Analysis and evaluation of audio-similarity algorithms for cover and live song identification

Martin Angelov

MSci Computer Science with Industrial Year, Student ID 1422087

School of Computer Science, University of Birmingham

Supervisor

Prof. Achim Jung

Acknowledgements

I wish to thank my family who supported me through every decision I made during my education, as well as my supervisor for his invaluable guidance throughout this project.

Abstract

This project presents a detailed analysis of audio similarity algorithms solving the task of cover and live song identification. After examination of the current state of the art several algorithms are chosen to be examined in detail. The algorithm analysis is conducted through a benchmark implementing the algorithms and measuring performance on cover and live song datasets. The algorithm results are compared through evaluation metrics and give an overview of best performing audio features and similarity methods suitable for song identification.

Contents

List of Figures

Chapter 1

Introduction

1.1 Project overview

Music information retrieval (MIR) is an area of analysis dedicated to extracting information from music. It combines many different disciplines of science including psychology, psychoacoustics, signal processing and computer science. One of the main aims when applying MIR techniques solving the task of song identification, i.e. matching an audio stream to a particular song [1]. This is usually achieved through a form of hashing applied on the digital signal and comparing the resulting representation to a reference fingerprint [2], [3]. This approach returns good results for the task, since we can easily quantify a good match between both fingerprints.

We can further modify the original song identification task to apply to cover songs. A cover song is a very creative reinterpretation of a released song usually performed by an artist different than the original. The cover can therefore differ significantly from the origin in tempo, pitch or song structure (*add more*). The amount of variation in a cover strongly depends on the genre of the primary track - Western popular music pieces are for example more likely to be transformed than ones from classical music [4]. Therefore the only remaining common feature between the cover song and the original is the underlying fundamental melody of the piece and potentially the lyrics.

Because of these potential disparities between two versions of a single song, the problem of identifying covers of songs is much more difficult than determing an identical match with the original. The above fingerprinting approach has been attempted [5] and the results are insignificant [6]. Direct comparison between the fingerprints of the song is unable to capture the remaining similarity within two audio files. Other MIR methods need to be considered in order to measure similarity when attempting cover song recognition.

The general advances of technology have allowed companies such as Spotify

[7], Apple [8], SoundCloud [9] and more to create large-scale music databases and offer them as commercial services. Proportionally to the increasing availability of large music collections grows the need for managing the volumes of audio information through MIR techniques, with cover song identification being one of them. As a consequence most modern mechanisms to cover song recognition work by comparing an audio track called *query song* against a large database of songs, a *reference database*. Each mechanism is evaluated based on its similarity estimation performance, as well as its scalability as we increase the database size.

This project analyses the principles of a set of non-hashing based cover song identification algorithms and evaluates their performance. Most of the examined algorithms are designed to work with large-scale databases and follow the workflow model described above. The evaluation considers only their similarity estimation results and does not account for scalability. After analysis of the results a hypothesis on the best performing audio similarity technique is established (*or maybe devised?*).

1.2 Report structure

The sections of this report are as follows:

- *Chapter 2* offers a summary of the background information required to understand and implement the audio similarity algorithms
- *Chapter 3* explores other state of the art methods of measuring similarity not examined in detail by the project
- *Chapter 4* provides a description of the evaluation task through which each algorithm is analysed
- *Chapter 5* contains detailed descriptions of each algorithm
- *Chapter 6* expands on implementation details related to the benchmark tool
- *Chapter 7* outlines the best results achieved and offers an analysis on them
- *Chapter 8* summarises potential further contributions to the project
- *Chapter 9* discusses the main challenges related to the project and the task of cover song identification
- *Chapter 10* is a summary of the project management techniques utilised during the project

Chapter 2

Background theory

Each type of information extracted from an audio stream is referred to as an *audio feature*. Audio features are mainly derived using various transformations on the signal based on some basic properties of sound. They allow for music data to be comparable and algorithmically accessible [10]. This section presents low-level theory required to understand the high-level description of how each feature is obtained further in the report. At this level of the project description we only require an understanding of general principles related to audio and signal processing, therefore the explanations are kept concise without going deeper into technical details.

2.1 Basic properties of audio signal

A *digital audio signal* is a representation of the continuous sound wave as a series of binary numbers. This representation helps preserve the *frequency* (the speed of the vibrations), *amplitude* (the fluctuations of the vibrations) and *period* (the time for which a wave completes one vibration) of the sound. The energy that each sound wave emits through vibrations is called *sound energy* and its rate is measured through *sound power*. The amount of energy in sound is related to its amplitude - The higher the amplitude the larger the amount of energy is.

The majority of audio features use frequency or power extracted over a period as primary audio information used to define the feature (*modify/change*).

2.1.1 Sampling and sample rate

The process of converting an analogue sound wave to a digital one involves a process of extracting points (samples) from the continuous signal and using them to describe the signal into a discrete form. This method is called *sampling* and

the average amount of samples collected per one second is *sample rate*. The representation of a song used during feature extraction is a sequence of samples extracted from the digital signal of the song based on its sample rate.

2.1.2 Tones

Western music is described using *tones*, steady periodic sounds [11]. They can be pure if the sound has sinusoidal waveform, or complex if they are a combination of pure tones with a periodic repetition pattern. Any of the waves contained in a complex tone is called a *partial*. The lowest of the tones determined by frequency is called a *fundamental*, with each of the tones higher than the fundamental defined as *overtones*.

Tones are distinguished by several basic perceptual properties of sound - pitch, loudness, timbre and duration [12]. A half tone is called a *semitone* and it is the smallest measure of period between sounds. Semitones are grouped into *octaves* where each octave contains 12 semitones. The acoustic opposite of tone is noise, a disordered sound which is unpleasant to the human brain and is disruptive to hearing [13].

2.1.3 Psychoacoustic properties

In order to understand how properties of a digital signal could form audio features we first need to examine how they relate to the ways of how people perceive music and sound.

Several basic perceptual properties of sound are regularly used to describe what is captured by each audio feature. They are pitch, loudness, timbre and duration. Klapuri et al. [14] form good definitions of the psychoacoustical terms outlined. They define *pitch* as a perceptual attribute which offers ordering of tones on a frequency-related scale. The term is very closely related to tone and sometimes both definitions are used interchangeably. Different pitches could be labelled through the Helmholz pitch notation [15] using letters, through scientific pitch notation [16] utilising letters and numbers, or directly using numbers representing the closest frequency in hertz (Hz). Despite being determined by clear and stable frequencies in sound, pitch is more importantly a subjective auditory sensation, so a strict mathematical relationship between frequency and pitch does not exist [17]. As a standard it is accepted the musical note of A above C (Helmholz notation) or A4 (scientific notation) has a frequency of 440 Hz [18]. The human ear perceives musical intervals on an approximately logarithmic scale with respect to a *fundamental frequency*, the lowest frequency available in a tone and therefore determining the overall pitch of the tone. The set of sounds with frequencies an integer multiple to the fundamental frequency is called *harmonic*

series with every member of the set being a *harmonic*. In relation to this any partial with a frequency matching a harmonic is called a *harmonic partial*.

Using this notion of logarithmic perception there are various mappings of pitches to frequencies, the most famous of which is the MIDI tuning standard [19].

Similar to tones, pitches are also ordered in octaves, where an octave is an interval between one pitch and another with double its frequency [20]. The set of all pitches which are within one octave of each other is called a *pitch class*.

Loudness is another subjective psychoacoustical attribute of sound [14]. Similar to how pitch is related to frequency, loudness is a perception of sound pressure. *Sound pressure* is a measurement of the pressure divergence caused by a sound wave to the ambient atmospheric pressure [21]. Loudness orders sounds on a scale ranging from quiet to loud [17].

Tones also have a "colour" attribute attached to them through the introduction of *timbre*. Timbre allows distinguishing sounds with potentially identical pitch, loudness and duration, but produced by different musical instruments [14]. In large part this is determined by the overtones in the sound generated alongside the fundamental frequency when a musical instrument is played. While we already established that the fundamental frequency determines the pitch of a tone, the added overtones create a more complex and richer sound, they add "colour" to it. Overtones cannot be perceived as separate notes, but we do hear them - they determine the timbre of an instrument.

Timbre is a complex concept which cannot be defined by a single property of sound. There are many attempts at breaking down the attribute into components. Robert Erickson [22] offers one of the most accepted decompositions where timbre relates to the following acoustic parameters of sound:

1. Tonal and noise characters

2. Time envelope - A *time envelope* describes how sounds changes over time [23]. It measures how much time it takes for the sound to reach an amplitude level when a musical instrument is activated (a key on the piano is pressed, for example) and subsequently how long it takes for the sound to go back to its initial level. The analysis of rise and fall of the amplitude in the sound wave is called *ADSR (attack, decay, sustain, release).*

3. Spectral envelope and any changes to it - when only the curve of the amplitude out of a time envelope is taken into consideration then a *spectral envelope* is used. Figures 2.1 and 2.2 illustrate the difference of between both envelopes.

4. Changes in the fundamental frequency

5. Onset dissimilar to the dominant vibration - with *onset* being the start of a musical note we look for 'anomalies' in the vibration of the wave compared to the vibrations following the anomaly.

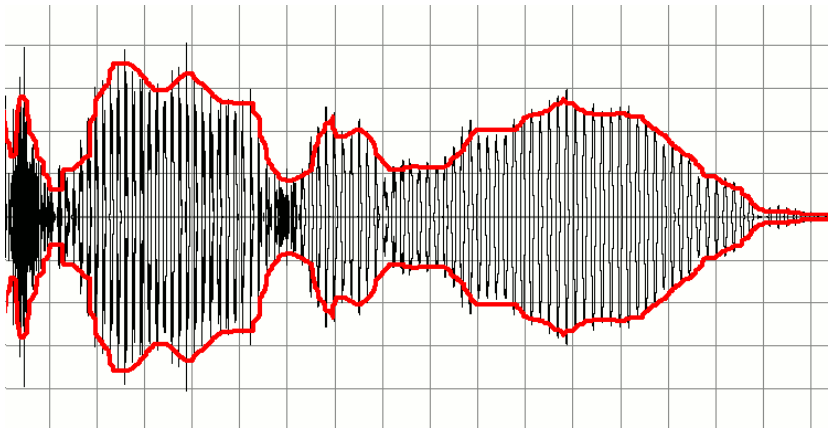

Figure 2.1: An example of a time envelope [24]

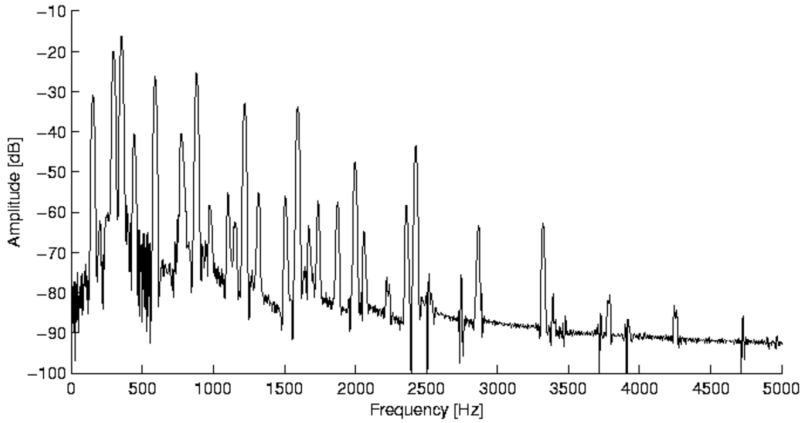

Figure 2.2: An example of a spectral envelope [25]

Out of all psychoacoustical properties introduced *duration* is the one which is the easiest to directly measure. It is an indicator of a length of any part of a musical composition - tone, pitch, the whole piece, etc [26]. The measurement is expressed using a base unit of time.

2.1.4 Beat

One final basic concept that we need to introduce is the *beat*. It signifies repeating portions in a song which define the overall rhythm of a music piece. Rhythm

is formed of "strong" and "weak" beats, with the former signifying a suitable moment for melody change.

2.2 Audio transformation techniques

2.2.1 Fourier transform

Any waveform including the audio ones could be presented as a sum of sinusoids of different frequencies [27]. In music each of the constituting frequencies represents a pure tone. In order to be able to work with the tonal representation of a song we need to perform *Fourier analysis*, that is, find a way to separate the different frequencies within the audio signal. The result is a complex-valued continuous function. We perform the analysis using a decomposition called *Fourier transform* - one of the most widely used audio transformations for conversion of a signal from a time domain representation into the frequency domain.

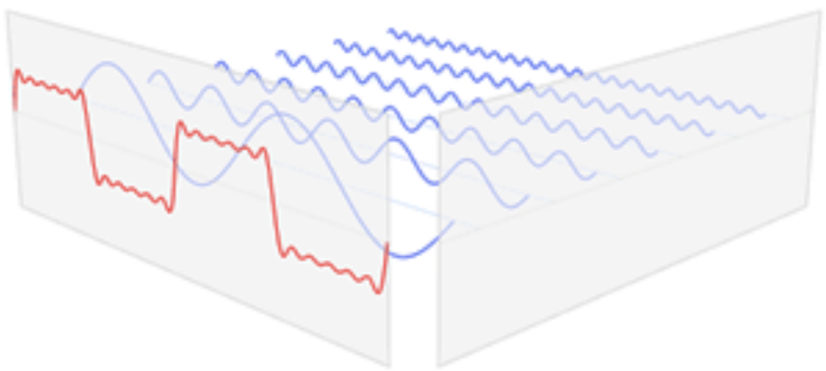

Figure 2.3: Fourier transform applied on a periodical function (in red). The transform distinguishes six sine functions (in blue) represented as an amplitude-frequency relationship [28]

Fourier transform could be applied to either periodic (resulting in a periodic function of combined harmonics called *Fourier series*) or non-periodic functions.

