University Name

Doctoral Thesis

A content-aware interactive explorer of digital music collections: The Phonos music explorer

Author:
John Smith

Supervisor:
Dr. James Smith

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Research Group Name
Department or School Name

March 2015

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

UNIVERSITY NAME (IN BLOCK CAPITALS)

Abstract

Faculty Name
Department or School Name

Doctor of Philosophy

A content-aware interactive explorer of digital music collections: The Phonos music explorer

by John Smith

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too…

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor…

Contents

List of Figures

List of Tables

Abbreviations

MFCC	**L**ist **A**bbreviations **H**ere
GUI	**L**ist **A**bbreviations **H**ere
HPCP	**L**ist **A**bbreviations **H**ere
MIR	**L**ist **A**bbreviations **H**ere
RS	**R**ecommender **S**ystems
BPM	**R**ecommender **S**ystems
FFT	**R**ecommender **S**ystems
OR	**R**ecommender **S**ystems
EM	**E**xpectation **M**aximization
EMD	**E**arth **M**overs **D**istance
SVM	**S**upport **V**ector **M**achine

For/Dedicated to/To my...

Chapter 1

Introduction

1.1 The importance of music analysis

The incredible growth of the Web over the last pair of decades has drastically changed many of our habits. One of the areas that have been highly affected by this fast-paced growth is our consumption of multimedia contents: the use of physically-stored content is seeing itself heavily reduced, as we are more and more getting used to the access of huge databases of multimedia content through the Web.
(it would be nice to cite [3] here to present the subject in a more elegant way) Music is one of these fields that have been revolutionized by this trend: the last decade has seen the rise of several Web services (iTunes, Spotify, Pandora, Google Music just to name a few) that offer their users an easy way to access their enormous catalogue of songs. Statistics show an increasing rate of annual growth for each of these services, in both the amount of users and of revenues: now they are among the most used ways of enjoying and discovering music.
However, the transition to this type of services has brought to some new problems. One of them relies on the vastness of these databases: given that users want to easily discover new music suitable to their tastes through intelligently created playlists, a way to reasonably pick songs and artists among the entire catalogue is needed.
This, among others, has been one reason of the rapid growth of **Music Information Retrieval** (MIR), an interdisciplinary research field whose subject is to provide new ways of finding information in music. Main techniques of describing music can be grouped into two categories:

- Metadata (literally *data describing data*), descriptors of music not directly retrieved from the audio signal but instead from external sources [1]
- Audio content descriptors, automatically computed from audio.

When it comes to choosing one method over the other, it becomes clear that both these categories of tools have their own pros and cons. Regarding metadata, major concerns arise from the questionable consistency of the descriptors among the entire catalogue catalogue of music, given that they may have been extracted from several sources. Other concerns also arise from how well they actually describe the audio track. On the other hand, audio content descriptors (especially the low-level ones) may have no musical meaning and therefore they could be hard to understand. Many efforts have be taken in order to improve the methods of information extraction of both these categories. In general, however, audio content descriptors are thought to be more flexible, since they can be easily and equally computed for any track. One advantage of this technique relies on the fact that these kind of descriptors could easily be computed not just for each kind of song, but also for any segment inside of it. This has for example been exploited by *Shazam*, a widely-used smartphone app for music identification that analyzes peaks in the frequency-time spectrum throughout all song length to build a very robust song identification system [1]. Another popular product that performs audio content analysis just for short segments of a song is The Infinite Jukebox[2], a web-application using *The Echo Nest* library and written by Paul Lamere, that allows users to indefinitely listen to the same song, with the playback automatically jumping to points that sound very similar to the current one. The Infinite Jukebox can be considered an application of the so-called *creative-MIR* [7], an emerging area of activity inner to MIR whose subject is to exploit MIR techniques for creative purposes. Other relevant software that exploits Echo Nest library for similar purposes is Autocanonizer[3] and Wub Machine[4]. However, there aren't many commercial or research-based software tools that exploit this kind of techniques for creative interaction or manipulation of audio tracks at the moment. Probably the most relevant commercial system is Harmonic Mixing Tool[5], that performs audio content analysis on the user's music collection in order to allow a pleasant and harmonic fade when mixing between songs. More recently, the research-based software

[1] There is a lack of agreement on the use of the term metadata, therefore its meaning could be different in other resources. For instance, it may be used to indicate all the data describing an audio file, including the ones derived from some computation on the audio signal itself.

[2] `http://infinitejuke.com`

[3] `http://static.echonest.com/autocanonizer`

[4] `http://thewubmachine.com`

[5] `http://www.idmt.fraunhofer.de/en/Service_Offerings/products_and_technologies/e_h/harmonic_mixing_tool.html`

AutoMashUpper has been developed with the intent of automating generating multi-song mashup[6] while also allowing the user a control over the music generated [8].

1.2 Phonos Project

Phonos project[7] is an initiative of the **Music Technology Group** (Universitat Pompeu Fabra, Barcelona) in collaboration with **Phonos Foundation**. Phonos was founded in 1974 by J.M. Mestres Quadreny, Andres Lewin-Richter and Luis Callejo, and for many years it has been the only studio of electroacoustic music in Spain. Many of the electroacoustic musicians in Spain attended the courses of the composer Gabriel Brncic at Phonos. It became Phonos Foundation 1982 and in 1984 it was registered at the Generalitat de Catalunya. In 1994, an agreement of co-operation with Music Technology group was established, with the purpose of promoting cultural activities related to research in the music technology. In 2014, the exhibition *"Phonos, 40 anys de música electrònica a Barcelona"*[8] has been planned at Museu de la Musica[9] (Barcelona) with the purpose of celebrating the 40th anniversary of Phonos and showing many of the instruments used in the studio, while allowing visitors to listen to the music works produced there during all these years. Given the songs' average length and their complexity, a way for the visitors to quickly and nicely explore a catalogue of songs produced in these 40 years was needed.

PHONOS FUNDACIÓ PRIVADA

FIGURE 1.1: Phonos logo.

1.3 GiantSteps

GiantSteps[10] is a STREP project coordinated by JCP-Consult SAS in France in collaboration with the MTG funded by the European Commission. The aim of this

[6] A mashup is a composition made of two or more different songs playing together.
[7] `http://phonos.upf.edu/`
[8] `http://phonos.upf.edu/node/931`
[9] `www.museumusica.bcn.es/`
[10] `http://www.giantsteps-project.eu/`

FIGURE 1.2: *Phonos, 40 anys de música electrònica a Barcelona*, Manifesto.

project is to create the "seven-league boots" for music production in the next decade and beyond, that is, exploiting the latest fields in the field of MIR to make computer music production easier for anyone. Indeed, despite the increasing amount of software and plugins for computer music creation, it's still considered very hard to master these instruments and producing songs[11] because it requires not only musical knowledge but also familiarity with the tools (both software and hardware) that the artist decide to use, and whose way of usage may greatly vary between each other. The GiantSteps project targets three different directions:

- Developing **musical expert agents**, that could provide suggestions from sample to song level, while guiding users lacking inspiration, technical or musical knowledge
- Developing improved **interfaces**, implementing novel visualisation techniques that provide meaningful feedback to enable fast comprehensibility for novices and improved workflow for professionals.
- Developing **low complexity algorithms**, so that the technologies developed can be accessible through low cost portable devices.

