Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingeneria Informatica

tesi di laurea

Scalable cover song identification based on melody indexing

Relatore: Prof. Nicola Orio

Laureando: Leonardo Romanato

Summary

In the last years, thanks to the growing market of smart-phones, tablets and others similar devices, access to applications for music information retrieval became easier for a wide variety of users. Among the others, many applications gained popularity offering services of music recognition. Basically these applications offer a service that, starting from a little excerpt of sound, can retrieve the song from which the excerpt belongs. The effectiveness of state of the art systems is really good if the recognition is carried out on studio version of the songs, but there is still an open field of research on the recognition of cover and live versions.
There are some difficult aspects to care about cover and live performance recognition: change in tonality, different instruments used, changes in the song structure, different tempo. In the case of pop and rock songs usually a lot of differences are noticeable between a cover and the original song.
In this work, we describe an efficient method for cover song identification focusing our target on pop and rock music genres. The procedure proposed is based on the fact that every pop/rock song has usually a main melody or a easily recognizing theme inside. Usually this theme or this melody, even if the cover song is really different from the original, is present in every version of the original song. This means that if we can identify the melody in each song we can identify also the original song.
First we searched for a good melody and main theme recognition using different tools available, then we tried different methodologies to obtain a good representation of the melody data. After these phases we had some data for every song that was possible to index and retrieve using Falcon, a search engine for music identification. At the end we tried to evaluate performance of the new system searching for the optimal configuration.
Test showed that the chosen approach can achieve encouraging results, opening the way to a deeper exploration of the system. Different parameters' configurations tried gave an idea of system's potentialities.

Contents

Chapter 1

Introduction

Music has always been present along human history. Starting from the prehistoric period when first humans used percussive sounds to celebrate rituals and ceremonies until nowadays where every single teenagers along the street is probably carrying an MP3 player, music is something that accompanies our lives and moments, with the incredible ability to change our mood simply using 7 notes.
Music styles really changed and evolved during history but the same we can say for music distribution methods.
Especially with the recent evolution of technologies the way people listen and have access to music change a lot, even compared to few years ago. The impact that digitalization had on music environment is not comparable to other arts: while the access to digital music allows users to enjoy the listening experience even more than what they were doing with analogic recordings is not possible to say the same for sculpture or painting; the possibility to see a photo of a sculpture or a paint, even in a 3D version, it can not substitute the direct interaction with the real work.
Thanks to large distribution of MP3 players, smartphones, tablets and other kind of devices, everybody can have several gigabytes of music directly in some centimeters inside a pocket. This fact, combined to the wider access that everybody has to Internet's resources, sometimes directly from the device they use to listen to music, gives to the people the possibility to access to an incredible amount of recordings.
Digitalization made also some really big changes even in the way music is created. Nowadays, with few money and a little bit of passion, the field of music creation, even with professional results, is accessible to everybody. This means that there is more music on the market now and, since the creation tools are everyday easier to use, there will be a consistent music creation increment year by year.
All of these changes within these years made the music market changing its face. While we face a crisis for music sellers at the same time we can notice a big

interest, and also a big economic profit, for who is involved in music services. On-line personalizable radio like Spotify or GrooveShark had a great success in this period and their economic profile can confirm the interest of people in these services. The same thing we can say for famous and popular applications for smartphones and portable devices like Shazam or SoundHound. These applications permit to everybody having a compatible device with a microphone to records a little audio excerpt from a songs and, after a query on the application's server, to obtain the name of the song and the author, usually correlated with different links to buy on-line the MP3 file. Both the applications mentioned above had in the last year a great success and some of the most interesting economic relevance in the portable devices applications market. In particular SoundHound won the title "Top Paid iPad App of All Time" in January, 2011 while Shazam surpasses in December, 2010, 100 Million Users, becoming one of world's leading discovery services.

1.1 Music Information Retrieval and applications

All the applications that we mentioned before use large *music digital libraries*. To deal with them they have to find specific and effective techniques being capable of indexing and retrieving music documents. The interdisciplinary science that deal with retrieving information from music documents is the *music information retrieval* (MIR). Some of the aspects involved in this branch of inormation retrieval are:

- music analysis and knowledge representation;
- applications of music identification and recognition (score following, automatic accompaniment, routing and filtering for music and music queries, etc.);
- computational methods for classification, clustering, and modeling like musical feature extraction, similarity and pattern matching management, information retrieval, etc.;
- software for music information retrieval like semantic web and musical digital objects, intelligent agents, web-based search and semantic retrieval, query by humming, etc.;
- music archives, libraries and digital collections;
- sociology and economy of music like music industry and use of music information retrieval in the production, distribution, evaluation of music IR systems, building test collections.

MIR is involved in a lot of different tasks and activities and there are many related disciplines to it. Anyway, one of the more challenging research field is the content-based document analysis and retrieval. In particular the *query by*

example paradigm has attracted a large amount of attention. This task consists, given a query in form of a music excerpt, to retrieve all the elements (may be scores or other audio excerpts), from a dataset or a digital library, that contain the query or have a similarity with it. The main idea of this problem is to have an unknown excerpt (in some meaningful sense), in order to discover more information about it, eventually finalized to index and store the new element in the library.
Some of the possible applications concerning the solution of this problem are described below.

Indexing of audio archives

One of the possible scenario it can be observed inside radio and television companies. Usually these companies have thousand of hours of live recording of music performances or rehearsals, stored in analog format on tape. Documents stored in this format are object of audio degradation and eventually of complete inaccessibility with the risk of losing an enormous amount of data. Also, this kind of recordings usually lacks of information about contents or even about who played the content. Manual labeling of these data require a considerable amount of time and musically trained people, which means an high cost for the company. An automatic tool which can be able to solve the *query-by-example* problem, would make the job completely automatic, or at least much more easier for the operator.

Automatic recognition of a song in real time

The possibility for somebody to know in real time which is the unknown song that they are listening to is one of the most famous situation nowadays thanks to some popular applications for portable devices. These applications solve the *query-by-example* problem in real time using the last technologies available. They can query a remote database obtaining in less than a minute a list of possible results granting a good reliability. Also, offering the possibility to buy on-line a digital version of the song, there is the concrete option to create a new market where the user can buy what he likes at the exact moment when he hears it somewhere.

Recognition of live performances for calculation of artists' profits

Lot of live shows are played around every day and it is physically impossible to control everyone of them to list and archive what it is played. This means that, for a lot of shows where no controls are made, money that has to be paid to songwriters is lost because people are not declaring what they play or people declare false information to gain money for themselves, while playing songs of somebody else. An application that can recognize automatically every song

played in a concert will help the authority which controls the music environment to distribute in a equal way the money due to the authors who retain the song's rights.

1.2 Recognition problem for live and cover songs

The *query-by-example* paradigm could be exploited in several contexts. As said before, the recognition of unknown audio excerpts is a deeply explored and well solved task. Unfortunately we can not say the same when we have a cover song or a live version of a song and we want to know which is the original song and the information about it. Live versions and cover songs present some different problems that must be take into account when we perform recognition: changes in tonality, changes in tempo, variations on the musical structures, different kinds of voice and instruments.
Taking a look on different papers and proposed work published within last few years is easy to see how actual is the problem of cover song identification. Different approaches to the problem were proposed. One of the best is the work submitted by Joan Serrà [1]. The proposed system exploits nonlinear time series analysis tools and standard methods for quantitative music description, and it does not make use of a specific modeling strategy for data extracted from audio. The system detailed in [1] presents the highest accuracies in the MIREX [2] "audio cover song identification task" to date (this includes the 2008, 2009 and 2010 editions).
Talking about classical music, the task of recognizing cover versions has been solved by different approaches, one is documented in [3]. We want to start from tools used in this approach to develop another one and trying to solve cover song identification for *pop and rock cover songs*.

1.2.1 Proposed Approach

In this work we want to use tools currently available for automatic audio recognition and, changing the features they used to perform indexing and retrieval, develop a new approach for cover song identification. With this goal in mind we propose, instead of using CHROMA [4] features, to index and retrieve songs using as descriptor the main melody or the main theme of every song. This because in pop and rock song there is always a relevant part, usually the melody sung in the chorus or a particular theme played by an instrument, that is used to obtain the "stuck-in-your-mind" effect, typical for these kind of music. Usually this really well-known and repetitive part is something that is never missing from a cover because it is the characterizing part of the song.
Other works adopt a similar approach to exploit the task of cover song identification. A number cover song identification systems use melody representations as a main descriptor ([5]; [6]; [7]; [8]). As a first processing step, these systems need to extract the predominant melody from the raw audio signal ([9]; [10]).

Melody extraction is strongly related to pitch perception and fundamental frequency tracking, both having a long and continuing history ([11]; [12]).
In our work we want also to develop a representation system that can be robust to all the problems we cited before for live version (tonality changes, tempo variations, etc.). In the end we want to underline how the representation we chosen can effectively achieve efficiency on our system. We are using the melody not for a simple *audio matching* task but we consider it as our key to index and retrieve songs. This means that we don't need the expensive computation necessary to the perfect match of two audio parts but that our melody representation is directly the key used to perform the research on the dataset.

1.3 Thesis structure

The thesis is structured as follow:

Chapter 2: describes the process used to extract pitch, and more in general the melody, from songs, giving an overview to all the tools tested and explaining in a more detailed way how the tool chosen is configured.

Chapter 3: describes the sequence of steps used to create an appropriate melody representation. Starting from the values detected with mirpitch different phases are used: pitch post-processing, intervals calculation, n-grams creation.

Chapter 4: describes the different experimental tests on the algorithm are described, by, particularly, taking care of the parameters that affect mostly the final result.

Chapter 5: gives some conclusions about the project and possible future works are treated.

Chapter 2

Melody Recognition

This chapter describes an approach to perform the recognition of the melody and main theme for a song, the tools we used and how we parametrized them. Also we will present a description of how the tool we chosen perform the pitch recognition.

2.1 What Melody is

Melody is a word that can take different meanings changing the area were it is used. Anyway most of the time the concept of melody is associated to a sequence of pitch notes. These definitions can be found in licterature: "A combination of a pitch series and a rhythm having a clearly defined shape" [13], and on Grove Music: "Pitched sounds arranged in musical time in accordance with given cultural conventions and constraints" [14]. In [15], melody is defined with some of these different connotations: "As a set of musical sounds in a pleasant order and arrangement as a theme in a music structure". The last definition we cited is the nearest thing we are searching for in our work. The concept of main melody that can be remembered within a song carry itself the underlined concepts of *pleasant order* and *arrangement as a theme*. Of course, everybody has a subjective perception of what can be figured out as melody or not but, given different songs with a melodic repetitive pattern placed in focal points, when we ask to identify the main theme, probably the result will be the same from every person. What we want to extract is exactly the characterizing melody, as defined above, using an automatic tool that take in input any kind of audio excerpt belonging to a song.

2.2 Pitch Recognition

Pitch recognition is the process used to identify the value of a sound. By value of a sound we intend the value of the absolute frequency of that sound. For example, given the record of a guitar playing a A4 note the pitch recognition tool will return a value of 440Hz if the guitar is tuned using the standard tuning convention. Also it is necessary to say that a sound can also have variations during time. This means that, taking the same example of the guitar presented above, a guitar perfectly tuned playing a A4 note can have a 440Hz pitch if it is considered the mean pitch value along the whole note but, if it is analyzed the note every 10ms for example is it possible to find some pitch variations that will stay around 440Hz. This is influenced from the different phases that usually a note has: attack, release, decay. For percussive strings instruments, like the guitar, the attack phase is usually out of tune due to the percussive movement on the string and then it will stabilize its value during the release and the decay. The same we can say for the voice. A singer is usually using the attack phase of the note, where he hears the note he is singing, to tune himself. Many different effects also affect the note pitch: tremolo, vibrato, modulation. As it is possible to see, pitch recognition is not an easy task even talking about monophonic recordings. Pitch recognition on polyphonic record will be a task even harder, especially considering a live situation. Many notes are summed together (bass, guitar, voice, keyboards, etc.) and also "non" sounds are present and disturb the record (general noise, crowd noise, drums, percussions).
In the work explained in this thesis we will not try to recognize the perfect pitch of every single time frame of a song but we will focus our attention on the concept of main melody or main theme of a song.
Especially in pop and rock music every song is characterized by a easy recognizable theme or melody and it is usually sung or played like a kind of solo by some instruments. Also, by listening to the song, this theme or melody is usually well audible in the records and is some decibel louder than the background music. Even if for our perception is easy to recognize a repetitive theme and identify it as an important feature for the song, this is not a really easy task for an automatic tool that must perform this recognition.
We start now showing different tools we tried in this work to extract the pitch melody, giving also some examples for everyone of them.

2.3 Approaches Analyzed

We will show now three different tools we tried to perform in a first phase pitch recognition on different kind of songs. These tools are the only ones available for free on-line. Even if different commercial products that included pitch detection function are available, only these have been created to be used for pitch detection only. Moreover, the way used by these tools to output results, permits an easy post-processing phase for pitch values. For everyone of them we performed a general test to have a global overview of which one

of them was the best of the three. We submitted to the tools different kinds of audio excerpts containing previously well known melodies and we analyzed results returned. The excerpts we used are:

- Jeff Buckley - Hallelujah (male voice and guitar)
- Giorgia - E Poi (female voce and piano)
- Celine Dion - My Hearth Will Go On (female voice and band)
- Vasco Rossi - Alba Chiara (male voce and band)

All the excerpts are live versions. The length of each excerpt is between 4 and 19 seconds. All of these files had a sampling frequency of 22050 Hz and they are monophonic recordings. All the files were in WAV format.