As we are working in the domain of music we are focussed on the Fourier workings on functions with periodicity. Using the transform our goal is finding an approximation for function $f(t)$ with period $T = 2L$ using a sinusoid functions each with period a multiple of T [28]. Figure 2.3 shows the destructuring that a Fourier transformation performs on a function. The resulting function $g(t)$ which is the transform of $f(t)$ takes the form of:

$$g(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n \cos\frac{n\pi x}{L} + b_n \sin\frac{n\pi x}{L}) \tag{2.1}$$

Figure 2.4: Fourier series representation of a periodic function [29].

The coefficients a_n and b_n determine the relative weights of each of the sinusoids [28]. They are calculated using sine and cosine integrals over the function period:

$$a_n = \frac{1}{L}\int_{-L}^{L} f(x)\cos\frac{n\pi x}{L}dx,\ n = 1, 2, 3, \ldots \tag{2.2}$$

$$b_n = \frac{1}{L}\int_{-L}^{L} f(x)\sin\frac{n\pi x}{L}dx,\ n = 1, 2, 3, \ldots \tag{2.3}$$

Figure 2.5: Derivation of the parameters for the Fourier transform [29].

From a frequency spectrum perspective the relative weights of the sinusoids also represents the amount of frequency present in the original function at a point of time [30].

There are different forms of Fourier analysis. In the area of MIR one of the most frequently used types is the *discrete-time Fourier transform (DTFT)*. DTFT is a *periodic summation* of the regular Fourier transform. A periodic summation is a representation of a periodic function with a period P by multiple aperiodic ones separated by periods with a length multiples of P [31]. A comparison between regular Fourier transform and DTFT is shown on top and bottom left of figure 2.6.

In order to obtain a finite sequence of DTFT samples we perform *discrete Fourier transform (DFT)* on equally-spaced samples from the original audio signal (bottom right of figure 2.6). Computing DFT directly is an operation on a sequence of complex numbers converting it into a new sequence of complex numbers. The computational complexity of the transformation is $\mathcal{O}(n^2)$. A class

of algorithms called *fast Fourier transform (FFT)* has been created to compute DFT more efficiently with complexity of $\mathcal{O}(n \log n)$ [32]. This is achieved by representing the DFT as a transformation matrix - a matrix which when multiplied with the input signal achieves the same result as normal DFT. To optimise the matrix multiplication process FFT algorithms factorise the transformation matrix into a product of several sparse factors [33]. In practice FFT is the main way to compute DFT and therefore both terms are used interchangeably.

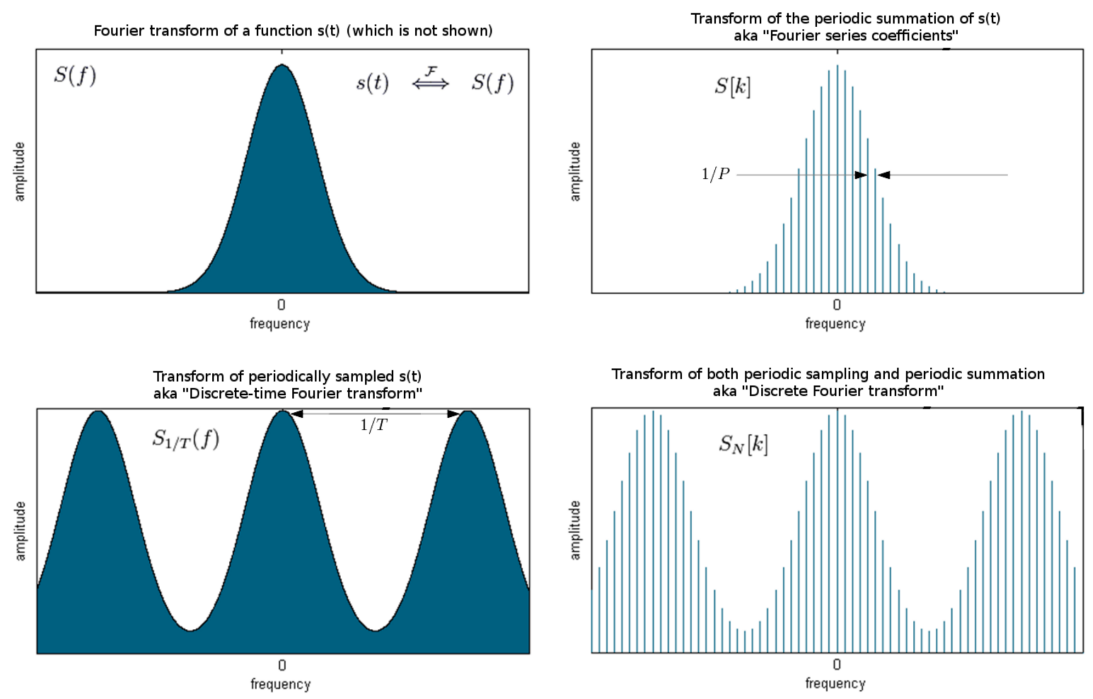

Figure 2.6: Regular fourier transform (top left), DTFT (bottom left) and DFT (bottom right) side by side. If we apply Fourier transform to a function $s(t)$ we get the standard representation in the top left corner. The top right corner shows the result of applying periodic summation to the function with period P and then applying the transformation. When the function $s(t)$ is sampled over interval T the Fourier transform produces DTFT (bottom left). If both periodic sampling and periodic summation are combined the transform produces DFT (bottom right) [31]

The result of a Fourier analysis is represented in a spectrogram (figure 2.7). The information contained in it is very valuable when determining sound properties such as pitch, power, timbre and more. The original audio signal could be re-obtained by performing *inverse discrete Fourier transform (IDFT)* on the spectrogram resulting from the Fourier analysis.

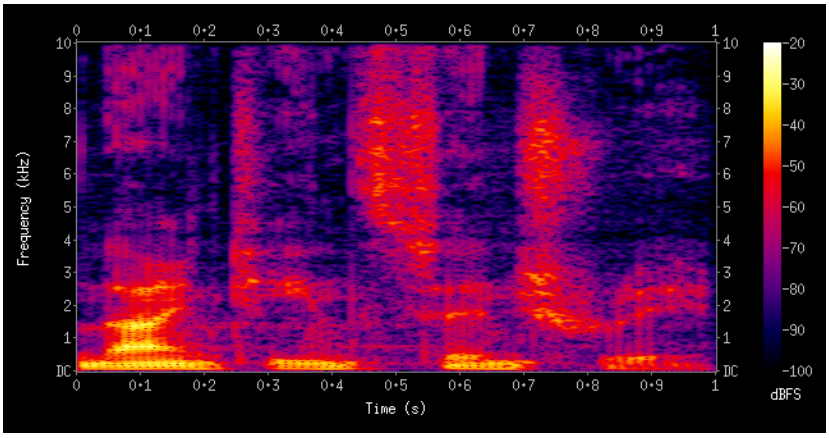

Figure 2.7: Spectrogram of a spoken word phrase. It shows the time versus frequency relationship produced by the Fourier transform, as well as the intensity (amount) of each frequency for a time frame [34].

2.2.2 Filters

During the processing of the audio signal we want to detect the frequency maxima and minima of the wave, or possibly attenuate certain frequencies. In order to do that we use *audio filters* - tools that pass certain frequencies and blocks others [35]. Filters are defined in terms of *bands*, the range of frequencies they pass. A combination of filters which helps us produce the required frequency cut-off is called a *filter bank*.

2.2.3 Scales

Earlier we mentioned that an octave contains 12 semitones. This is an established standard - the semitones are equal on a logarithmic scale and form the twelve-tone equal temperament or *chromatic scale*. It is the ordering that most Western music compositions employ and therefore the ordering we use to determine relative positions in tones of audio signals. It is also the scale which musical instruments often utilise. A good example of this are the keys of a piano, which are ordered based on an equal temperament scale.

Another scale which is widely used to arrange values of a psychoacoustic variable is the mel scale. It presents a perceptual ordering of pitches which are determined to be equally far away from each other [36]. The scale introduces a unit of perceptual pitch called *mel*, where $1000\,mel = 1000\,Hz$. The rest of the mel-frequency mapping is displayed in figure 2.9. The conversion from Hz to mel is calculated as:

$$Mel(f) = 2595 \log (1 + \frac{f}{700}) \tag{2.4}$$

Figure 2.8: Converting a frequency of fHz into $mels$

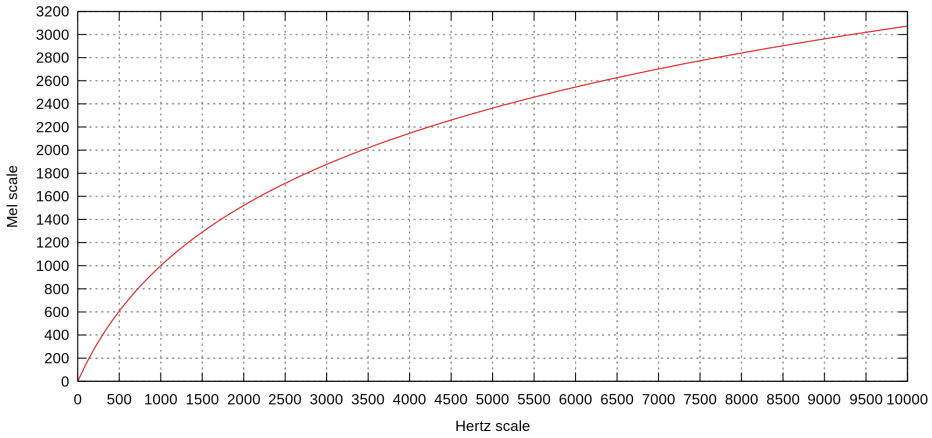

Figure 2.9: Perceived pitches on a Mel scale versus the Hertz frequency scale [37]

2.3 Machine learning techniques

Some cover song identification algorithms create machine learning models based on which they measure similarity when a query song is provided. This section outlines the principles of each method used later in the algorithms.

2.3.1 Random forests

Random forests are a form of ensemble learning for classification and regression [38]. Ensemble methods train multiple learners in an attempt to solve the same problem [39], and in the case of random forests the type of learner is a decision tree. The method works by building a predefined number of decision trees during training, and return a result combining their individual outcomes. In the case of classification the mode of all returned class predictions is taken, while the mean of the predictions is used during regression.

Random trees learning generally builds upon the idea of recursive partitioning used in decision tree learning. Partitioning builds a decision tree from the training data based by asking questions related to the independent variables (attributes) formed by the data samples. The 'answer' or split points for the tree are called cut points. They are calculated to be the local optima values which best split the data into homogenous sets based on the attribute. This principle fails to create

generalised models, since as the tree becomes deeper and deeper the more it tends to overfit on their training data [40].

Random trees help avoid the overfitting tendency of individual decision trees by implementing bootstrap aggregating, also known as bagging. The learning algorithm creates several small sets of uniformly sampled observations from the training set and constructs trees using them. Depending on the configuration some samples can be picked in more than one set. The end result is achieved through the voting approaches explained above.

2.3.2 K-means clustering

K-means clustering is an unsupervised machine learning approach which aggregates a collection of data points together because of some similarity between them. The end result is a separation of the points into k distinct clusters. The outcome of the clustering is evaluated through the sum of squares defined as:

$$Sum\ of\ squares = \sum_{i=1}^{n}(x_i - \overline{x})^2 \tag{2.5}$$

Figure 2.10: The sum of squares for n items where x_i is the value of the i-th item and $\overline{x}$ is the mean of the set n.

The intent is to reduce the sum of squares as much as possible. Clustering is complete when either the sum of squares does not significantly change any more, or the algorithm runs a pre-determined number of iterations.

In the area of digital signal processing K-means clustering is used to perform *quantisation* - mapping a large (possibly continuous) set of values to a countable smaller set which is easier to work with [41].

Chapter 3

Related work

Cover song identification has been a very active topic of research in the last 10 years. The task has spawned many solutions, so for the scope of this project it is infeasible for each technique to be analysed in detail - from technical implementation details to results on datasets. To fully cover the active developments in the research area should also examine related research which is not the direct focus of the project.

The chapter presents an analysis of the algorithms submitted in the *Cover song identification* task, part of the Music Information Retrieval Evaluation eXchange (MIREX) conference [42]. The format is similar to a competition, where participating researchers submit potential solutions to a task and each proposal is evaluated under equal conditions. The evaluation environment is preserved throughout the MIREX editions, so results from different years are comparable between each other.

The study of the conference was conducted before the selection of the main audio similarity algorithms for the project and their implementation in a benchmark. This allowed for an informed decision on what audio features are performing best and also ensured that algorithms not included in MIREX are picked, to potentially present a contribution to the field.

3.1 Cover song identification in MIREX 2005 - 2018

In 2006 the MIREX conference created the *Cover Song Identification* task. The datasets used for evaluation are the MIREX 2006 US Pop Music Cover Song dataset (1000 tracks, each track is 16-bit monophonic, 22.05 kHz sampling rate, wav format). The dataset has 30 cover songs each represented by 11 different versions for a total of 330 files. Each of the files are then individually run as

queries for the algorithms. Each algorithm then attempts to return the other 10 versions of the same song. The only output the submitted algorithms should return is a *number of queries* $\times$ *number of candidates* distance matrix.

Out of the 40 submissions related to the cover song task in the MIREX conference 25 were considered.

3.1.1 Results evaluation

In order to determine which algorithms perform best we need to look at the results achieved each year. From them we would then be able to determine what audio similarity techniques have worked best so far and examine them further in depth.