Started on November 2013, GiantSteps will last 36 months and the institutions involved are:

[11] "Computer music today is like piloting a jet with all the lights turned off." (S. Jordà). `http://vimeo.com/28963593`

- **Music Technology Group**, Universitat Pompeu Fabra, Barcelona, Spain
- **JCP-Consult SAS**, France
- **Johannes Kepler Universität Linz**, Austria
- **Red Bull Music Academy**, Germany
- **STEIM**, Amsterdam, Netherlands
- **Reactable Systems**, Barcelona, Spain
- **Native Instruments**, Germany

Figure 1.3: GiantSteps logo.

1.4 Purpose of this work

The purpose of this work is to develop a software to be used by visitors during the exhibition *Phonos, 40 anys de música electrònica a Barcelona* and that allows users to easily explore a medium-sized collection of music. This software is intended to exploit latest MIR findings to create a flow of music, composed of short segments of each song, concatenated in a way that the listener can barely realize of the hops between different songs. The application must also allow users to interact with it in order to have some control over the generation of the playlist; specifically, the user should be able to give a general direction to this flow (through some sliders or others GUI elements) in regards to some relevant music features, in a way that the change in the musical output can be perceived. The application developed is meant to be part of the GiantSteps project and therefore should follow the three guidelines explained in the previous page. In addition to this, given its future use on a public place, the application is required to be easy to use also for non-musicians, as many of the visitors of the exhibition could be.

1.5 Introduction to the problem of Playlist Generation

The problem of playlist generation has already been addressed by many popular music platforms, such as *Last.fm* [12], *Pandora* [13] and *Musicovery* [14]. The main objective of such services is to help users find tracks or artists that are unknown to them and that they may like, providing *personalized radio playlists*. However, a playlist may be defined, in a broad way, just as a sequence of tracks [17] and therefore its use could be more general. For instance, a common use of the term plylist refers to the broadcasting radio playlists, i.e. playlists made by DJs in a radio stations and often involving popular tracks. We can therefore define the problem of playlist generation as follows [17]:

> Given (1) a pool of tracks, (2) a background knowledge database, and (3) some target characteristics of the playlist, create a sequence of tracks fulfilling the target characteristics in the best possible way.

This task is made particularly challenging by the average size of the music database on which the generation of the playlist is needed: already, personal music libraries can be huge [18], hence the corresponding amount of information to be processed in order to the generate the playlist leads to very heavy computational tasks. Depending on the need of the application, these tasks may also be performed offline, although a real-time user interaction should be supported in many cases in order to allow the user to have some control over this generation process (such as in the case study of this work). As we will see in Chapter 2, extracting information from an audio signal is not a trivial task and many algorithms have considerable time-complexity, and this may lead to very long computational times already for the analysis of small-sized catalogues. Playlist generation is a well-known problem inside MIR [19] [20], since this task can be considered as a retrieval task if its definition is limited to the selection of tracks satisfying a user query [17]. Other major topics of MIR also include extraction of features and similarity analysis, that can be seen as a basis for building a playlist generation system [21].

[12] `http://last.fm`
[13] `http://www.pandora.com`
[14] `http://musicovery.com`

1.6 Structure of the dissertation

This dissertation is organized as follows:

- The first part will at first give an overview regarding music analysis techniques, explaining *metadata*, audio content analysis and the differences between them. Then, common techniques of music similarity computation will be explained.
- The second part will be about the methodology, explaining the different stages of the development, the problems faced and the techniques used. A presentation of the case study will introduce to an explanation of the reasons that lead to prefer the use of some techniques over others.
- Finally, experimental results will be shown, together with some ideas regarding future development of the application.

Part I

Background

Chapter 2

Music analysis techniques: state of the art

The main subject of MIR regards the *extraction and inference of musically meaningful features, indexing of music* (through these features) and the development of *search and retrieval schemes* [2]. In other terms, the main target of MIR is to make all the information regarding music over the world easily accessible to the user [2]. During the last two decades, several approaches have been developed, which mainly differ in the nature of the features they deal with. The main categories of sources from which features are extracted are generally considered to be the following ones: *music content*, *music context*, *user properties* and *user context* [4]. *Music content* data deals with aspects that are directly inferred by the audio signal (such as melody, rhythmic structure, timbre) while *music context* data refers to aspects that are not directly extracted from the signal but are strictly related to it (for example label [24], artist and genre information [25] [26], year of release [27], lyrics [28] and semantic labels). Regarding the user, the difference between *user context* and *user properties* data lies on the stability of aspects of the user himself. The former deals with aspects that are subject to frequent changes (such as mood or social context), while the latter refers to aspects that may be considered constant or slowly changing, for instance his music taste or education [4].
In this chapter, we will focus on the differences between *music content* and *music context* data.

2.1 Metadata

By metadata we mean all the descriptors about a track that are not based on the audio content[1]. Therefore, they are not directly extracted from the audio signal but rather from external sources. They began to be deeply studied since the early 2000s, when first doubts about an upper threshold of the performance of audio content analysis systems arised [5]. Researchers then started exploring the possibility of performing retrieving tasks on written data that is related to the artist or to the piece.
At first, the techniques were adapted from the Text-IR ones, but it was immediately clear that retrieving music is fairly more complex than retrieving text for several reasons:

- If metadata are used in a music recommendation system, they should take into account also the musical taste of the user who performed the query.
- Text documents are in general able to provide information about its content. The user who performs the query in the hope of retrieving a particular text generally has a good idea of how to form his query. For instance, in [3] Orio provides the example that if a user is looking for a text that is somehow linked to or setted in a tempest, he could just query “tempest”. On the other hand, music's abstractness makes it very hard for the user to formalize a precise query in order to retrieve audio.

As a consequence of this fact, it is generally believed that music characteristics can be described only in musical terms [3]. Yet the task of describing music remains hard, for musical terms are mainly referred to structural features rather than the content, and therefore terms like *concerto* or *ballad* are not useful to discriminate among the different hundreds or thousands of concerti or ballads [3].
During last years, many techniques exploiting metadata have been developed; they may differ both in the sources used for retrieving data and in the way of computing a similarity score, and clearly the performance of a system using metadata for similarity computation is highly affected by both of these factors. Sources may include [13]:

- Manual annotated data: description provided by experts; they may be referred to genre, mood, instrumentation, artist relations.
- Collaborative filtering data: data indirectly provided by users of web communities, in the form of user ratings or listening behaviour information.