2.3.1 YIN

YIN is a pitch recognizer tool developed by Alain de Cheveigné [16]. It is a Matlab tool and the software is distributed with a directory with two .m files, one that contains the executable function: yin.m, and another one that perform a test with two different sound files included in the packet: testyin.m. Further details on YIN implementation are available in [17].
This tool is really easy to use and it offers the possibility to change different parameters. Anyway to modify them you need to edit directly the file yin.m changing manually every time the value of every variable. We tried the tool with the standard configuration.

2.3.2 LabRosa Melody Analyzer

LabRosa Melody Analyzer is a tool developed by Graham Poliner and Dan Ellis. This tool was developed for the audio melody extraction contest at MIREX, year 2005 [2] and it came in a hybrid version of Matlab functions and a precompiled Java routine. You can find the tool here [18]. The tool outputs a text file containing all the frequencies recognized on the audio excerpt, one every 0.10 seconds. The documentation of this tool is really poor and we found hard to modify parameters to obtain different results or using different values for time intervals in pitch detection. Also the computation is really long and it takes around two minutes to compute an audio excerpt of 4 seconds. We decided to not take further test on this tool and this is the reason why you will not find any result coming from LabRosa Melody Analyzer in the Test Result section.

2.3.3 MIR ToolBox - mirpitch

MIR ToolBox is a complete suite of audio tools developed by Olivier Lartillot, Petri Toiviainen and Tuomas Eerola. The suite is a set of Matlab files and everything is working inside the Matlab environment. The tool is open source

and distributed under GNU General Public License version 2. You can find more info on MIR ToolBox and the link to download it here [19]. MIR ToolBox offers a really good guide to users [20] and an mailing-list where Olivier Lartillot usually replies very quickly to every question [21].
In particular for our work we focus our attention on the module mirpitch. This module is in charge of detecting pitch of an audio file and it returns results using absolute frequencies values. For the preliminary test we used this tool with the standard configuration.

2.4 Test Results

In this section we proposed the results of some tests made with YIN and the mirpitch module. The command used to obtain this graphs are:

YIN: `yin 'nameOfTheSong.wav'`

mirpitch: `mirpitch('nameOfTheSong.wav','Frame','Max',600)`

About the command used for mirpitch, using it without `Frame` option returns a single pitch value that is the mean of the pitch samples calculated all along the audio excerpt. The command `Max` is used to restrict the view on the graph, showing only the frequencies from 0 Hz to 600 Hz in the y-axis.
About figures 2.1, 2.2, 2.3, 2.4, the first graph represents pitch values as a function of time (in octaves re: 440 Hz); Different colors represent how much is reliable the sample: more close to blue is the color more reliable is the sample. The second graph represents the aperiodicity of each frame. The third graph represents the power of each frame.
About figures 2.5, 2.6, 2.7, 2.8, the graph represents pitch values as a function of time. Values are represented as absolute frequencies. If more than a pitch value is detected for a single frame, pitch values are represented using different colors.

As it possible to see directly from figures mirpitch offers a better representation of the melody we are searching for. The blue points' values on the graph return graphically the melody direction. Only the last graph presented in figure 2.8 has a really noisy representation but the sung melody is still identifiable in the range from 200Hz to 400Hz. Anyway, after the refining of the parameters for this tool results become better. Results coming from graph had a final confirm watching at the numerical results given by the two tools: mirpitch is slightly more accurate than YIN for the kind of application that we need. Mirpitch resulted also faster than YIN by some seconds of difference. To analyze the excerpt of *E Poi - Giorgia* mirpitch taken 5.4 seconds while YIN taken 7.7 seconds. In conclusion we have also to say that mirpitch offers the possibility to change parameters, for trying different types of pitch recognition configurations, directly on the command line. This offers the possibility to made different tries

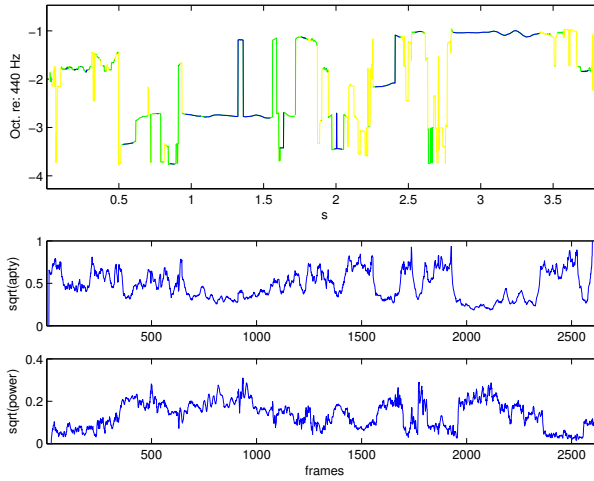

Figure 2.1: YIN Pitch Extractor: Hallelujah - Buckley.

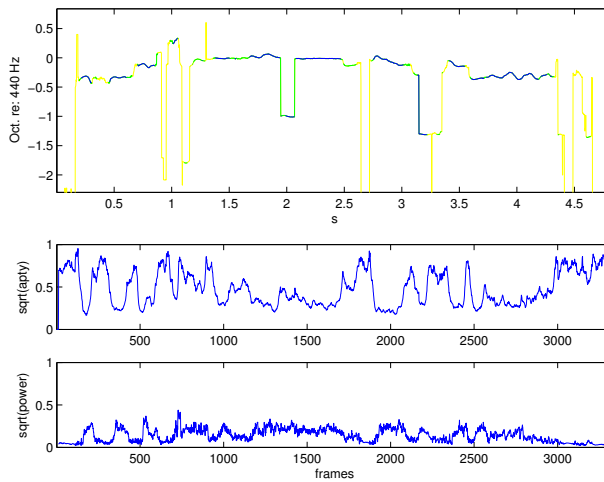

Figure 2.2: E Poi - Giorgia - Pitch Extraction made by YIN.

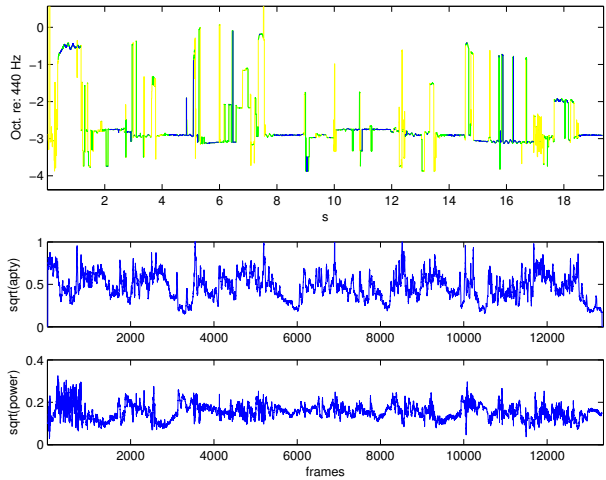

Figure 2.3: My Hearth Will Go On - Celine Dion- Pitch Extraction made by YIN.

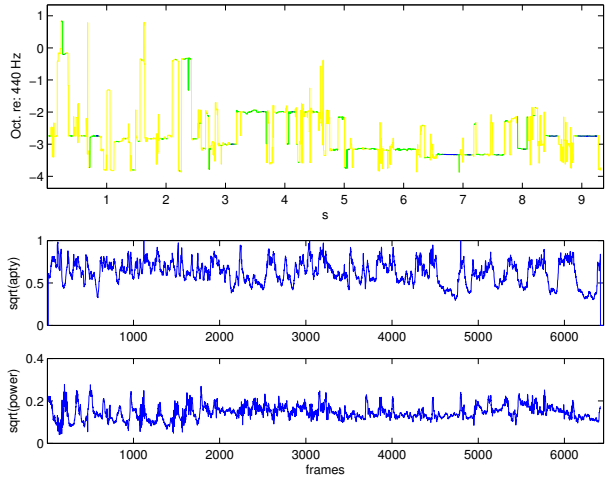

Figure 2.4: Albachiara - Vasco Rossi - Pitch Extraction made by YIN.

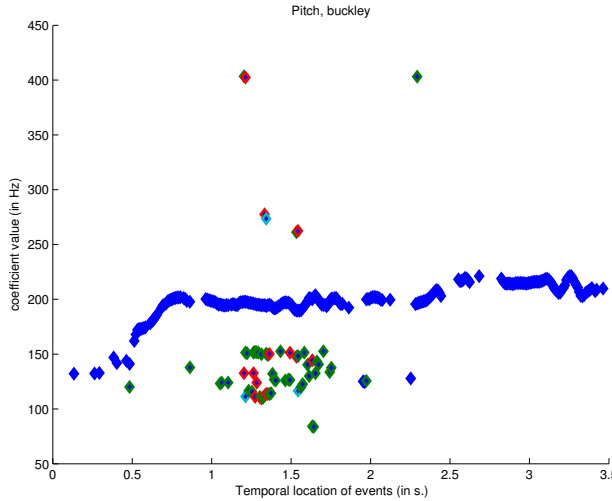

Figure 2.5: Hallelujah - Jeff Buckley - Pitch Extraction made by mirpitch in MIR ToolBox.

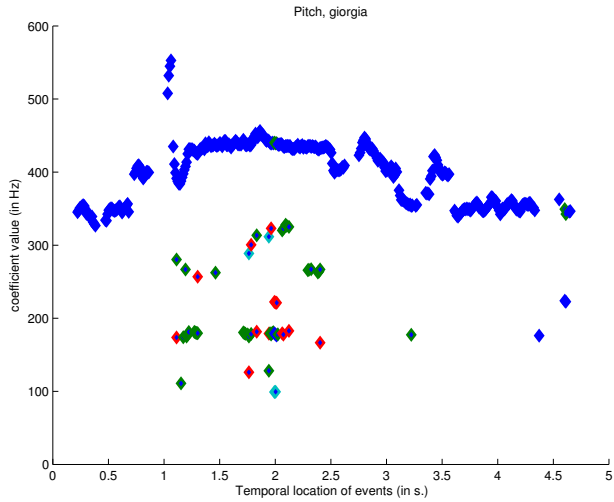

Figure 2.6: E Poi - Giorgia - Pitch Extraction made by mirpitch in MIR ToolBox.

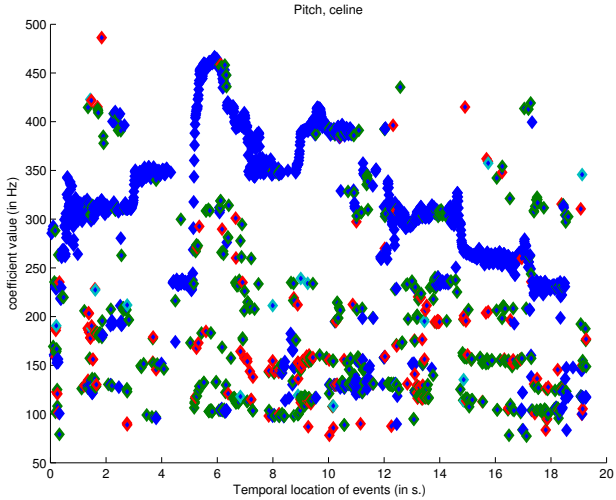

Figure 2.7: My Hearth Will Go On - Celine Dion - Pitch Extraction made by mirpitch in MIR ToolBox.

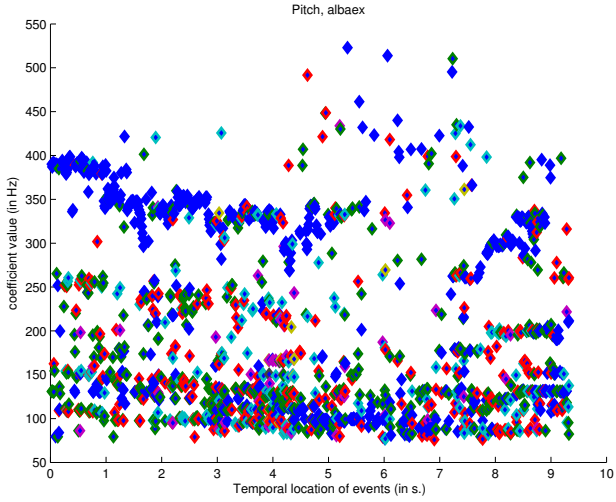

Figure 2.8: Albachiara - Vasco Rossi - Pitch Extraction made by mirpitch in MIR ToolBox.

with a lot of different experimental setups, without modify every time variables in the code.

We decided to use mirpitch as the tool for pitch recognition in this work. In the next section is presented a short description on how mirpitch is computing pitch recognition and the best way we found to parametrize it.

2.5 Mirpitch: workflow and parametrization

2.5.1 Mirpitch workflow

MIR ToolBox is structured like a chain of functions and, usually, everything coming from an output of a function can be used as input for another one, making it possible to perform different kinds of computation on audio files. For an overview on how MIR Tool Box chain structure is composed is possible to take a look at [20] pag. 9.