The results outlined here are based on the statistics provided by MIREX each year for the results of the cover song identification task. Each year MIREX publishes the total number of covers identified (TNCI), the mean number of covers identified (MNCI), the mean of average precisions (MAP) and the mean rank for first correctly identified cover (MRFCIC) for each algorithm.

Let us look at the best THCI results plotted over the years for the MIREX cover song identification task:

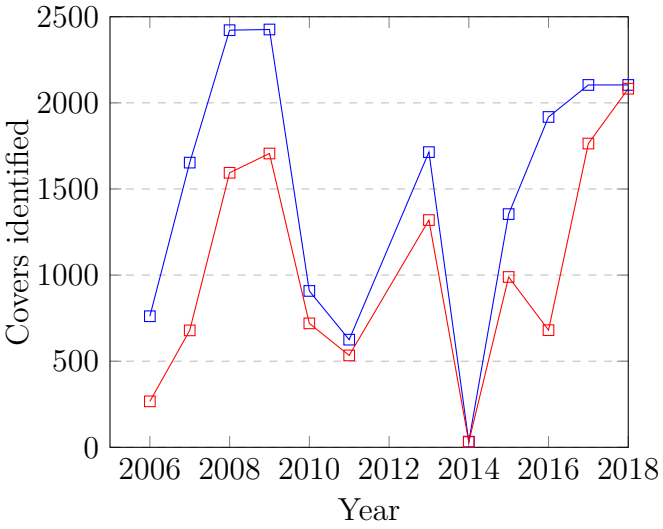

Figure 3.1: Results of the winning submissions in the period 2005-2018. The blue line represents the max THCI achieved for each year (i.e. the challenge winners). The red line outlines the average THCI for each year the task has taken place. The challenge did not run in 2005 and 2012.

We could see that the best performing submissions are the ones published in 2008, 2009 and 2018.

To determine if there is a tendency into what type of audio feature is used, a table comprising of all audio features used is compiled from the available submissions:

Audio feature	Times used	Winner
Chroma features	12	2006, 2018
Harmonic Pitch Class Profiles (HPCPs)	5	2007, 2008, 2009, 2016
Mel-frequency cepstral coefficients (MFCCs)	2	
HPCP, MFCC and MFCC SSM combined	1	
Basic Pitch Class Profiles (PCP)	1	
Custom technique	1	2014
Relationship (distance) to other songs in set	1	
Statistical Spectrum Descriptors	1	
Pitch line estimation	1	

Figure 3.2: Number of times a type of audio feature is used in MIREX 2005-2018. The *Winner* column indicates whether the best algorithm for the year utilises the feature.

Based on this evaluation we could see that the best audio features for measuring song similarity are chroma features and HPCPs. Unfortunately only 7 out of the 13 winning papers were available online, so it is unclear whether they are also using any of the features for information extraction. The *custom technique* mentioned in the table refers to an algorithm implementing a non-conventional audio feature which achieves the worst winning result in the MIREX competition so far - only 33 covers identified (out of 3300 possible).

Some of the main audio similarity algorithms explored in the project use chroma features and MFCCs as audio features. Therefore in this section we only explore the structure of HPCPs.

3.1.2 Harmonic Pitch Class Profiles

For each pitch representing a semitone part of an equal temperament scale a *pitch class* is the set of all pitches which are all separated by an octave from each other. Knowing that we can create a *pitch class profile (PCP)* - a vector which represents the intensities of all twelve pitch classes [43]. PCPs become *harmonic* when they represent the relative intensities of the pitch classes within an analysis frame in a signal [44]. Straying from the general definition of a PCP they could also be extracted over a 24 or 36-bin pitch spectrum, representing the relative intensity of every 1/2 or 1/3 semitones respectively.

HPCPs are generally extracted using a Fourier transform to determine the constituting frequencies in the audio stream. The resulting spectrogram is then

filtered to include only an audible spectrum of frequencies, which are then mapped to their pitch classes. Each time frame is then normalised to obtain the HPCP audio feature representation of a song.

3.1.3 Similarity methods

Besides introducing the audio features which provide a song representation, a majority of the MIREX submissions also outline the processes of measuring similarity between songs. A set of notable similarity techniques is of interest to background of the task.

Dynamic time warping (DTW) is an algorithm for measuring similarity between two temporal sequences, which may vary in speed [45]. The algorithm performs a sequence alignment method, which 'bends' the sequences non-linearly in the time dimension in order to determine the similarity independent of other non-linear variations in the time dimension. After sequences are aligned a rule-based matching is applied which determines the final similarity score. Dynamic time warping has been very successful when creating automatic speech recognition systems. The algorithm achieves great results when used in a voice recognition system with MFCC audio features [46], a task not too dissimilar to cover song identification.

Because a digital audio signal is essentially a binary stream some submissions appropriate classic information theory techniques such as *Normalised compression distance (NCD)* to establish whether two songs are covers. The origins of NCD come from the notion of information distance, a metric that represents the size of the shortest program required to convert one binary object into another. We apply normalisation to the result to obtain similarity based on the object lengths. If objects a and b are of length 1000000 bits and their distance is 1000 bits, then this should be a more significant result than having objects c and d of length 1000 bits with the same distance [47]. Given this outcome a normalised information distance (NID) indicates that a and b are more similar than c and d.

NID however is proven to not be computable or semicomputable [48]. It is therefore approximated using a compressor algorithm applied on the objects, with the NID applied to the compressed results.

Chapter 4

Evaluation task

An evaluation task must be set based on which the algorithms are analysed, evaluated and compared.

4.1 Design

The design of the evaluation task follows the general structure for an algorithm targeted towards utilising a large-scale database. In includes taking a query song and creating pairwise comparisons of it with any song from a song database. The similarity score of each pair is then calculated using a cover song identification algorithm and an ordering is created. The pair with the highest similarity includes the database song that the algorithm has identified to be most similar to the query song (and potentially its cover version).

As a song database the evaluation task uses several music datasets. Each dataset contains sets of different versions of a song. The unique identifier that combines all versions of a single song is called a *clique*. For instance, the original version song 'Take On Me' by the Norwegian band A-ha and its cover version created by the American punk rock artist MxPx are in the same clique named *Take On Me*. Each dataset is split into a symbolical *train* and *test* sets. The train portion of the songs is initialised as a song database, while each of the tracks in the test set is used as a query song. During the split it is ensured that there is at least one song from each clique in both the train and test set, so that for every query song a cover ready to be identified is found in the song database.

The task results are produced through a benchmark written using Python [49] and storing the song database into a MongoDB [50] instance. Figure 4.1 shows a high-level architecture diagram of the benchmark, which could help understand the general design of the evaluation task.

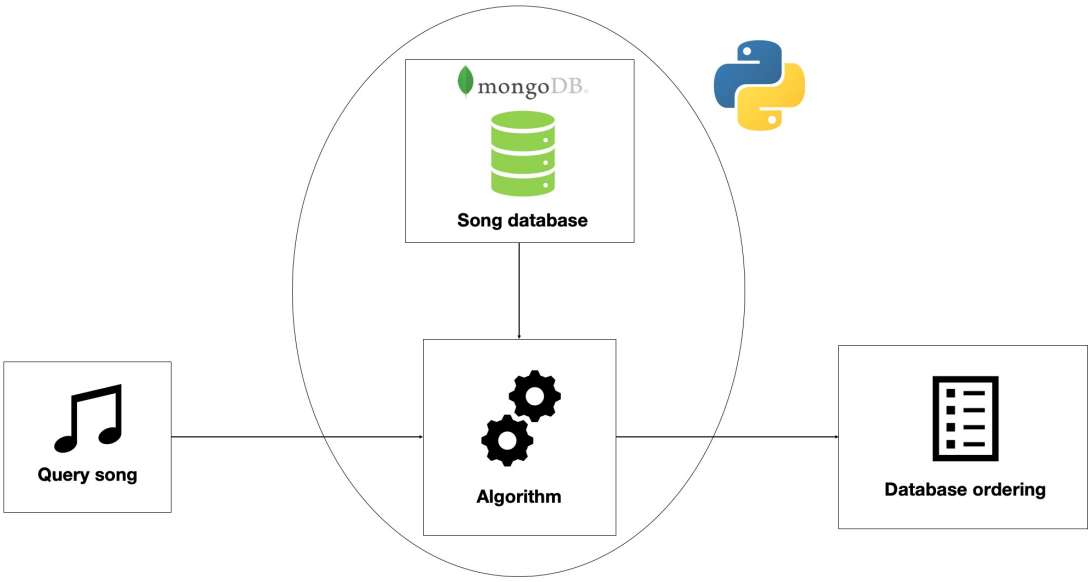

Figure 4.1: Benchmark architecture diagram

4.2 Evaluation methods and metrics

All algorithms are evaluated using metrics adapted from the MIREX *Cover Song Identification* competition format. The benchmark implements the following measurements:

- *mean rank (MR)* of the position of a true cover (a song from the same clique as the query song) in the database ordering produced by an algorithm
- *mean reciprocal rank (MRR)* of the position of a true cover in the database ordering produced by an algorithm
- covers identified in the *Top-K* positions of the database orderings for all songs, where K is a number
- covers identified in the *Top-P* percentile of the database orderings for all songs, where P is a number

4.3 Datasets used for evaluation

One of the main challenges in the project was the lack of appropriate datasets, therefore some of the datasets are self-curated using my personal music collection. The algorithms were evaluated using the following datasets:

- *covers80* - a widely used by MIR researchers consisting of 80 cliques and 164 songs [51]. Contains original compositions and cover versions of them.
- *live_dataset* - a self-curated dataset consisting of 70 cliques and 145 songs. Contains original compositions and professionally recorded, mixed and mastered live performances of them.
- *yt_dataset* - a self-curated dataset consisting of 16 cliques and 32 songs. Contains original compositions and live performances of them recorded from an audience perspective using a smartphone.

A list of all songs in each dataset is provided in the Appendix, as well as in the benchmark source code GitLab repository [52].

Chapter 5

Examined algorithms

The main algorithms analysed in the project are attempted to be as different as possible from each other. This allows covering as many audio features and similarity measuring techniques as possible. My motivations of the choice of algorithms were influenced by the study on MIREX submissions, as well as the overall scope of features they encompass.

Julien Osmalsky coins the term *rejector* to define a pairwise comparison function that given two audio tracks it returns a score ranking of similarity between both tracks [53]. This term was adopted by the project to refer to each of the algorithms examined. The relation between an algorithm paper and its rejector term is outlined in section 5.1.

The final selection includes 5 individual algorithms and an algorithm aggregating results from 4 of them. Each of the algorithms is examined individually, with the results from some of them combined through the rank aggregation algorithm. A lot of the algorithm details are deduced based on related literature or past papers written by the academics who created the algorithm.

Any implementation information about an algorithm is either taken from the algorithm paper or through my own experimentation and research.

5.1 Algorithms list

The analysis if performed on the following algorithm:

- **Weak rejector** - Osmalskyj, Julien, Marc Van Droogenbroeck, and Jean-Jacques Embrechts. *Enhancing cover song identification with hierarchical rank aggregation* [54]
- **Cross-correlation rejector** - Ellis, Daniel PW, Courtenay V. Cotton, and Michael I. Mandel. *Cross-correlation of beat-synchronous representations for music similarity* [55]

- **Quantisation rejector** - Osmalskyj, Julien, et al. *Combining features for cover song identification* [53]
- **Timbre rejector** - Tralie, Christopher J., and Paul Bendich. *Cover song identification with timbral shape sequences* [56]
- **Aggregated rank rejector** - Osmalskyj, Julien, Marc Van Droogenbroeck, and Jean-Jacques Embrechts. *Enhancing cover song identification with hierarchical rank aggregation* [54]
- **Fingerprint rejector** - Rafii, Zafar, Bob Coover, and Jinyu Han. *An audio fingerprinting system for live version identification using image processing techniques* [57]

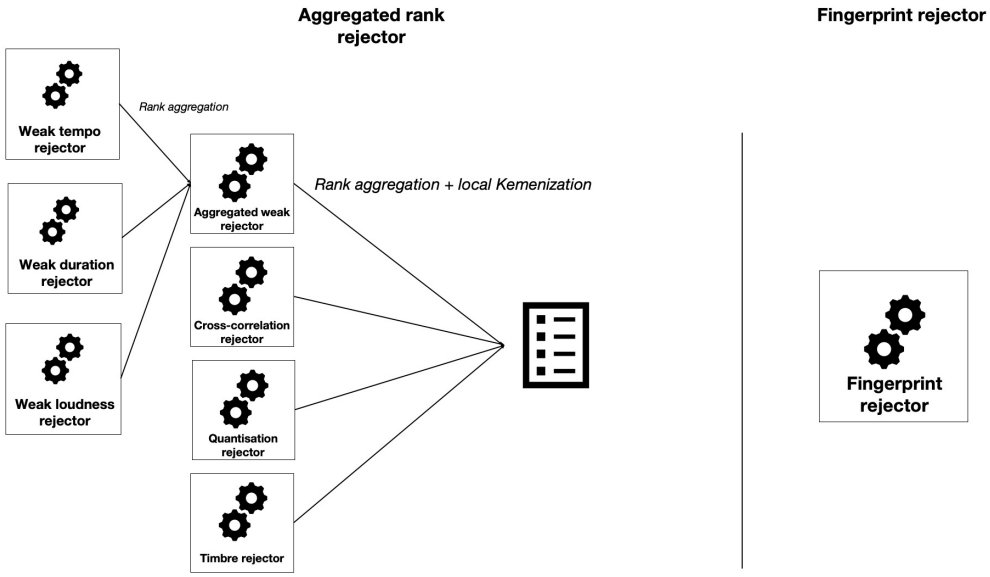

Figure 5.1: Diagram of chosen algorithms and the relationships between them. The aggregated rank rejector combines the results of 3 types of weak rejectors, as well as the quantisation, cross-correlation and timbre rejectors

5.2 Weak rejector

The weak rejector takes its name from the fact that it uses global single-valued features of sound to calculate a similarity score. Weak features include tempo, duration, loudness, average number of beats, etc. Those audio properties are considered 'weak' since by themselves they cannot uniquely describe a song - the

majority of pop tracks from a certain era are composed with the same tempo, for example [58]. As a consequence the results from weak rejectors are inherently insignificant are usually not considered individually, but as part of an aggregation with other results.