[1]We recall again that there is a lack of agreement on the use of the term metadata; elsewhere it could be used with a different meaning, for instance it may indicate all the data regarding a music piece, including the one extracted from the audio signal itself.

- Social tags: data directly provided by users of social network of music (such as *Last.fm*[2]) or social games.
- Information automatically mined from the Web. Sources in these cases may include web-pages related to music or microblogs (for instance Twitter).

The availability of some of them greatly depends on the size of the music collection under consideration; for instance, as manual expert annotations might be very accurate, they would be extremely costly and probably infeasible on large collections [9]. In contrast, collaborative filtering data may be the most studied technique, given that it has already been intensively used in other different fields, for instance in movie recommendation. It is the predominant approach in the field of Recommender Systems (RS) [37] and is mainly focused on user ratings, generally leading to better results [12]. However, some concerns are related to this technique. First, collaborative filtering approaches have not been designed to be used for playlist generation, but mainly for recommending artists or music. Second, the availibility of datasets for user ratings in the field of music is very limited compared to other fields, and research is often based on very small samples [38]. Regarding listening behaviour information, they might be inaccurate for they don't keep track of song durations and of the user activities while listening to music [40]. In addition, there's no way of collecting negative feedback (*dislikes*) through them and, more in general, listening to a specific song doesn't necessarily imply liking that song [13].
Sources are also picked in relation to the subject of the research or of the system, that may be for example a recommendation or a similarity computation system. At this point, it's important to highlight the difference between the two of them: a recommendation system not only has to find similar music, but has also to take into account the personal taste of the user, and therefore it's generally considered as a basic tool to produce recommendation [23]. In any case, the terms "similarity" and "recommendation" cannot be substituted, given that a good similarity computation system doesn't necessarily equate to a good recommendation system [39]. The computation of similarity may happen through a Vector Space Model (a technique adapted from the Text-IR) and co-occurence analysis. In the next subsections we will briefly explain the characteristics and the performance of these techniques.

[2] `http://last.fm`

2.1.1 Vector Space Model

The main idea of this technique lies on building a bag-of-words representation[3] of a retrieved document, and then computing a term weight vector for each document. It's a frequently used technique in Text-IR (and in Computer Vision) which can safely be used when retrieving web pages related to music, in an attempt of computing similarity. In [41], one of the first work in this field, Whitman and Lawrence provided an analysis of this kind on music-related web pages retrieved with the queries (to the *Google* search engine) "artist" `music review` and "artist" `genre style`, where words such as music and review where added to improve the chances of automatically retrieving webpages related to music. After downloading the pages, the authors apply a Part-of-Speech[4] tagger to assign each word to its suited test set. Similarity between pairs of artists is then computed on the tf-idf (term frequency - inverse document frequency) vectors of the respective artists. In general, the tf-idf assigns more weight to words that occur frequently in a specific document but rarely in others [61]. This approach has later been refined, by applying better filtering to the corpus of the page (in order to consider only significant words) and with different similarity measures. The use of Vector Space Model has not been limited to webpages. For instance, it has been applied to microblogs (specifically Twitter) in [62], achieving a mean average precision (MAP) of 64% on a collection of 224 artists. Interestingly, results of more than 23000 single experiments showed that:

- Queries composed only by the artist name performed best, while adding other keywords showed a decrease in accuracy.
- Highest MAP values are achieved without filtering the corpus of the webpage (thus increasing the computational costs)

Vector Space Model has been used also with collaborative tags and games with a purpose. Collaborative tags are small annotation (usually just one or two words) created by users of social music platform, such as *Last.fm* of *MusicStrands*[5]. The main advantage of these tags over the microblog data lies in the limited dictionary used for indexing and in the meaningfulness of these tags; in other words, tags are generally less dispersive, less noisy and more musically meaningful. Furthermore, tags are available not only at artist level, but also at the level of albums or tracks. On the other hand, they require a large

[3]A bag-of-words can be basically seen as an extension of a programming language "dictionary": it collects words (that sometimes may just be an abstraction of much more complex features, such as computer vision descriptors) from a document, and then computes the frequency with which each of them appears in the document. Two different documents are considered similar if they contain the same or similar words with a comparable frequency.

[4]Software or algorithm for automatically assign each word in a text to a speech.

[5]`http://music.strands.com`

Figure 2.1: The user interface of *TagATune*.

and active user community. In this kind of community, moreover, the phenomenon of "popularity bias" is more frequent: much more data is available for popular tracks and artists, while no data at all might be found for artists in the *long tail*. If this kind of data is used inside a music recommendation system, a further phenomenon of "the rich gets richer" may happen, as more popular songs are more subject to be recommended and therefore will be tagged by even more users. Another problem with tags of *Last.fm* is that many of them are irrelevant to create a descriptive artist or song profile [4], as personal and subjective tags such as "love" or "favorite" may be found. This problem can be solved with the use of data coming from games with a purpose (for instance *TagATune*[6]), that are usually sources of much more meaningful data. In *TagATune*, two players are played a song, which could be either the same or different: they have to understand as soon as possible (in order to get a higher score) if the some they're being played it's the same.

2.1.2 Co-Occurence Analysis

The main idea behind co-occurence analysis is that two items that frequently co-appear in some kind of music-related document are similar. Data sources on which this kind of analysis is performed are typically music playlists [63] (compilation discs, radio

[6] `http://musicmachinery.com/tag/tagatune/`

stations or even simply users' playlist), music collections shared on peer-to-peer networks [41], as well as web-pages [64] [65] and microblogs [67] [66]. On microblogs, hashtags such as `nowplaying` or `music` are used to identify the individual listening events of a user and therefore building a simple user model. Co-occurence analysis with music playlists is commonly used in music recommendation system, while the same kind of analysis with web-pages is not common as Vector Space Models have yielded better results on the same kind of data [68].

2.2 Audio content analysis

The main idea behind this kind of analysis is to directly extract useful information, through some algorithms (or library of algorithms), from the audio signal itself. The type of content information extracted may greatly vary in relation to the need of the research, but we can mainly distinguish four categories [13]:

- *Timbral* information: related to the overall quality and color of the sound. There is not a general definition for music timbre; in [3], Orio has defined it as *"the acoustic feature that is neither pitch nor intensity"*. In other words, timbre embraces all the features that make a C4 played by a violin sound clearly different from the same note played by a piano.
- *Temporal* information: related to rhythmic aspects of the composition, such as tempo or length of measures.
- *Tonal* information: directly linked to the frequency analysis of the signal and to the pitch. It can describe what notes are being played or the tonality of a given track.
- *Inferred semantic* information: information inferred (usually through machine learning techniques) from the previous categories, in the attempt of giving a more defined and understable shape to the data collected. This kind of information may include descriptors such as genre, valence or arousal.

