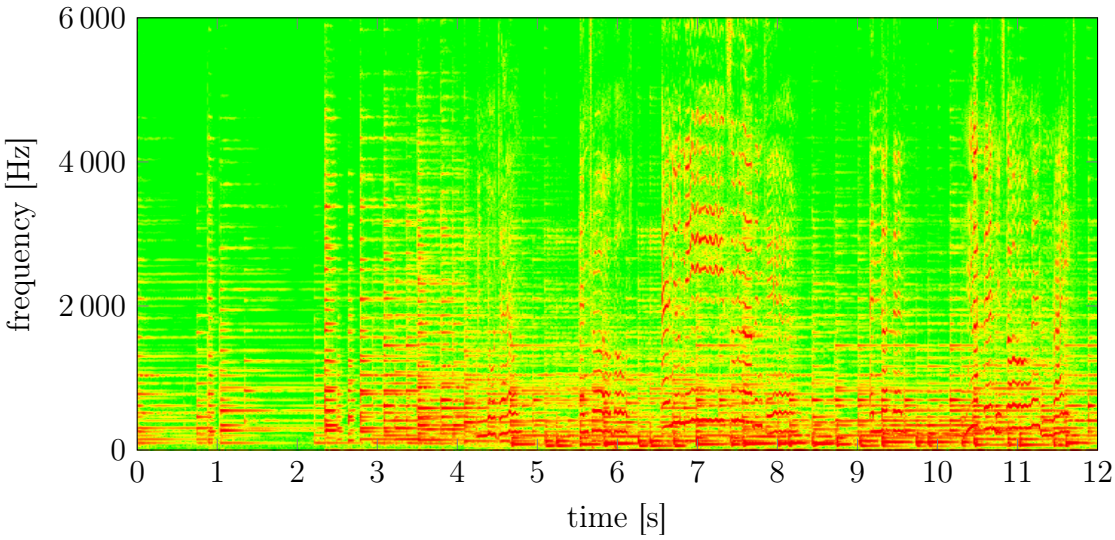


Figure 2.4.: Spectrogram using a linear frequency axis and logarithmic coloring. Audio sample is an excerpt of "You Have a Friend" – Carole King.

be interpreted as magnitude $\left|\widehat{f}(\xi)\right|$ and phase $\arg(\widehat{f}(\xi))$ of the sinusoids. For many applications the phase information is considered irrelevant. This may be related to the fact that the human ear is largely insensitive to phase as the perception of a tone does not considerably change when altering the phases of its frequency components. [Howard and Angus, 2009, p. 62] Consequently, one defines the *magnitude spectrum* to be $\left|\widehat{f}(\xi)\right|$ which no longer contains phase.

Related to the magnitude spectrum is the *power spectrum* which is expressed by the square of the magnitude: $\left|\widehat{f}(\xi)\right|^2$. "Power" here refers to the power in physics that is required to play a sound by making the air vibrate and which is proportional to the square of the amplitude. For our algorithm we will employ the magnitude spectrum only and will refer to it as "spectrum".

Magnitude and power spectra are usually visualized as two-dimensional diagrams called spectrograms that have a linear x-axis corresponding to time and a linear or logarithmic y-axis corresponding to frequency. Each point in the image is colored according to the amplitude at the respective frequency and time, often employing a logarithmic scale (such as decibels). Figure 2.4 shows an example of what a spectrogram typically looks like.

3. Music theory

What follows is a short dive into music theory that will be needed to fully understand the ideas in the next chapter. The reader is assumed to have a certain prior knowledge of concepts such as notes and intervals.

3.1. Notation

When speaking of notes we will use scientific pitch notation which consists of the single letter name of the note, followed by an octave number, e.g. C3, B4, F♯2. For modeling notes in algorithms and formulas a more suitable pitch representation is needed. For the purpose of this work, we use the corresponding MIDI[1] number to describe pitch. The MIDI number of A4 is defined to be 69 and for each half tone higher/lower the number increases/decreases by one. The pitch range of a piano in scientific pitch notation A0...C8 is therefore represented by MIDI notes 21...108.

3.2. Equal temperament

The theory of chords is best described in terms of notes and intervals but a chord estimation algorithm will work on frequencies. The mapping between notes and frequencies can happen in different ways, one of them is a system known as *equal temperament*. Practically all modern music uses equal temperament today and considering that it is also probably the simplest to model mathematically make it a good choice to use in chord recognition. "Equal" here refers to the fact that the frequency of every note and the half tone above it have a fixed ratio. The advantage of this is that music can then be played in different keys without sounding differently, a major disadvantage of earlier tuning systems. [Howard and Angus, 2009, p. 163]

Since 12 half tones make a full octave and an octave is characterized by a doubling in frequency, the fixed ratio follows to be: $\sqrt[12]{2} \approx 1.06$. We can calculate the frequency of any (MIDI) note n as follows:

$$\text{freq}_{f_{A4}}(n) := \sqrt[12]{2}^{n-69} \cdot f_{A4} = 2^{\frac{n-69}{12}} \cdot f_{A4}$$

[1] Musical Instrument Digital Interface, a technical standard for music accessories.

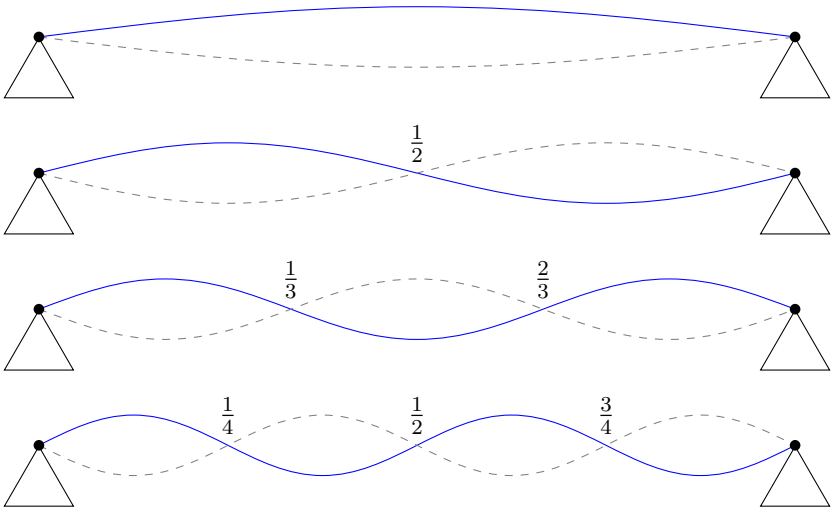


Figure 3.1.: From top to bottom: fundamental, 2nd, 3rd and 4th harmonic standing waves in a plucked string.

where f_{A4} denotes the frequency of A4 (also called *concert pitch*). Standard values for f_{A4} have changed over time and are currently around 440 Hz.

3.3. Harmonics

Blowing air through a wind instrument or plucking the string of a guitar produces a vibration that is perceived as sound. The pitch we hear depends on the frequency of the vibration which in turn is determined by the air volume in the instrument or the length of the string. The notion that the string vibrates at only one frequency, the *fundamental*, is a simplified model, however. In fact, there exist oscillations at integer multiples of the fundamental called *harmonics* or *partials* (see Figure 3.1). The n-th harmonic is a sine wave with a frequency of n times the fundamental. [Benson, 2006, p. 144] The fundamental matches in most cases the perceived pitch of a tone and is often the partial with the strongest intensity. [Loy, 2006, p. 29]

In general, the intensities of harmonics drop the higher the harmonic. The exact distribution over the harmonics, however, varies greatly between instruments and is, in fact, a substantial part of what defines the timbre of a sound. [Howard and Angus, 2009, p. 247]

Most natural instruments like woodwinds or stringed instruments only have partials at or close to integer multiples of the fundamental but some instruments like bells, drums and brass instruments can exhibit frequencies that are not a multiple of the fundamental. [Benson, 2006, p. 144] The sound they produce is said to be *inharmonic*. [Loy, 2006, p. 29]

3.4. Chords

Simply put, chords are multiple notes played at the same time by the same instrument. They can be characterized by the notes that are part of it, e.g. C-E-G denoting notes C, E and G played together. Alternatively, chords can be defined by a root note and the intervals (in half tones) from this root note to all other notes that make up the chord, e.g. C:1,4,7 specifying a C-E-G chord with C being the root note.

The majority of chords consist of three (triads) or four notes (tetrads) although there is no upper limit to the number of notes. If the root note is not the lowest note in a chord, the chord is called *inverted*. Inverted chords generally sound similar to their non-inverted counterparts since they are built out of the same notes, just in a different order.

By far the most common chords are simple *major* and *minor chords*. A major chord consists of the root note, the major third (4 half tones) above it and the perfect fifth (7 half tones) above it. Likewise, a minor chord consists of the root note, the minor third (3 half tones) above it and the perfect fifth above it. Major chords are often written out as a single capital letter indicating the root note of the chord, minor chords have an additional "m" appended. For example, C, Fm, A♭m express C major, F minor and A♭ minor chords, respectively. Chord notation sometimes includes a suffix indicating the note the bass plays, which may or may not be a note belonging to the chord, e.g. C/E or G/A.

Apart from the above definition of "chord", the term can also refer to a higher-level concept, namely that of *harmonies* or *changes*. In this sense, a chord is no longer understood as a single set of notes played at once but acts as a general guideline as to which notes should be played by an ensemble of instruments, valid for a specific number of beats or bars, until it is exchanged for another chord that is the new guideline. According to this guideline, the instruments that are responsible for harmonies play some or all of the notes that are part of the current chord. This is of course a very simplified model but should suffice here.

Segments in a music piece that cannot be attributed any chords are sometimes marked by *N.C.*, an abbreviation for "no chord".

When talking of detecting or estimating chords, it is this second definition that we are referring to, i.e. we aim to recognize the underlying chord structure that all played notes are based on.

4. The algorithm

4.1. Overview

Work on this algorithm was initially started without any prior knowledge of existing concepts and research works on the topic of chord recognition. It later turned out that many of the developed techniques already existed in similar form. This means that, even though we will refer to the concepts in our algorithm by established terms and with references to related work, they were to a large part developed independently.

At the core of the algorithm lies the concept of *chord templates*, first proposed by [Fujishima, 1999], to derive chord probabilities from pitch salience. We do not employ machine-learning techniques or statistical models like hidden Markov models, Bayesian inference or artificial neural networks that form the basis of many alternative proposals. This has the advantage that no separate learning phases are required, the algorithm is less susceptible to overfitting, easier to implement and computationally cheaper. The drawback of a simple design is that there are fewer opportunities to optimize and train the algorithm, which in general translates to poorer detection accuracy compared to more advanced techniques.

Our system is designed to be primarily used in non-real-time scenarios, i.e. the algorithm first reads in the data of an entire song and then outputs all chords. Real-time chord detection is considered challenging since the chords have to be estimated on the fly without information on the near-future. [Cho and Bello, 2009] have shown that accuracy greatly improves when providing algorithms a look-ahead of the data to come.

The set of chords our system estimates is restricted to the 24 major and minor chords (C, Cm, D♭, D♭m, ..., B, Bm) and the N.C. symbol (if no chord is deemed to be present). There are several reasons for this rather limited set. Firstly, these are by far the most common chords present in popular music (about 75% of all, see Section 5.2 for more details). Secondly, detection accuracy drops significantly the larger the set of chords to estimate is. Besides, it is often possible to describe more complex chords meaningfully by a simple major or minor chord, the one that shares root note and most of the intervals (e.g. Gm7 by Gm, Dmaj7 by D, etc.). The detection of passages where no chords are present is primarily there to allow meaningful comparisons with other algorithms that support these.