Each weak rejector takes a single audio feature as song representation. The algorithm extracts the feature from each song in a dataset and from them it creates a training set by combining each pair of songs in different ways. For songs A and B with weak feature values of $f1$ and $f2$ respectively the minimum ($min(f1, f2)$), maximum ($max(f1, f2)$), sum ($f1 + f2$), quotient ($\frac{f1}{f2}$) and absolute value of quotient of the difference divided by the sum ($abs(\frac{f1-f2}{f1+f2})$) are taken. The generated data is used as attributes to train an ExtraTrees classifier model. Extra randomised trees or ExtraTrees classification is a form of random forest classification. In contrast to the regular random forest implementation, ExtraTrees uses the whole training data for each decision tree. Rather than calculating an optimal cut-point, it also uses a random one. The direct benefits of using this variation of random trees are higher computational efficiency while achieving similar perfomance on average, in addition to better results on some specific problems [59].

The evaluation datasets used in this project are too small to generate an ExtraTrees model which performs even adequate classification of songs. To gather sufficient training data the weak features of 12,104 songs are extracted from the music streaming service Spotify [7]. The tracks are chosen based on clique groupings from the SecondHandSongs (SHS) [60] dataset - the largest dataset of covers songs aimed at academic research. While the full dataset is difficult to obtain, information about the cliques structure within it is easy to find. The extracted data is then combined into pairs using the procedure outlined in the previous paragraph. A binary class is then assigned to each pair entry depending on whether the pair of songs are covers or not. Afterwards the training data is further split into a training set of 3773 cliques (8653 songs) and validation set containing 1573 cliques (3451 songs). The metric used to validate the model performance is the area under the receiver operating characteristic (ROC) curve - a curve resulting from the plot of the true positive against the false positive rate of the validation results. ROC analysis of such form is suitable as a validation metric for the selection of an optimal model because it is independent of any potential cost function or cost distribution [61]. Because we are solely working with weak feature information the we can only validate based on correct and incorrect produced results.

After the training phase is complete each of the evaluation datasets is converted into the training set format and passed to the model for class prediction. The output result is a probability of every pair of songs being covers. Group-

ing the results per song produces a database ordering that is required by the evaluation task.

5.3 Cross-correlation rejector

The cross-correlation rejector uses chroma features as the main audio feature describing each song. Chroma is another name for pitch class profiles, i.e. a 12-dimensional representation of the intensity distribution in each of the twelve pitch classes part of the chromatic scale [43]. Any feature that at its core uses chroma to represent a song is a chroma feature. That is why the terms harmonic pitch class profiles (HPCP) and chroma features have a very similar definition, in fact HPCP is a type of chroma feature. The main difference between chroma features is their method of extraction. Using chroma features produces a chromagram - a pitch class versus time plot showing the intensity of each pitch class at any time point (Figure 5.2).

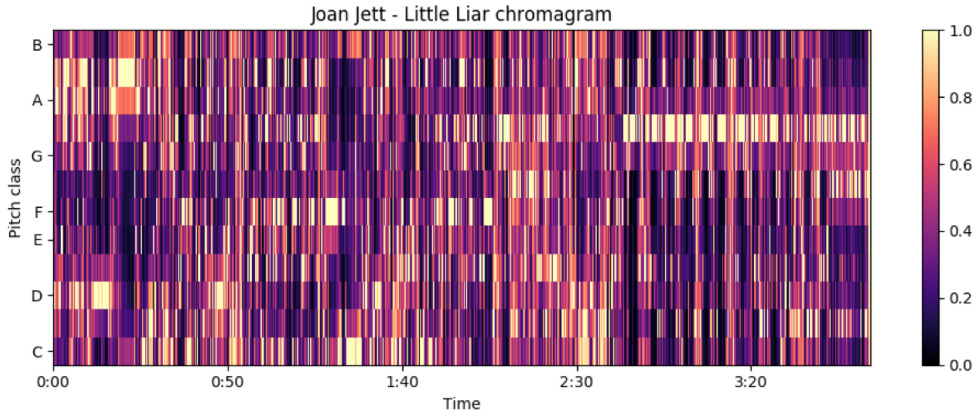

Figure 5.2: Chromagram of the song *Little Liar* by Joan Jett

The chroma features in the cross-correlation rejector are extracted by performing short-time Fourier transform (STFT) analysis. It consists of running Fourier transform on short segments of a signal to determine how the frequency content changes over time [62]. STFT improves upon the concepts of regular Fourier transform by not only telling us what frequencies are contained within the audio stream, but also at what time intervals does each frequency appear.

The transformations are applied using the fast Fourier transform (FFT) algorithm. The short segments are calculated over sliding overlapping windows as illustrated on figure 5.3. The size of the sliding window is determined by *FFT window size* and the overlap between each window depends on a parameter called

hop length. The actual window overlap is calculated as the difference between the window size and the hop length.

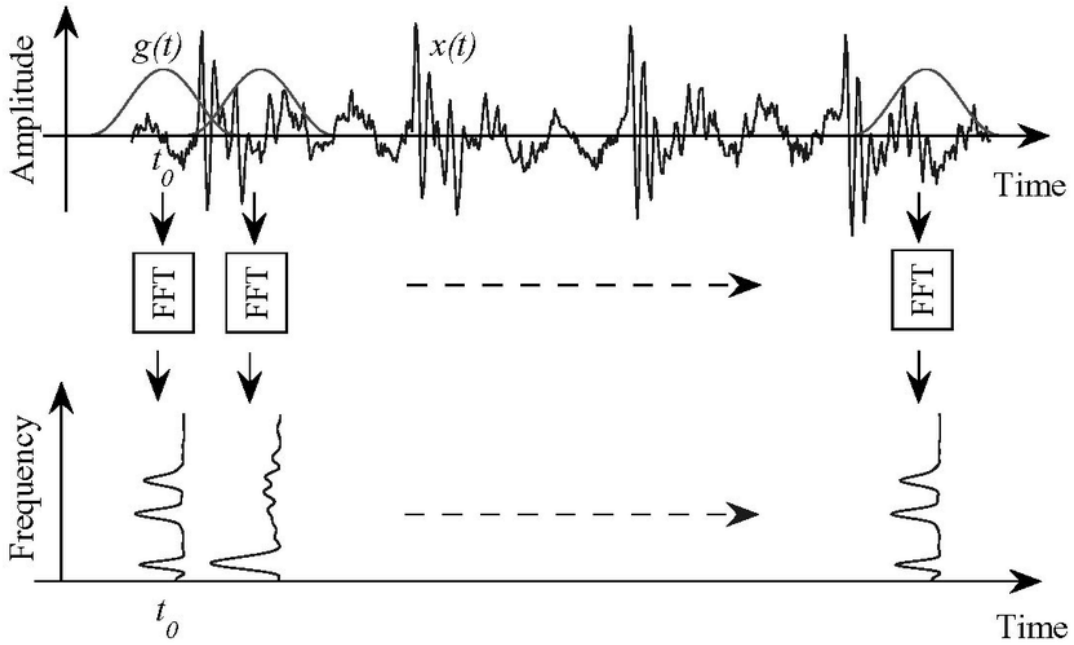

Figure 5.3: Workings of STFT on an audio signal $x(t)$. $g(t)$ is a function defined using the FFT window size and the hop length. The result of each FFT transformation is the amplitude and phase of frequency over the time period. [62]

The next step into obtaining chroma features for the cross-correlation rejector is converting the spectrogram resulting from the STFT into a chromagram. Using the standard mapping of pitch $A4 = 440Hz$ and knowing that pitches in music lay on an equal-tempered scale we can calculate the remaining pitches (relative to A4) from the frequency spectrum. We then aggregate all pitches belonging to the same pitch class and that gives us the chroma value of the chroma value corresponding to the class.

The conversion from frequency to pitch is regulated using a specification defining a function to be used when the pitches relative to $A4$ are calculated. This standardisation is commonly referred to as *tuning standard*. The paper describing the cross-correlation rejector does not specify what tuning was used to generate the chroma features, so a library implementation of chroma feature extraction was used in the benchmark. To further explain the conversion of frequency to chroma an example using the *MIDI tuning standard* is presented. Figure 5.5 follows each stage of the procedure.

The MIDI tuning covers a range of 128 frequencies enumerated from 0 to

127. A4 (which again corresponds to $440Hz$) is assigned number 69 and all other pitches are calculated using the equation:

$$d = 69 + 12\log_2(\frac{f}{440Hz}) \tag{5.1}$$

Figure 5.4: Calculating pitch from frequency f based on MIDI tuning standard [19].

This gives us a song representation similar to the one from 5.5 c). The final step is to sum all MIDI pitch coefficients into their corresponding pitch classes and obtain the chromagram from 5.5 d).

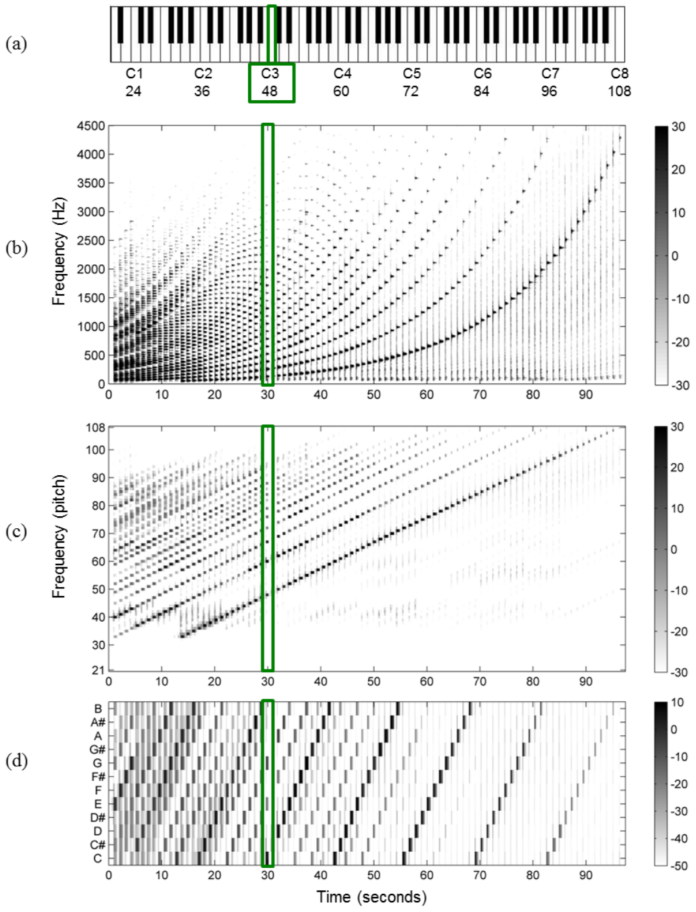

Figure 5.5: The separate stages of a spectrogram to chromagram conversion of a piano recording of the chromatic scale from A0 to C8 using MIDI tuning. [10]. a) The equal-tempered scale represented by the keys of a piano, with the pitch C3 mapped to 48 according to MIDI Tuning b) A spectrogram of the song, with the time frame where C3 was played marked in red c) Pitch log-frequency representation of the spectrogram d) Resulting chromagram

The chroma features in the cross-correlation rejector are *beat-synchronous*. This means that the period over which a chroma feature is obtained is every beat of the song, rather than a shorter time segment. The technique is very popular within cover song algorithms due to its ability to make comparisons between songs invariant to tempo, a property which is often modified by the cover song artists. Understanding beat detection in detail was beyond my abilities and a library implementation of a beat tracker created by the algorithm creators is directly used to obtain a result [63].

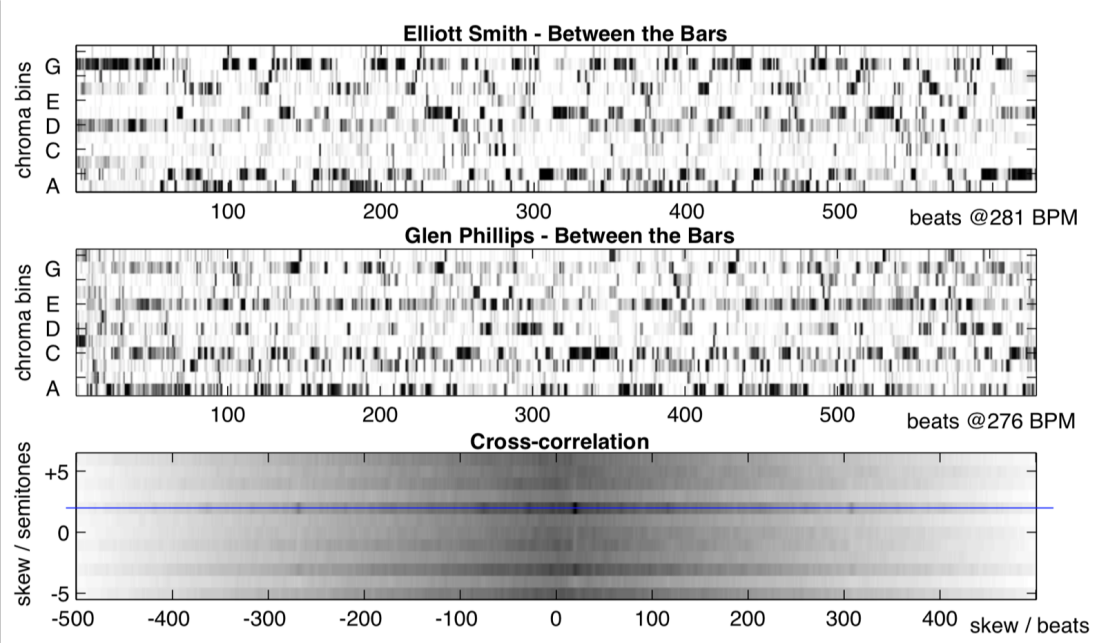

Figure 5.6: Extracted chromagrams of two cover songs and the resulting cross-correlation [4]

The resulting chromagrams for two songs are then scaled to unit length, so that the sum of all dimensions of the chroma vector has a sum of 1. The representation of the longer song is also shortened to be equal to the one of the shorter song. The chromagram matrices are then cross-correlated. To account for variations in the structure or pitch of the cover songs the shifts of the chromagrams between -500...500 beats are considered. The resulting intervals of peak values of peaks in the resulting cross-correlation matrix indicate a strong match between both tracks. Figure 5.6 shows the correlation process of two chromagrams. The blue line in the cross-correlation matrix indicates the strong similarity detected.