We describe now how this chain is used to compute pitch sampling. First of all the audio file is passed inside a filter called *'2Channel'*. This filter decomposes the audio file in two different channels: the first is obtained with a Low-Pass filter using a frequencies band of 70Hz-1000Hz and the other is obtained with a High-Pass Filter using a frequencies band of 1000Hz-10000Hz. Furthermore, on the second channel, two other steps are performed: first the signal is passed into an Half Wave Rectifier and then the signal is passed again in a Low-Pass filter with a frequencies band of 70Hz-1000Hz.
After the filtering phase, both channels are divided into frames where the frame length and the overlap will be specified later when we will talk about how we parametrize the tool. Then, the frame-divided two-channels audio file takes simultaneosly two different ways.
On the first one MIR ToolBox performs the autocorrelation function on both channels independently. Then the autocorrelation results are summed together and another module takes care of finding peaks to identify points with best autocorrelation values. On the other path the audio file obtained after filtering operation and frame division is passed inside a function that calculates the spectrum of the signal. Results returned by this function highlight the repartition of the amplitude of the frequencies.
After these operations are computed, results of spectrum function and autocorrelation peaks are merged together to find the best pitch for every frame. The pitch value is returned as absolute frequency value.
A graphic representation of this workflow is presented in figure 2.9.

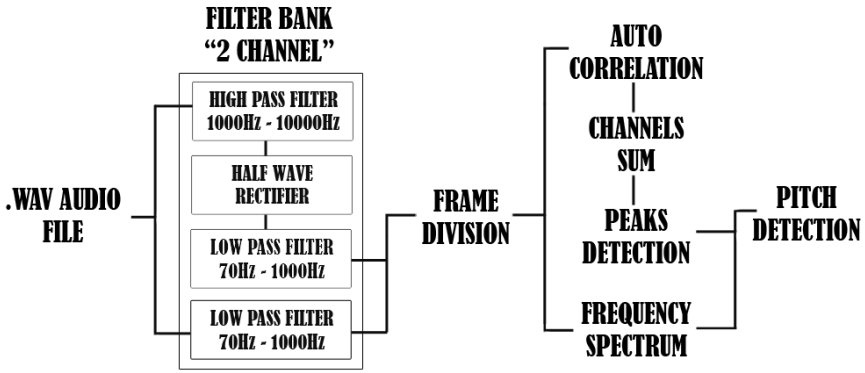

Figure 2.9: Mirpitch: graphic representation of pitch extraction's step.

2.5.2 Mirpitch parameters

The command we used to extract pitch values is:

```
mirpitch('fileName', 'Frame', frameLength, 's', overlap, 's', 'Min',
180, 'Max', 800, 'Mono')
```

Where:

fileName: is the name of the .wav file we have to analyze;

frameLength: is the length of the frame used to divide the audio file after the filtering process. (It is necessary to put the argument `'Frame'` before this value to indicate to mirpitch that we want frame division and specify `'s'` after it to indicate seconds as measure unit for frame length);

overlap: is the overlap for every frame. In example if the frame is 100 seconds and the overlap is 75 seconds the second frame start 25 seconds after the beginning of the first. (It is necessary to put the argument `'s'` after the overlap value to indicate seconds as measure unit for overlap);

'Min', 180, 'Max', 800: indicate to mirpitch that we are interested only in pitch values inside the range of frequency 180Hz-800Hz. This because vocal range and solo instrument parts are usually inside this interval; we have the best probability to find main melody or song's theme in this range;

Mono: indicate to mirpitch to compute a single pitch value for each frames. This because mirpitch usually computes more than a pitch value for each frame. With this command the tool automatically computes the best pitch and returns only it.

Chapter 3

Melody Representation

In the previous chapter we showed how we perform pitch detection on a song. In this chapter we start from this step and we describe the procedure we took to compute pitch values and transform them in a representation that can underline the melody or the theme of a song. We want to highlight how the representation we chosen can be robust to all the problems we underlined in Chapter 1 (tonality changes, variations on musical structure, tempo variations, etc.). At the end of the chapter we will see how we transformed the robust melody representation in a way that can be indexed and retrieved by Falcon.

3.1 Dataset Configuration

3.1.1 Query songs

The optimal parameters setting for the melody representation has been made using a dataset of 50 songs. All of these songs are live versions and some of them were also performed by a different artist than the songwriter. So we have a dataset including live songs and live cover songs. Table 3.1 presents the list of the songs that we used.

3.1.2 Dataset description

Starting from these 50 files we created four different sets. Every set is different from each others for the value of the frame length and the overlap that we used to execute the mirpitch module. The list of sets with detailed information is presented below:

Number	Title - Author - Note	Number	Title - Author - Note
01	24000 Baci - Adriano Celentano	26	Englishman In New York - Sting (Cover)
02	4 Marzo 1943 - Lucio Dalla and Ornella Vanoni (Cover)	27	Enjoy The Silence - Depeche Mode
03	50 Special - Cesare Cremonini (Cover)	28	Every Breath You Take - The Police
04	A Chi Mi Dice - Blue	29	Everybody Hurts - R.E.M.
05	A Hard Days Night - The Beatles	30	Frozen - Madonna
06	A Kind Of Magic - Queen	31	Girls Just Want To Have Fun - Cyndi Lauper
07	A Song For You - Whitney Houston	32	Hallelujah - Jeff Buckley (Cover)
08	Albachiara - Vasco Rossi	33	Hey Joe - Jimi Hendrix
09	Almeno Tu Nell'Universo - Elisa (Cover)	34	I Got You (I Feel Good) - James Brown
10	American Pie - Don McLean	35	In the air tonight - Phil Collins
11	Baby One More Time - Britney Spears	36	Io vagabondo - I Nomadi
12	Beat It - Michael Jackson	37	Knockin on Heaven's Door - Bob Dylan and Sheryl Crow (Cover)
13	Besame Mucho - Andrea Boccelli (Cover)	38	La Bamba - The Gypsies (Cover)
14	Caruso - Lucio Dalla	39	Light My Fire - The Doors and Eddie Vedder (Cover)
15	Celebration - Kool And The Gang	40	Like A Virgin - Madonna
16	Certe Notti - Ligabue	41	Mas Que Nada - A Cappella (Cover)
17	Che sará - Ricchi e Poveri	42	More Than Words - Extreme
18	Could You Be Loved - Ziggy Marley (Cover)	43	My Hearth Will Go On - Celine Dion
19	Crossroads - Eric Clapton (Cover)	44	Nessuno - Mietta
20	Day Tripper - The Beatles	45	O Sole Mio - Claudio Villa (Cover)
21	Diamante - Zucchero	46	Questo Piccolo Grande Amore - Claudio Baglioni
22	Dieci Ragazze - Lucio Battisti	47	Satisfaction - Rolling Stones
23	Dont Cry For Me Argentina - Joan Baez	48	Smells Like Teen Spirit - Nirvana
24	E Poi - Giorgia	49	Smooth Criminal - Michael Jackson
25	Emozioni - Lucio Battisti	50	What A Wonderful World - Louis Amstrongs

Table 3.1: Query files list - 50 Songs: live versions and live cover versions.

Dataset 1: *Frame length:* 0,30 seconds - *Overlap:* 0,15 seconds;

Dataset 2: *Frame length:* 0,20 seconds - *Overlap:* 0,10 seconds;

Dataset 3: *Frame length:* 0,10 seconds - *Overlap:* 0,05 seconds;

Dataset 4: *Frame length:* 0,05 seconds - *Overlap:* 0,025 seconds.

We used every one of these sets to compute data analysis and to obtain results that will be presented in the next sections.

3.2 Pitch Post-Processing

After receiving in input a song, mirpitch outputs a list of pitch values that we can then elaborate and modify to obtain the melody representation we desire. As an example of what it is the output of mirpitch we take a small excerpt of the result files of *Albachiara* by *Vasco Rossi* which was also one of the song that we used to test pitch detection tools in Chapter 2.

.., NaN, 392.632744, 394.685048, 235.829549, 395.703268, 392.720145, 198.186974, 390.995919, 195.404064, 266.179055, 255.844199, 374.861648, 366.588746, NaN, 372.088625, 222.229265, 211.547243, NaN,213.477265, NaN, ..[1]

As we can see values returned are absolute frequencies. Sometimes we find a special symbol: *NaN*. This symbol means that mirpitch is unable to compute a pitch value for that particular frame.

Starting from these values we have to compute and create a consistent representation of the song's melody. Every operative commands used in this thesis it is listed and explained in [22].

3.2.1 Notes Duration

The first analysis we made on pitch values was concerning notes' duration. This was made to understand how many short notes we were detecting with mirpitch. Usually, especially for sung part, short notes mean a transition phase from a note to another. Take in example the figure 3.1 where is reported a part of the melody of the chorus taken from *More Than Words* by *Extreme*.

The passage highlighted by the red circle is sung without pause from the D# to the B. This means that the singer is making a transaction "sliding" with the voice from the first note to the second. In this case mirpitch detects first the D#, then some different values from D# to B, including D, C#, C, and then the note B. We were not interested in all the short notes between D# and B so we wanted to remove them. Also, some of the short notes that we can find on mirpitch results could be either noise or a wrong detection of the pitch.

[1] Is it possible to find further details on commands and instructions to obtain pitch data, parsed with this method, in [22].

To perform this cleaning task we first analyzed which was the notes' length distribution on the overall set of our query songs. Figure 3.2, 3.3, 3.4, 3.5 present histograms for each dataset we mentioned in the list 3.1.2.

As it possible to see the distribution is more or less the same for every dataset considered. The only thing that changes is the number of notes detected. This is normal because having smaller frames it means that we need more frames to cover a whole songs. So, having more frames, consequently we have more pitch values.

Considering the high frequencies of very short notes we decided to remove every note which length was equal to the frame length. So practically we removed the first column for every histogram we presented in figure 3.2, 3.3, 3.4, 3.5. This was done to obtain a less noisy distribution, remove effects of wrong pitch detections and to delete transaction notes. Notes deleted were sobstituted with the value of the longer note on the left or on the right.

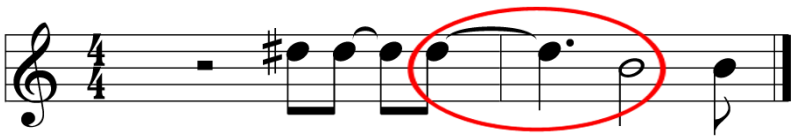

Figure 3.1: More Than Words - Extreme: chorus melody.

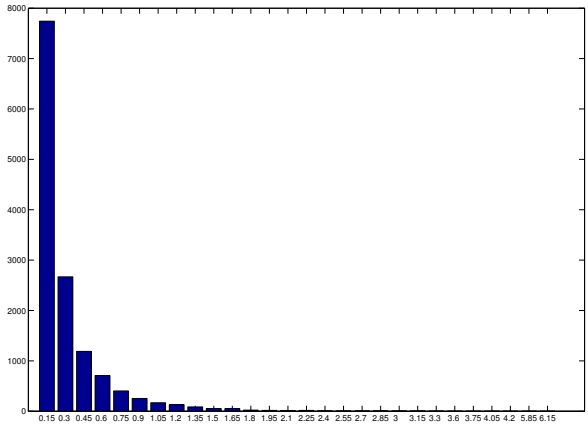

Figure 3.2: Notes' lengths distribution - Dataset 1.

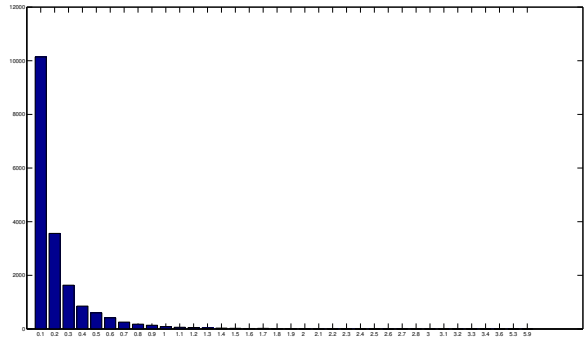

Figure 3.3: Notes' lengths distribution - Dataset 2.

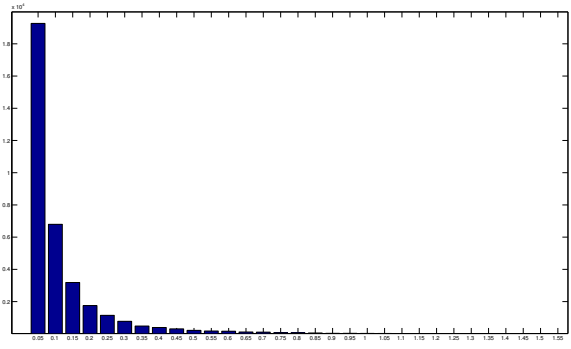

Figure 3.4: Notes' lengths distribution - Dataset 3.

3.2.2 Pitch correction and cleaning

Together with the deletion of "one-frame-notes" some other operations were applied to pitch results to improve the overall reliability. Analyzing pitch results was possible to notice that often in pitch detections were present holes between values of the same note. Here it is an example:

.. 366.588746, NaN, 372.088625 ..

The *NaN* value is placed between the same note[2]. Another similar situation is when, instead of a *NaN* value in the middle of a note, we find a complete different note value. Here is an example of this situation:

.. 392.632744, 394.685048, 235.829549, 395.703268, 392.720145 ..

The pitch value *235.829549* is in the middle of a note. This can happen because that part of the song is particularly disturbed by the presence of noise and mirpitch made a wrong pitch detection for that frame.