Information extracted through this family of techniques may also be categorized in the following way:

- Low-level data: information that has no musical meaning and that, more in general, is not interpretable by humans. Examples of this kind of descriptors are Mel Frequency Cepstral Coefficients (MFCCs) and Zero Crossing Rate (ZCR).

Preliminary steps

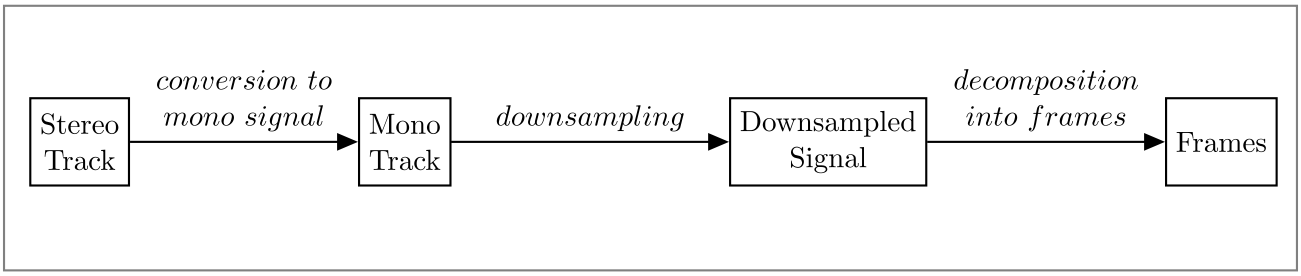

Figure 2.2: Standard procedure preliminary to the extraction of audio features.

- Mid-level data: information that has musical meaning but that is related to low-level music features. This kind of category mainly includes temporal and tonal descriptors.

- High-level data: corresponding to inferred semantic information.

Many of the studies conducted on the computation of music similarity through audio content descriptors have solely focused on low-level and timbral information, because this has been proved to bring alone to acceptable results with proper similarity measures [14]. However, more recent studies have shown some evidence of advantages in using high-level descriptors [15] [16] and, more in general, the most performant systems use data from all of these categories. When computing low and mid-level descriptors, the procedure requires the following operations:

- Conversion of the signal from stereo to mono, in order to compute all the descriptors for just one signal

- Down-sampling of the signal to improve the performance while computing the descriptors

- Segmentation of the signal into frames, short segments (usually from 512 to 2048 audio samples). Consecutive frames are usually not disjoint: the so-called *hop-size* determines the hop of samples between the beginning of a frame and the next one, and is normally half or a quarter as big as the *frame size*.

- Computation of Fast Fourier Transform, with an appropriate prior windowing technique[7].

The computation of descriptors is then performed on each frame, and finally a single value for each descriptor is computed by the means of specific statistical analysis. Mean,

[7] Although this last step may not be strictly seen as a necessary operation, many descriptors rely on frequency analysis of the signal and therefore they require the computation of the Fourier Transform.

median, variance and covariance are the most used statistical tools for calculating representative global values out of the enormous *pool* of values computed in each frame. Some more operations may sometimes be needed, such as de-noising [8] of time-scaling of the signal. In the next sections, a more detailed look among most important descriptors will be given.

2.2.1 Low-level descriptors

2.2.1.1 MFCC

Mel-Frequency Cepstral Coefficients (MFCCs) are a set of descriptors that have been widely used in MIR. They have been introduced in [69] for speech recognition: since then, they are used in state of the art systems for speech recognition; furthermore, they have shown prominent results on music similarity systems when a single or multivariate Gaussian distribution is computed over their values.
MFCCs are strongly related to human auditory system behaviour, which can be modeled by a set of critical bandpass filters (called *"auditory filters"*) with overlapping bands, as already shown by Hermann von Helmholtz in 1863. The term *critical band* was introduced by Harvey Fletcher in the 1940s, and indicates a range of frequencies around a specific one that may be not perceived in a totally independent way if played together to this reference frequency. This phenomenon is due to the inner behaviour of the cochlea, the sense organ of hearing within the inner ear. The mel bands are a set of bands that try to replicate auditory filters and therefore to somehow capture relevant features in the perception of music. Mel bands are based on the mel frequency scale, a perceptual scale of pitches judged by listeners to be equal in distance from one another. Mel frequency scale is linear at low frequencies (below 1000Hz) and logarithmic above. A popular formula from converting from f Hertz to m mels is:

$$m = 2595 \log_{10}\left(1 + \frac{f}{1000}\right) \tag{2.1}$$

Though there is not a standard procedure for computing MFCCs values, steps are generally motivated by perceptual or computational consideration, and follow approximately this procedure:

1. Computation of Fourier Transform of the signal (of the frame)

[8] A set of operations which purpose is to reduce the amount of background noise in a signal, therefore incrementing the signal-to-noise ratio (*SNR* or sometimes *S/N*).

2. Mapping of the powers obtained in step 1 onto mel bands. Then the logarithm of each power for each mel band is computed, to approximate the cochlea in the human ear more closely [14].

3. Computation of the discrete cosine transform of these logarithm values, in order to eliminate unnecessary redundancies.

4. Extraction of the MFCCs as amplitudes of the resulting spectrum.

Differences in the procedure often involve the shape of the windows used for mapping the spectrum into these bands, the choice of pre-filtering of the signal after step 1 or even the total number of MFCCs to extract.

2.2.1.2 Bark bands

The Bark scale, proposed by Eberhard Zwicker in 1961, is a psychoacoustical scale that tries to improve the mel scale, where each "Bark" stands for one critical bandwidth. Bark bands are 24, and are described as bands over which masking phenomenon and the shape of cochlea filters are invariant, which is strictly not true. To convert a frequency from f Hertz to B Barks we can use the formula:

$$B = 13a\tan\left(\frac{f}{1315.8}\right) + 3.5a\tan\left(\frac{f}{7518}\right) \tag{2.2}$$

The published Bark bands (given in Hertz) are[9]:

[0, 100, 200, 300, 400, 510, 630, 770, 920, 1080, 1270, 1480, 1720, 2000, 2320, 2700, 3150, 3700, 4400, 5300, 6400, 7700, 9500, 12000, 15500]

with corresponding band centers at:

[50, 150, 250, 350, 450, 570, 700, 840, 1000, 1170, 1370, 1600, 1850, 2150, 2500, 2900, 3400, 4000, 4800, 5800, 7000, 8500, 10500, 13500]

Again it can be seen that width of frequency bands grows slowly below 1000Hz, while showing an exponential growth at higher frequencies.