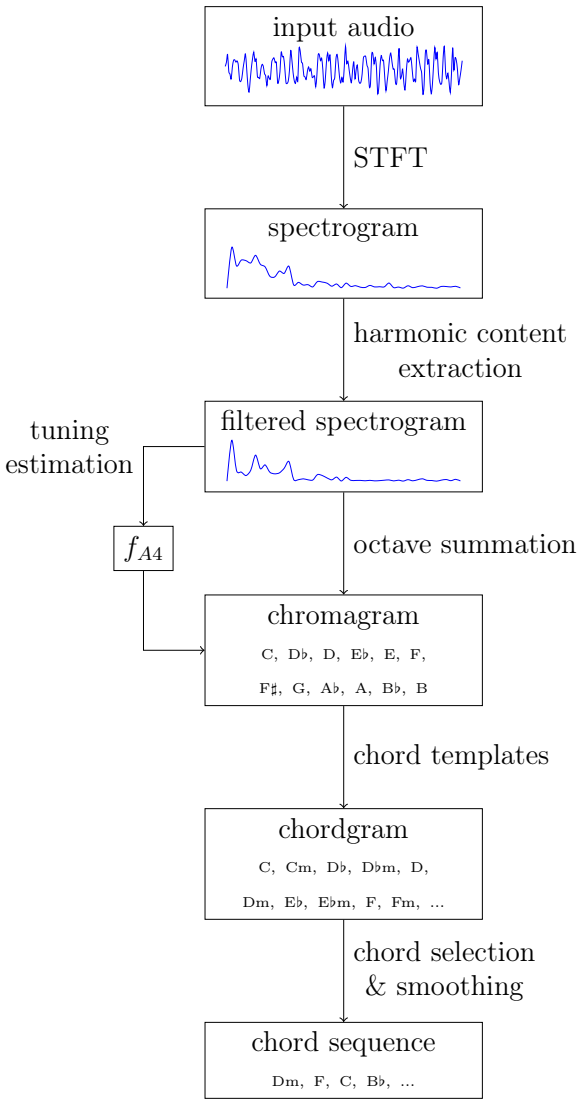


Figure 4.1.: Architecture of the proposed chord recognition system

The algorithm takes uncompressed mono PCM audio data as input and outputs estimated chords and their durations. The raw input data is highly redundant considering that we are only interested in the chords and there is no way of extracting that information directly from the PCM data. Instead, the idea is to transform the input data in a series of steps into progressively lower-dimensional representations, better suited for extracting chord information and discarding data irrelevant to the task.

Figure 4.1 illustrates the architecture of our chord recognition system. In the following, each of the steps will be described in detail.

4.2. Spectrogram calculation

This first step is practically self-explanatory. For simplicity we assume that the input audio data is sampled at $\sigma = 48\,000\,\text{Hz}$. For other sample rates the input could be resampled to 48 000 Hz before being processed or the parameters of the algorithm that depend on the sample rate (e.g. FFT window size) could be adjusted accordingly. The short-time Fourier transform of the input data is computed with window size $N = 16384$ and overlap factor $O = 8$ using the Blackman-Harris window:

$$S(r, m) := \text{STFT}(r, m),$$

$$r \in \{0, ..., L-1\}, m \in \{0, ..., B-1\}$$

Here, L denotes the number of samples on the time axis of S and B denotes the frequency bin corresponding to the Nyquist frequency.

4.3. Harmonic content extraction

In order to detect chords in the input audio it is essential to determine which notes are played. The spectrogram is already a good starting point for this task as it provides us access to the frequency distribution in the signal. To quantify the presence of a note at a time step r one may be inclined to take $S(r, m)$ as indicator where m is the bin that corresponds best to the note's frequency. This approach ignores the fact though, that instruments usually do not produce pure sine waves but exhibit several partials at multiples of the fundamental frequency that also appear as such in the spectrogram.

As an example, suppose a single piano note A4 is played and the tuning frequency is 440 Hz. The spectrogram will not only show a high magnitude at 440 Hz, the fundamental of the tone, but also peaks at 880 Hz and 1320 Hz, the 2nd and 3rd harmonic. Having the 2nd harmonic in the spectrogram might be acceptable from a chord detection point of view since it corresponds to the same tone as the fundamental, just one octave higher. The 3rd harmonic, however, is more problematic because it coincides very closely with the perfect fifth E6. As a result, chances are that the above method would see at least three notes A4, A5, E6 played even though it was really a single note.

Another issue that arises with the simple approach above is that a peak in the spectrogram may not necessarily be part of a played tone at all. Consider inharmonic sounds like clapping, drums or sound effects. These usually have a broad spectrum and may happen to have a frequency component at the frequency of a note that gets erroneously detected.

One way to circumvent these problems is to perform some kind of filtering on the spectrogram before interpreting its peaks as fundamentals. The method proposed

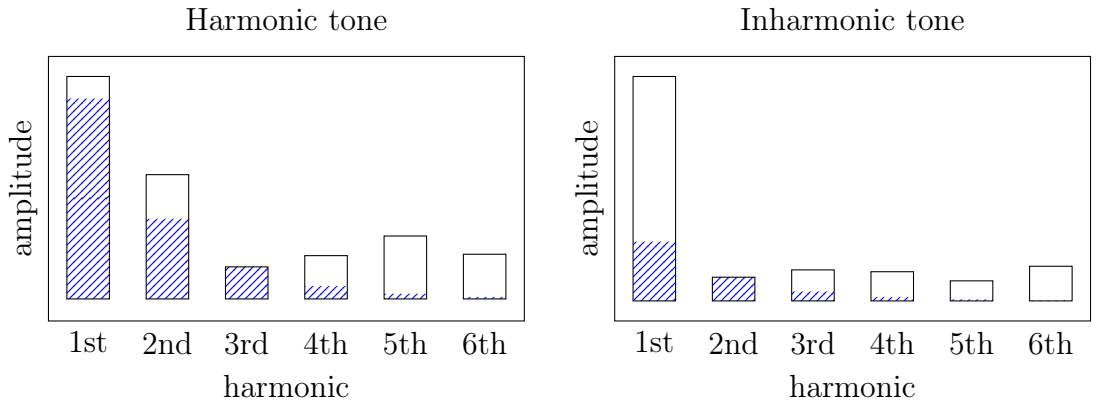


Figure 4.2.: Illustration of the principal behind the extraction of harmonic content. For two example spectrograms, the height of the bars indicate the amplitudes at multiples of some fundamental f. To find the harmonic content at frequency f, an exponentially falling curve is determined that has maximum height but still lies at or below the amplitudes. The value of this curve at the 1st harmonic is then taken as harmonic content.
Left: All frequencies of the first six harmonics are present. The harmonic content at f is not significantly lower than the unfiltered value.
Right: The spectrogram shows a high peak at frequency f but has little energy at the 2nd to 6th harmonics. The harmonic content at f is determined to be very low in amplitude.

here was designed to distinguish fundamentals from higher harmonics as well as to filter out inharmonic noise.

For modeling harmonics we adopt the model proposed by [Gómez, 2006], albeit with different parameters. The harmonics' amplitudes of every harmonic sound are assumed to all drop exponentially with the same decay rate ϵ:

$$A_n = A_1 \cdot \epsilon^{n-1}, \quad n \geq 2$$

where A_n is the amplitude of the n-th harmonic. For ϵ we suggest a value of 0.4.

The harmonic content extraction works by determining the maximum possible amplitude of an harmonic signal with the properties mentioned above, that could possibly be present in the spectrogram at a specific time and frequency (see Figure 4.2). In every case is the filtered value lower or equal to the unfiltered value.

The filtered spectrogram $F(r, m)$ is calculated as follows, checking the first 10 harmonics:

$$F(r, m) := \min \left\{ S(r, h \cdot m) \cdot \left(\frac{1}{\epsilon}\right)^{h-1} \mid h \in \{1, ..., 10\} \right\}$$

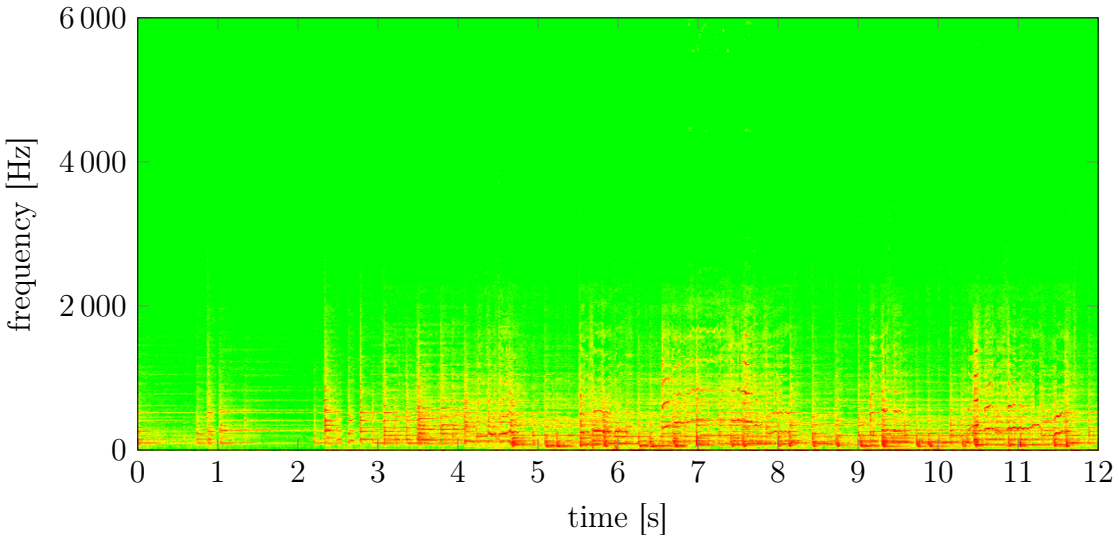


Figure 4.3.: Filtered version of the spectrogram from Figure 2.4. The filtering is by no means perfect as can be seen at around 7 seconds where upper harmonics are still visible.

An equivalent definition that better conveys the idea could be:
$F(r, m)$ is defined to be the maximum value X so that

$$\forall h \in \{1, ..., 10\} : \quad \underbrace{X \cdot \epsilon^{h-1}}_{\substack{\text{amplitudes} \\ \text{according to model}}} \quad \leq \quad \underbrace{S(r, h \cdot m)}_{\text{observed amplitudes}}$$

The reason why this method filters out inharmonic content is simple. If a peak in the spectrogram does not belong to the fundamental of an harmonic tone it is likely that at least one of the multiples of its frequency has a small amplitude. According to the definition of the filter this then leads to a reduction of the peak. Higher harmonics are filtered out due to the very same reason.

4.4. Tuning estimation

In order to map frequencies in the spectrogram to the pitch of musical notes we assume instruments are tempered according to equal temperament. We still need to know the concert pitch, i.e. the frequency of A4, denoted by f_{A4}. If the concert pitch is not an external input to the algorithm (e.g. set by the user in advance) it has to be estimated.[1]

[1]Practically, one could of course skip this step and just assume a standard tuning of 440 Hz for any song and still get reasonably good results in the majority of cases.

We are employing a very simple and efficient method to get a good approximation of f_{A4} which is accurate enough for our purpose. The main idea is the fact that we can expect significantly more energy in the spectrogram at frequencies that correspond to musical notes than at frequencies in-between (see Figure 4.4). This is particularly the case when the audio sample is comprised of mostly harmonic instruments and few inharmonic sounds. Locating these peaks in the frequency spectrum allows us to gain information about the tuning frequency simply by determining the distance that the peaks are shifted from the 440 Hz mark.

There is one caveat though, in that f_{A4} can theoretically be arbitrarily far off from today's standard 440 Hz. For instance, the traditional tuning for baroque music is located at around 415 Hz [Boyd and Butt, 2003] which corresponds to A♭4 when a tuning of 440 Hz is assumed. There is no way of deciding if a played tone is an A4 with $f_{A4} = 415\,\text{Hz}$ or an A♭4 with $f_{A4} = 440\,\text{Hz}$. In order to evade this ambiguity our algorithm always chooses the concert pitch that is closest to 440 Hz. For a very large percentage of test cases this is the right choice as the majority of songs today are tuned reasonably close to 440 Hz.