Both of these situations are observed to be usually related with a hole of one or two frames. To solve them a Matlab routine was developed. The routine is in charge to substitute "hole-values" with the value of the note where they are placed in the middle.[3]

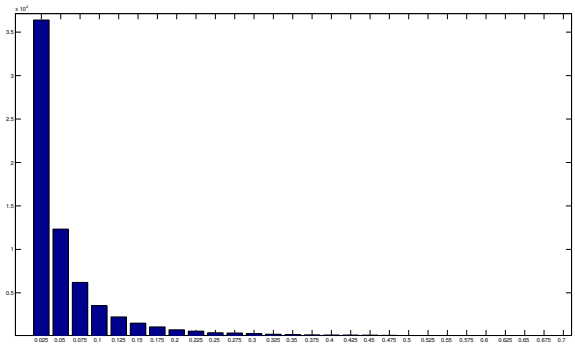

Figure 3.5: Notes' lengths distribution - Dataset 4.

[2]For an overview on how to calculate a note value starting from the absolute frequencies see Appendix A

[3]The command to recall the routine and how it is parametrized is explained in [22].

3.3 Intervals Representation

After we obtained a reliable list of pitch values for each song we had to decide to represent the melody in a way that must be not affected by problems we discussed in Chapter 1. The representation we chosen is the *intervals representation*.

An *interval* is the difference between the pitch of two sounds. Is possible to express it, from a physical point of view, as the ratio between the two sounds' frequencies. Talking about it from the musical point of view, considering that the sound available to us are twelve (the seven basic notes plus their alterations), an *interval* is a difference between two sounds that define it. [23]

A graphic demonstration on how we intend to elaborate our melody representation is given in figure 3.6. Starting from the original song that can be something like what is represented in Score A we use it as input for mirpitch module. What we supposed to obtain is only the melody of the song that is what is represented in Score B. As we said before mirpitch works extracting a pitch value for each frame the song is divided. Since the frames' length is fixed we have more pitch values for long notes than for shorter ones. An example of mirpitch extraction process for the first and the second bar is given below the two scores. Just below the examples of pitch values extraction, we found the interval representation for the first two bars. A 0 means a unison interval (created by the two G notes on the first bar) while a 2 means an ascendent second interval (like the one created by the D and the E on the second bar). If the second interval is descendent, like the one created by the E and the D in the second bar we indicate that as -2.

We consider different notes with the same pitch as a unique unison interval. For example, the four C in the last bar in figure 3.6 are considered like a unique C and the correspondent interval is 0. We are doing this because mirpitch detects pitch values using fixed temporal intervals and not with dynamic alignment on notes' attack. This means that for us, four different notes with the same pitch are the same thing that a single note with a duration equal to the sum of the single durations of the previous four notes. We tolerate this behavior because the melody concept we are working on is not based on the perfect transcription of the melody but on the recognition of the melody direction. Also, not representing the tempo make our representation robust to tempo variations. The chosen interval representations is suitable to this aim.

The intervals representation we chosen is robust to problems given by cover and live songs. We are going to explain for every single problem why the representation is robust against it.

Tonality changes: if the tonality of a song is changed the melody is played on different notes but intervals between notes stay the same.

Tempo variations: since the chosen approach is not representing the tempo but only the melodic direction given by intervals, even if the song is played faster or slower we just need to find an appropriate frame length for

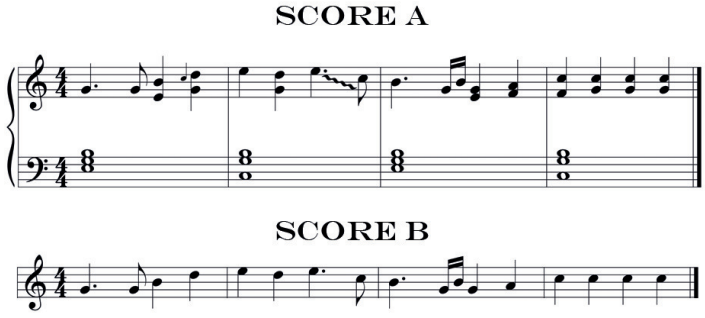

Figure 3.6: Example of melody extraction.

mirpitch module. Once this value is found, mirpitch is able to detect at least every short and necessary note to calculate melody direction.

Different voice or instrument: this problem is more concerned to the pitch detection tool that to the representation. Anyway, if the pitch results given are reliable the representation of the melody will be the same, even if the part is sung by a male or female, or played by a guitar or a oboe.

Changes on musical structure: even if in our song some musical parts are arranged in a different order, compared to the original song, the main melody or the main theme will stay the same. Once we recognize it we just need to identify it matching the proper song's segment. This is possible because we also segment our melody representation in different parts (we will talk about in on 4).

3.3.1 Representations

For each dataset presented in list 3.1.2 we tried several types of representation[4]. Below is a list of the representations we tried:

Representation 1: Semitones. For this representation we used semitones, the smallest interval that is possible to find in music (or in the Western music culture at least. For a further overview on this topic see Appendix A). Intervals range in this situation is wide but it is possible to represent every single variation between two notes.

Representation 2: Intervals - Fine. For this representation we used the classical intervals' division used in the music field. We put together different variations of the same interval: for example an interval of second includes the minor and the major second interval. The figure 3.7 explain how we grouped together semitones to compose intervals.

Representation 3: Intervals - Coarse. This representation is simply a more generalized version of the *Representation 2*. Here different intervals are grouped together. The figure 3.8 explain how we grouped together intervals to compose this representation.

3.3.2 Data analysis on representation

For each of the representations listed in 3.3.1 we conducted a data analysis to observe if they were congruent to typical melody for pop and rock songs. Also this analysis was useful to observe how much noise was detectable in the different representations. When we say "congruent to typical melody " we mean that usually, there are some intervals that are more used than others. For

[4]Functions and routines for computation of representation are explained, as before, in [22]

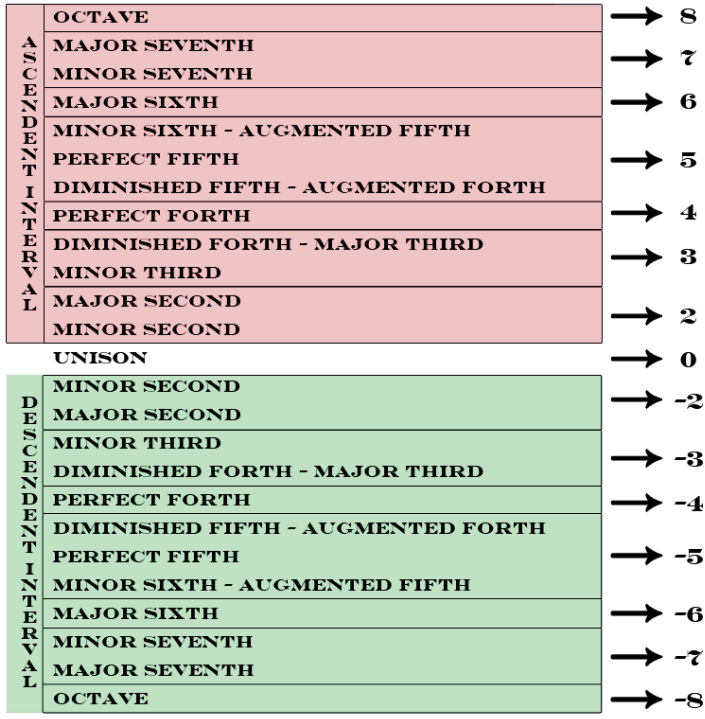

Figure 3.7: Representation 2 - Grouping of the intervals.

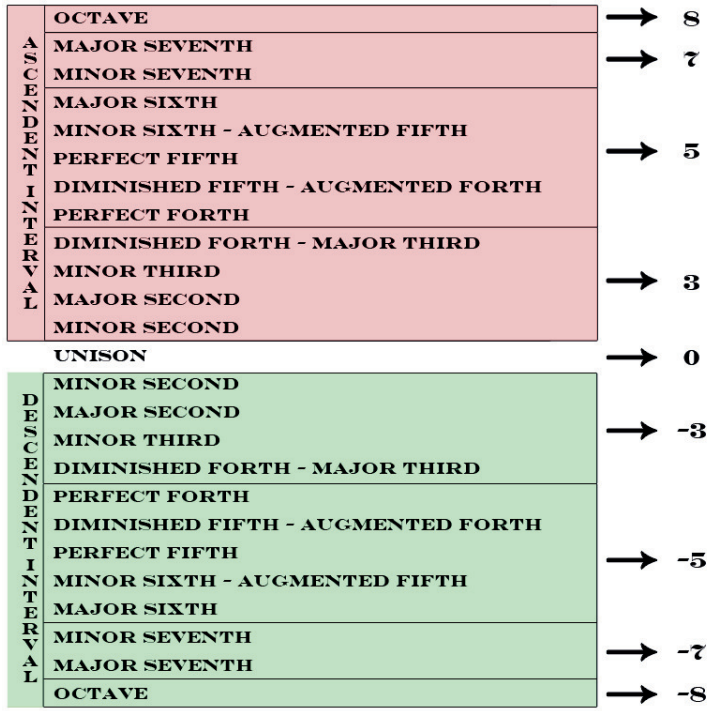

Figure 3.8: Representation 3 - Grouping of the intervals.

example 3rd interval or 5th interval are usually more used than 6th interval. Looking at intervals in a more detailed way, 5th diminished interval is less used that perfect 5th interval. Also we expected to find a majority of small intervals (2nd intervals and 3rd intervals) because melodies, especially sung ones, are usually constructed on close notes.

We present now some histograms showing the distribution of intervals calculated starting from pitch values detected on songs presented in 3.1. Histograms presented here are the ones computed on the Dataset 2. We present only these histograms because the behavior for the other datasets is really similar. Anyway this dataset was the one with the most uniform distribution of two semitones intervals compared to one semitone intervals. Considering that, it appeared that this dataset was probably the one with the lowest presence of noise coming from pitch detection phase. It is possible to consult the full set of histograms for each dataset here [24].

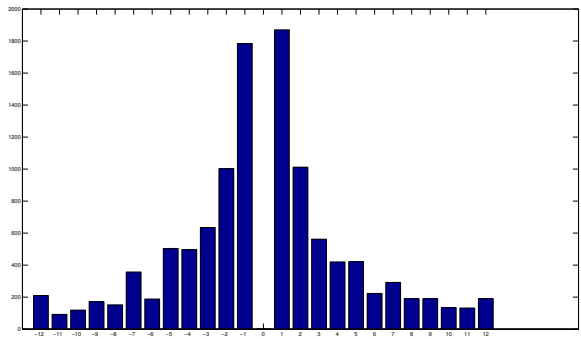

Figure 3.9: Representation 1 - Semitones distribution - Dataset 2.

The same data analysis we carried out on our query set was done also on the large dataset we used for experimental tests of cover song recognition. The experimental dataset is composed of studio songs and for some title also by live versions (but not the same live we used as query files). There are usually ten different versions of the song for each title that is present in our list in **??**. For a detailed list of the files in the dataset see [25]. The data analysis on this dataset was carried out to verify if the distribution given by our representation was compatible a priori between the two sets. As it can been seen from the histograms this condition was verified. There are some differences between the two datasets but the overall distribution of the large dataset is very similar to the distribution of the query dataset. Pitch values for the large dataset were calculated using the same parameters of *Dataset 2* detailed in list 3.1.2.

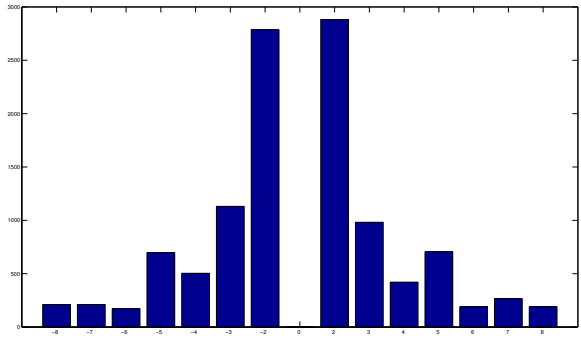

Figure 3.10: Representation 2 - Fine intervals distribution - Dataset 2.

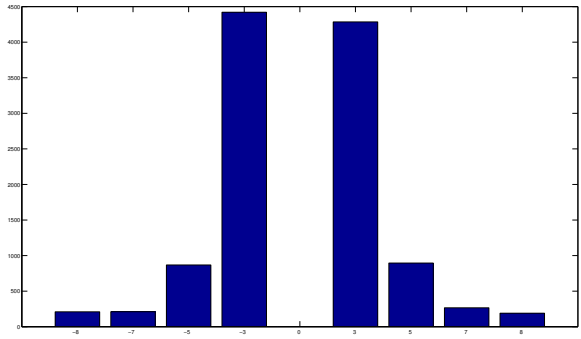

Figure 3.11: Representation 3 - Coarse intervals distribution - Dataset 2.