The similarity score that the rejector returns is the reciprocal value of the highest peak value in the cross correlation matrix.

5.4 Quantisation rejector

The dominant audio feature in the quantisation rejector is again chroma vectors which are beat-synchronous and unit normalised. The extraction implementation is the same, with some modifications to the FFT window size and hop length parameters. The general idea of the rejector is to represent songs using *code-*

words which are generated using K-means clustering of a large number of chroma vectors. The resulting cluster centers are the codewords. Each song is then represented using a histogram showing the codewords frequency for each track. As illustrated in figure 5.7 the approach leads to visible similarity between cover songs.

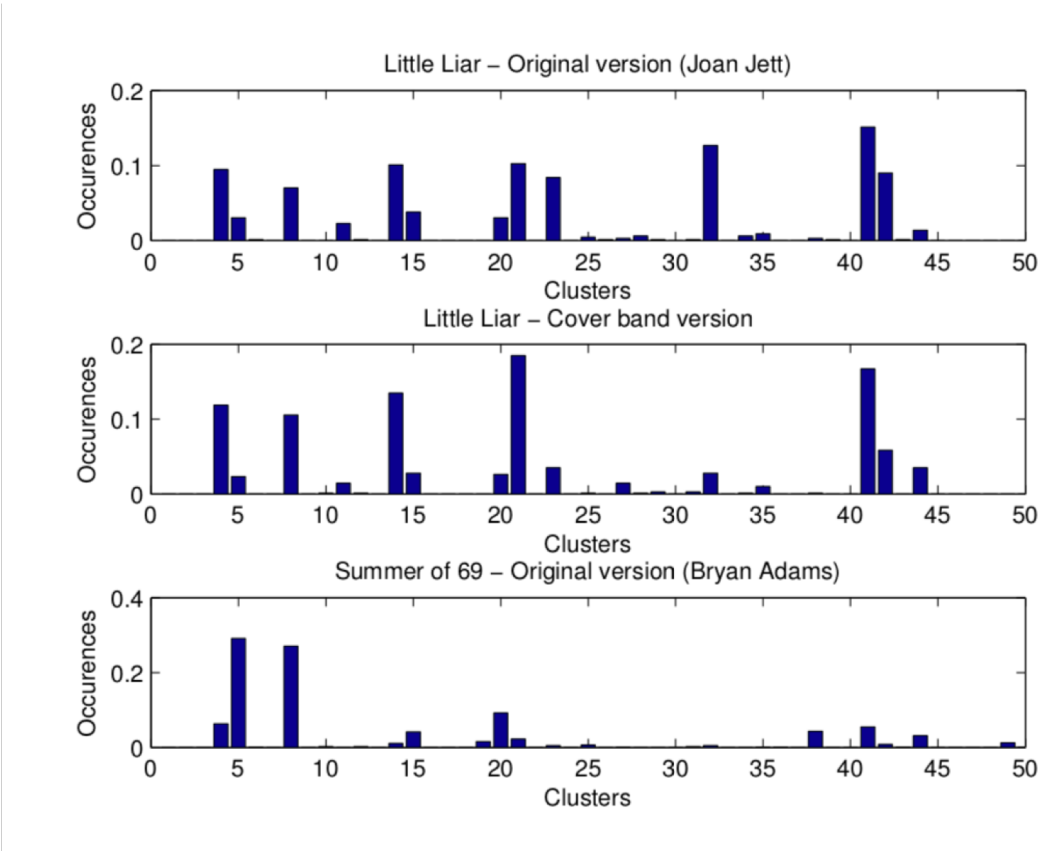

Figure 5.7: Histogram representations of songs utilising codewords. The first two histograms illustrate the visual similarity occurring between two cover, in comparison to the third song which is not related to the first two [64]

Sometimes a cover song can shift up or down the overall pitches of the original by a fixed interval. This technique is called *key transpositions*. To take possible transpositions into consideration the quantisation rejector implements *Optimal Transposition Index (OTI)* [65]. It is a calculation on the number of positions one of the histograms needs to be shifted in order to achieve best possible alignment with the other. For two histograms represented as n-dimensional vectors $\vec{A}$ and $\vec{B}$, where n is the number of codewords used, the OTI is derived from:

$$OTI(\vec{A}, \vec{B}) = \underset{0 \leq m \leq n-1}{\operatorname{argmax}} \{\vec{A} \cdot circshift_R(\vec{B}, m)\} \tag{5.2}$$

Figure 5.8: OTI equation

$circshift_R$ is a function shifting the vector $\vec{B}$ m positions to the right. The shift is circular, which means that the last element of the vector becomes the first one after every position shift. The operator $\cdot$ specify a dot product operation between both vectors. The OTI is used to shift one of the vectors to achieve best alignment between both songs:

$$\vec{A^{Tr}} = circshift_R(\vec{A}, OTI(\vec{A}, \vec{B})) \tag{5.3}$$

Figure 5.9: Shifting $\vec{A}$ to best align with $\vec{B}$

The performance of each K-means clustering model during training is evaluated through the sum of squared distances of each clustered point to their closest cluster center, a metric called *inertia* [66]. A low inertia indicates a a good separation of the points and therefore a good model.

The final similarity metric returned by the quantisation rejector is the cosine similarity between both histograms.

5.5 Timbre rejector

Beat synchronisation deals with tempo variations, but cannot handle any key changes in a cover song compared to the original. Chroma features also mix all octaves into pitch classes which obscures the nuances in the "colour" of sound. The timbre rejector aims to present a solution by examining the relative change of timbre over time.

The timbre rejector uses *Mel-frequency cepstrum coefficients (MFCCs)* as audio features due to their ability to preserve the timbral information after transformation of the signal [56]. MFCCs take the cepstrum of a song and project it on a mel scale. The main purpose of the feature is to serve as a bridge between the ways humans perceive sound (logarithmic frequency distinction, mel scale, etc) with the mechanics of musical instruments, in particular harmonics generation.

Cepstrum was originally created to describe a technique for detecting echoed signals in measured time series [67]. Since then it has been applied in many speech recognition problems [46] [68], and now it is also used in other MIR tasks such as cover song identification.

Cepstrum is informally defined as a nonlinear "spectrum of a spectrum". All types of cepstrum are usually obtained through a configuration of Fourier transform of a Fourier transform. For MFCCs we need to obtain the *power cepstrum* $c[n]$ defined as:

$$c[n] = |\mathcal{F}^{-1}\{\log(|\mathcal{F}\{x[n]\}|^2)\}|^2 \tag{5.4}$$

Figure 5.10: Derivation of a power cepstrum, where $\mathcal{F}$ denotes a discrete Fourier transform (DFT), $\mathcal{F}^{-1}$ is the inverse discrete Fourier transform (IDFT) and $x[n]$ is the audio signal to be transformed [69]

Another representation of the operation is available on figure 5.11.

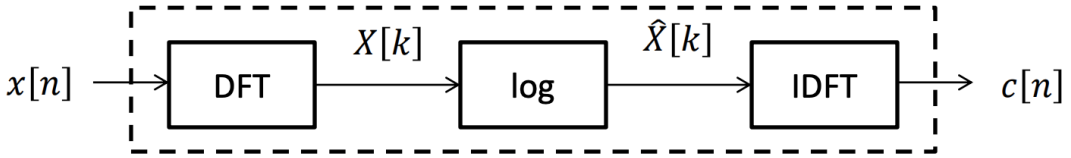

Figure 5.11: Sequence of Fourier transforms to obtain power spectrogram [70]

By applying DFT we obtain the power spectrum of a song - a projection of sound power versus frequency resulting in a log spectrogram such as the one from figure 5.12. The aim of the power cepstrum is to measure the periods of the resulting functions - on figure 5.12 the frequency with a longer period (described by the dashed line) is a representation of the fundamental frequency, while the one with lower period represents the harmonic overtones. By applying IDFT to the spectrogram we derive the power cepstrum in the second graph of figure 5.12. The peak in the the cepstrum represents the fundamental frequency and its period as at *quefrency* T. A quefrency is intended to denote the inverse of frequency as a time domain unit [71]. On the left side of the quefrency axis we have peaks which represent the periods of each harmonic.

By having the period sizes of the harmonics relative to each other we can gain information about the harmonic evolution over time. This harmonic information tends to be present even in cover songs significantly different than the original [56].

MFCCs are coefficients that are part of the *Mel-frequency cepstrum*, a variation of the power cepstrum where the extracted peaks of each harmonic are projected on a mel scale based on their frequencies. They are extracted as 20-dimensional vectors from a song.

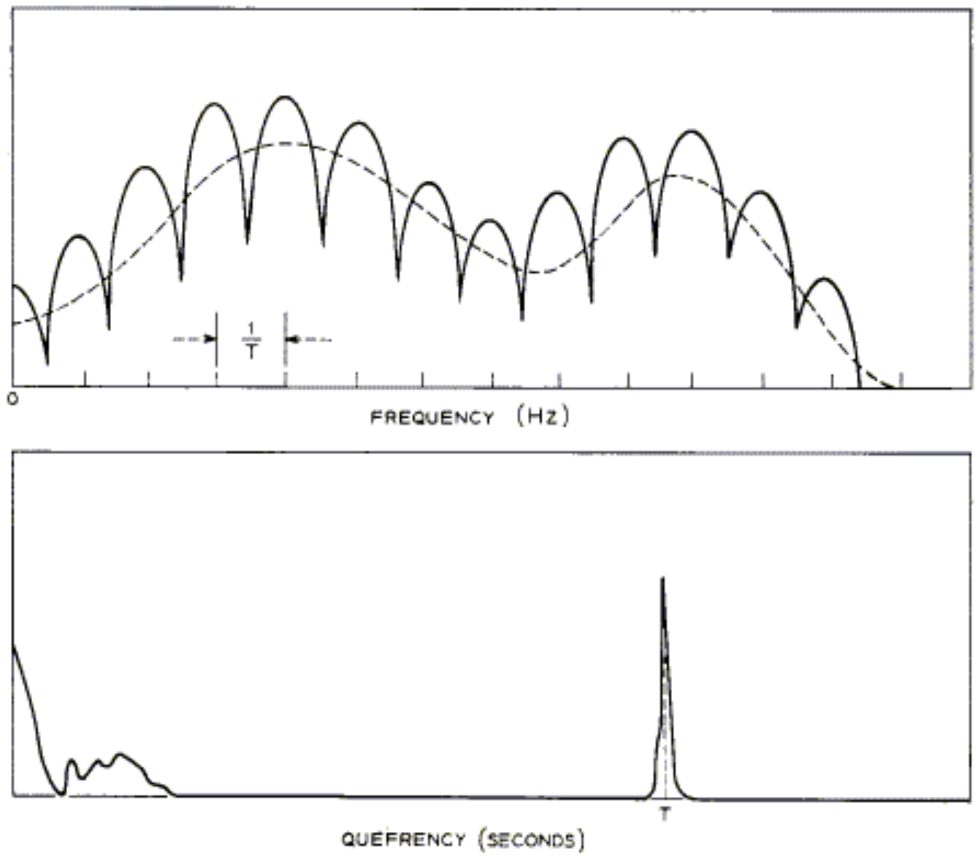

Figure 5.12: Spectrogram after applying DFT on the signal and a power cepstrum of it. $F(W)$ is equivalent to $\mathcal{F}\{x[n]\}$ [72]

Just like most of the previous rejectors, MFCCs in the timbre rejector are extracted using over the identified beat intervals in a track. There is one significant difference however - each extraction time frame is a *block* enclosing B time-contiguous beat intervals. Every two successive blocks overlap in a way that only the first beat of the first block and the last beat of the second block differ between two blocks. For a song which contains N beat intervals, the total number of blocks is $N - B + 1$. The MFCC features are extracted from each block using *discrete Cosine transform (DCT)*, a Fourier transform very similar to the discrete Fourier transform. As with STFT, DCT requires a sliding window size and hop length. The specific parameters for the MFCC extraction are discussed in chapter 6. One MFCC vector is extracted over every sliding window.

The group of 20-dimensional MFCC vectors within one block is called a *20-dimensional point cloud*. A self-similarity matrix (SSM) is constructed from every

point cloud. If we have a cloud $X \in \mathbb{R}^{M \times 20}$ where M is the number of points in the cloud, the derived SSM matrix D has dimensions $M \times M$. As a pre-processing method aiming to normalise for loudness we center the cloud on its mean and scale each point to unit norm:

$$\hat{X} = \{\frac{x - mean(x)}{||x - mean(x)||_2} : x \in X\} \tag{5.5}$$

Where the distance metric is *L2 vector norm*, defined as $||x||_2 = \sqrt{\sum_i x_i^2}$. Each element of the SSM is then calculated using:

$$D_{ij} = |\hat{X}[i] - \hat{X}[j]||_2 \tag{5.6}$$

The length of a beat segment depends strictly of the tempo of a track, therefore it is common that we end up with different SSM dimensions for each song. To enable comparisons between matrices from different songs, all SSMs are resized to a dimension $d \times d$, where d is an empirically selected parameter.