As a first step we build an average spectrum over all time steps from the (filtered) spectrogram:

$$\widetilde{F}(m) := \frac{1}{L} \cdot \sum_{r=0}^{L-1} F(r, m)$$

There are two reasons for this. Firstly, it is fair to assume that the tuning stays during the whole song constant, so we can confidently ignore temporal changes in the spectrum. Secondly, there may be parts in the song that contain no harmonic content (e.g. a drum solo) and thus can't be used for tuning estimation. By taking the average spectrum we can be sure to have all useful information included.

Next, we define a scoring function for quantifying how likely a possible tuning frequency f is. We take all notes between a lower bound N_l and an upper bound N_h, get their corresponding frequency bins under the assumption that f is the concert pitch, look up the amplitudes in $\widetilde{F}$ and sum the results.

$$\text{score}(f) := \sum_{n=N_l}^{N_h} \widetilde{F}(\text{bin}_f(n))$$

$\text{bin}_f(n)$ maps a MIDI note n to the nearest frequency bin in the spectrogram that corresponds to the note's frequency when taking f as tuning frequency. It can be defined as:[2]

$$\text{bin}_f(n) := \text{round}\left(\frac{\text{freq}_f(n)}{\frac{\sigma}{2}} \cdot B\right)$$

[2]Instead of rounding to the nearest frequency bin we use a slightly more sophisticated conversion using linear interpolation in our implementation.

We then estimate the tuning frequency to be the one in the range $[f_{A4,l}, f_{A4,h}]$ that yields the highest score:

$$f_{A4} := \underset{f \in [f_{A4,l}, f_{A4,h}]}{\arg\max} \ \text{score}(f)$$

$f_{A4,l}$ and $f_{A4,h}$ are set so that they span an interval of one half tone with $f_{A4,l}$ lying exactly between A♭4 and A4, and $f_{A4,h}$ between A4 and B♭4, assuming a concert pitch of 440 Hz:[3]

$$f_{A4,l} := 440 \cdot \sqrt[12]{2}^{\left(-\frac{1}{2}\right)} \approx 427.5\,[\text{Hz}]$$
$$f_{A4,h} := 440 \cdot \sqrt[12]{2}^{\frac{1}{2}} \approx 453\,[\text{Hz}]$$

In case the concert pitch lies below $f_{A4,l}$ or above $f_{A4,h}$ it will be falsely detected as the one frequency in the range that is a whole number of half tones higher or lower. Ultimately, this will lead to the root notes of all estimated chords being shifted one or more half tones up or down (e.g. Fm will be estimated as Em, B as B♭, etc.).

Implementations are expected to evaluate the scoring function for a discrete set of points in $[f_{A4,l}, f_{A4,h}]$, for example in uniform steps of 0.5 Hz, depending on the desired accuracy. For N_l and N_h we suggest the values 36 and 95, corresponding to notes C2 and B6, respectively. Figure 4.5 shows typical plots of the scoring function for different tunings.

4.5. Chromagram calculation

The next step in our system involves the computation of a so-called *chromagram*. These are, just like spectrograms, two-dimensional structures where one dimension represents time. While the columns of a spectrogram describe a frequency spectrum they are in the case of chromagrams 12-dimensional vectors called *chroma vectors* (first proposed by [Fujishima, 1999] as *Pitch Class Profiles*) whose elements correspond to the 12 pitch classes:[4]

$$\text{PITCHES} := \{C, D♭, D, E♭, E, F, F♯, G, A♭, A, B♭, B\}$$

Each detected note is assigned to one of these pitch classes (the one that describes its relative pitch ignoring octave position, e.g. F♯3 → F♯) and the elemens of chroma

[3] In other words, $f_{A4,l}$ is one half of a half tone lower than $A4$ and $f_{A4,h}$ one half of a half tone higher.

[4] There is no deeper meaning behind the choosing of D♭ over C♯, E♭ over D♯, etc. Since a twelve-tone equal temperament is assumed the different notations are enharmonically equivalent. Fujishima labels the pitch classes C, C♯, D, D♯, E, F, F♯, G, G♯, A, A♯, B while we prefer to use the spellings from the circle of fifths centered on C.

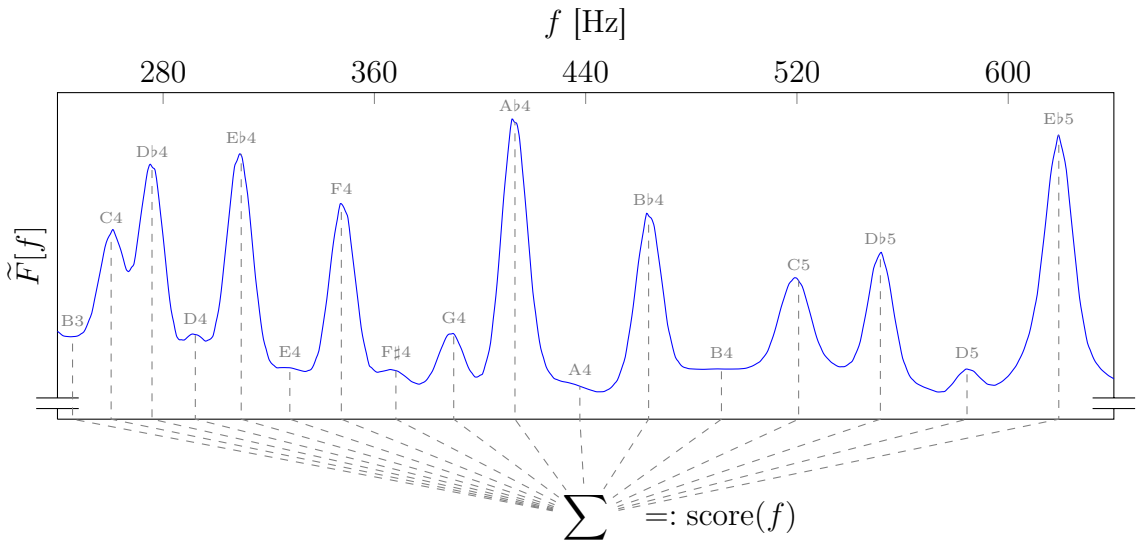


Figure 4.4.: Plot of the averaged spectrum $\widetilde{F}$ of "You've Got a Friend" – Carole King. The dashed lines indicate the frequencies whose amplitudes are summed up by the scoring function when assuming $f_{A4} = 438\,\text{Hz}$. They coincide very closely with the peaks in the averaged spectrum, leading to the scoring function reaching its maximum and suggesting that 438 Hz is the correct tuning.

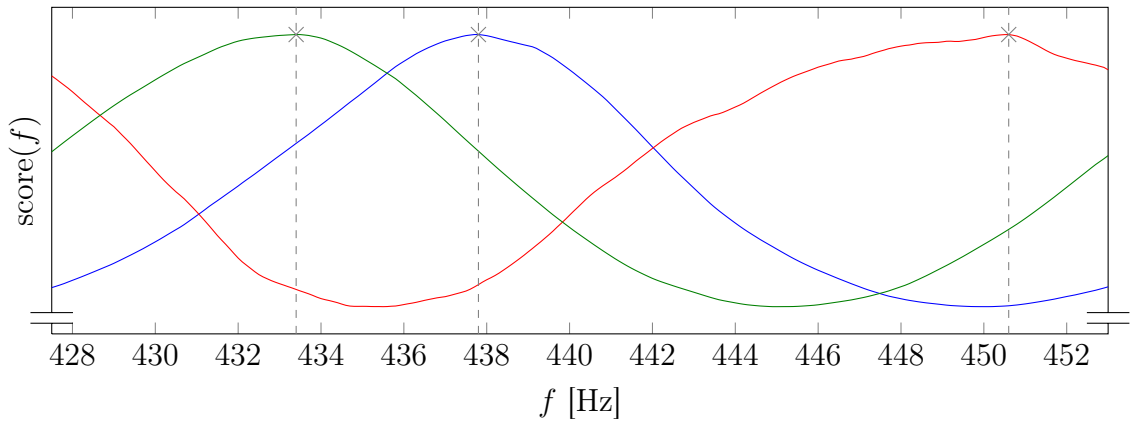


Figure 4.5.: Plot of the tuning estimation scoring function for three representative songs, "You've Got a Friend" – Carole King (blue), "California Dreamin'" – The Mamas & The Papas (red) and "Das Boot" – Klaus Doldinger (green). The three curves have been normalized to a common minimum and maximum for easier comparison. The vertical lines mark the estimated tunings which correspond to the maximum of the scoring function.

vectors can be interpreted as indicators of how strongly each of the 12 pitch classes is present in the input at a given time.

More formally, we can define the chromagram $C(r, a)$ as follows:

$$C(r, a) := \frac{1}{|M_a|} \cdot \sum_{m \in M_a} F(r, \text{bin}_{f_{A4}}(n)),$$

$$M_a = \{N_l, ..., N_h\} \cap (a + 12\mathbb{Z}), \quad a \in \{0, ..., 11\}$$

For convenience, we introduce a notation for the chroma vector at time step r:

$$C[r] = (C(r, 0), C(r, 1), ..., C(r, 11))$$

The chromagram computation can be thought of happening in two steps. First, all frequencies that correspond to a note between N_l and N_h are picked out from the filtered spectrogram (e.g. frequencies of C2, D♭2, D2, ..., B6). Then, their respective amplitudes are determined and averaged over all octaves (average of C2, C3, ..., C6, average of D♭2, D♭3, ..., D♭6, etc.). The result is a chroma vector with 12 elements for each time step r.

The motivation behind this is the following. The spectrogram samples a broad frequency range (0 up to the Nyquist frequency) relatively densely. This was required for determining the exact tuning frequency. Now that f_{A4} is known our focus shifts to only those frequencies that correspond to a note in some range $N_l ... N_h$ because notes are the building blocks for chords that we aim to detect later.

Discarding the octave information by taking the average of the amplitudes over all octaves is used as a convenient way to reduce the dimensionality of the data without loosing meaningful features that could be helpful for the classification of chords. It turns out that the absolute octave position that a tone is played at is relatively unimportant (provided that the bounds N_l and N_h are set reasonably). Still, it is worth noting that for more advanced chord estimation systems keeping octave information and actually using it to support chord detection has proven to be a good way to improve recognition quality and to expand the set of detectable chords.[5]

As with spectrograms, chromagrams are best visualized as two-dimensional images with the horizontal axis representing time, the vertical axis comprising the 12 pitch classes and the values of the chroma vector elements being color-coded. See Figure 4.6 for an example.

[5] One conceivable approach could be the detection of chord inversions by leveraging octave information. Another possibility would be to interpret notes in very low octaves as bass notes and enable the detection of chords with bass annotation (see Section 3.4)

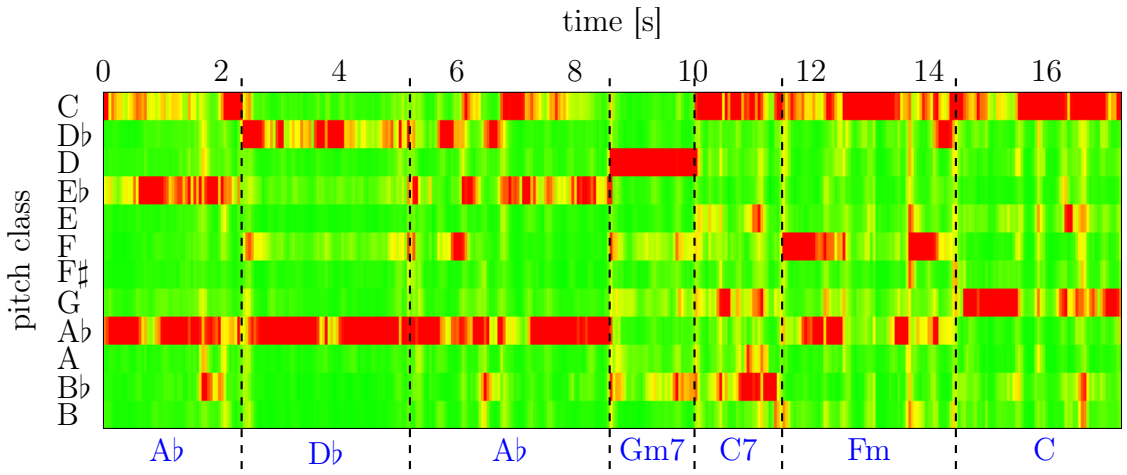


Figure 4.6.: Chromagram of the intro of "You Have a Friend" – Carole King. For better illustration, the individual chroma vectors have been normalized (green $\widehat{=}$ 0, red $\widehat{=}$ maximum value in the vector). Note that the strongest pitch classes match the annotated chords (blue) very well, e.g. for the duration of the first A♭ chord the pitch classes C, E♭ and A♭ are clearly the most prominent.