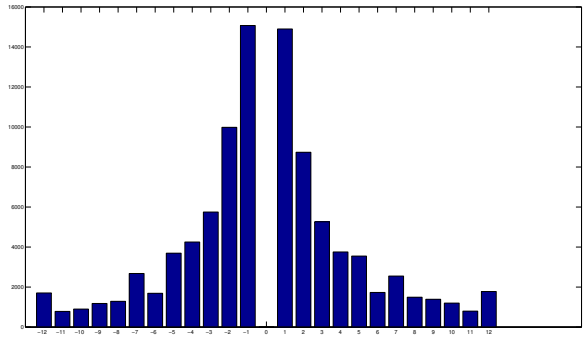

Figure 3.12: Representation 1 - Semitones distribution - Experimental Dataset.

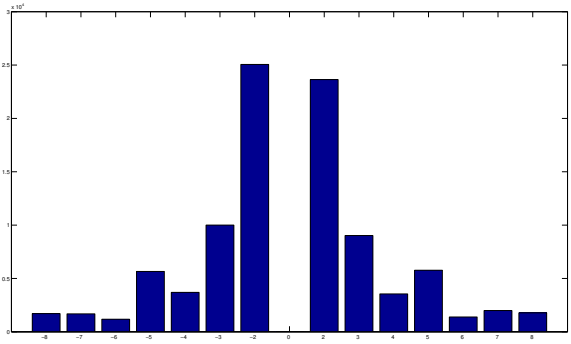

Figure 3.13: Representation 2 - Fine intervals distribution - Experimental Dataset.

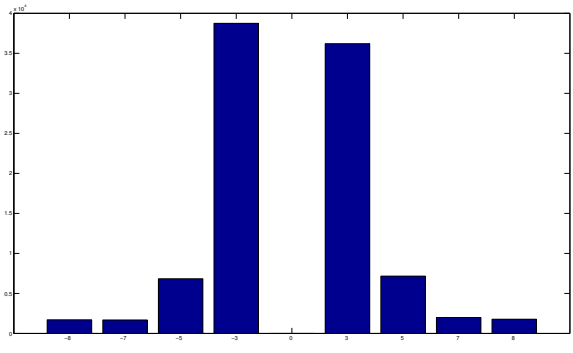

Figure 3.14: Representation 3 - Coarse intervals distribution - Experimental Dataset.

3.4 Transformation to N-Grams

After we found a good representation for the melody we had to find a way to index every song and retrieve it. In particular we need to create some index terms starting from the intervals melodic representation. The concept that we adopted in this work is based on the concept of N-Grams.

A N-Gram is simply a sequence of N symbols. We wanted to create N-Grams of different lengths and we had to find a way to code them in a manner that could give us a single number to represent a N-Gram, something like an hash. The N-Grams we created have a percentage of overlap on each other. In particular we created a N-Gram starting from every interval of the sequence we had for a song. The figure 3.15 can clarify the concept that we used to create our N-Gram representation.

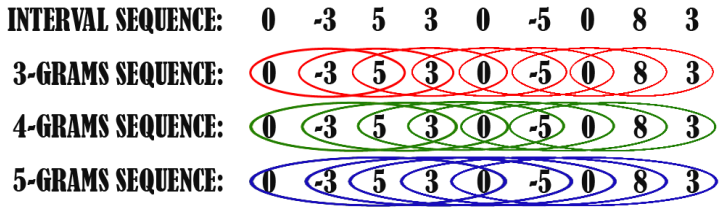

Figure 3.15: Example of N-Grams creation.

We need to create N-Grams of different length for the necessity to obtain the best representation possible of the song. Short N-Grams could be good descriptors for a song since it is plausible that they will be found both in the query song and in the original one, giving us a good point to match. On the other hand short N-Grams are also really common in each song. For example the 3-Gram *0 3 0* is probably really common in every song since the passage on the third is a really common melodic structure for sung parts. This means that short N-Grams, even if they have a good matching score, sometimes are not really useful to identify a single song.

On the other hand, long N-Grams are more characterizing for each song since they represent a long melodic sequence. Anyway they are probably really few so if we miss to create them for some problem coming from pitch detection we lost the songs' match.

Another good reason to mix-up different types of N-Grams in the representation is the different robustness that they have to wrong pitch detections. Imagine that, even after pitch post-processing operations, some wrong pitch values are present in the results. This means that some wrong intervals are present in the sequence used to create N-Grams. In the case of 3-Grams a wrong interval in a sequence it means that we have between one and three wrong 3-Grams (remember N-Grams' overlap). On the other hand, considering 6-Grams, one wrong interval in the sequence it means that we have between one and six

wrong 6-Grams. Considering the numerosity of 3-Grams and 6-Grams that can be created starting from the same intervals' sequence, is easy to see that 3-Grams are less affected by wrong intervals presence than 6-Grams.

For these reasons we decided to combine together all types of N-Grams creating for each song 3-Grams, 4-Grams, 5-Grams and 6-Grams. In figure 3.16 we can see how N-Grams are distributed in our *Dataset 2*. Dark blue lines are 3-Grams, light blue lines are 4-Grams, yellow lines are 5-Grams and red lines are 6-Grams. As it possible to see the distribution is dominated by 3-Grams and 4-Grams but soon we find also some 6-Grams that are quite common in the distribution. The overall number of N-Grams for our collection is around 210000. The number of distinct N-Grams is 3163.

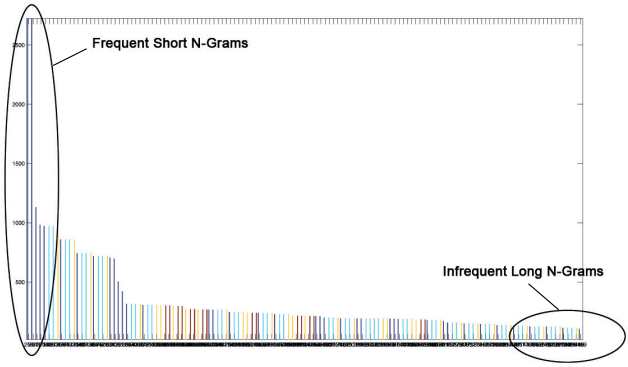

Figure 3.16: Extract from the distribution of 3-Grams, 4-Grams, 5-Grams and 6-Grams - Dataset 2.

3.4.1 N-Gram hashes

To represent each of the N-grams with a different symbol we decided to assign a numerical value to each of them using interval names to create this value. The concept behind it is simple: consider that we are talking about 3-Grams using the fine intervals representation presented before. In this situation we have 3 symbols for each N-Gram that must be computed on the sequence of symbols, where each symbol is taken from an alphabet of 15 symbols. This because the intervals we considering in *fine intervals representation* are -8, -7, -6, -5, -4, -3, -2, 0, 2, 3, 4, 5, 6, 7, 8; different representations have different number of symbols. So we transformed every 3-Gram in a three ciphers number using base 15. In our procedure we had a transformation for unison interval and negative intervals. The table 3.2 presents this transformation.

After this transformation the creation of an hash is an easy task. Consider for example the 3-Gram *0 3 0*. To compute this hash the formula is:

Interval	Transformed Interval
-8	15
-7	14
-6	13
-5	12
-4	11
-3	10
-2	9
0	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8

Table 3.2: Transformation of intervals' names in useful values for N-Grams hashes computation.

$$1 * 15^2 + 3 * 15^1 + 1 * 15^0 = 271$$

Using this type of hashing functions we are able to calculate a value for each single N-Gram without the problem of overlapping values for different types of N-Grams. This because the value of a 3-Gram hash will never be as high as the value of a 4-Gram hash and also the value of a 4-Gram hash will never be small as the value of a 3-Gram hash. This because we are not using the value 0 to calculate our hashes. The last fact means also that we have the possibility to use the value 0 (and many other values in fact) as a special value for other functions that could be useful in our work.

Chapter 4

Experimental Results

In this chapter we will present the results coming from different experiments we made on the data. We started with a basic configuration and we changed several parameters trying to achieve better results, looking for an optimal configuration of the system.

4.1 Basic configuration

The configuration we chosen to start our test is the following:

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds (note below this threshold are removed);
N-Gram combination: 3-Grams, 4-Grams, 5-Grams, 6-Grams.

These values were decided as a starting point after the data analysis performed in Chapter 3. Results obtained from intervals distribution for this configuration were the most uniform looking at the disposition of one semitone and two semitones intervals. Anyway there are no guarantees that this configuration can give the best results. We had to find the optimal configuration through different experiments, refining parameters step by step.

The procedure of the test is composed by three simple steps:

Step 1: indexing of the collection of 555 files (see [25]). The files are processed

in the same way as the query files. This means that we used the same mirpitch frame length, same mirpitch overlap, etc. that we used also for processing the query files. Indexing task is computed by Falcon.

Step 2: retrieval for the 50 query files (see 3.1). Query files are given in input to Falcon. Then, Falcon outputs for each file a list containing, from the first to the last, ordered by match score, the 555 songs composing the dataset. The first song in the list is the one that Falcon believes more probable to be the original song. This step is executed using three different types of ranking functions: BM25, MINHF and HFIDF. These ranking functions are described in the next section.

Step 3: Falcons' results are elaborated by an evaluation function and results are summarized together.

All the operative commands used in this phases are listed and explained in [22].

4.2 Ranking functions

Falcon implements three different types of ranking functions used to measure the score match of every query against songs that are in the dataset. These ranking functions are: *BM25* [26], *HFIDF* [27], *MINHF* [3].
These ranking functions are used to calculate a matching score for a query. BM25 and HFIDF[1] are not designed for a specific application but they are known in general in the field of information retrieval. MINHF was specially developed for the usage on Falcon.

4.3 Performance Measures

For each experiment usually two main measures are presented: *MRR - Mean Reciprocal Rank* and *MAP - Mean Average Precision.*

Mean Reciprocal Rank - MRR

The Reciprocal Rank, or RR, is the reciprocal of the first rank position where the correct song is found. In example, if the first song is correctly matched in second position the MRR is:

[1] Also knows as TFIDF. The H of HFIDF means "hashes", because we are working on N-Grams hashes instead than text terms that is where T of TFIDF derives.

$$1/2 = 0,5$$

The MRR is the mean of all the RRs calculated on a query set. This means that all the RRs for every single query are summed together and then divided by the queries number. In example, given a set of 3 query on a dataset of 5 files, let the results be: *Query 1*: 1st place, *Query 2*: 3rd place, *Query 3*: 2nd place. In this case the MRR is:

$$(1/1 + 1/3 + 1/2)/3 = 0,61$$

It can be easily shown that MRR is better when its value tends to 1, while result are worst when its value tends to 0.

Mean Average Precision - MAP

The Average Precision, or AP, is the sum of the reciprocal of all the rank positions founded for a query divided by the number of matches found. In example, if a song is detected correctly in position 2, 4 and 6 the AP for the query is:

$$(1/2 + 1/4 + 1/6)/3 = 0,31$$

MAP is the mean of all the APs calculated on a query set. This means that all the APs for every single query are summed together and then divided by the queries number. It can be shown that MAP is better when its value tends to 1, while result are worst when its value tends to 0.

4.4 N-Grams Segmentation

The first parameter we started to evaluate was the segmentation length for N-Grams. Falcon permits to evaluate each song considering it as a sequence of segments allowing also overlap between them. This is a desirable feature considering that, for a future application of the system, it is more probable that a user will try to recognize only small excerpts of a song rather than a complete song. For our tests we used the entire song as query files but the implementation of segmentation is a good starting point for future tests on small excerpts. To exploit segmentation of our N-Grams sequence we simply used a delimiter which value was not used in computation of hashes: *0*. Remember from the section 3.4.1 where we talked about N-Grams, the value 0, due to the process of N-Gram creation, is never used.
As an example, the N-Grams sequence coming from the segmentation made in table 4.1 is:

367 765 456 4563 5674 45637 654763 0 654 456 456 6786 6754 67546 54623 675498 0 1 0 456 654 789 6754 2345 76859 675839 0

	Segment 1	Segment 2	Segment 3	Segment 4
3-Grams	367 765 456	654 456 456		456 654 789
4-Grams	4563 5674	6786 6754		6754 2345
5-Grams	45637	67546 54623		76859
6-Grams	654763	675498		675839

Table 4.1: Example of N-Grams segmentation.

As it possible to see different types of N-Grams belonging to the same segment are grouped together in order of N-Gram's size. Segments are separated by the delimiter 0. A special case is the segment 3 where no N-Grams are present. This is because not always is possible to detect pitch values for every frame and the lack of this information can conduct to situations of segments without any N-Grams. These void segments are coded in our sequence with the value 1. This is another value that is never used during N-Grams creation.

Falcon was instructed to interpret value 0 and 1 respectively as "segment delimiter" and "empty segment".

4.4.1 Segmentation Results

The first experiments set was conducted to test which type of segmentation obtain the best results. For this scope we used three different types of segmentation:

- Long segments with wide overlap
- A unique segment
- Short segments with small overlap

Below here each configurations is detailed and results are presented.

Configuration 1

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 3-Grams, 4-Grams, 5-Grams, 6-Grams;
Segment length: 40 seconds;
Segment overlap: 30 seconds;

Configuration 2

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 3-Grams, 4-Grams, 5-Grams, 6-Grams;
Segment length: 600 seconds;
Segment overlap: 0 seconds;

In this set using a segment length of 600 seconds and 0 seconds of overlap we consider a song as a unique segment.