[9] `https://ccrma.stanford.edu/~jos/bbt/Bark_Frequency_Scale.html`

2.2.2 Mid-level descriptors

2.2.2.1 Rhythm

In traditional music notation, there are several notations for tempo. It may be expressed in BPM (beats per minute), MPM (measures per minute; commonly used in ballroom dance music) or by semantic notations indicating a range of BPM; an example of this last category of notations may be the popular system of Italian markings, such as *presto* (168-200 BPM), *andante* (84-90 BPM) or *allegro* (120-128 BPM).
In the field of MIR, accurate notations are needed, therefore semantic annotations are disregarded in favour of more precise notation such as BPM and Onset Rate (OR).

Onset rate
Onsets are generally defined as the beginning of a new musical event, and onset rate is therefore defined as the number of onsets in a time interval. This definition however hides several difficulties: several instruments might have a long attack time, therefore it is not trivial to determine the actual beginning of the note. Furthermore, in polyphonic music nominally simultaneous notes may be spread over tens of seconds, making this definition more blurred [42]. Onsets are generally considered to be different from notes: for instance, a glissando might involve many different notes but only one onset.

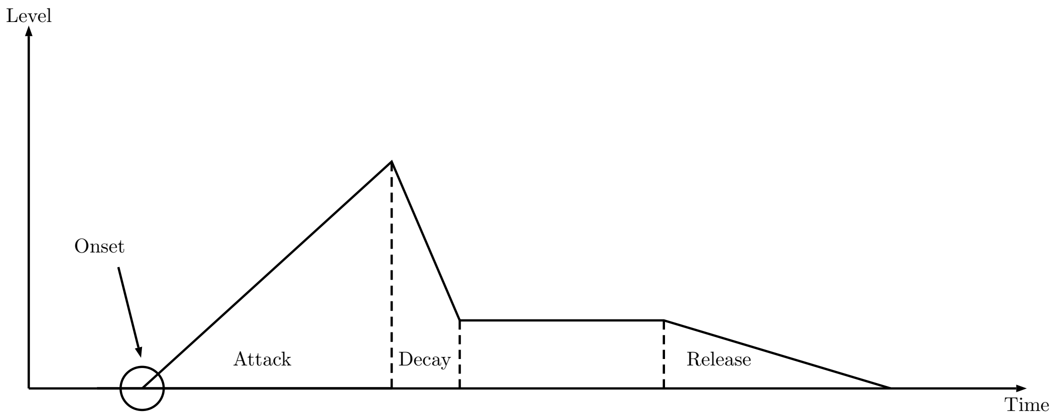

Figure 2.3: Onset and sound envelope.

Several ways of computing an onset detection function have been proposed, according to what aspects are taken into account for defining an onset. Actually, onset detection may be performed in time domain (when looking for significant changes in the overall energy), frequency domain (if looking for events regarding just a specific range of frequencies), phase domain or complex domain. Important algorithms for this task are:

- *HFC*, the High Frequency Content detection function that looks for important changes on highest frequencies. It is very useful for detecting percussive events.
- Spectral Flux, that decomposes the entire audible range of frequencies (approxitamely the interval 20-20000 Hz) into bins, measures changes in magnitude in each bin, and then sums all the positive changes across all the bins.
- the Complex-Domain spectral difference function [43] taking into account changes in magnitude and phase. It emphasizes note onsets either as a result of significant change in energy in the magnitude spectrum, and/or a deviation from the expected phase values in the phase spectrum, caused by a change in pitch.

HFC was proposed by Masri in [44]. Given the short-time Fourier transform (STFT) of the signal $x(n)$:

$$X_k(n) = \sum_{m=-\frac{N}{2}}^{\frac{N}{2}-1} x(nh+m)w(m)e^{-\frac{2j\pi mk}{N}} \tag{2.3}$$

(where $w(m)$ is again an N-point window, and h is the hop size, or time shift, between adjacent windows), the idea behind HFC is to give more weight to higher frequencies, by defining a onset function whose values are computed in the following way:

$$HFC(n) = \frac{1}{N} \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} k|X_k(n)|^2 \tag{2.4}$$

The HFC function produces sharp peaks during attack transients and is notably successful when faced with percussive onsets, where transients are well modeled as bursts of white noise [45].
On the other hand, the Spectral Flux SF function is defined as follows:

$$SF(n) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} H(|X(n,k)| - H(|X(n-1,k)|) \tag{2.5}$$

where $H = \frac{x+|x|}{2}$ is the half-wave rectifier function. This algorithm greatly characterizes changes in magnitude spectrum but it is quite weak to frequency-modulation phenomena (such as vibrato). To this end, the recently proposed variant SuperFlux [46] seems to achieve much better results.
Another interesting onset function is the *Complex Domain*, that calculates expected the expected amplitude and phase of the current bin $X(n,k)$ based on the previous two bins $X(n-1,k)$ and $X(n-2,k)$. By assuming constant amplitude the expected value

$X_T(n, k)$ is then computed:

$$X_T(n, k) = |X(n-1, k)|e^{\psi(n-1,k)+\psi'(n-1,k)} \tag{2.6}$$

and therefore a complex domain onset detection function CD can be defined as the sum of absolute deviations from the target values [42]:

$$CD(n) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} |X(n, k) - X_T(n, k)| \tag{2.7}$$

Given an onset function (for instance one of the already cited $HFC(n)$, $SF(n)$ or $CD(n)$), onsets are then extracted by a peak-picking algorithm which finds local maxima in the detection function, subject to various constraints. Threshold and constraints used in the peak-picking algorithm has a large impact on the results, specifically on the ratio of false positives[10] to false negatives[11]. For instance, a higher threshold may lead to a lower number of false negatives but to a higher number of false positive, while a lower threshold may have the opposite effect. A compromise, mostly specific to the application, has to been found.

BPM

We can roughly define beats as those instants in which we tap with a foot while listening to track; beats therefore correspond to moments of musical emphasis in an audio signal. The algorithms for detecting the beats-per-minute (generally called *beat tracking algorithms*) greatly rely on onset detection funcions. The basic idea is to look for some time-pattern that may explain the distribution of onsets over time, and hence derive BPM. They usually require more than one onset detection function to achieve good results. One of the beat tracking algorithm achieving very high reliability is TempoTapDegara, presented by N. Degara et al. in [47]. This algorithm models the time between consecutive beat events and exploits both beat and non-beat signal observations, estimating not only beats position but also the expected accuracy. It specifically analyzes the input music signal and extracts a beat phase[12] and a beat period salience observation signal, and then computes the beat period[13] from these values.

The beat tracking probabilistic model then takes as input parameters the phase observation signal and the beat period estimation, returning the set of beat time estimates. The quality of the beat period salience observation signal is finally assessed and a k-nearest neighbor algorithm is used to measure the reliability of the beat estimates.