4.6. Chordgram calculation

Having reduced the input to a series of chroma vectors, the next step is to derive chord probabilities from them.[6] First, we define the set of chords that our algorithm can detect, all 24 major and minor chords and a symbol for "no chord":

$$\begin{aligned}\text{CHORDS} := \{&C, \quad D\flat, \quad D, \quad E\flat, \quad E, \quad F, \quad F\sharp, \quad G, \quad A\flat, \quad A, \quad B\flat, \quad B,\\ &Cm, \quad D\flat m, \; Dm, \; E\flat m, \; Em, \; Fm, \; F\sharp m, \; Gm, \; A\flat m, \; Am, \; B\flat m, \; Bm,\\ &N.C.\}\end{aligned}$$

For each chroma vector in the chromagram and for each of these detectable chords a score is computed that expresses how likely it is that the chord is correct.

In order to compute this score we will make use of the concept of *chord templates* first described by [Fujishima, 1999]. The general idea behind chord templates is that, assuming a specific chord is played, the chroma vectors point into a distinct direction that can be captured in a chord template which in turn can then be used for recognizing this specific chord. In other words, the distribution of the energy in the chroma vectors over their 12 elements is indicative of the chord being played and by comparing it with the typical distributions of all supported chords, the correct chord can be estimated.

[6] "Probability" is not to be understood in a strict mathematical sense but is meant as a general numeric score indicating the likelihood of a chord.

Templates are effectively 12-dimensional vectors, just like chroma vectors, and they also refer to the same 12 pitch classes. It is not necessary to define a separate template for each detectable chord. Defining it once for both chord types (major and minor) and one root (e.g. C) suffices, the template for all other roots can then be obtained by simply rotating the elements in the vector (e.g. to get the template for F♯m the template for Cm is rotated right by 6 elements). We will denote templates by τ_c where $c \in$ CHORDS is one of the detectable chords.

The most basic templates are those that have been proposed in [Fujishima, 1999]. They are binary templates consisting of entries 1 and 0, depending on whether the corresponding note is part of the chord or not:

τ_C =

C	D♭	D	E♭	E	F	F♯	G	A♭	A	B♭	B
1,	0,	0,	0,	1,	0,	0,	1,	0,	0,	0,	0

τ_{Cm} =

C	D♭	D	E♭	E	F	F♯	G	A♭	A	B♭	B
1,	0,	0,	1,	0,	0,	0,	1,	0,	0,	0,	0

The ones refer to the notes C, E, G and C, E♭, G, respectively, the pitches that make up C major and C minor chords.

For our algorithm we are proposing the following weights which give us significantly better results than the binary templates (see Section 5.4 for more details):

τ_C =

C	D♭	D	E♭	E	F	F♯	G	A♭	A	B♭	B
1,	0,	0.3,	0,	1,	0,	0,	1,	0,	0,	0.3,	0.1

τ_{Cm} =

C	D♭	D	E♭	E	F	F♯	G	A♭	A	B♭	B
1,	0,	0.3,	1,	0,	0,	0,	1,	0,	0,	0.3,	0

The idea is to improve detection in cases where not only simple major or minor triads are played but more complex chords. Of these, most common are chords having an additional minor seventh (e.g. C7 or Cm7) and/or ninth present. In the case of the root C these are the notes B♭ and D. The corresponding elements in the template are set to 0.3, indicating that these notes may be present but are of lower importance than the main triad with weighting 1. In addition to this adjustment, the template for the major chords has the entry corresponding to the major seventh set to 0.1. This slight change aims to model the fact that major sevenths appear far more frequently in major chords (e.g. Cmaj7) than in minor chords (e.g. Cminmaj7).

It took admittedly several attempts to obtain a template that performs significantly better than the binary one, and the necessary adjustments were found to a large extent through trial-and-error.[7]

For the detection of passages with no chords (N.C.) a template is employed that weights all pitch classes equally:

[7] One early attempt was to add non-zero weights for all notes of the diatonic major/minor scale in the templates, although with only mediocre results.

$$\tau_{N.C.} = ($$

C	D♭	D	E♭	E	F	F♯	G	A♭	A	B♭	B
1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1,	1

$$)$$

The absence of any harmonic structure is expected to lead to a uniform energy distribution over the 12 pitch classes and the above template mirrors this idea.

In order to assess how well a chroma vector matches a chord template, we determine the cosine similarity between the two. The cosine similarity is equivalent to the cosine of the angle between two vectors and is defined as follows:

$$\text{cossim}(u, v) := \frac{\langle u, v \rangle}{||u||_2 \cdot ||v||_2} \quad \text{where} \quad u, v \in \mathbb{R}^n, \quad u, v \neq 0$$

For non-negative u and v the cosine similarity takes on values in $[0, 1]$, one indicating an identical direction of u and v and zero indicating orthogonality.

Note that the cosine similarity is undefined if u or v are zero. Chord templates are guaranteed to be non-zero but chroma vectors could be zero in the case of complete silence in the input audio. To accommodate, we can just fallback to the value zero as similarity index in that case.

For more control on the sensitivity of the templates an additional weighting mechanism is introduced where each chord class (major, minor and N.C.) can be assigned different weights. This has proven to be necessary especially for the N.C. template.

$$\omega_c = \begin{cases} 1 & \text{if } c \text{ is major chord} \\ \omega_{minor} & \text{if } c \text{ is minor chord} \\ \omega_{N.C.} & \text{if } c = N.C. \end{cases}$$

With $\omega_{minor} < 1$ minor chords are penalized and with $\omega_{minor} > 1$ they are given preference. The same applies to the $\omega_{N.C.}$ weighting. As default values $\omega_{minor} = 1$ and $\omega_{N.C.} = 0.7$ are suggested.

Putting all together and computing the likelihood of all chords for all chroma vectors gives us what we will call a *chordgram*. The term has been adopted from [Papadopoulos and Peeters, 2009] and is supposed to hint at its close resemblance to the chromagram, the difference being that energy is no longer distributed over a set of pitch classes but a set of chords. The chordgram is defined as follows:

$$H(r, c) = \begin{cases} \text{cossim}(C[r], \tau_c) \cdot \omega_c & \text{if } C[r] \neq 0 \\ 0 & \text{if } C[r] = 0 \end{cases},$$

$$c \in \text{CHORDS}$$

Chordgrams can be visualized analogously to chromagrams, see Figure 4.7 for an example.

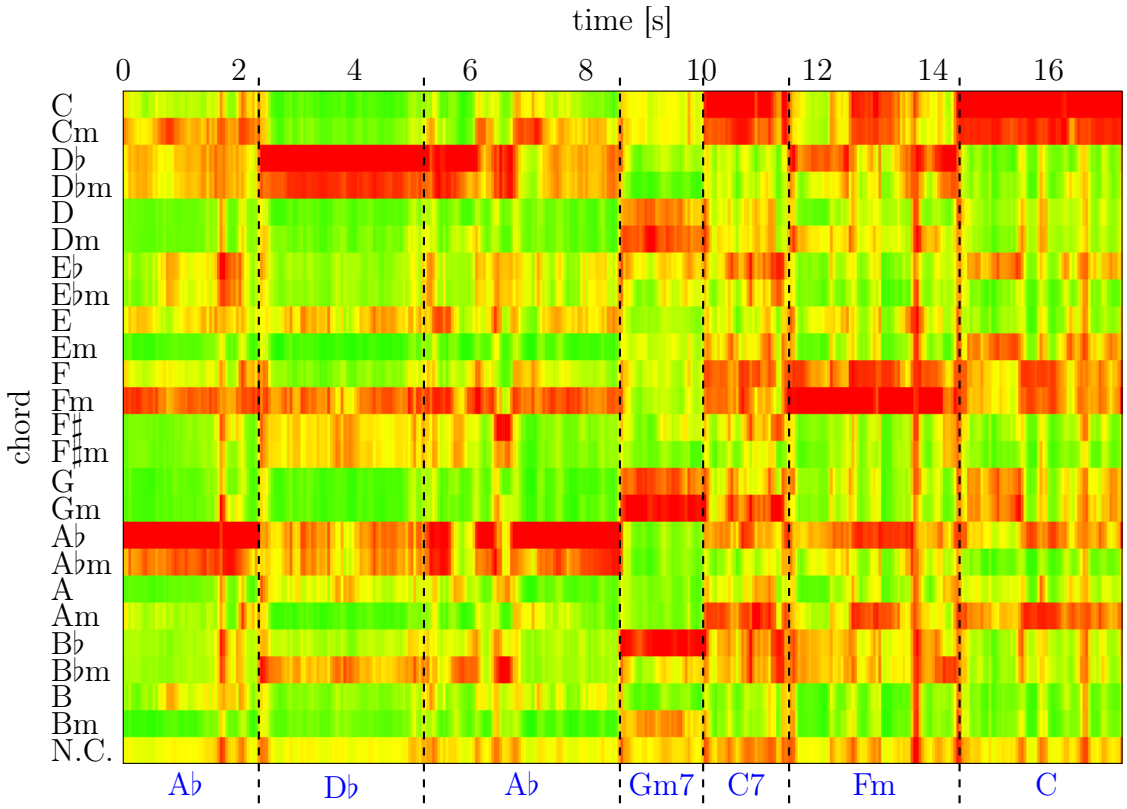


Figure 4.7.: Chordgram of the intro of "You Have a Friend" – Carole King. For better illustration, the individual probability vectors have been normalized (green $\widehat{=}$ 0, red $\widehat{=}$ maximum value in the vector). For the most part, the chords with the highest probability match the annotated correct chords (blue), the exception being the Gm7 chord where Gm and B♭ received nearly identical scores. Gm is the preferred estimation here because it retains the root of Gm7.

4.7. Chord sequence estimation

The remaining step in our system is to estimate a chord sequence (chords and their durations) from the chordgram. For this, for each time step the chord with the highest probability in the chordgram is determined:

$$R(r) = \underset{c \in \text{CHORDS}}{\arg\max}\ H(r, c)$$

The chord sequence obtained by this method is very fragmented because of noise in the chord scoring function (see Figure 4.8). This makes the chord sequence R inappropriate for many use cases and some form of smoothing is needed if a less noisy output is required. Smoothing not only makes the algorithm output better suited for further processing or presenting to a user, it also significantly improves the detection quality (in terms of the quality metric defined later in Section 5.1).

For smoothing, we apply a mode filter, a non-linear filter similar to the median filter. The mode filter works by computing the mode of all values in a neighborhood to determine the filtered value. The mode of a sequence of values is defined to be the value that appears most frequently in it.[8] Thus, the smoothed chord sequence R' can be written as:

$$R'(r) = \text{mode}\left\{R(r - \frac{w}{2}), ..., R(r + \frac{w}{2})\right\}$$

Here, w denotes the filter size, i.e. the width of the window used for computing the mode. Larger filter sizes translate to stronger smoothing and are to be preferred when the input song only contains slow-changing chords. Likewise, shorter filters preserve more details in the chord estimations and are appropriate for music with quickly-changing chords. We have found a value of $w = 30$ to be a good compromise that works well for most songs.