Configuration 3

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 3-Grams, 4-Grams, 5-Grams, 6-Grams;
Segment length: 12 seconds;
Segment overlap: 6 seconds;

Results

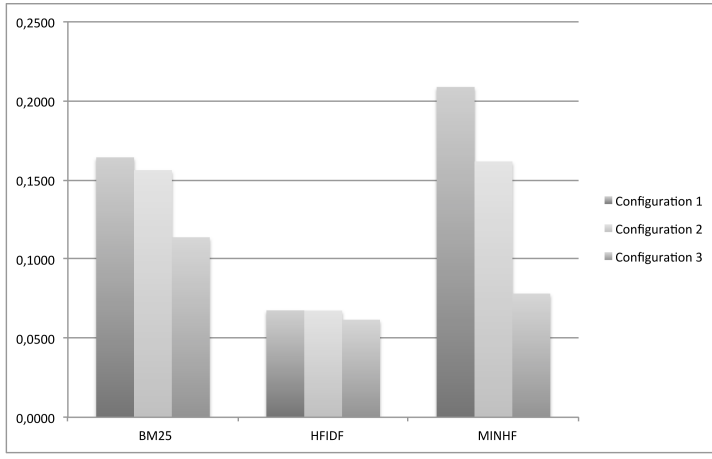

Figure 4.1: MRR of first experiment on N-Gram segmentation.

As it possible to see from the results reported in figure 4.1, the best performances are obtained using the segmentation with high segment value and high percentage of overlap. In particular the best value of MRR is obtained using

the ranking function MINHF.
Also we can notice that HFIDF is the ranking function that is working always worst compared to the others. We made also other different tests that confirmed this evaluation on ranking functions. Two of these additional tests were made on the dataset using as query original files in the dataset, one time keeping them in the dataset and one time removing them. Also these tests showed that HFIDF is the worst ranking function compared to BM25 and MINHF. Assuming this, from now on, our tests are based only on BM25 and MINHF.

Starting from the results obtained with the first experiment we explored deeply the possibilities offered by segmentation. We created other configurations with different values, still looking for the best segmentation, focusing on long segments with a wide overlap. Below here different configurations used during our tests are reported.

Configuration 4

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 3-Grams, 4-Grams, 5-Grams, 6-Grams;
Segment length: 30 seconds;
Segment overlap: 20 seconds;

Configuration 5

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 3-Grams, 4-Grams, 5-Grams, 6-Grams;
Segment length: 50 seconds;
Segment overlap: 40 seconds;

Configuration 6

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 3-Grams, 4-Grams, 5-Grams, 6-Grams;
Segment length: 60 seconds;
Segment overlap: 50 seconds;

Configuration 7

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;

Threshold for note deletion: 0.10 seconds;
N-Gram combination: 3-Grams, 4-Grams, 5-Grams, 6-Grams;
Segment length: 70 seconds;
Segment overlap: 60 seconds;

Configuration 8

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 3-Grams, 4-Grams, 5-Grams, 6-Grams;
Segment length: 100 seconds;
Segment overlap: 85 seconds;

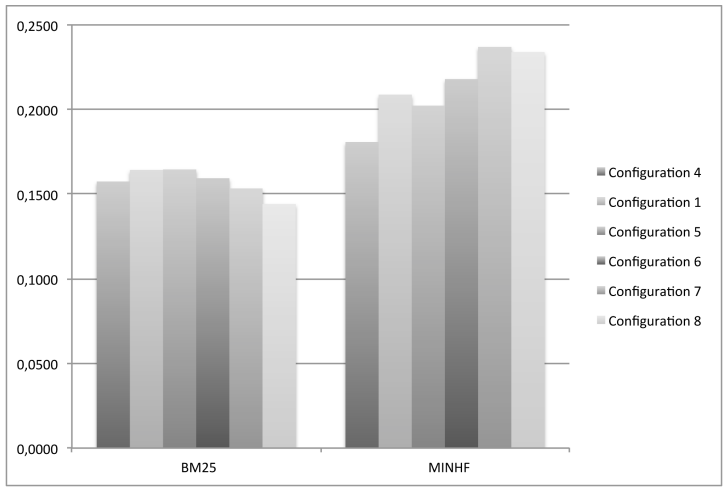

Figure 4.2: MRR of second experiment on N-Gram segmentation.

In figure 4.2 results of the second experiment are presented. Notice that the *Configuration 1*, the same as the first experiment is reported again also in this one to have a comparison with other configurations. From this new experimental setup we have seen that increasing until a certain limit the segment length and its overlap conduct to better results. In particular *Configuration 7* gave best results.

4.5 Overlap Configuration

After we found an optimal value for segments using a segment length of 70 seconds and an overlap of 60 seconds we decided to change the overlap parameter to try to obtain better results. We tried with different configurations keeping the segment length value to 70 seconds and changing only the overlap value. All the tests, carried out both with BM25 and MINHF ranking functions, gave worst results that the ones obtained with the *Configuration 7*. So we decided to keep these values for segment length and overlap for further tests.

4.6 Results with Different Combinations of N-Grams

After having fixed values for segment length and overlap we tried to mix different types of N-Grams, always with the hope to obtain better results. The first experiment we tried was carried out analyzing the behavior of 4 different configurations using only a type of N-Gram each. The configurations for this test are listed below:

Configuration 9

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 3-Grams;
Segment length: 70 seconds;
Segment overlap: 60 seconds;

Configuration 10

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 4-Grams;
Segment length: 70 seconds;
Segment overlap: 60 seconds;

Configuration 11

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 5-Grams;
Segment length: 70 seconds;

Segment overlap: 60 seconds;

Configuration 12

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 6-Grams;
Segment length: 70 seconds;
Segment overlap: 60 seconds;

Since from previous tests we have seen that BM25 has always a significative lower value that MINHF, from now on, tests are carried out using only MINHF ranking function.

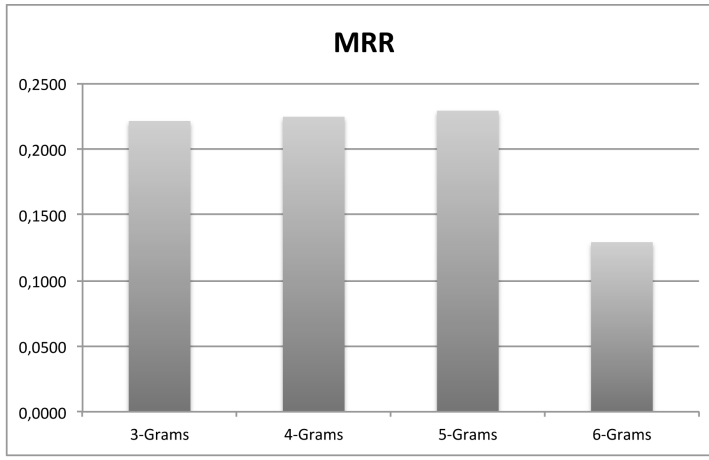

Figure 4.3: MRR for configurations using only one type on N-Grams.

As it possible to see from figure 4.3 and 4.4 *Configuration 12* that uses only 6-Grams is the one with the worst results. On the other hand, talking about others kinds of N-Grams we have a inverse behavior for MRR and MAP. While MRR is increasing going from 3-Grams to 5-Grams MAP is decreasing. This is something that we expected considering what we said about N-Grams and their properties of frequencies, information rate and robustness to wrong pitch detections.
After the analysis of results coming from this experiment we tried to achieve both good MRR and MAP values using a combination of 3-Grams and 5-Grams. What we found is the state of the art of our system. Using the configuration with the fine intervals representation and values of frame length and overlap for mirpitch respectively equal to 0.20 seconds and 0.10 seconds, the combination

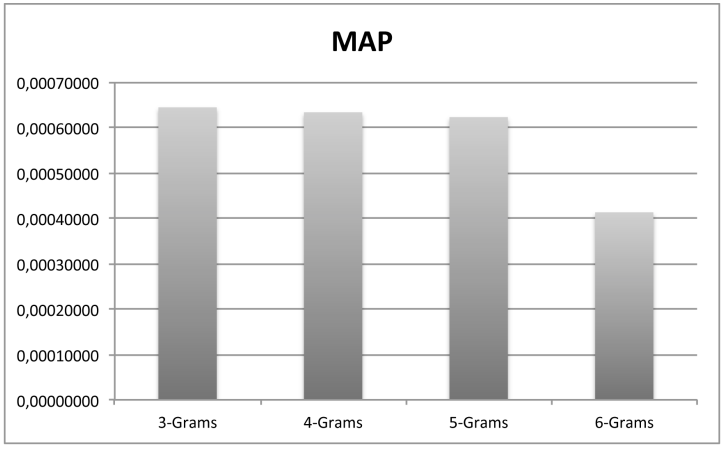

Figure 4.4: MAP for configurations using only one type on N-Grams.

of 3-Grams and 5-Grams gave the best results. A detailed explanation of the configuration is presented below.

Configuration 13

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 3-Grams, 5-Grams;
Segment Length: 70 seconds;
Segment Overlap: 60 seconds;

Configuration	Measure	BM25	HFIDF	MINHF
Configuration 13	MRR	0,1446	0,0697	0,2537
	MAP	$4,92 * 10^{-4}$	$2,45 * 10^{-4}$	$6,92 * 10^{-4}$

Table 4.2: State of the art of the system. MRR and MAP values obtained with Configuration 13.

Configuration	Query Type	1st	<= 10	<= 25	<= 50	<= 100
Configuration 13	BM25	8%	34%	58%	74%	80%
	HFIDF	2%	18%	42%	50%	68%
	MINHF	14%	44%	64%	82%	90%

Table 4.3: Results on correct song matching obtained with Configuration 13.

As it possible to see from table 4.2 the *Configuration 13* presents the best values for MRR and MAP using MINHF ranking function. Table 4.3 presents also the percentage of correct songs individuated in some predefined intervals. Performances are appreciable for this configuration, considering that this is the first version of the system.

4.7 Additional Tests

After having found a good configuration for our system we decided to carry out some other tests aimed to explore other possibilities offered by our dataset and tools. In particular we tried other configurations changing at the base parameters of pitch detection. Also we tried other configurations changing parameters for pitch post-processing phase and, finally, changing entirely the type of representation. Results for these other tests are described in the sections below.

4.7.1 Changes on pitch post-processing parameters

The first tests we made were focused to see the differences coming from the modification of the parameters of pitch post-processing. In particular we tried to see how different approaches on deletion of notes affect results of our system. We kept all the parameters belonging to *Configuration 13* and we changed only the *threshold* one. This parameter defines which is the minimum notes' length to keep on pitch values during the phase of pitch post-processing. Our new configurations involved a non-post-processed set (all the notes were kept), a set with threshold parameter equal to two times the parameter for *Configuration 13* and a set with threshold parameter equal to three times the parameter for *Configuration 13*. All the results obtained by these sets were inferior to the results obtained by *Configuration 13*. In particular, only the set without post-processing obtained inferior but similar result as *Configuration 13* while the other two sets output very poor results, especially the set with the biggest threshold value.

4.7.2 Changes on values for pitch detection

Some other tests were made changing the parameters used on mirpitch to detect pitch values. These parameters are the *frame length* and the *frame overlap*. We created different configurations, one for each dataset listed in 3.1.2. All other parameters are the same as the ones used in *Configuration 13*. Results coming from this set of tests underline that the best configuration was still *Configuration 13*. The only configuration with similar results, compared to the ones of *Configuration 13*, is the one with frame length equal to 0.10 seconds and overlap equal to 0.05 seconds. Other two configurations report poor values for MRR and MAP in the tests. Furthermore, to explore more in detail this experimental setup, we carried out some other tests using frame length equal to 0.10 and overlap equal to 0.05 to see if we could obtain better results. To do this we modified the *segment length* and *segment overlap* parameters. Anyway results were not better that what found before. This was another confirmation that a segment length of 70 seconds and a segment overlap of 60 seconds are a kind of "magic combination " for our dataset and our system.

4.7.3 Changes on representation

After changing parameters of post-processing phase and pitch detection module we decided then to try to change radically the melody representation. At the beginning, to create the starting situation we decide to represent melody using a fine intervals representation. Anyway we had disposable two other types of representation: *coarse interval* and *semitones*. Both of them were presented in Chapter 2. Changing this parameters means that we change radically also the N-Grams' values because they are calculated on a different amount of symbols. Our system has no problem to manage these new values and we were curious to see how they affected the system.

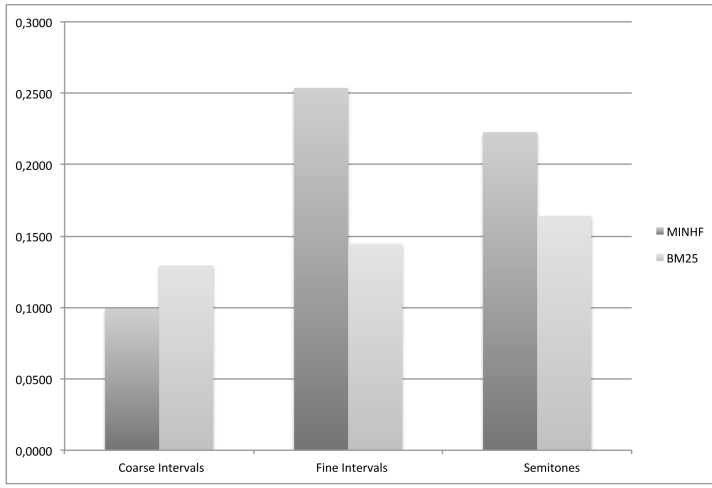

Figure 4.5: MRR value for different type of representation using BM25 and MINHF ranking functions.