[10]True onsets that are not detected by the algorithm.

[11]Points that are classified as onsets by the algorithm, while they are actually not.

[12]The location of a beat with respect to the previous beat.

[13]Regular amount of time between beat events.

A *complex spectral difference* method is used for computing the beat phase observation signal that will allow to compute all the other features. This onset function has shown good behavior for a wide range of audio signals and has been used with satisfying results in other beat tracking systems [70].

2.2.2.2 Tonality

Many efforts have been taken in order to improve the techniques for detecting tonality or harmonic content of a song, as this is one of the most main aspects of western music (a direct consequence of tonality is the detection of predominant melody; to understand why this is so important just try to think how many times you whistled or sang a song to let other people recognize it). Many studies focused on this aspect of music were not oriented toward the computation of similarity between tracks, but instead toward different tasks, such as *automatic trascription of a polyphonic audio signal* (mainly into a MIDI representation) and *source separation*, that is the task of isolating a single and specific instrument among many playing together.
From a music point of view, in western music an octave is made of 12 different pitches, and seven differents notes take place in this discrete range. According to the pitch assigned to each note, we may have different *keys*, that are a combination of a *tonic* (the central pitch) and the mode. *Major* and *minor* are the most popular modes.

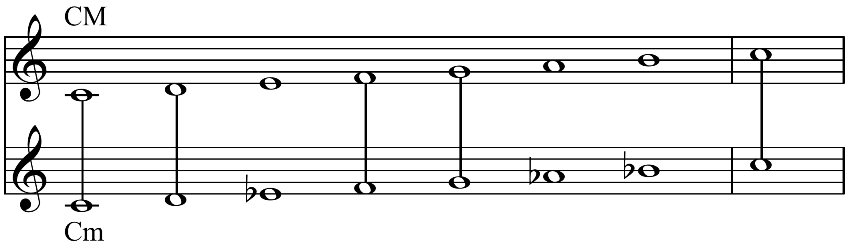

FIGURE 2.4: Major and minor modes of C.

Harmony is a term that denotes the simultaneous combination of notes, called *chords*, and over time, *chord progressions*. One of the most important descriptor for extracting information related to tonality is called Harmonic Pitch Content Profile (***HPCP***, also called chromagram). This is directly related to tonality and chord detection: chords can be recognized from from the *HPCP* without even precisely detecting what notes are being played, and tonality can also be inferred by *HPCP* (and in this case a previous estimation of chords is not necessary).
An *HPCP* is a $12k$ size vector indicating the level energy for each profile class. If $k = 1$, the *HPCP* represents the intensities of the twelve semitone pitch classes, otherwise

of subdivision of these[14]. In [49], Gómez proposes to distinguish tonality features on temporal scales:

- Instantaneous: features attached to an analysis frame.
- Global: features related to a wider audio segment, for instance a phrase, a chorus or the whole song.

Furthermore, Gómez proposes to split tonal descriptors in both low-level and high-level descriptors. We hence obtain the representation of tonal descriptors shown in Table 2.1.

Name	Temporal Scale	Level of abstraction
HPCP	Instantaneous	Low
Chord	Instantaneous	High
Average HPCP	Global	Low
Key	Global	High

Table 2.1: Main tonal descriptors.

The general approach for computing *HPCP* can be summarized as follows:

- At first, some pre-processing of the audio signal may be performed. For instance, a transient detection algorithm may be used to detect and eliminate regions where the harmonic structure is noisy. This step is usually performed to decrease the computational cost of the *HPCP* without affecting its output [50].
- At this point, spectral analysis is required: once the signal is segmented into frames of a proper size and a windowing function is applied, the Fast Fourier Transform (*FFT*) is computed to get the frequency spectrum. Frames should not be too short, in order to have a better frequency resolution.
- A peak-picking algorithm is run to find those frequencies corresponding to local maxima in the spectrum. Usually, these algorithms are not run only on the interval [100, 5000] Hz: this has shown much better results, because outside this range the signal is predominantly noisy, due to some percussion and instrumental noise [49].
- The *HPCP* is finally computed; many approaches have been developed for this task, all based on the pitch content profile algorithm presented by Fujishima in [48]. At first, a mapping of each frequency bin of the *FFT* to a pitch class is

[14] It may be extremely useful to study subdivision of semitone pitch classes when dealing with non-chromatic scales, that are very popular in eastern music.

needed (for instance, *FFT* bins corresponding to frequencies like 430Hz, 432Hz or 444Hz are mapped to the A at 440Hz). Then, the amplitudes inside each region are summed up and divided by the number of bins inside that region. Finally, the bins obtained are collapsed, so that bins referring to the same note but in a different octave (for example A4 and A5) are collapsed in a single bin for that note, indicating the overall energy of it in the frame. The *HPCP* is different from the PCP in the sense that a higher resolution may be used for *HPCP* bins (decreasing the quantization level to less than a semitone) and a weight function is introduced into the feature computation. The *HPCP* value of the n-th *HPCP* bin is calculated as:

$$HPCP(n) = \sum_{i=1}^{nPeaks} w(n, f_i) a_i^2 \tag{2.8}$$

where a_i and f_i are respectively the magnitude and the frequency of the ith peak, $nPeaks$ is the number of spectral peaks considered, and $w(n, f_i)$ is the weight of the frequency bin f_i when considering the *HPCP* bin n.
The performance of the *HPCP* builder strongly relies on the weight function [51]. Note that, for how the common procedure of building *HPCP* is defined, *HPCP* are usually considered robust to noise and different tuning references.
HPCP values are usually normalized in order to store the relative importance of the nth *HPCP* bin:

$$HPCP_{normalized}(n) = \frac{HPCP(n)}{Max_n(HPCP(n))} \tag{2.9}$$

Once the *HPCP* are computed, additional tonal features may be computed, such as tonality or chords. Regarding tonality estimation, this is generally computed through a correlation analysis between the *HPCP* obtained and a matrix of *HPCP* profiles corresponding to different keys.