Applying the filter once may not eliminate all fluctuations in the estimated chord sequence. We obtained good results by performing the smoothing repeatedly a number of times (e.g. up to 10 times). While this still does not guarantee all chord changes to be some number of time steps apart, we consider it sufficient for most use cases. If needed, all chords with a duration lower than some threshold could be removed in a subsequent step.

With the mode filter having been applied a number of times, the resulting chord sequence is the final output of the algorithm. Figure 4.9 shows an example for the algorithm output.

[8] Note that the mode is not necessarily unique since two or more values may appear with the same counts in the sequence. In our case this has little practical consequence, though, and implementations are free to choose any of the candidates arbitrarily.

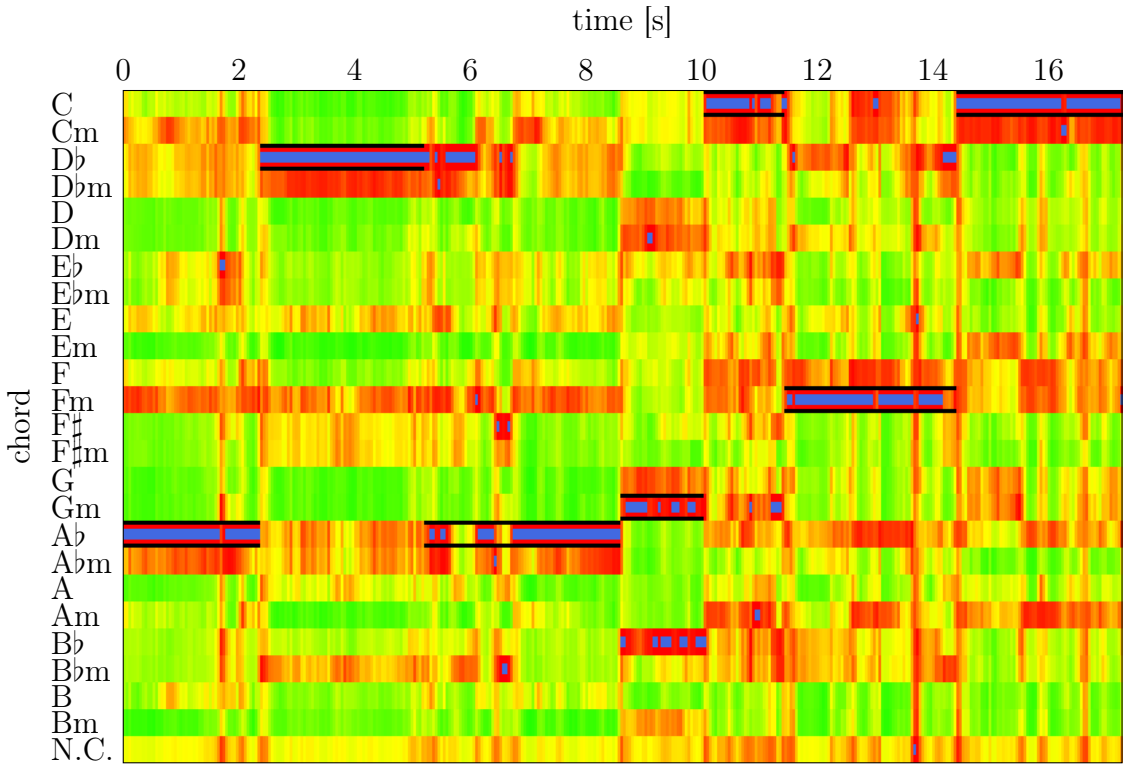


Figure 4.8.: Chordgram of the intro of "You Have a Friend" – Carole King. The chords with the highest probability have been marked in blue. Note that the resulting chord sequence is in some places heavily fragmented. The ground truth is indicated by two black lines.

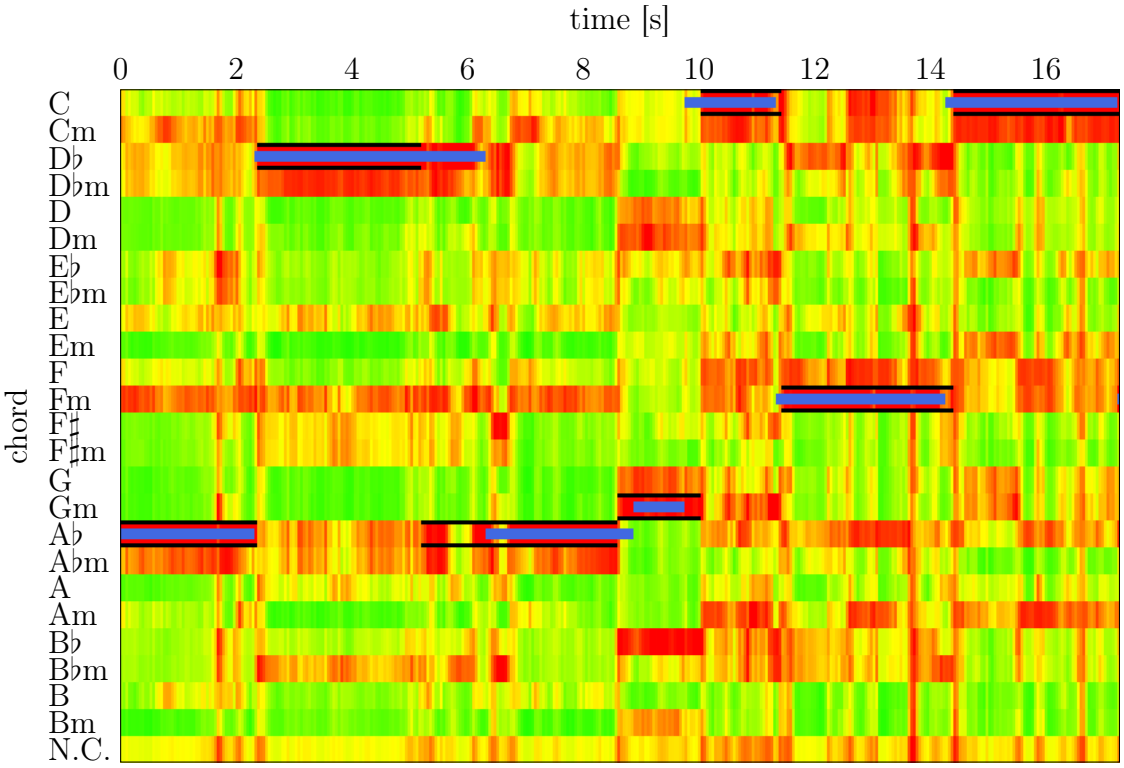


Figure 4.9.: Chordgram of the intro of "You Have a Friend" – Carole King. Overlaid in blue is the final estimated chord sequence. The ground truth is indicated by two black lines.

5. Evaluation

MIREX (Music Information Retrieval Evaluation eXchange)[1] is a yearly event run by the University of Illinois at Urbana-Champaign (UIUC) since 2005 that performs large-scale evaluations of community submissions in a number of disciplines, all in the field of audio and music processing (among others: genre/mood classification, beat tracking, melody extraction, structure segmentation, cover song identification and query-by-singing/tapping). Of special interest for us is the Audio Chord Estimation task which has been part of MIREX since 2008. Since its inception, MIREX has quickly turned into the de-facto standard framework for benchmarking chord recognition methods and has seen 10–20 submissions every year from international research groups.

In order to evaluate the detection performance of our chord recognition system and compare it to MIREX submissions of recent years, we created an implementation of it as part of this thesis (see Appendix A for more information). This chapter shall provide in-depth results of this evaluation, point out limitations and possible approaches, and examine the significance and influence of most of the parameters in the algorithm.

5.1. Measuring detection quality

Being able to quantitatively measure how well an algorithm detects chords is essential when trying to understand the effect of different parameter choices on the final detection quality. One can then tune the parameters so that the algorithm reaches the best possible performance in terms of the quality metric and a training set. It also allows an objective comparison of different algorithms and methods.

For measuring the error in a chord detection it is required to have access to the correct chords, also called ground truth. Training and evaluation is best performed on an extensive test set ideally covering a variety of artists and genres in order to gain a statistically significant result that is universally valid.

Compiling a training set from the ground up can be very time-consuming since the correct chords for each song have to be determined by someone who has the ability to recognize them by ear. This may be the reason why the number of publicly

[1] `http://www.music-ir.org/mirex/wiki/MIREX_HOME`

available chord annotation sets is very low, unfortunately. Moreover, the songs they are based on are often copyrighted, hard to obtain or very specific in their genre. The transcriptions vary in quality and accuracy, and may obviously contain human errors.

The following is a list of all public chord annotation datasets we are aware of:

- Beatles dataset[2]
  annotations for 180 songs from all 12 released Beatles albums by Christopher Harte
- Queen and Zweieck dataset[2]
  annotations for 20 Queen songs and 18 songs of the Zweieck album Zwielicht by Matthias Mauch
- McGill Billboard dataset[3]
  annotations for more than 1000 songs taken from the Billboard magazine's Hot 100 compilations from 1958 to 1991 by Ashley Burgoyne
- Carole King dataset[2]
  annotations for the Tapestry album by Carole King comprising 7 songs
- RWC Music Database: Popular Music dataset[4]
  annotations for 100 royalty-free songs mostly in the style of Japanese pop music

Chord detection quality is usually measured in terms of *overlap ratio* (OR), *average overlap ratio* (AOR) and *weighted average overlap ratio* (WAOR):

$$OR = \frac{\text{duration of correctly identified chords}}{\text{duration of all identifiable chords}}$$

$$AOR = \frac{\sum_{i=1}^{N_{songs}} OR_i}{N_{songs}}, \quad WAOR = \frac{\sum_{i=1}^{N_{songs}} OR_i \cdot L_i}{\sum_{i=1}^{N_{songs}} L_i}$$

where N_{songs} is the number of songs and L_i are the respective song lengths. [Glazyrin, 2012,Glazyrin, 2013,Harte, 2010] OR is a score between 0 and 1 measuring the detection quality of one single song. It can be thought of as the fraction of song duration where chords were estimated correctly. AOR and WAOR both are indicators of the average detection quality over a set of songs. AOR is the average OR over all songs while the WAOR score is the average OR weighted by the lengths of the songs. WAOR is considered by many to be a better estimate of the detection performance because very short songs do not bias the end result too much.

The question when an estimated chord counts as correct or not needs some further consideration. The main challenge is that chord estimation algorithms generally only

[2] `http://isophonics.net/content/reference-annotations`
[3] `http://ddmal.music.mcgill.ca/billboard`
[4] `https://staff.aist.go.jp/m.goto/RWC-MDB/AIST-Annotation/`

support a subset of all the chord types that appear in the ground truth. Chord annotation datasets are typically using very detailed and specific chord annotations in order to be as precise as possible but for chord estimation, algorithms are restricted deliberately to a small subset of chord types (such as in our case to pure major and minor triads only) for simplicity and detection quality reasons. It follows that there has to be some mapping of complex chords to simpler ones that are supported by the estimation algorithm in order to evaluate the performance of an algorithm on a test set. Complex chords may also be omitted completely from the evaluation, particularly when chords cannot be mapped uniquely to one of the simpler chords in any reasonable way (e.g. Dsus4 consisting of notes D, G and A neither fits into the major nor minor chord class). [Pauwels and Peeters, 2013]

Since the goal is to compare our algorithm performance with the participants of the MIREX contest, we chose to use the metric that contestants were ranked with at MIREX 2009. In 2010 and later years, the metric used in the MIREX competition was revised and optimized for evaluations on larger chord dictionaries since the participating algorithms started to support a growing number of different chord types. With our algorithm only supporting major and minor chords, we think the MIREX 2009 metric remains the appropriate choice for this evaluation, even if it makes comparisons with contestants of recent years harder.