As it possible to see from the histogram on figure 4.5 coarse intervals representation presents the worst performance. This is probably due because this type of representation is too general and can not represent in a proper way differences between songs. Furthermore, is important to underline the good results coming from semitones representation. MRR value calculated with MINHF has a lower value than the one calculated using fine intervals representation but MRR value calculated with BM25 is slightly higher than the value calculated using fine intervals representation.
It is necessary to underline that we obtained this result simply with casual attempts of configuration sets. This means that, considering semitones representation, with some work aimed to refine parameters we can maybe obtain results better than the ones obtained with our *Configuration 13* that use fine interval representation.

4.7.4 Tests on semitones representation

Since we obtained encouraging results trying to test different types of representations, we decided to push a little bit more on this side. So, using the *semitones* representation we did several other tests to see if it was possible to improve the performance of the system. The procedure adopted was a duplicate of what we did to test the first basic representation. We first tried different types of segment length always using a wide overlap. Then, once the best value for segment length was founded, we tried different types of overlap value for the segment. Finally, we also tried to mix different types of N-Grams.

What came out from the results it is that the semitones representation presents results similar to fine intervals representation. Anyway, the best result obtained is still the one coming from *Configuration 13*. What we have to say is that semitones representation presents results with less variability than results obtained by fine intervals representation. This means that, even changing parameters for segment length and overlap, results value present less differences one from each others compared to the ones obtained using fine intervals representation. Moreover, results obtained using BM25 with semitones representation are better than results obtained using BM25 on fine intervals representation even if they do not reach the same good results obtained with MINHF. In figure 4.6 and 4.7 tests' results are reported, comparing fine intervals representation and semitones representation. The four configurations used, both for fine intervals representation and semitones representation are:

Configuration 1

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 3-Grams, 5-Grams;
Segment length: 70 seconds;
Segment overlap: 60 seconds;

Configuration 2

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 3-Grams, 5-Grams;
Segment length: 60 seconds;
Segment overlap: 50 seconds;

Configuration 3

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 3-Grams, 5-Grams;
Segment length: 50 seconds;
Segment overlap: 40 seconds;

Configuration 4

Mirpitch frame length: 0.20 seconds;
Miripitch frame overlap: 0.10 seconds;
Threshold for note deletion: 0.10 seconds;
N-Gram combination: 3-Grams, 5-Grams;
Segment length: 40 seconds;
Segment overlap: 30 seconds;

The best configuration found for semitones representation is *Configuration 3*. For this configuration we tried also several different types of overlap values. Furthermore, we changed also the combination of N-Grams, carrying out some other tests. None of these other configurations gave better results.

Detailed results, showing also MRR and MAP values, for *Configuration 3* are presented in tables 4.4 and 4.5. Notice that for this configuration, even if the MRR value is lower MAP value is a bit higher than the one obtained with the best configuration using fine intervals representation.

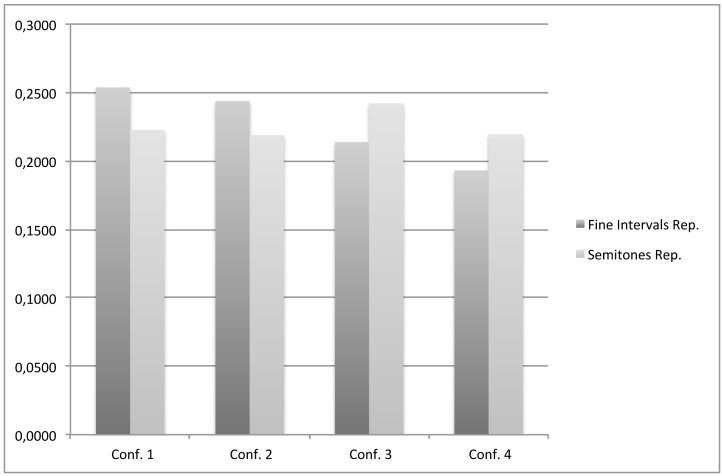

Figure 4.6: Comparison on MRR values for semitones representation and fine interval representation using MINHF.

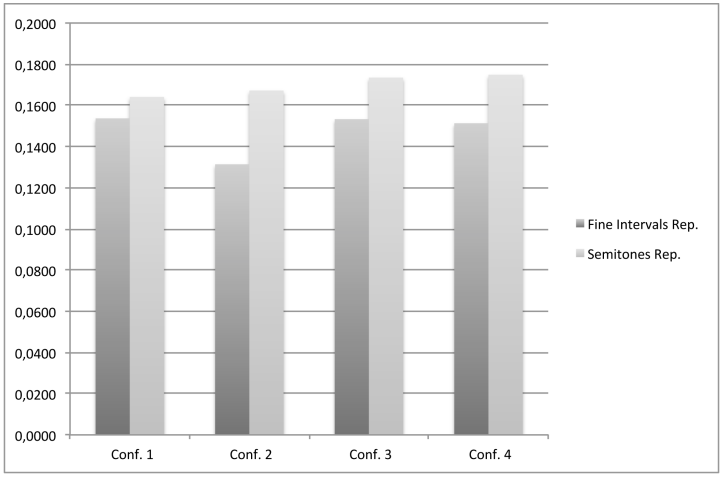

Figure 4.7: Comparison on MRR values for semitones representation and fine interval representation using BM25.

Configuration	Measure	BM25	HFIDF	MINHF
Configuration 3	MRR	0,1735	0,0614	0,2421
	MAP	$5,56 * 10^{-4}$	$2,33 * 10^{-4}$	$6,95 * 10^{-4}$

Table 4.4: MRR and MAP values obtained with Configuration 3.

Configuration	Query Type	1st	<= 10	<= 25	<= 50	<= 100
Configuration 3	BM25	8%	34%	52%	74%	88%
	HFIDF	2%	16%	32%	46%	68%
	MINHF	16%	42%	54%	70%	88%

Table 4.5: Results on correct song matching obtained with Configuration 3.

4.7.5 Test with post-processing on N-Gram frequencies

Other experiments where carried out after adding a new module to our system.[2] This module is in charge to count the numerosity of a N-Gram on the entire dataset. The distribution that we obtained after we counted how many occurences of the same N-Gram we had in the dataset permitted to us to execute an extra phase of post-processing on data. In particular we focused our attention on the N-Grams generated starting from intervals. Our target was to remove the "head" and the "tail" of the N-Grams distribution, too see if this operation could improve the performance of our system. Operatively this operation consist of avoiding to include in our segments all the N-Grams belonging to the first or the last part of the graph showed in figure 3.16. How wide is the "head" or the "tail" to remove is decided by parameters passed to our module.

We tried few tests after we processed our dataset and the query set with this new function, but without achieving better results than the ones obtained with non processed sets. Briefly, we removed, first together and then separately, the head and the tail of the N-Gram distribution. The remotion we performed for the head was done on the first two N-Grams more frequent in the collection. On the other hand the remotion we performed on the tail aimed to almost all the N-Grams which presented only one or two occurences in the whole dataset.

We computed all of the three post-processing operations both on *Configuration 13* and *Configuration 4*.

As it possible to see from figure 4.8 MRR values after N-Grams deletion of both more frequent and less frequent N-Grams, are worst than values obtained with non processed sets. Results are worst for both configurations, losing more or less both a 0,3 value for the MRR measure (computed using MINHF ranking function). Also is possible to see that responsible for these lower values is only the cutting operation performed on the head while the test where we performed only the remotion on the tail has same MRR values than configurations without

[2]Commands to use this module of the system are listed in the last part of [22].

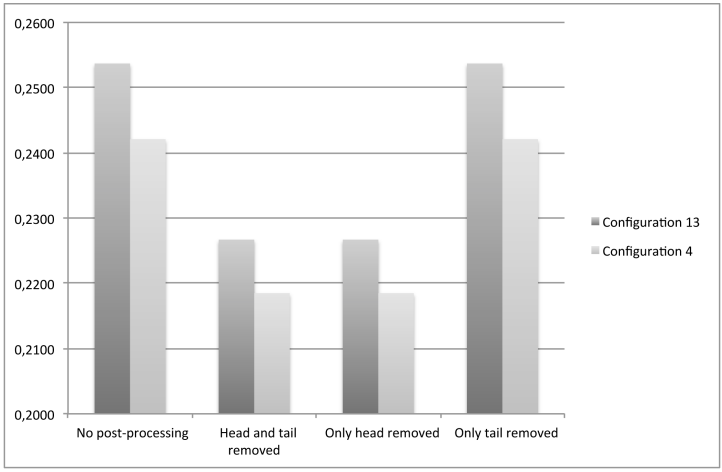

Figure 4.8: MRR value for Configuration 13 and Configuration 4 after having applied different N-Grams post-processing operations. MRR value calculated with MINHF ranking function.

post-processing. Anyway it is possible that, using higher values for the cutting parameters, measures values will improve.

4.8 Computational time

In order to conclude the analysis of the application, it is important to consider briefly the computation time of the algorithm. As mentioned in Chapter 2 the most part of the functions of this systems are implemented using Matlab. The only exception is Falcon that is implemented using Java. In fact, Matlab is a very good tool to test an algorithm, but it does not fit for a commercial application because of its structure. Matlab is optimized for the matrix calculation and it does not work so fast when, as in this context, a lot of hidden cycles are present.
The main parameters that affect the running time of the application are:

- Frame length used to detect pitch values;
- Frame overlap used to detect pitch values;
- Segment length used to segment the N-grams sequence;
- Segment overlap used to segment the N-grams sequence.

Off course, the length of a song is playing a fundamental role on computational time but, since we are talking about pop and rock genres, usually songs belonging to these kinds of music have a length between three and four minutes.
The system used to test the application is composed of an Intel Core 2 Duo 2.4 Ghz with 4 GB as RAM memory and Mac OS X 10.6.8 as operating system.

Basing on parameters used for *Configuration 13* we can split the total time in different pieces that compose the total amount of computation.

Pitch detection: pitch detection process takes usually between 20 and 40 seconds to process each song, depending on the song's length. Considering that we had to detect pitch values for the 50 query files and the full dataset of 555 files the pitch detection module taken more or less 5 hours to compute pitch values for every song.

Pitch post-processing, intervals creation, n-gram creation, segmentation: all of these operation were really fast. We can ignore the time taken by these phases.

Falcon indexing and retrieval: Falcon is really fast, since it is implemented in Java. Usually for the indexing of the dataset of 555 files it takes more or less 8 seconds. For the creation of the ranked list of all the 50 query Falcon it usually takes 30 seconds. Since we are usually creating 3 different result files, one for each query type, Falcon takes more or less 1,5 minutes for the indexing and retrieval tasks.

Results evaluation: the results evaluation routine is written in Matlab and it takes always around 3 minutes to compute MRR and MAP for each set of 50 queries. The different segmentation or changing values for pitch detection do not afflict time taken by this function.

As it possible to see the bottleneck is the calculation of pitch exploited by mirpitch. Anyway we have to say that calculation of pitch values for the entire dataset has to be done only once. The results, thus, are not so bad and a different implementation of the pitch detection algorithm using, for example, the Java language, will improve surely the computation time.

4.9 Memory usage

After the overview on computational time, to have a complete point of the situation for the system, it is worth to present also details about usage of memory of our system. For convenience, as we did for computational time, we divide the usage of memory in different parts, each one of them compose the total amount of memory used.

Sound files: even if they are not really needed to perform indexing and retrieval, since we are working only on text files containing results of melody recognition phase, we need them in the first phase to compute all the pitch values we used for following operations. All of these files had a sampling frequency of 22050 Hz and they are monophonic recordings. All the files were in WAV format. Considering that usually, a song, has a length comprised between three and four minutes the average dimension for a file is 10 Megabytes. This is confirmed by the fact that our system (that works on 605 files: 555 files for the dataset and 50 query files) present a usage of memory for sound files equal to 6,02 Gigabytes.

File descriptor: after all the phases focused on the creation of the melody representation, data for every song is stored in a text file, saving for every segment of the song the N-Grams' hashes. Values for memory usage of these files are depending by a lot of parameters (segment length, segment overlap, number of types of N-Grams used, etc.). Considering the two best cases for fine intervals representation and semitones representation memory usage for the 50 query files is respectively 7,3 Megabytes and 7,2 Megabytes. We can deduce that the mean memory usage for a single file is approximately 150 Bytes. The total memory usage for our system is around 90 Megabytes.

Falcon index: as said before Falcon need to create an index from the dataset of 555 files. The size of this index is usually comprised between 3,5 Megabytes and 4,5 Megabytes depending on the parameters used to process the dataset values.

Chapter 5

Conclusions

During this work, we took care of the live and cover song recognition problem. The idea consists of retrieving the original song starting from an unknown audio recording that can be also a live version or a cover of a song. The original song is part of a dataset previously indexed.
The proposed approach to this problem consists in the use of the melody to index and retrieve the original song.
First of all we extracted the pitch values for every song. This operations is exploited by a Matalab function included in MIR ToolBox: mirpitch. Then we made some post-processing operations aimed to obtain a more reliable sequence of pitch values.
After the pitch values were properly processed we created a representation of the melody using intervals. This representation is robust to tonality changes and, since it does not represent tempo, it is also robust to tempo variations.
After the creation of intervals representation we transformed it into N-Grams representation. This was done to have a more efficient way to index our melody and subsequently to retrieve it. In fact, for every single N-Gram we gave a number identifying it. This means that we simply created an hash for each N-Gram we had.
Once all this codification phase was done we started, using Falcon, to test our system using 50 query files. These files were composed of live versions and also live versions played by different artists than the songwriter. These files were tested again a dataset of 555 other files representing the original songs.
After a wide range of tests on different parameters, starting from an arbitrary basic situation, we figured out an optimal configuration for our system. This configuration had a MRR value of 0,2537 and a MAP of 0,000692. These values, even if they are not so close to the perfection, gave us a first hope that the way we started is was right.
After we found a first stable configuration more other tests were carried out trying different types of representation or changing other kinds of parameters in our algorithm.