2.2.3 High-level descriptors

Though there is not an univocal definition for it, the term *"high-level descriptor"* is generally used for indicating a descriptor that is the result of a machine learning process or of a statistical analysis of low and mid-level descriptors. The motivation behind them is to "translate" low-level features into descritors associated with a semantic explanation. A growing research attention in MIR has been targeted to some MIR classification problems, especially genre classification, artist identification and mood detection [72]. Music genre classification is usually performed on the basis of a machine learning process: a classification algorithm is trained on low-level data, such as timbre or rhythm descriptors. The quality of the system for predicting music genre highly depends

not only on the quality of the low-level data employed, but also on the typology and variety of music in the training set. For instance a system trained on a dataset of pop-rock music would perform badly on a classical music dataset. In addition, the process of labelling songs hides further difficulties, as labels themself could be misunderstood: for example, with the term "danceable" we usually refer to electronic dance music, but the term might correctly be applied to *waltz* music as well. The task of classification might be performed to detect also valence and arousal, and results are generally considered improvable. To this end, it is worth to mention the recent project *AcousticBrainz*[15], that is aimed at building a huge open database of descriptors computed (through Essentia, see Section 2.2.4) on the catalogue of songs of its users. One of its main purposes is to exploit this huge catalogue of music and data to explore the usefulness of current algorithms for low-level audio features extraction in music classification algorithms.
Automatic segmentation of audio is another task targeted by the MIR community: it consists of splitting songs into its compositional main parts (such as intro, refrain or outro) or its main rhythmic patterns, such as bars. Regarding the segmentation into compositional main parts, approaches generally follow the one proposed in [71], in which Foote performs a local self-similarity analysis in order to locate points of significant change. Segmentation is then performed computing a self-similarity matrix for the entire track, in which the similarity between excerpts starting at points of significant change is measured on the basis of low-level descriptors regarding rhythm, melody or timbre. Similar excerpts are then considered to belong to the same structural pattern of the song.

2.2.4 Main tools for extracting audio content

Many tools are available for the extraction of audio content descriptors from an audio signal. Many of them have been developed by researchers following the research necessities of MIR. This great variety of tools offers support to several operative systems (mainly Linux, Mac OS X and Windows) and programming languages (Java, C++, C, Python, Matlab). Some of this tools may be offered as standalone software or as a Vamp plugin. Not all the tools for extracting audio content are open-source, therefore they may not be particularly useful for the research community. In the following paragraphs, we will briefly show the features of the tools taken into account on the development of this work.

FIGURE 2.5: Essentia Logo.

Essentia

Essentia[16] is an open-source C++ library of algorithms for audio analysis and audio-based music information retrieval. It has been developed at Music Technology Group[17], Universitat Pompeu Fabra, and has released under the Affero GPL license[18]. In its current version 2.0.1, it contains a large collection of spectral, temporal, tonal, and high-level music descriptors, algorithms for audio input/output functionality, standard digital signal processing blocks and statistical tools. The library can be complemented with Gaia [19], a C++ library to apply similarity measures and classifications on the results of audio analysis. Both these libraries include Python 2.* bindings and support Linux, Mac OS X and Windows. Essentia has been in developed for over 7 years, incorporating the work of more than 20 researchers and developers through its history. It offers two different modes: standard and streaming, the first being imperative while the latter being declarative. Each processing block is offered as an algorithm, and has three different types of attributes: inputs, outputs and parameters. Different blocks may be linked in order to perform the required processing task. In Figure 2.6 a block diagram of a processing task is shown, composed of several different algorithms linked

HPCP COMPUTATION

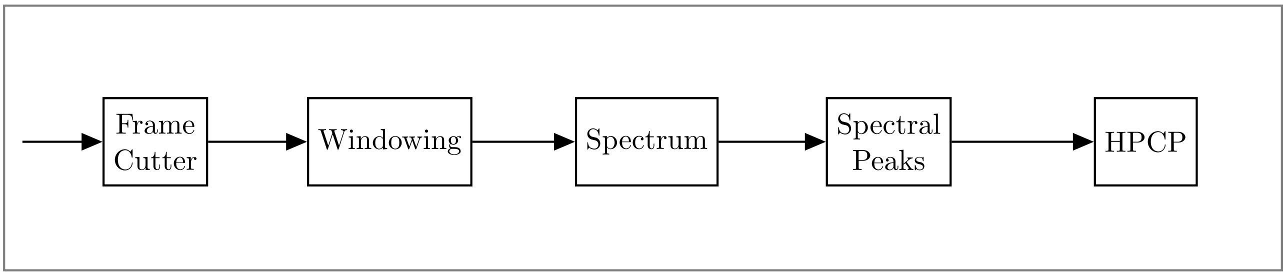

FIGURE 2.6: Computation of HPCP with Essentia. Each block corresponds to a different algorithm of Essentia.

[15] http://acousticbrainz.org/
[16] http://essentia.upf.edu/
[17] http://mtg.upf.edu/
[18] http://www.gnu.org/licenses/agpl.html
[19] https://github.com/MTG/gaia

together. See Appendix A for consulting the full list of descriptors provided by Essentia 2.0.1.

The Echo Nest

Figure 2.7: Echo Nest logo.

The Echo Nest[20] is a company that provides access, through Web API, to a collection of audio descriptors for a catalogue of over 36 million songs and almost 3 million artists. In order to access to this database, an API key is required, and some rate limits are imposed to the use of a free license (for instance, the maximum number of HTTP calls for minute is subject to a limit, generally 20). Users can decide to upload their collection into this database, so that descriptors will be computed for new songs and made available to other users. The performance of this library greatly depends on the chance that a song that is about to be analyzed has already been uploaded into this service. If this is not the case, the upload time has to be taken into account for performing the analysis task.

The Echo Nest provides a great amount of descriptors for each track (see appendix B for the entire list), ranging from very accurate audio content information to metadata, and has been used by several commercial solutions, developed by *Spotify*, *Rdio*, *Warner Music Group* and many others. Many official and unofficial libraries provide access to *The Echo Nest* service; among these, the most important one is probably the official Python library *Pyechonest*[21], that provides full access to all of the Echo Nest methods including artist search, news, reviews, blogs, similar artists as well as methods for retrieving detailed analysis information about an uploaded track. Furthermore, the library *Echo Nest Remix*[22] worths mentioning, as it is a library for audio manipulation and mixing and has been used by many *web-applications*, including The Infinite Jukebox.

However, the source code of *The Echo Nest* API service is not provided, therefore it has

[20] `http://the.echonest.com/`
[21] `http://echonest.github.io/pyechonest/`
[22] `http://echonest.github.io/remix/`

little usefulness to the research community. *The Echo Nest* has been aquired by *Spotify* on March 2014.

jMIR

jMir[23] is an open-source software suite implemented in Java and intended for use in music information retrieval research. Its development has been guided by Cory McKay (Marianopolis College), with many researchers from several institutions contributing to it. *jMir* is composed of several components differentiated in their scope, spacing from audio content analysis (performed by *jAudio*), to web mining of metadata and machine learning algorithms for classification.
The most relevant components of this suite are as follows:

- *ACE*: Pattern recognition software that utilizes meta-learning.
- *jAudio*: Software for extracting low and high-level features from audio recordings.
- *jSymbolic*: Software for extracting high-level features from MIDI recordings.
- *jWebMiner*: Software for extracting cultural features from web text
- *jSongMiner*: Software for identifying unknown audio and extracting metadata about songs, artists and albums from various web services.