The MIREX 2009 scoring works as follows. Chords are mapped to either a major or minor chord, depending on which one better "fits". Chords that do not fit well to either of the two are defined to be included in the major chord group.[5] The following table gives a more precise definition of the mapping used: [Harte, 2010]

mapping	**annotations**
maj	maj, aug, maj7, 7, maj6, 9, maj9, sus4, others
min	min, min7, minmaj7, min6, min9, dim, dim7, hdim7, sus2

Note that AOR and WAOR may be easy to evaluate metrics but are far from being without issues, one being that they only measure the average time that chords were identified correctly but do not give information about the structure of the estimation. For instance, an algorithm may provide overall good chord predictions but miss chords for short time periods and thereby produce a highly fragmented output (as in Figure 4.8). In order to measure how well the segmentation of the estimated chords matches that of the target chords, alternative metrics have been proposed. Noteworthy is the segmentation score described by [Mauch and Dixon, 2010] that is based on directional hamming divergence, a concept originating from the field of image segmentation. [Abdallah et al., 2005] Unfortunately, segmentation

[5] This rule has been criticized by some as it leads to a bias towards the major chords. A better solution could have been to exclude chords that don't fit well into one of the two groups completely from evaluation.

submission	main features	WAOR (%)
our result	**templates[6], harmonics[7]**	**72.4**
CH[8]	templates	57.6
DE[9]	HMM[10], beat-detection	69.7
KO1[11]	HMM, language modeling	69.7
KO2		70.8
MD[12]	DBN[13], beat-detection, structural segmentation	71.2
OGF1[14]	templates, harmonics	70.6
OGF2		71.1
PP[15]	HMM, beat-detection	67.3
PVM1[16]	HMM, harmonics	68.2
PVM2	templates, harmonics	65.4
RRHS1[17]	rule-based, symbolic music	62.9
RRHS2		54.5
RRHS3		53.8
RUSUSL*[18]	HMM, harmonics	70.1
WEJ1*[19]	structured SVM[20]	70.4
WEJ2*		72.3
WEJ3*		72.3
WEJ4*		74.2

Table 5.1.: Results from all groups participating in the MIREX 2009 chord recognition task including the performance of our algorithm on the same test set. The submission names are the identifiers of the various participating groups that were assigned by the MIREX organizers, numbered identifiers indicate multiple submissions from the same group. Entries marked with an asterisk (*) were train-test systems. [MUSIC IR, 2009]

scores have not seen an as wide adoption in algorithm comparisons as would be desirable.

5.2. MIREX dataset results

For parameter optimization, evaluation and comparison of our algorithm we use the exact same dataset that came to use in MIREX 2009: a combination of parts of the Beatles, Queen and Zweieck datasets making up a total of 206 songs.

[6] matching of chord templates/pitch class profiles as main mechanism
[7] any form of filtering/special treatment of harmonics
[8] [Harte and Sandler, 2009]
[9] [Ellis, 2009]
[10] hidden Markov model, probabilistic model

Table 5.1 lists the results of all participants in the MIREX 2009 contest in terms of WAOR including the result we obtained with our algorithm. Entries in the table marked by an asterisk (*) are systems that were explicitly trained and then tested on the dataset as part of the evaluation. All other submissions were pre-trained systems, i.e. any training necessary was performed in advance on an independent training set as the evaluation test set was not publicly available at that point. Consequently, direct comparisons between pre-trained and train-test systems should be drawn with care as train-test algorithms are expected to have a certain advantage here.

Since our algorithm has been optimized using the evaluation test set it would rather belong to the train-test group. However, considering that the number of parameters that were optimized is rather low and assuming that their optimal values are relatively independent from the training set, we think that a comparison with pre-trained systems is fair.

In terms of WAOR, our algorithm performs very competitively among the contenders on this specific test set, with only one submission reaching a higher score than us. Although the complexity of the various systems varies it is hard to see any correlation thereof in the scores. In particular, our proposal and the submissions by Oudre et al. (OGF1 and OGF2) use a relatively simple approach and yet reach very good scores. We conclude that some of the other systems either did not unleash the full potential of their models or are generally harder to train and optimize.

Figure 5.1 provides a more detailed view of our algorithm's performance on individual songs. Almost all songs get scores between 50% and 90% with only 8 songs getting scores lower than 50% and 4 songs achieving scores higher than 90%. The top score achieved is 91.3%, the lowest score being 14.5%.

Reaching scores close to 100% is very hard as some of the missing percentage points always stem from inaccuracies in the ground truth or not completely identical chord transition times. This is why scores above 80% may already be considered as very good and probably suffice for most applications. On the other hand, estimations with overlap ratios below 50% are relatively useless as these scores are already reached by very rudimentary estimations (e.g. a piece following the classical Blues form in C[21] may be detected as one single C chord and have an overlap ratio of 50%).

[11] [Khadkevich and Omologo, 2009]
[12] [Mauch et al., 2009]
[13] dynamic Bayesian network, probabilistic model
[14] [Oudre et al., 2009]
[15] [Papadopoulos and Peeters, 2011]
[16] [Pauwels et al., 2009]
[17] [Rocher et al., 2009]
[18] [Reed et al., 2009]
[19] [Weller et al., 2009]
[20] support vector machine, supervised learning model
[21] four bars C, two bars F, two bars C, two bars G, two bars C

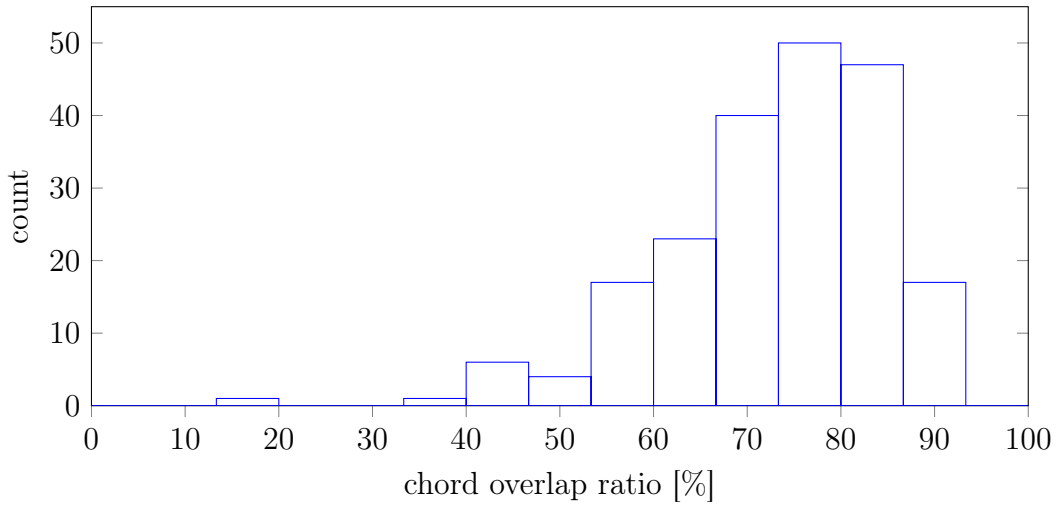


Figure 5.1.: Histogram showing how the individual scores of all songs in the MIREX 2009 dataset are distributed.

When looking at the songs that receive very low scores, the reasons for the bad performance on them are in most cases clearly evident even though they differ from case to case. These are primarily songs that do not have any instruments that consistently play chords, contain lots of noise, or have extensive sections of no-chords (N.C.). The remarkably low score of 14.5% of one song (The End – Beatles), however, turns out to be due to a failed tuning estimation. With the correct tuning, an overlap ratio of 73.9% would be obtained for this particular song.

A breakdown of the detection ratio depending on different chord types is given in Table 5.2. It can give insights into which chord classes are most problematic and may provide clues on how to further optimize detection rates. It has to be noted that the scores of some of the more infrequent chords are be based on a very limited number of songs in the test set and should therefore be seen as less reliable.

Interestingly, minor chords are significantly less often correctly recognized than major chords. This appears to hold for seventh chords as well, when comparing the score of min7 with that of 7 and 9 chords.

Seventh and ninth chords get considerably worse scores than simple major chords, the same can be observed with minor seventh chords compared to simple minor chords. This is to be expected because of the increased complexity of the chords.

N.C. detection is revealed to perform rather poorly. We see this as a consequence of how $\omega_{N.C.}$ is set by default. The $\omega_{N.C.}$ parameter determines the aggressiveness in detecting N.C. As the relative amount of N.C. in the ground truth is relatively low (3.9%), it makes sense to decrease $\omega_{N.C.}$ and detect N.C. only in cases of very high confidence. As a consequence though, the detection ratio of N.C. suffers in this statistic.

chord	frequency (%)	WAOR (%)
maj	58.6	78.9
min	16.8	67.6
7	7.0	72.2
N.C.	3.9	38.5
min7	3.0	59.1
9	1.2	64.7
maj7	1.0	58.8
other	8.5	60.2

Table 5.2.: Relative amount of various types of chords in the ground truth and our algorithm's respective detection rates. Frequencies are calculated by taking the total duration of all instances of the chord type divided by the total duration of the dataset. For the WAOR only segments are taken into account where the ground truth is of the respective chord type.

5.3. Typical detection issues and possible approaches

In the following, some of the types of detection issues we most often encountered in the MIREX dataset and with other songs are discussed. In fact, the majority of cases where detection fails can be attributed to one or more of the following issues.

Incorrect root selected

The estimated chord may share notes with the correct chord but has the wrong root (see Figure 5.2a).

For example, a Gm7 chord (G-B♭-D-F) is recognized as a B♭ chord instead of Gm, a Dmaj7 chord (D-F♯-A-C♯) is recognized as F♯m instead of D, or a A/E chord is recognized as E or Em instead of A. Occurs particularly often with complex chords.

The octave-summation into a small set of 12 pitch classes and the subsequent simple template-based scoring are to be blamed for these errors. The 12 pitch classes no longer carry any information about the original absolute octave positions. These could be used for determining the bass note of the chord (which in turn likely matches the root of the chord). One way to gain more information on the bass part is the introduction of a separate bass chromagram that covers low notes as proposed in [Mauch and Dixon, 2010]. A more sophisticated template scoring is another way to further increase chances of correct detection here.

Chord changes in quick succession get lost

Chords lasting only for short periods of time[22] are not recognized, instead the preceding and following chord are extended so they span the gap of the undetected chord (see Figure 5.2b).

For one part, this is an inevitable consequence of the smoothing step. Lowering the filter size w does help here but it in turn leads to a more fragmented result. A second reason is the fact that it is generally easier to detect longer chords than very short ones, simply because more data on the chord is available and eventual transients during the attack phase of instruments play a less important role.

Ways to provide better detection rates in this regard could be alternative techniques for smoothing or beat tracking. We did experiment with an idea from [Lyon, 1987] where chord boundaries are found by magnitude peaks of the chroma vector gradient, and while it indeed showed promising improvements in the detection of short chords, it fared significantly worse when evaluated on WAOR.

Incorrect mode

The root of a chord was correctly identified but the wrong mode is selected (major instead of minor and vice versa, see Figure 5.2c).

This issue is frequently encountered in a form where the estimated mode switches one or several times during a time frame where an actually constant chord is present. It occurs when the respective major and minor chords get scores very close to one another and due to fluctuations the top scoring chord changes multiple times.

We prototyped a simple heuristic approach that replaces chord transitions matching $\mathrm{X} \rightarrow \mathrm{Xm} \rightarrow \mathrm{X}$ or $\mathrm{Xm} \rightarrow \mathrm{X} \rightarrow \mathrm{Xm}$ by one single X or Xm but ultimately decided against any special handling of these cases for simplicity reasons and due to negligible improvements in WAOR.

Difficulties with loud lead vocals/instruments

Chord recognition is mislead when vocal or instrumental melody lines are significantly more present than the accompaniment.