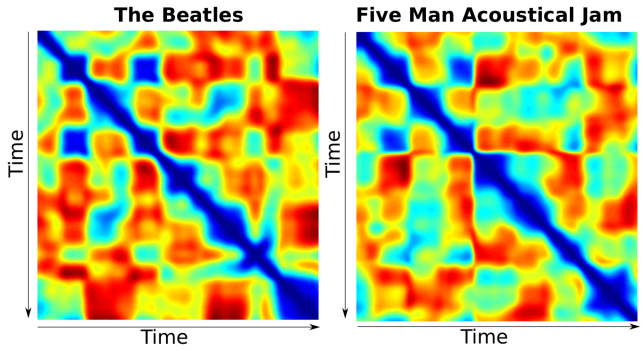

Figure 5.13: Example self-similarity matrices across 4 beats of the song "We Can Work It Out" by The Beatles (left) and its cover version created by Five Man Acoustical Jam (right). Some similarity is clearly visible [56]

The final song representation is a set of all SSMs extracted from a song and ordered by blocks. The matrices for each database song is generated and stored, as it is needed for comparisons with the query song.

When a query song is passed to the timbre rejector, the algorithm extracts the MFCCs and constructs the SSM representation of a song as described above. The query song is then paired with every database song and a cross-similarity matrix (CSM) is created for each pair. By having a set of N SSMs for song A and a

set of M SSMs for song B, the CSM has dimensions $N \times M$, with an element calculated using:

$$CSM_{ij} = ||SSMA_i - SSMB_j||_2 \tag{5.7}$$

$SSMA_i$ and $SSMB_j$ identify the i^{th} block from song A and the j^{th} block from song B respectively. The distance metric is the *Frobenius norm*, where given a matrix X of size $D \times E$:

$$||X||_2 = \sqrt{\sum_{i=1}^{D}\sum_{j=1}^{E}|a_{i,j}|^2 : \ a \in X} \tag{5.8}$$

To obtain a similarity score, we need to binarise the CSM. The binary matrix is computed using mutual fraction κ nearest neighbours, i.e. $B_{ij}^M = 1$ if CSM_{ij} is within the κM^{th} smallest values in row i and also within the κN^{th} smallest values in column j of the CSM [56]. The value of κ is a proportion between 0 and 1.

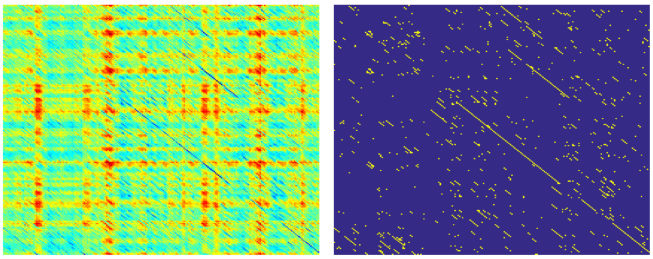

Figure 5.14: Cross-similarity matrix (left) which is then converted to a binary matrix (right). The two songs examined are two versions of "We Can Work It Out" by The Beatles and Five Man Acoustic Jam [56]

The final step of the rejector is to measure similarity using *Smith-Waterman (SW) algorithm*. It presents a technique for detecting the best local alignment between tracks by evaluating the binary matrix from the previous step. The rejector implementation prefers local over global alignment due to the fact that cover versions of songs could add custom intro, outro and bridge sections, but be very similar otherwise to the original. Smith Waterman creates another matrix D from the binary matrix based on the following recursive function:

$$D_{ij} = max \begin{cases} D_{i-1,j-1} + (2\delta(B_{i-1,j-1}^M, 1) - 1) + \Delta(B_{i-2,j-2}^M, B_{i-1,j-1}^M), \\ D_{i-2,j-1} + (2\delta(B_{i-1,j-1}^M, 1) - 1) + \Delta(B_{i-3,j-2}^M, B_{i-1,j-1}^M), \\ D_{i-1,j-2} + (2\delta(B_{i-1,j-1}^M, 1) - 1) + \Delta(B_{i-2,j-3}^M, B_{i-1,j-1}^M), \\ 0 \end{cases} \tag{5.9}$$

The sign δ represents Kronecker delta function, which given two variables i and j it returns values:

$$\delta_{ij} = \begin{cases} 0 \; if \; i \neq j, \\ 1 \; if \; i = j \end{cases} \tag{5.10}$$

Based on the Kronecker delta the term $2\delta(B^M_{i-1,j-1}, 1) - 1$ assigns score of +1 if there is a match between the two songs, and -1 if there is mismatch between both tracks.

The Δ function is called *affine gap penalty* and assigns a score to an identified gap. The scoring takes the form of:

$$\Delta(a, b) = \begin{cases} 0 \; if \; b = 1, \\ -0.5 \; if \; b = 0, a = 1, \\ -0.7 \; if \; b = 0, a = 0 \end{cases} \tag{5.11}$$

The Smith-Waterman algorithm configuration is adapted to prioritise diagonal alignment, rather than horizontal or vertical one. Any gaps that occur on the diagonal are appearing in both songs almost simultaneously in time, which is also a form of similarity. At the same time if large gaps happen in a horizontal or vertical line, they would indicate a large mismatch between both songs.

An illustration of the different matrix evaluated is given in figure 5.15. The resulting SW matrix for the two versions of "We Can Work It Out" is shown on figure 5.16.

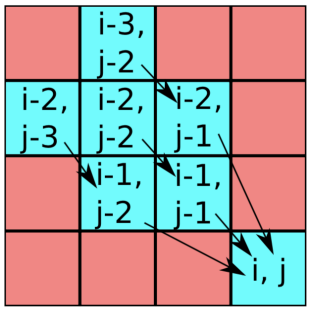

Figure 5.15: SW algorithm traversal

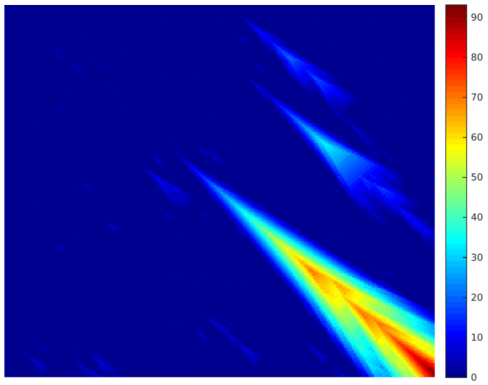

Figure 5.16: Smith-Waterman matrix displaying the scores calculated using equation 5.9

The finalised similarity score assigned after a run of the Smith-Waterman algorithm is the sum of the scores along the longest identified local alignment.

5.6 Aggregated rank rejector

The aggregated rank rejector applies several rank aggregation techniques in an attempt to improve individual results. The algorithm cannot be used on its own to identify cover or live songs.

A *rank aggregation rule* is a function takes the ranks of each database song within the rankings returned by other rejectors. A new rank is then calculated using either the *minimum, maximum, median* or *mean* of the previous rank positions.

The rank aggregation examined combines the results of all rejectors described so far. A diagram of the exact rank aggregation is available on figure 5.17. The original algorithm specification includes rank aggregation on more rejectors - an average chroma and average number of beats weak rejector, as well as a 2D Fourier transform magnitude (2DFTM) rejector [54]. Unfortunately due to lack of sufficient amount of training data the implementation of the missing algorithms is not possible within the scope of this project.

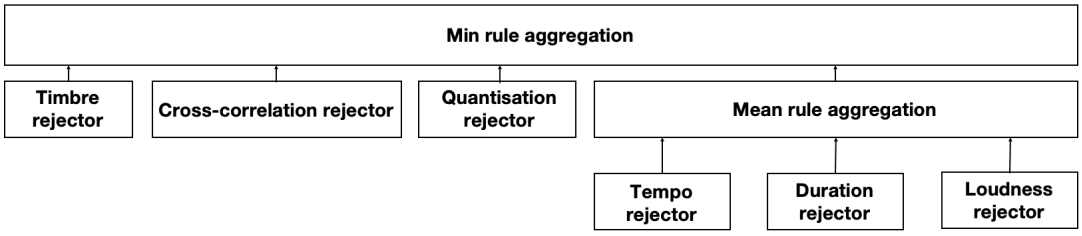

Figure 5.17: Rank aggregation applied to the results of rejectors. The types of rules at each stage are decided based on the evaluation metric results they return

A final rank optimisation step called *local Kemenization* is applied following rank aggregation. Kemenization refines the final aggregated ranking by taking into account all ranking results returned by the participating rejectors. For every pair of adjacent songs (i, j) in the final aggregated ranking, the refinement method checks whether a swap would reduce the *aggregated Kendall measure*. A Kendal measure takes two rejector rankings is defined by:

$$\tau = \frac{n_c - n_d}{n(n-1)/2} \tag{5.12}$$

where n_c counts the number of song pairs (i, j) for which i is ahead of j in both rankings. The count of song pairs that does not satisfy this condition, i.e. j is above i in any of the rankings, is indicated using n_d. The total number of song pairs is defined by n.

The aggregated Kendall measure is defined by the sum of Kendal measures of all ranking combinations.

5.7 Fingerprint rejector

The final rejector examined is originally created to tackle the problem of live song identification. It works on short query segments (usually from 6 to 10 seconds) or *snippets* of live performance recordings. The sources of the recordings vary from official live releases to amateur capturing of a live performance through an audience-based device (smartphone, portable recorder, etc.). The snippets are then compared to a song database through several image processing techniques. It is the only analysed rejector which does not work with full versions of songs.

As an initial step the rejector obtains frequency information using *Constant Q Transform (CQT)*. This technique, similarly to FFT, transforms a signal from the time to frequency domain and produces a spectrogram. While the frequency bins (the intervals of each frequency) in a FFT spectrogram are uniformly spaced out, the bins resulting from CQT are logarithmically spaced and follow the equal tempered scale. This is a better representation of the human perception of frequencies. It also leads to a better frequency resolution for the low and mid frequencies, as well as better time resolution for high frequencies [73].

The computation complexity of CQT is considerably high compared to FFT. The exact implementation is not straight forward and it involves details beyond the scope of my knowledge in the area. A further specification of CQT is therefore unavailable in this report. In the benchmark a library implementation of CQT is used throughout.

The first stage of the rejector is the *fingerprinting stage*. After CQT is applied on a song, the resulting spectrogram is converted into binary image using adaptive thresholding. Threshold is a simple form of image segmentation most frequently used to covert grayscale images into binary form. Thresholding varies in relation to the type of image information used, adaptive thresholding modifies each pixel value based on a local pixel neighbourhood of predetermined size. For a neighbourhood around a pixel (i, j) of matrix X we calculate the median of the neighbouring values:

$$M(i,j) = median\ \{\ X(I,J) \mid i - \Delta_N \leq I \leq i + \Delta_N \text{ and } j - \Delta_N \leq J \leq j + \Delta_N\ \} \tag{5.13}$$

In the median calculation Δ_N is the neighbourhood size. Using the local median value the neighbourhood the point $X(i, j)$ can be binarised:

$$B(i,j) = \begin{cases} 1, & \text{if } X(i,j) > M(i,j) \\ 0, & \text{otherwise} \end{cases} \tag{5.14}$$

By repeating the procedure for each element of the spectrogram we obtain the binary matrix B. The ones in the representation indicate local areas with high sound energy (the peaks of the spectrogram). A binary representation is extracted for every song in the database. In the interest of clarity we refer to the binary matrix of a database song as a *reference fingerprint*.

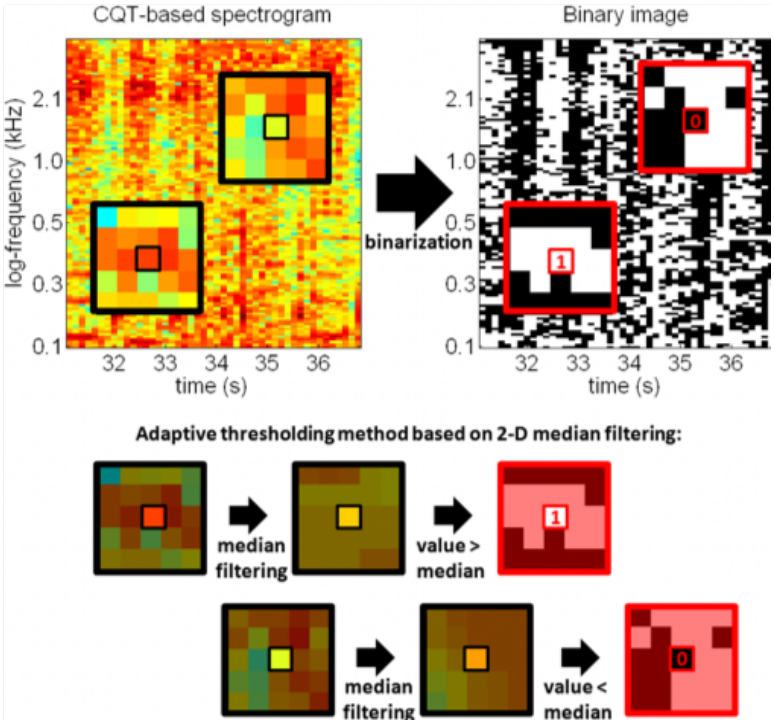

Figure 5.18: Overview of the *fingerprinting* stage of the rejector [57]

The following *matching* stage starts by constructing a similarity matrices from the binary image of the query song and each of the reference fingerprints. The similarity metric used is the Hamming similarity, i.e. the percentage of values that match between both matrices. First, the reference fingerprints are modified using the function $A(x) = 2x - 1$ and then multiplied with the query fingerprint. The product is converted with the inverse of A ($f_{-1} = (x+1)/2$). The resulting matrix is normalised by the frequency dimension of the CQT spectrogram specification, which gives us a matrix containing Hamming similarity scores for each pair of time frames between the query segment and the reference.