What came out from last tests was a sub-optimal configuration that use *semitones representation* instead of *fine intervals representation*. This configuration had a MRR value of 0,2421 and a MAP of 0,000695. Even if this configuration does not add better results to the ones founded with the optimal configuration, it can be a good starting point for future tests.
The system gave encouraging results on precision and accuracy. Also, we achieve the target of efficiency: the system takes around 3 seconds to perform the query operation.

5.1 Future developments

Talking about future developments there are many directions that can be explored.
On of the is a research, or a new implementation, of a pitch detector that works better than mirpitch or, also, to test different configurations for mirpitch. The focal point is that, having a less disturbed and more reliable pitch values, we can perform a better melody representation.
Also other approaches could be tested, using different types of representation and changing parameters for test configurations. Even a rank fusion procedure between BM25 and MINHF can be explored. This because we noticed on our tests that often, when MINHF has bad values BM25 has better values.
Another option is to combine results coming from melody-based system with results coming from CHROMA-based system. Melody approach can, in some way, affect CHROMA-based systems raising performance of several percentage points. In a more general way, the approach of results fusion with other system available, performing audio recognition, even if they are not dedicated to live and cover songs, could be a directions to explore.
Considering a possible future application for this system we can say that a target to focus on is a new implementation aimed to be used as a stand alone application. This can permit to integrate currently available tools of music recognition with features exploited by our system.

Appendix A

This section is intended to give a quick and general overview about some musical concepts involved in the work detailed in this thesis.

Tones and Semitones

We start explaining the concept of *semitone*. A *semitone* is the minimum interval that is possible to found in music, as known in the current theory for musical harmony. Two semitones together form a *tone*. *Tones* and *semitones* are used to indicate the distance of a note from another. Figure 5.1 represent the distance between notes in our musical system.

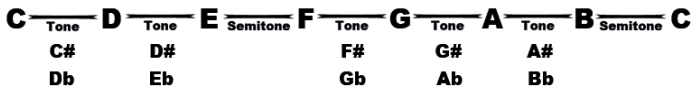

Figure 5.1: Distance between notes using tones and semitones.

As it possible to see from figure 5.1 intervals of a tone are divided in a half by altered note (C#/Db, D#/Eb, etc. where C# and Db are the same pitch value called with two different names considering if we want an altered C or an altered D). So, considering all the notes, including altered ones, we can say that every note is separate from the next or the previous one by an interval of a semitone.

Intervals

As seen in the previous section we can always identify an interval between two notes just counting the number of semitones encountered moving from one note to the other. To identify in a faster way an interval between a note and another, tones and semitones are grouped together to create *intervals*. Each interval has a name and a quality. The process to create intervals grouping

Interval name	Interval quality	Example	Semitones
Unison	diminished	C - Cb	-1
Unison	perfect	C - C	0
Unison	augmented	C - C#	1
Second	diminished	C - Dbb	0
Second	minor	C - Db	1
Second	major	C - D	2
Second	augmented	C - D#	3
Third	diminished	C - Ebb	2
Third	minor	C - Eb	3
Third	major	C - E	4
Third	augmented	C - E#	5
Fourth	diminished	C - Fb	4
Fourth	perfect	C - F	5
Fourth	augmented	C - F#	6
Fifth	diminished	C - Gb	6
Fifth	perfect	C - G	7
Fifth	augmented	C - G#	8
Sixth	diminished	C - Abb	7
Sixth	minor	C - Ab	8
Sixth	major	C - A	9
Sixth	augmented	C - A#	10
Seventh	diminished	C - Bbb	9
Seventh	minor	C - Bb	10
Seventh	major	C - B	11
Seventh	augmented	C - B#	12

Table 5.1: Creation of intervals grouping tones and semitones.

tones and semitones is explained in the table 5.1.
Intervals presented in the table are ascendent intervals because are created considering that the second note has a higher pitch than the first one. For descending intervals name are the same, just the second note has a lower pitch than the first one.

As it possible to see from values in semitones column on table 5.1 some intervals with the same number of semitones difference can take different names. This is used in music to solve ambiguity between notes with the same pitch but with different name.

What we did in the work of this thesis was simply to calculate, starting from the pitch value of two notes, the difference between them in terms of semitones.

Calculation of semitones difference between two pitch values

To calculate the distance between two pitch values in term of semitones we simply start from a formula that calculate the frequency of a note knowing the difference of the note from the *A4* in terms of semitones. When we talk about *A4* we intend the note that is usually used as reference to tune instruments. This note is an A with an absolute frequency of 440 Hz.
The formula used to calculate the frequency of a note is:

$$frequency = 440 * 2^{n/12} Hz$$

where n is the number of semitones separating the desired note from the *A4*. In example, if we want to calculate the frequency of C5 we just need to substitute n with the value 3.

For our scope we used the formula in a reverse manner. In this way, starting from a frequency value, we calculated the interval from the founded note to the A4 in terms of semitones. The formula adopted is:

$$semitones = 12 * log_2(PitchValue/440)$$

Once the distance in semitones is computed we simply used a subtraction to obtain the difference in semitones between two notes.

$$intervals = round(note2 - note1)$$

In the formula above *note2* is the difference in semitones from A4 for the second note of the interval and *note1* is the difference in semitones from A4 for the first note of the interval. The function *round(..)* simply compute a rounding operations on final value because, as it possible to see from table 5.1, to compute an interval value we need a number of semitones without decimal digits.

Bibliography

[1] J. S. Julià, "Identification of versions of the same musical composition by processing audio descriptions," 2011. PhD Thesis, Universitat Pompeu Fabra, Department of Information and Communication Technologies.

[2] "Mirex home page." `http://www.music-ir.org/mirex/`. Last visited: December 2011.

[3] E. D. Buccio, N. Montecchio, and N. Orio, "Falcon: Fast lucene-based cover song identification," 2010.

[4] R. Miotto and N. Orio, "A music identification system based on chroma indexing and statistical modeling," 2008.

[5] M. Marolt, "A mid-level representation for melody-based retrieval in audio collections.," 2008. IEEE Trans. on Multimedia, 10(8), 1617–1625.

[6] C. Sailer and K. Dressler, "Finding cover songs by melodic similarity. music information retrieval evaluation exchange (mirex) extended abstract," 2006. Available online: `http://www.music-ir.org/mirex/abstracts/2006/CS_sailer.pdf`.

[7] W. H. Tsai, H. M. Yu, and H. M. Wang, "A query-by-example technique for retrieving cover versions of popular songs with similar melodies.," 2005. In Proc. of the Int. Conf. on Music Information Retrieval (ISMIR), pp. 183–190.

[8] W. H. Tsai, H. M. Yu, and H. M. Wang, "Using the similarity of main melodies to identify cover versions of popular songs for music document retrieval.," 2008. Journal of Information Science and Engineering, 24(6), 1669–1687.

[9] E. Gómez and P. Herrera, "The song remains the same: identifying versions of the same song using tonal descriptors.," 2006. In Proc. of the Int. Conf. on Music Information Retrieval (ISMIR), pp. 180–185.

[10] A. Ehmann, D. P. Ellis, E. Gómez, B. S. Ong, G. E. Poliner, and S. Streich, "Melody transcription from music audio: approaches and evaluation.,"

2007. IEEE Trans. on Audio, Speech and Language Processing, 15(4), 1247–1256.

[11] A. D. Cheveigné, "Pitch perception models.," 2005. In C. J. Plack, A. J. Oxenham, R. R. Fray, & A. N. Popper (Eds.) Pitch: neural coding and perception, chap. 6, pp. 169?233. New York, USA: Springer Science.

[12] A. D. Cheveigné and H. Kawahara, "Comparative evaluation of f0 estimation algorithms.," 2001. In Proc. of the Eurospeech Conf., pp. 2451–2454.

[13] L. J. Solomon, "Solomon's glossary of technical musical terms," 1996.

[14] A. Latham, "The oxford companion to music," 2002.

[15] C. Isikhan and G. Ozcan, "A survey of melody extraction techniques for music information retrieval," 2008.

[16] A. de Cheveigné, "Yin - a tool for pitch relevation." Online available `http://www.ircam.fr/pcm/cheveign/sw/yin.zip`. Last visited: January 2012.

[17] A. de Cheveigné and K. Hideki, "Yin, a fundamental frequency estimator for speech and music," 2002. J. Acoust. Soc. Am., 111, 1917–1930.

[18] D. Ellis, "Labrosa melody analyzer website." `http://labrosa.ee.columbia.edu/projects/melody/`. Last visited: January 2012.

[19] O. Lartillot, "Mirtoolbox home page." `https://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mirtoolbox`. Last visited: January 2012.

[20] "Mirtoolbox primer - user guide." `https://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mirtoolbox/Primer`. Last visited: February 2012.

[21] "Mirtoolbox mailing list." `http://www.freelists.org/list/mirtoolboxnews`. Last visited: February 2012.

[22] L. Romanato, "Operative manual on how to perform pitch extraction and melody representation creation for the thesis: Scalable cover song identification based on melody indexing." The manual is included in the CD inside DEI thesis library.

[23] O. M. D'Antona and L. A. Ludovico, "Una classificazione formale degli intervalli musicali," 2002.

[24] L. Romanato, "Data analysis report for notes' lengths and interval representation on the query set and on the dataset of files where to retrieve results." Report is included in the CD inside DEI thesis library.

[25] "Dataset - 555 files - songs' names and authors." Dataset list is included in the CD inside DEI thesis library.

[26] "Okapi bm25 on wikipedia." http://en.wikipedia.org/wiki/Okapi_BM25. Last visited: March 2012.

[27] "Tfidf on wikipedia." http://en.wikipedia.org/wiki/TF-IDF. Last visited: March 2012.

[28] E. D. Buccio, N. Montecchio, and N. Orio, "Falcon home page." http://ims.dei.unipd.it/falcon. Last visited: December 2011.

List of Tables

List of Figures

Acknowledgments

This thesis represent the end of my university career. So, after five years (and a little bit more) spent for this experience, there are some people that I want to thanks.

The first thought is for my parents, my mother Ermelinda and my father Massimo. They always supported and helped me, since the beginning of my studies. It is thanks to them if i could have done the work on this thesis because of their support also on the musical side of my studies. The second thought is for my brother Alessandro, with the hope that, between satisfactions and sacrifices, his university career may be an important life experience as was mine.

A huge thanks to my friends that, within last few years, completely redefine my concept of friendship, helping me probably in the worst period of my life. So, a big thanks is first of all to Giacomino *Jibber* James for his support, the long nights talking and his "livin' easy style". A big thanks also to Mattia, my fellow in the Nerdship Alliance, Sole and RockSun, for the incredible experience in summer 2010, Deborah, Marta, Manu. A thank also to my childhood friends: Chiara, Giulia, Marco, Stefano (people change...)

A special section is dedicated to the guys that tried and are trying to live with me a dream. This is for you: Gillo, Chicco, Luca and Beppe. The band that we created is something that I was dreaming since I began to play. I never expect to have something like that in my life and this is thanks to all of you. We start to created our path, and i hope that our path may be still long and full of satisfactions..... obviously destroying every stages in the world!

Now is the time to thanks a girl that keep supporting me in all of these months, especially in the last period, where I really was not the best boyfriend on the earth. So, thanks Sara. You have been in all of these months a constant source of inspiration, new ideas and someone that was always there for me. Distance is not an easy thing to manage, but we are doing it great, and I hope we will go on like this.

A huge thanks to all the people that I met in one of the most amazing experience of my life: Erasmus. 9 months in Ireland would not be so incredible without you. Thanks to Laura, Enrico, Fee, Filippo, Pietro, Roldan, Ines, Odelette (fake chinese names....), Leslie, Cristian and all the guys that I met there. I came home surely a different person and I hope different can mean better. It is also thanks to this experience if I can wrote this thesis in this language using its “hated by me ” s.

A thanks to all the Sweet Basil people. Acknowledgments are for Lorenzo, all my guitar teachers (Jacopo, even if was not in Sweet Basil, Claudio, Raffaello, Khaled, Daniele), all other teachers (Fabio, Valentino) and incredible musicians that I met there. A special thanks to Michele, for the opportunity gave me some years ago; it really changes my life, my mind and moreover myself.

Finally i want to thanks my advisor Nicola Orio. Once again, after the experience of my first degree thesis, i lived a great experience also doing the work for this thesis. He has given me the possibility to make a very interesting work and to understand the pleasure of making research. He is the one that made me discover the great side of academic world with his passion for his work and his willingness.

At the really end, something that i really like, just because i hate to forget people in my acknowledgments: thanks to everybody I forget in this section!

Thanks to everyone...

Leonardo Romanato