While accompanying instruments are usually intended to establish harmonies and are thereby restricted to notes from the current chord, melody lines have often greater freedom in this regard and may contain notes that are not strictly part of the chord. Since our method does not differentiate between lead vocals/instruments and backing their respective volumes determine how much they influence chroma

[22] typically durations of 1 beat and less

vectors. Hence, if melody lines are significantly louder than the backing this can lead to false chords being estimated.

We explored the idea of leveraging methods for vocal removal[23] aiming at reducing the influence that vocals have on the chroma features. An external tool[24] had been utilized to perform the filtering. Unfortunately, detection ratios did not improve consistently. We assume that this is because in addition to vocals other instruments that would have been beneficial may have been canceled out, or because in many songs the vocals actually contribute to a correct recognition of chords.

Difficulties when harmonies are only hinted at and not explicitly played

Chord recognition fails when chords are not explicitly played by any instrument but are only recognizable in the melody line or in very slow arpeggios[25,26].

By contrast, songs containing many layers of instruments (especially synth pads, organs and string sections holding harmonies in the background of a song) may appear more complex and thereby more difficult to handle but they turn out to be much more amenable to chord recognition. This is also the reason why estimation performance during choruses is noticeably superior to that during verses where typically fewer instruments play.

5.4. Impact of various stages on detection accuracy

Table 5.3 lists WAOR measurements of different algorithm variants where for each variant one of the algorithm stages is skipped. In this way, the effect that the various steps have on the final detection quality can be measured.

Improvements by tuning estimation are higher the farther song tunings are away from 440 Hz. While improvements are for some songs higher than 40 percentage points, the overall improvement due to tuning estimation is much lower as the majority of songs have tunings close to 440 Hz.

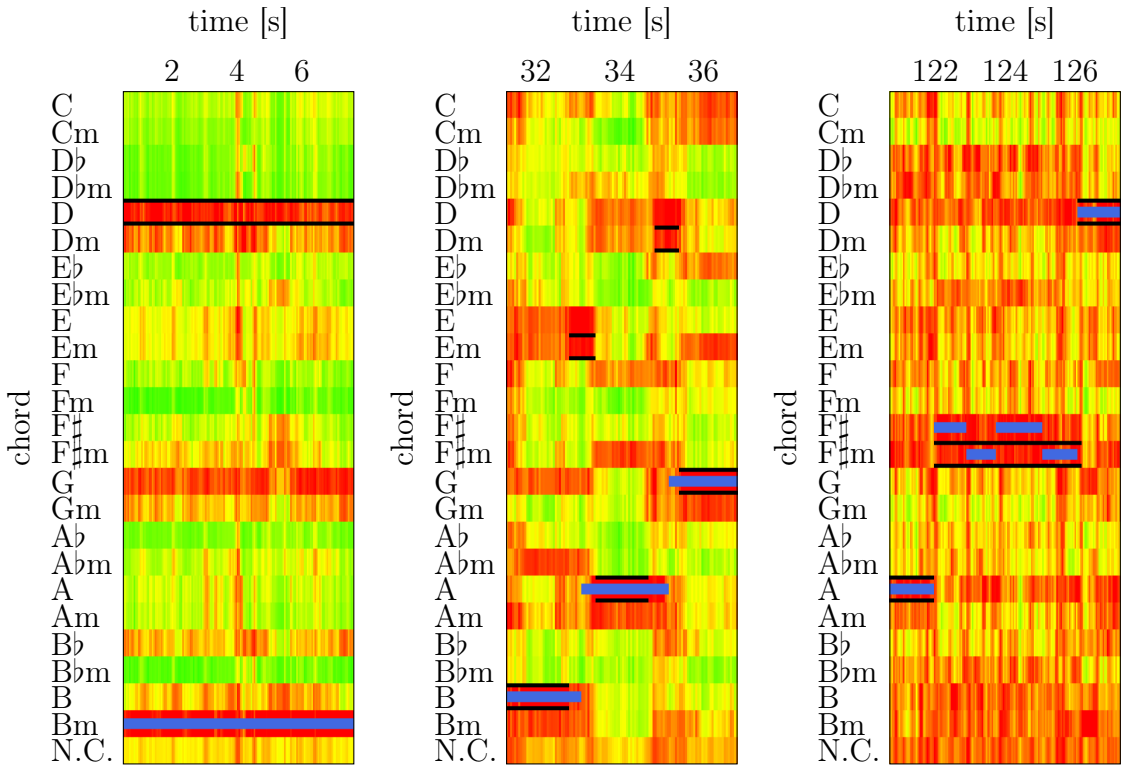
Filtering inharmonic content has an extensive effect on detection quality as well and because of this we see it (or any comparable method) as an essential component of

[23] The principal behind all vocal removers is that vocals are usually mixed into the center of a stereo recording. Simple removal methods involve subtracting one channel from the other obtaining one mono channel with the vocals removed.

[24] GoldWave (`http://goldwave.com`)

[25] chords decomposed into single notes played in succession

[26] faster arpeggios do not pose a problem if the STFT window size is large enough since individual notes of the arpeggio are not discernible then



(a) Beginning of "The Fool on the Hill" – The Beatles. The correct chord is D6 (D-F♯-A-B) but the root is incorrectly determined to be B (as part of a Bm7 chord with the same pitches B-D-F♯-A).

(b) Excerpt of "Moon River" – Henry Mancini. Two very short chords fail to be recognized. Note that the chords do get assigned the highest scores (red) but are ignored during smoothing.

(c) Excerpt of "Please Mr. Postman" – The Beatles. F♯ and F♯m get comparable scores leading to a characteristic pattern in the detection.

Figure 5.2.

variant	WAOR (%)
no tuning estimation[27]	69.7 (-2.7)
no harmonic content extraction	67.4 (-5.0)
binary templates	67.4 (-5.0)
no smoothing	51.9 (-20.5)

Table 5.3.: Effect of skipping certain stages in the algorithm on final detection quality (as measured on the MIREX 2009 test set). Included in parentheses is the relative change compared to the original score of 72.4%.

any serious chord recognition system.

Simple binary templates clearly perform far worse than our proposal. We see more sophisticated templates as a great way to improve detection without increasing complexity in the algorithm.

Smoothing not only enhances algorithm output in regard to reducing fragmentation, it also leads to a huge increase in terms of overlap ratio. Therefore, even if a fragmented output was acceptable for some use cases a smoothing step is still advisable.

5.5. Parameter optimization

Our algorithm incorporates a set of parameters that can be tuned to reach optimal detection performance. The default values that we propose have been determined by evaluations on the MIREX 2009 dataset and were already listed in Chapter 4. To give a better idea of the effect that the various parameters have on detection quality, this section provides evaluations and plots for different settings. For each plotted parameter, the values of all other parameters are kept fixed at their default values.[28] Note that all evaluations are conducted as averages over the entire dataset and plots of overlap scores (OR) of individual songs may look different.

FFT window size N, overlap factor O

Prior experiments indicated that, independently from the FFT window size, an FFT overlap factor of $O = 8$ or higher is required for optimal results. This finding seems to be in accordance with a rule-of-thumb in [Roads et al., 1997], stating that a

[27] a fixed tuning of 440 Hz is assumed

[28] It is assumed that the optimum value of any of the parameters is largely independent of all other parameters. A fully global optimization was unfeasible due to the large search space but we are confident that our results are very close to the global optimum.

FFT window size N	**WAOR (%)**
4096	50.2
8192	71.5
16384	**72.4**
32768	67.3
65536	54.4

Table 5.4.: WAOR performance on the MIREX 2009 test set for different FFT window sizes.

minimum of four-fold overlapping is needed for proper sampling of the input audio data. We see this as a confirmation that a value of $O = 8$ is a good choice and set this parameter fixed for the FFT window size evaluations.

Table 5.4 lists WAOR performance numbers on the MIREX 2009 test set for five different FFT window sizes. Only sizes that are powers of two are considered here due to limitations in the FFT implementation we are using. For these measurements the smoothing filter size w has been adjusted to always amount to the same timespan (measured in seconds).

A drop in detection quality for very low or very high window sizes is to be expected as these lead to poor resolution in frequency or time. It is harder to reason though, why exactly a window size of 16384 performs optimal here. The optimum value is obviously a function of the sampling rate, the frequency range of interest, and the required temporal resolution which in turn is in some way defined by the typical timespan between chord transitions. When taking differences in the sampling rate into account, 16384 is in line with what other groups suggest for their algorithms as reported in [Mauch, 2010].

Pitch range $[N_l, N_h]$

The lower and upper bound N_l and N_h are both explored in steps of one octave. Apart from a reduction in the search space, this also guarantees that N_l and N_h always span a whole number of octaves. The latter is important since it ensures that for all pitch classes an equal number of notes are mapped to the pitch class in the chromagram stage. Otherwise a bias would be introduced towards certain pitch classes.[29]

Table 5.5 lists the results of these measurements. Among ranges spanning only one or two octaves, [C4, B4] and [C3, B4] perform best by a large margin, suggesting that this is where most relevant information for chord recognition lies. Conversely,

[29] The division by the number of notes that contribute to a pitch class in the chromagram definition (see Section 4.5) turns out not to be sufficient in this regard.

N_h \ N_l	C0	C1	C2	C3	C4	C5	C6	C7
B0	2.8							
B1	3.8	2.7						
B2	24.5	23.7	25.9					
B3	55.7	55.6	56.5	58.4				
B4	70.7	70.7	71.0	70.1	65.7			
B5	72.1	72.2	72.3	70.4	62.8	44.6		
B6	72.1	72.1	72.4	70.2	61.2	40.1	19.4	
B7	72.2	72.1	72.4	70.3	61.2	40.0	18.7	2.1

Table 5.5.: WAOR performance (%) on the MIREX 2009 test set for different pitch ranges $[N_l, N_h]$. Values for N_l and N_h are specified here in scientific pitch notation and results are color-coded from red (0%) to green (100%) for better illustration.

ranges that do not include [C4, B4] (i.e. $N_h \leq$ B3 or $N_l \geq$ C5) score, as expected, very poorly.

Highest detection quality is achieved with intervals [C2, B6] and [C2, B7]. There is a larger set of ranges, though, that performs only marginally worse (less than 0.5 percentage points), those that include at least interval [C2, B5]. We conclude that, as long as the pitch range is set large enough to cover certain essential octaves, the end result is close to optimal. The inclusion of additional, potentially unneeded octaves at the very low or high end appears to have little influence on detection quality.

Harmonics decay rate ϵ

Figure 5.3 shows a plot of detection quality as a function of harmonics decay rate ϵ.

For values of ϵ close to zero, harmonics are assumed to fall off very fast and harmonic content extraction thus approaches to having no effect at all. For large values of ϵ (i.e. closer to 1), harmonics are required to be present with little decay and as a result frequencies are more likely to be filtered out. If ϵ is chosen too large ($\epsilon > 0.6$), sounds that are actually harmonic get filtered out and detection quality drops quickly.

Best results are obtained with values around 0.4, a value differing slightly from what is proposed in [Gómez, 2006] and was later adopted by [Papadopoulos and Peeters, 2011, Papadopoulos and Peeters, 2007]. In these works a decay rate of 0.6 is used and only the first six harmonics are considered. It remains unclear why exactly these parameters were chosen and Gómez admits that further experiments are needed to study these. But assuming they were determined by a similar evaluation as was

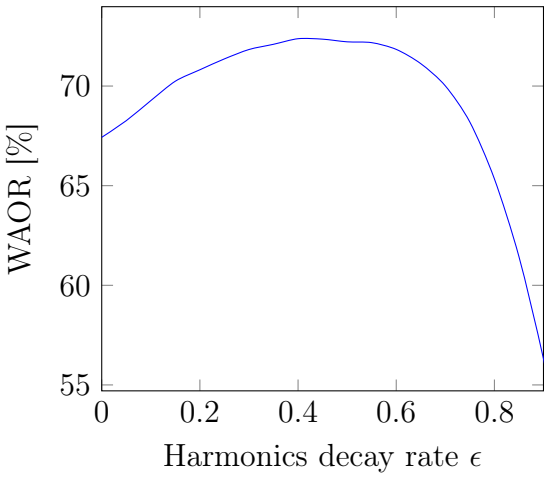


Figure 5.3.: WAOR performance on the MIREX 2009 test set for different harmonics decay rates ϵ.