After a further binarisation of the Hamming matrix using an empirically chosen pre-defined fixed value, the final similarity score returned by the rejector is calculated using the largest cumulated Hamming similarity defined by a line in the matrix. A line containing the highest sum of similarity values signifies the best alignment between a query segment and a reference fingerprint. Similarly

to the timbre rejector, because one of the dimensions represents alignment along the time domain. Therefore the expected best alignment should correspond to a diagonal line within the Hamming similarity matrix.

To detect best cumulative Hamming similarity in the matrix a method called *Hough transform* is utilised. The technique is a used to detect straight lines within an image (or set of points) using the following line representation on a coordinate system:

$$\rho = c\cos\theta + y\sin\theta \tag{5.15}$$

where ρ represents the shortest distance of the defined line from the origin and θ is the angle between the x axis and the perpendicular from the origin to the line (figure 5.19).

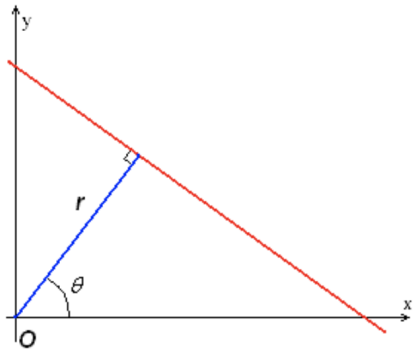

Figure 5.19: Geometric representation of the Hesse normal form of a line

This equation helps us define a line using (ρ, θ). Plotting the set of all values of (ρ, θ) for a fixed point describes a sine curve:

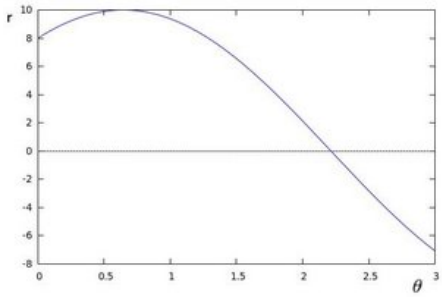

Figure 5.20: Plot of the set of all lines through a single point

Given multiple points the straight line passing through all of them is defined by the point in the (ρ, θ) plane which is the intersection of all sinusoidal curves.

Given a line the result of Hough transform is the set of coordinates of the points lying on the line.

The fingerprint rejector exploits the idea of representing a line using a distance and an angle. A range of values for ρ and θ is defined and the formed lines are passed to a Hough transform. The cumulative Hamming similarity for the line is calculated using the locations within the matrix returned by the transform.

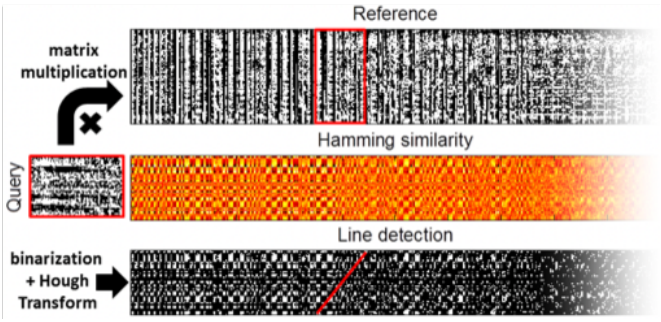

Figure 5.21: Overview of the *matching* phase of the rejector

Chapter 6

Evaluation benchmark

The benchmark implementing the algorithm was targeted for solely personal use, therefore the majority of its functionality is not easily accessible through a form of a user interface. The output that most functions generate is verbose and is intended to assist with the analysis of the inner workings of an algorithm. Overall it is recommended the benchmark to be used as a software library, rather than a standalone tool.

Following on the design of the evaluation task any usage of the benchmark requires a MongoDB instance to store audio features and act as a song database on figure 4.1. A Docker Compose recipe is available in the *misc/docker* directory of the project GitLab repository for quick spawning a MongoDB container.

6.1 Implementation details

As mentioned before the whole benchmark is implemented in Python 3.7. To perform audio feature extraction from a song the benchmark utilises Librosa, a library for audio and music processing [74]. Librosa offers various functions to extract all required audio features using different Fourier transforms, as well as the ability to separate a song into beat frames. The software was also written by the authors of the cross-correlation rejector [55], and its beat tracker implementation is the core of beat tracking for the timbre rejector implementation [56]. Because of that Librosa was the preferred choice for audio analysis library over other options [75] [76].

Connections between the benchmark and the database are accommodated through the official MongoDB Python package called *PyMongo* [77]. The audio feature extraction functionality in the benchmark could potentially serve as a middleware between Librosa and PyMongo in future MIR-related projects.

Other general matrix operations are achieved using *NumPy*, a popular scien-

tific computation Python package [78].

The benchmark is mainly targeted to be run on sets of query songs, rather than individual songs. Each algorithm implementation however has a *run_on_test_set* and *run_on_query_song* functions, which initiate an algorithm runs on a set of query songs or an individual query track.

6.2 Algorithm implementations in the benchmark

This section offers documentation on the exact parameters used during each algorithm implementation. It complements all information from chapter 5.

6.2.1 Weak rejectors

Tempo, loudness and duration weak rejectors have been produced by the benchmark. Each rejector is in the form of a classifier model, with an implementation for ExtraTrees classification provided by the *scikit-learn* Python library [66]. The trained models are stored into the database, so that they can be reused to produce classifications during an algorithm execution. The maximum depth for each tree in the model is set to 20. Due to memory concerns the classifier is trained using batches of data rather than on the full training set at the same time. A tree is added for each training phase using a batch of data. For batches of approximately 42,000 pairs the overall number of trees in the trained model is 1819. The low maximum depth of a tree combined with the number of trees helps reduce model overfitting [54].

The ROC scores returned by the trained models used in the results generation are 0.5 for the duration and tempo classifiers and 0.53 for the loudness classifier.

6.2.2 Cross-correlation rejector

Chroma features for the cross-correlation rejector are extracted using STFT with a window size of 2048 samples and a hop length of 512 samples. The extraction of the beat frames of a song is done using an initial tempo estimate of 240 beats per minute (bpm), a value mentioned in the original algorithm implementation. The cross-correlation of two chromagrams is performed on row pairings using the NumPy *correlate* function. The resulting cross-correlation matrix is normalised by the size of the smaller chromagram so that the maximum possible peak value is 1.

6.2.3 Quantisation rejector

The process of extracting chroma vectors in the quantisation rejector is completely identical to the cross-correlation rejector implementation. The trained K-means model forms 200 cluster centers and uses *k-means++* (initial random selection of first cluster center with every following center selected using a weighted probability based on distance to all previously chosen centers) as an initialisation model. The K-Means clustering implementation was provided by scikit-learn. The training phase involved approximately 42,000 chroma vectors extracted from the train sets of the datasets. The finalised model is again stored into the database for persistence.

The achieved inertia of the best performing K-means model was 5633.

6.2.4 Timbre rejector

The extraction of MFCCs for the timbre rejector was achieved through the *mfcc* function provided by Librosa, which inherently uses a discrete cosine transform (DCT), a Fourier-related transform similar to DFT. The window size of the transform is dependent on the tempo of a song and is determined by the function $\frac{60}{tempo} * sr$, where sr is the sample rate of the song.

The beat tracking takes 3 initial guesses for tempo (60, 100, 180) and 3 dimensions (100×100, 200×200, 300×300) are examined for size of the self-similarity matrix for each track. This leads to the construction of 9 self-similarity matrices (SSMs) for each database or query song.

An implementation of the Smith-Waterman algorithm provided by the creator of the algorithm is directly used to generate similarity scores.

The current version of the timbre rejector runs considerably slower compared to the other audio similarity algorithms. While a run of the rest of the algorithms usually takes a relatively short time (up to 30 minutes per dataset) the timbre-based algorithm takes a variable time between 2 and 6 hours to return a result for a single query song of the dataset. A potential reason for the long run time could be the high computational costs associated with handling 9 SSM representations for a single database song. Each SSM represented as a NumPy matrix takes a lot of storage space - the total size of the SSMs for all database songs for the *covers80* dataset takes almost 90 GB in MongoDB. Retrieving such a significant amount of data from the database for every query song, combined with the construction of a cross-similarity matrix for each SSM results in a very slow task. Even utilising lazy loading, a pre-processing step required due to all SSM results from the database being too big to fit in memory, does not result in any noticeable improvement of the time for execution.

6.2.5 Rank aggregation rejector

The rank aggregation rejector takes advantage of the *min, max, median* and *mean* functions in NumPy to determine an aggregated rank from an array of ranks.

6.2.6 Fingerprint rejector

As per algorithm specification, the constant Q transform (CQT) applied on query segments and database songs uses a window size of 512 samples, a range of 120 frequency channels and a minimum frequency of 130.81 Hz (equivalent to the C3 note). The conversion of the resulting spectrogram to a binary image is performed using a adaptive thresholding implementation provided by the scikit-image, a Python image processing library. The neighbourhood sizes considered are 15 and 35. Hamming similarity is calculated using the *hamming* function from *SciPy*, another scientific calculation package.

6.3 Benchmark result format

The result of an algorithm run on a query song is stored as an entry in a MongoDB collection. Apart from the actual similarity scores each entry also includes metadata allowing for easy understanding of the results:

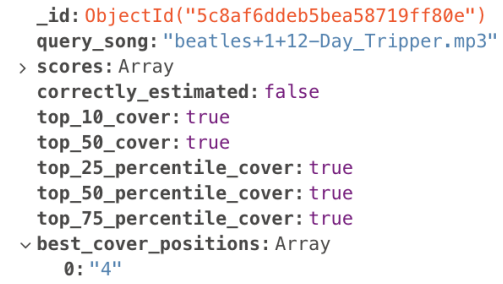

Figure 6.1: An example of an algorithm result for a single query song. The representation accommodates for a quick retrieval of the *Top-K* and *Text-P* evaluation metrics

The verbose similarity scores are available in the *scores* array:

Figure 6.2: Verbose scores stored in the database

The benchmark supports exporting the evaluation metrics defined in chapter 4 as CSV files.

Chapter 7

Results and experimentation

Summary

Details all the results of your study here (exploits graphics for results visualisation). This chapter should also contain a full discussion, interpretation and evaluation of the results. wha what

7.1 Best results

7.2 Results analysis

7.3 Experimentation

Chapter 8

Further work

Details all the results of your study here (exploits graphics for results visualisation). This chapter should also contain a full discussion, interpretation and evaluation of the results.

Chapter 9

Challenges

Throughout the time frame of the project several challenges were encountered and overcome to the best of my abilities. Some of them heavily influenced the algorithms analysis, implementation and achieved results. This section describes these obstacles.

9.1 Lack of supporting academic infrastructure

The task of cover song identification is still a relatively new one within the MIR area. Active contributions to it are significantly less than other admittedly more fundamental and important problems such as chord estimation or beat tracking [42].

Because of this relative lack of innovation, the task suffers from shortage of good evaluation criteria. The only metrics used to examine the performance of an algorithm are the ones used in the *Audio Cover Song Identification* part of the MIREX conference (and consequently this project). Their current definition however fails to establish a reliable method of comparing results from different datasets, as discussed in Chapter 7.

The amount of big publicly available datasets is also insufficient. Due to copyright concerns most datasets decide to provide extracted audio features from a set of songs rather the songs themselves. While it has been established which features perform best in relation to the task (eg. chroma features), the lack of the original audio streams applies an inherent limitation of the flexibility and originality of a future research project. The biggest dataset of features from cover songs is the *SecondHandSongs (SHS)* dataset [60]. It is sufficiently big in size (18,196 tracks in total), however it is not easily obtainable as it is distributed as part of the *Million Song Dataset* - a dataset consisting of 1,000,000 songs and having a size of around 300 GB. Because of its large magnitude the dataset

is available only to researchers using cloud-based virtual machines to interact with it. Apart from it the only other publicly available datasets include *covers80* (used in this project) and *covers1000* (offering a small number of audio features extracted from 1000 songs).

9.2 Project scope

The general project scope was initially underestimated by a margin, possibly due to my inexperience in the area of digital signal processing (DSP) prior to starting the project. The overall increasing complexity of it was unexpected and the required theoretical background and development skills presented many challenges. Almost all of them were overcome through thorough research into related concepts and techniques. Consequently this analysis on the broader DSP field contributed to the depth of knowledge the project aims to achieve.

Despite my best efforts some concepts still present a challenge for me to understand.

9.3 Ambiguity in the algorithm papers

Implementing all algorithms was another significant hurdle to prevail over. The main purpose of each paper describing a new way of identifying cover or live songs is to inform about the introduced innovations and the results achieved as a consequence. A lot of implementation details are omitted which makes the task of reproducing the algorithm one that requires a lot of experimentation and improvisation.

Some academics have shared their algorithm codebase online - this proved to be very helpful in filling the empty spaces of the algorithm specification defined in the published paper.

Chapter 10

Project management

A successful execution of the project required careful time management and organisation of priorities. This section briefly outlines how this was achieved.

10.1 Using GitLab

GitLab was the primary tool used for project management. The system offers functionality of project tracking through Issues and Milestones. Issues were used to define small individual tasks of the benchmark development, such as implementing fingerprinting using constant Q transform, splitting the datasets to train and test sets, etc. Any progress to a task was logged using comments under the corresponding issue. GitLab milestones were used to group issues based on the corresponding algorithm they relate to.

The git repository provided for the project on GitLab served as a main backup and version control destination. Development of separate benchmark features was done using *feature branches* - git branches which encapsulate the implementation process of a functionality before being merged to the master branch after the feature is complete. Regular commits were regularly pushed to store updated versions of the codebase.

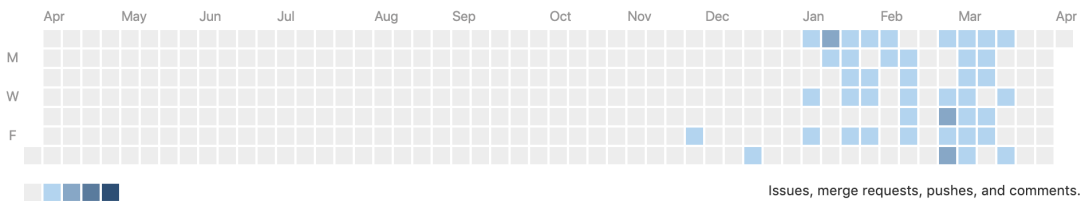

Figure 10.1: Overview of the frequency of commits per day of the month. Other related graphs are available in the appendix.

10.2 Canvas logs

All Canvas logs have been submitted with information about what was done each week. Any other logging was done through GitLab.

10.3 Supervisor meetings

Productive meetings with my supervisor took place each week. During each of them the progress of the project was evaluated, followed by a discussion on possible direction and targets for the upcoming week. The meetings helped me and the project preserve its aim and focus towards the end goal.

Chapter 11

Conclusion

Conclusions should summarize the problem, the solution and its main innovative features, outlining future work on the topic or application scenarios of the proposed solution.

References

[1] E. Weinstein and P. Moreno, "Music identification with weighted finite-state transducers," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, vol. 2, pp. II–689, IEEE, 2007. 1

[2] A. Wang *et al.*, "An industrial strength audio search algorithm.," in *Ismir*, vol. 2003, pp. 7–13, Washington, DC, 2003. 1

[3] J. Haitsma, T. Kalker, and J. Oostveen, "Robust audio hashing for content identification," in *International Workshop on Content-Based Multimedia Indexing*, vol. 4, pp. 117–124, Citeseer, 2001. 1

[4] D. P. Ellis and G. E. Poliner, "Identifyingcover songs' with chroma features and dynamic programming beat tracking," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, vol. 4, pp. IV–1429, IEEE, 2007. 1, 27

[5] T. Bertin-Mahieux and D. P. Ellis, "Large-scale cover song recognition using hashed chroma landmarks," in *2011 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pp. 117–120, IEEE, 2011. 1

[6] D. P. Ellis and B.-M. Thierry, "Large-scale cover song recognition using the 2d fourier transform magnitude," 2012. 1

[7] Spotify, "Music for everyone — Spotify," 2019. [Online; accessed 29-March-2019]. 2, 22

[8] Apple, "Music — Apple (UK)," 2019. [Online; accessed 29-March-2019]. 2

[9] SoundCloud, "Soundcloud —- Listen to free music and podcasts on SoundCloud," 2019. [Online; accessed 29-March-2019]. 2

[10] M. Müller, "Short-time fourier transform and chroma features," 3, 26

[11] Wikipedia contributors, "Musical tone — Wikipedia, the free encyclopedia," 2019. [Online; accessed 31-March-2019]. 4

[12] Acoustic Glossary, "Sound power - acoustic glossary - article," 2019. [Online; accessed 29-March-2019]. 4

[13] The Physics Hypertextbook, "Music & noise - the physics hypertextbook," 2019. [Online; accessed 31-March-2019]. 4

[14] A. Klapuri and M. Davy, *Signal processing methods for music transcription*. Springer Science & Business Media, 2007. 4, 5

[15] H. Helmholtz, *On the sensations of tone*. Courier Corporation, 2013. 4

[16] H. Fletcher, "Loudness, pitch and the timbre of musical tones and their relation to the intensity, the frequency and the overtone structure," *The Journal of the Acoustical Society of America*, vol. 6, no. 2, pp. 59–69, 1934. 4

[17] A. S. of America. Secretariat and A. N. S. Institute, *American National Standard Psychoacoustical Terminology*. American National Standard, American National Standards Institute, 1986. 4, 5

[18] R. W. Young, "Terminology for logarithmic frequency units," *The Journal of the Acoustical Society of America*, vol. 11, no. 1, pp. 134–139, 1939. 4

[19] MIDI, "The complete midi 1.0 detailed specification," 2019. [Online; accessed 31-March-2019]. 5, 25

[20] Wikipedia contributors, "Octave — Wikipedia, the free encyclopedia," 2019. [Online; accessed 4-April-2019]. 5

[21] Engineering Toolbox, "Sound pressure," 2019. [Online; accessed 31-March-2019]. 5

[22] R. Erickson, *Sound structure in music*. Univ of California Press, 1975. 5

[23] Wikipedia contributors, "Envelope (music) — Wikipedia, the free encyclopedia," 2019. [Online; accessed 31-March-2019]. 5

[24] Wikipedia contributors, "Timbre — Wikipedia, the free encyclopedia," 2019. [Online; accessed 8-April-2019]. 6

[25] Diemo Schwarz, "Spectral Envelopes," 2019. [Online; accessed 8-April-2019]. 6

[26] B. Benward, *Music in Theory and Practice Volume 1*. McGraw-Hill Higher Education, 2014. 6

[27] The Fourier Transform, "Fourier transform," 2019. [Online; accessed 31-March-2019]. 7

[28] The Fourier Transform, "Fourier series," 2019. [Online; accessed 31-March-2019]. 7, 8

[29] Zachary S Tseng, "Second order linear partial differential equations," 2019. [Online; accessed 31-March-2019]. 8

[30] Wikipedia contributors, "Fourier transform — Wikipedia, the free encyclopedia," 2019. [Online; accessed 31-March-2019]. 8

[31] Wikipedia contributors, "Periodic summation — Wikipedia, the free encyclopedia," 2019. [Online; accessed 6-April-2019]. 8, 9

[32] Wolfram MathWorld, "Fast fourier transform," 2019. [Online; accessed 4-April-2019]. 9

[33] Wikipedia contributors, "Fast fourier transform — Wikipedia, the free encyclopedia," 2019. [Online; accessed 4-April-2019]. 9

[34] Wikipedia contributors, "Spectrogram — Wikipedia, the free encyclopedia," 2019. [Online; accessed 2-April-2019]. 10

[35] Betty Lise Anderson, "Audio filters," 2019. [Online; accessed 1-April-2019]. 10

[36] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937. 10

[37] Wikipedia contributors, "Mel scale — Wikipedia, the free encyclopedia," 2019. [Online; accessed 1-April-2019]. 11

[38] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1, pp. 278–282, IEEE, 1995. 11

[39] Z.-H. Zhou, *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012. 11

[40] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, vol. 1. Springer series in statistics New York, 2001. 12

[41] Wikipedia contributors, "Quantization (signal processing) — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Quantization_(signal_processing)&oldid=889000626, 2019. [Online; accessed 1-April-2019]. 12

[42] MIREX, "Mirex home." https://www.music-ir.org/mirex/wiki/MIREX_HOME, 2019. [Online; accessed 1-April-2019]. 13, 47

[43] T. Fujishima, "Real-time chord recognition of musical sound: A system using common lisp music," *Proc. ICMC, Oct. 1999*, pp. 464–467, 1999. 15, 23

[44] Wikipedia contributors, "Harmonic pitch class profiles — Wikipedia, the free encyclopedia," 2016. [Online; accessed 2-April-2019]. 15

[45] Wikipedia contributors, "Dynamic time warping — Wikipedia, the free encyclopedia," 2019. [Online; accessed 2-April-2019]. 16

[46] L. Muda, M. Begam, and I. Elamvazuthi, "Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques," *arXiv preprint arXiv:1003.4083*, 2010. 16, 29

[47] Wikipedia contributors, "Normalized compression distance — Wikipedia, the free encyclopedia," 2019. [Online; accessed 2-April-2019]. 16

[48] S. A. Terwijn, L. Torenvliet, and P. M. Vitányi, "Nonapproximability of the normalized information distance," *Journal of Computer and System Sciences*, vol. 77, no. 4, pp. 738–742, 2011. 16

[49] Python, "Python," 2019. [Online; accessed 3-April-2019]. 17

[50] MongoDB, "Mongodb," 2019. [Online; accessed 3-April-2019]. 17

[51] D. P. W. Ellis, "The covers80 cover song data set," 2019. [Online; accessed 3-April-2019]. 19

[52] Martin Angelov, "mod-ug-proj-2018/mxa487," 2019. [Online; accessed 5-April-2019]. 19

[53] J. Osmalsky, J.-J. Embrechts, P. Foster, and S. Dixon, "Combining features for cover song identification," in *16th International Society for Music Information Retrieval Conference*, 2015. 20, 21

[54] J. Osmalsky, M. Van Droogenbroeck, and J.-J. Embrechts, "Enhancing cover song identification with hierarchical rank aggregation," in *Proceedings of the 17th International for Music Information Retrieval Conference*, pp. 136–142, 2016. 20, 21, 35, 41

[55] D. P. Ellis, C. V. Cotton, and M. I. Mandel, "Cross-correlation of beat-synchronous representations for music similarity," in *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 57–60, IEEE, 2008. 20, 40

[56] C. J. Tralie and P. Bendich, "Cover song identification with timbral shape sequences," *arXiv preprint arXiv:1507.05143*, 2015. 21, 29, 30, 32, 33, 40

[57] Z. Rafii, B. Coover, and J. Han, "An audio fingerprinting system for live version identification using image processing techniques," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 644–648, IEEE, 2014. 21, 37

[58] Elias Leight, "Producers, songwriters on how pop songs got so slow," 2019. [Online; accessed 3-April-2019]. 22

[59] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006. 22

[60] Labrosa, "The secondhandsongs dataset — million song dataset," 2019. [Online; accessed 4-April-2019]. 22, 47

[61] Wikipedia contributors, "Receiver operating characteristic — Wikipedia, the free encyclopedia," 2019. [Online; accessed 4-April-2019]. 22

[62] R. X. Gao and R. Yan, "Non-stationary signal processing for bearing health monitoring," *International journal of manufacturing research*, vol. 1, no. 1, pp. 18–40, 2006. 23, 24

[63] Librosa, "librosa.beat.beat_track," 2019. [Online; accessed 5-April-2019]. 26

[64] J. Osmalskyj, S. Piérard, M. Van Droogenbroeck, and J.-J. Embrechts, "Efficient database pruning for large-scale cover song recognition," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 714–718, IEEE, 2013. 28

[65] J. Serra and E. Gómez, "Audio cover song identification based on tonal sequence alignment," in *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 61–64, IEEE, 2008. 28

[66] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. 29, 41

[67] R. Martin and C. Burley, "Power cepstrum technique with application to model helicopter acoustic data," 1986. 29

[68] O. Viikki and K. Laurila, "Cepstral domain segmental feature vector normalization for noise robust speech recognition," *Speech Communication*, vol. 25, no. 1-3, pp. 133–147, 1998. 29

[69] Wikipedia contributors, "Cepstrum — Wikipedia, the free encyclopedia," 2019. [Online; accessed 5-April-2019]. 30

[70] Ricardo Gutierrez-Osuna, "L9: Cepstral analysis," 2019. [Online; accessed 5-April-2019]. 30

[71] John Coleman, "Cepstral analysis," 2019. [Online; accessed 5-April-2019]. 30

[72] A. M. Noll, "Cepstrum pitch determination," *The journal of the acoustical society of America*, vol. 41, no. 2, pp. 293–309, 1967. 31

[73] C. Schörkhuber and A. Klapuri, "Constant-q transform toolbox for music processing," in *7th Sound and Music Computing Conference, Barcelona, Spain*, pp. 3–64, 2010. 36

[74] LabROSA, "Librosa," 2019. [Online; accessed 6-April-2019]. 40

[75] Aubio, "Aubio," 2019. [Online; accessed 6-April-2019]. 40

[76] pyAudioAnalysis, "pyaudioanalysis," 2019. [Online; accessed 6-April-2019]. 40

[77] MongoDB, "Pymongo," 2019. [Online; accessed 6-April-2019]. 40

[78] Travis Oliphant, "NumPy: A guide to NumPy." USA: Trelgol Publishing, 2006. [Online; accessed 6-April-2019]. 41

[79] J. G. Roederer, *The physics and psychophysics of music: an introduction*. Springer Science & Business Media, 2008.

[80] Wikipedia contributors, "Fourier series — Wikipedia, the free encyclopedia," 2019. [Online; accessed 31-March-2019].

[81] MIREX, "2018:audio cover song identification." `https://www.music-ir.org/mirex/wiki/2018:Audio_Cover_Song_Identification`, 2018. [Online; accessed 13-November-2018].

[82] J. Serra and E. Gómez, "A cover song identification system based on sequences of tonal descriptors," *Music Information Retrieval Evaluation eXchange (MIREX)*, 2007.

[83] J. Serra, E. Gómez, and P. Herrera, "Improving binary similarity and local alignment for cover song detection," *MIREX extended abstract*, 2008.

[84] J. Serra, M. Zanin, and R. G. Andrzejak, "Cover song retrieval by cross recurrence quantification and unsupervised set detection," 2009.

[85] Wikipedia contributors, "Discrete-time fourier transform — Wikipedia, the free encyclopedia," 2019. [Online; accessed 4-April-2019].

[86] OpenCV, "Hough lines," 2019. [Online; accessed 6-April-2019].