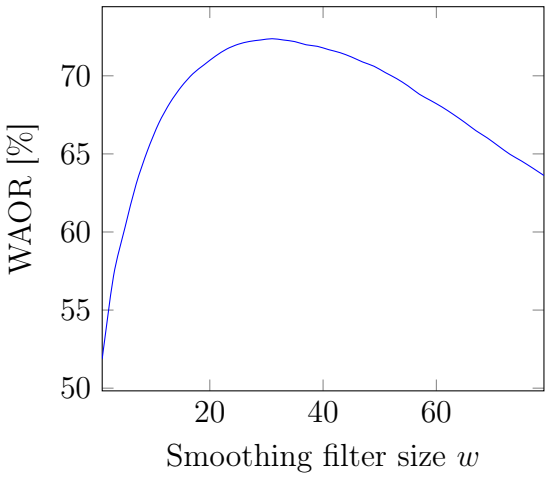


Figure 5.4.: WAOR performance on the MIREX 2009 test set for different smoothing window sizes.

done here, the discrepancy could be due to different datasets used[30] or differences in how harmonics are handled in the algorithm[31]. Generally, it has to be kept in mind that the optimal value determined as part of our evaluations remains the value for which our algorithm performs best. It may or may not be indicative of the actual decay behavior of harmonics.

Smoothing window size w

Figure 5.4 shows evaluations for different smoothing window sizes. Unsurprisingly, too light or too heavy smoothing has a negative effect on detection ratio. WAOR performance stays almost identical for sizes between 25 and 40, but what is not immediately obvious is that estimations actually vary there between being more fine-grained or coarser.

[30] Ultimately, the best choice for ϵ depends on which instruments are present in the audio as different instruments show different decay behavior of their harmonics.

[31] Gómez handles harmonics as part of his *Harmonic Pitch Class Profile*, mapping a frequency to multiple pitch classes simultaneously (one for each harmonic), and Papadopoulos incorporates harmonics directly into chord template design.

6. Conclusion

This work was aimed at providing an introduction to the task of automatic chord recognition and relevant concepts in the field of signal processing and music theory. For this, a simple chord detection algorithm was designed and its detection performance evaluated.

We began by introducing a number of mathematical tools required. This included continuous and discrete variants of the Fourier transform as well as the short-time Fourier transform for time-frequency analysis. The magnitude spectrum of a signal was then presented as a suitable form for further analysis of audio data.

We continued with a description of the equal temperament tuning system that provided us a mapping of musical notes to frequencies. The phenomenon of harmonics in the sound of natural instruments was outlined which amounted to a composition of sine waves of multiples of a fundamental frequency. Lastly, we finished on the topic of music theory by providing a basic introduction to the structure and meaning of chords in Western tonal music.

The proposed chord recognition algorithm was introduced, elaborating on a number of design decisions that were made such as the focus on a system that requires no training and a restriction to 24 major/minor chords. We then detailed the overall detection process which involved the calculation of a spectrogram, the filtering of higher harmonics and inharmonic noise, a tuning estimation, the computation of feature vectors in a chromagram through octave summation, the derivation of chord probabilities from it through chord templates, and a final chord selection and smoothing step.

Next, the typical process of evaluating a chord detection system was outlined which requires a test set of songs with corresponding ground truth annotations and a quality metric such as the weighted average overlap ratio (WAOR). We evaluated our algorithm on the MIREX 2009 test set and provided in-depth interpretations of the results. In terms of WAOR the algorithm fared very competitively among MIREX 2009 contenders although direct comparisons have to be drawn with care because of differences in how the systems were trained. Major issues proved to be the correct detection of chord roots for complex chords, the retaining of quick chord changes, as well as the differentiation between major and minor modes. Lastly, the effect of different algorithm parameter choices on detection accuracy was examined, demonstrating how suggested default values were determined.

Given that work on chord recognition only started around 15 years ago and the tremendous amount of progress that has been made especially in the last few years, it remains exciting to see how far automatic chord recognition can be taken. Today, top algorithms already accomplish scores of 83% detection accuracy on the MIREX 2009 test set and up to 73% on completely unseen test data. [McVicar et al., 2014] We see a huge potential for future research on not only the task of chord estimation, but also on all the other promising disciplines tested by MIREX, hopefully enabling exciting new applications and experiences in the future.

A. CD content & Implementation

The CD-ROM supplied with this thesis contains:

- this document in digital form (PDF)
- binaries and source code of the implementation of the algorithm presented, including licensing information and compilation notes

The algorithm was implemented in the C# programming language and is licensed under the MIT license.

The focus was put on a straightforward implementation with readability and extensibility in mind. The code has been only optimized to a point where performance is sufficient for typical use cases (around 15 seconds to process a 3–4 minutes long music file on a 2 year old laptop), we expect there to be a large room for further optimizations if high-performance was a requirement.

Spectrograms are calculated using FFT routines. In addition, the calculation of the spectrogram and the harmonic context extraction step are parallelized to the number of CPU cores available.

The code references two third-party libraries, NAudio[1] for the decoding of audio files and Math.NET[2] for the FFT implementation.

Both a graphical and a command-line interface have been implemented for the algorithm. Main features include:

- loading audio files and running the algorithm on them
- loading ground truth data and computing the overlap ratio
- changing various parameters of the algorithm
- displaying/exporting the generated spectrogram, chromagram and chordgram
- exporting the estimated chord sequence
- playing an audio file and displaying the current estimated chord (graphical interface only)

[1] `http://naudio.codeplex.com/`
[2] `https://www.mathdotnet.com//`

List of Figures

List of Tables

Bibliography

[Abdallah et al., 2005] Abdallah, S., Noland, K., Sandler, M., Casey, M., and Rhodes, C. (2005). Theory and evaluation of a Bayesian music structure extractor. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 420–425.

[Benson, 2006] Benson, D. J. (2006). *Music: A Mathematical Offering*. Cambridge University Press.

[Boyd and Butt, 2003] Boyd, M. and Butt, J. (2003). *J.S. Bach*. Oxford Composer Companions. Oxford University Press.

[Burgoyne et al., 2011] Burgoyne, J. A., Wild, J., and Fujinaga, I. (2011). An expert ground-truth set for audio chord recognition and music analysis. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 633–638.

[Cho and Bello, 2009] Cho, T. and Bello, J. P. (2009). Real-time implementation of HMM-based chord estimation in musical audio. In *Proc. Int. Computer Music Conference*, pages 117–120.

[Dongarra and Sullivan, 2000] Dongarra, J. and Sullivan, F. (2000). Guest editors' introduction: The top 10 algorithms. *Computing in Science and Engineering*, 2(1):22–23.

[Ellis, 2009] Ellis, D. P. W. (2009). The 2009 LabROSA pretrained audio chord recognition system. LabROSA, Columbia University, New York.

[Fujishima, 1999] Fujishima, T. (1999). Realtime chord recognition of musical sound: a system using Common Lisp Music. In *Proc. Int. Computer Music Conference*, pages 464–467.

[Glazyrin, 2012] Glazyrin, N. (2012). Audio chord estimation using chroma reduced spectrogram and self-similarity.

[Glazyrin, 2013] Glazyrin, N. (2013). Mid-level features for audio chord estimation using stacked denoising autoencoders.

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

[Harte, 2010] Harte, C. (2010). *Towards Automatic Extraction of Harmony Information from Music Signals*. PhD thesis, University of London.

[Harte and Sandler, 2009] Harte, C. and Sandler, M. (2009). Automatic chord recognition using quantised chroma and harmonic change segmentation. Centre for Digital Music, Queen Mary University of London.

[Howard and Angus, 2009] Howard, D. M. and Angus, J. A. S. (2009). *Acoustics and Psychoacoustics*. Focal Press.

[Khadkevich and Omologo, 2009] Khadkevich, M. and Omologo, M. (2009). Improved automatic chord recognition. Fondazione Bruno Kessler, University of Trento.

[Loy, 2006] Loy, G. (2006). *Musimathics: the mathematical foundations of music*. MIT Press.

[Lyon, 1987] Lyon, R. F. (1987). Speech recognition in scale space. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 12, pages 1265–1268.

[Mauch, 2010] Mauch, M. (2010). *Automatic Chord Transcription from Audio Using Computational Models of Musical Context*. PhD thesis, School of Electronic Engineering and Computer Science, Queen Mary University of London.

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

[Mauch et al., 2009] Mauch, M., Noland, K., and Dixon, S. (2009). MIREX submissions for audio chord detection (no training) and structural segmentation. Centre for Digital Music, Queen Mary University of London.

[McVicar et al., 2014] McVicar, M., Santos-Rodríguez, R., Ni, Y., and De Bie, T. (2014). Automatic chord estimation from audio: A review of the state of the art. *IEEE Transactions on Audio, Speech and Language Processing*, 22(2):556–575.

[MUSIC IR, 2009] MUSIC IR (2009). Audio chord detection results. `http://www.music-ir.org/mirex/wiki/2009:Audio_Chord_Detection_Results`. Retrieved 08/16/2014.

[Oudre et al., 2009] Oudre, L., Grenier, Y., and Févotte, C. (2009). MIREX chord recognition system system 1 : major and minor chords. Institut TELECOM, TELECOM ParisTech, CNRS LTCI.

[Papadopoulos and Peeters, 2007] Papadopoulos, H. and Peeters, G. (2007). Large-scale study of chord estimation algorithms based on chroma representation and HMM. In *CBMI'07. International Workshop on Content-Based Multimedia Indexing*, pages 53–60. IEEE.

[Papadopoulos and Peeters, 2009] Papadopoulos, H. and Peeters, G. (2009). Local key estimation based on harmonic and metric structures. In *Proceedings of the International Conference on Digital Audio Effects*, pages 408–415.

[Papadopoulos and Peeters, 2011] Papadopoulos, H. and Peeters, G. (2011). Joint estimation of chords and downbeats from an audio signal. *IEEE Transactions on Audio, Speech and Language Processing*, 19(1):138–152.

[Pauwels and Peeters, 2013] Pauwels, J. and Peeters, G. (2013). Evaluating automatically estimated chord sequences. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 749–753.

[Pauwels et al., 2009] Pauwels, J., Varewyck, M., and Martens, J.-P. (2009). Audio chord extraction using a probabilistic model. Department of Electronics and Information Systems, Ghent University, Belgium.

[Reed et al., 2009] Reed, J. T., Ueda, Y., Siniscalchi, S., Uchiyama, Y., Sagayama, S., and Lee, C.-H. (2009). Minimum classification error training to improve isolated chord recognition. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, pages 609–614.

[Roads et al., 1997] Roads, C., Pope, S. T., Piccialli, A., and De Poli, G. (1997). *Musical Signal Processing*. Swets & Zeitlinger Publishers.

[Rocher et al., 2009] Rocher, T., Robine, M., Hanna, P., and Strandh, R. (2009). Dynamic chord analysis for symbolic music. In *Proceedings of the International Computer Music Conference (ICMC)*, Montreal, Quebec, Canada.

[Weller et al., 2009] Weller, A. V., Ellis, D. P. W., and Jebara, T. (2009). Structured prediction models for chord transcription of music audio. In *ICMLA*, pages 590–595. IEEE.

Declaration of Authorship

I do solemnly declare that I prepared this thesis independently and that the thoughts taken directly or indirectly from other sources are indicated accordingly. The work has not been submitted to any other examination authority and also not yet been published.

....
(Date) (Signature)