MIRtoolbox

MIRtoolbox[24] is a set of functions for Matlab, dedicated to the extraction of audio content features from audio files. The design is based on a modular framework: algorithms are decomposed into stages, formalized using a minimal set of elementary mechanisms, with the objective of offering an overview of computational approaches in the MIR research field. MIRtoolbox has been developed at the Jyväskylän Yliopisto (University of Jyväskylä, Finland), by Olivier Lartillot, Petri Toiviainen and Tuomas Eerola.

[23] `http://jmir.sourceforge.net/`
[24] `https://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mirtoolbox`

2.3 Computing music similarity with audio content descriptors

Many sizable issues have to be taken into account when trying to compute the music similarity between two pieces. Main concerns are:

- Features whose values somehow reflect similarities between tracks must be discerned; this is especially important with audio content features, since many different descriptors can be extracted.
- Discerned features must be comparable by the means of proper distance functions;
- There is no definition of similarity; specifically it may vary according to the kind of music and to the users' background.

Thus, computing music similarity is a difficult task, because its wideness involves not only the choice of the similarity functions but also the descriptors and the data on which the computation of similarity is performed. Furthermore, music similarity cannot be thought as a static concept, as it highly depends on the context. For instance, in [73] Stober reports some context factors that may come into play when analyzing music similarity:

- Users: their preferences and background highly affect their perception of music similarity. Factors that may come into play are instruments played, listening habits and history, musical preferences.
- Retrieval task: according to the particular retrieval task, the concept of music similarity may be greatly affected. For instance, comparing the task of cover song retrieval takes into account factors that wouldn't be considered when comparing a more general music similarity distance between two different tracks.

The lack of a static concept of music similarity makes the evaluation task of a music similarity computation system difficult. The problem of evaluation of such systems will be presented in Chapter 3.

Approaches to music similarity based on audio content data are generally based on low-level timbre descriptors [5], rhythm features and tonal information. Before presenting important results in the field of music similarity, we will give a brief overview on similarity measures.

Similarity measures

A **similarity measure** (also called *"similarity function"* or simply *"similarity"*) on a set X is a function defined as:

$$s : X \times X \rightarrow R^{+} \tag{2.10}$$

satisfying the following conditions for all x, y and z in X:

- Equal self-similarity: $s(x, x) = s(y, y)$
- Simmetry: $s(x, y) = s(y, x)$
- Minimality: $s(x, x) \geq s(x, y)$

On the other hand, a **distance metric** (also called *"distance function"* or simply *"distance"*) on a set X is defined as the function:

$$d : X \times X \rightarrow R^{+} \tag{2.11}$$

satisfying the following conditions for all x, y and z in X:

- **Non-negativity**: $d(x, y) \geq 0$
- **Identity of indiscernibles**: $d(x, y) = 0$ if and only if $x = y$
- **Simmetry**: $d(x, y) = d(y, x)$
- **Triangle inequality**: $d(x, z) \leq d(x, y) + d(y, z)$

Non-negativity and identity of indiscernibles together produce **positive definiteness**. Some "light" distance metrics are also defined:

- pseudometric: allows violation of identity of indiscernibles
- quasimetric: allows violation of simmetry. If d is quasimetric, the symmetric metric d' can easily be formed as:

$$d'(x, y) = \frac{1}{2}(d(x, y) + d(y, x)) \tag{2.12}$$

- semimetric: allows violation of triangle inequality.

A similarity measure s and a distance measure d in a set X are compatible if and only if the following condition holds for all x, y, a and b in X:

$$d(x,y) \geq d(a,b) \Leftrightarrow s(x,y) \leq s(a,b) \tag{2.13}$$

A distance measure can be easily transformed into a similarity measure (or viceversa) by mapping $s = 1$ to $d = 0$ and $s = 0$ to $d = MAX$, and then reversing the order of objects in between (for instance with the Shepard formula $s(x,y) = e^{-d(x,y)}$) [73]. Therefore, we will use the two terms almost indistinctively.

Distance measures may be vector-based, sequence-based, set-based or distribution-based. Important vector-based distance measures are Minkowski distance, Mahalanobis distance and cosine similarity. Minkowski distance between vectors x and y is defined as:

$$d_{minkowski}(x,y) \equiv \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{2.14}$$

Frequently used special cases of Minkowski distance are:

- Manhattan distance: $p = 1$
- Euclidean distance: $p = 2$

On the other hand, Mahalanobis distance between two n-dimensional vectors x and y is defined as:

$$d_{mahalanobis}(x,y) \equiv \sqrt{(x-t)^T W (x-y)} \tag{2.15}$$

where W is a $n \times n$ weight matrix that can be derived from covariances in the data. When $W = I$, the Mahalanobis distance corresponds to an Euclidean distance. The cosine distance is finally defined as:

$$\cos(x,y) = \frac{\sum\limits_{i=1}^{n} x_i y_i}{\|x\| \|y\|} \tag{2.16}$$

and it corresponds to the angle between two n-dimensional vectors x and y. It is commonly applied in text retrieval.

Regarding sequence-based distance measures, a very important one is Hamming distance, which performs a character-by-character comparison of two sequences of equal length, returning the total number of indexes in which the two sequences differ. It is commonly applied in bioinformatics for DNA and RNA sequencing.

Regarding distribution based, two distances measures commonly used in MIR are Kullback-Leibler divergence and earth mover distance (EMD). Kullback-Leibler divergence between two discrete probability distributions A and B is defined as:

$$D_{KL}(A\|B) = \sum_i A(i) \ln \frac{A(i)}{B(i)} \tag{2.17}$$

However, there are some aspects that must be taken into account with this specific measure:

- It cannot be considered a real distance measures, as it fails to fulfill simmetry and triangle inequality
- When A and B are multivariate normal distributions, the formula can be rewritten as:

$$KL(A,B) = \frac{1}{2}\left(tr(\Sigma_B^{-1}\Sigma_A) + (\mu_B - \mu_A)^T \Sigma_B^{-1}(\mu_B - \mu_A) - k + ln\left(\frac{det\Sigma_B}{det\Sigma_A}\right)\right) \tag{2.18}$$

  where tr is the *trace* operation of a matrix (i.e., the sum of the elements in its main diagonal), μ_i the mean vector of the MFCC values of point i, Σ_i is its the covariance matrix, Σ_i^{-1} its inverse covariance matrix, and k the size of μ_i.

These specific problems will be addressed in Chapter 5.2. Regarding earth mover distance, it is a *cross-bin* distance function that addresses the problem of lack of alignment between bins domain. EMD defines the distance between two distributions as the solution of the transportation problem that is a special case of linear programming [75]. It is commonly used in image retrieval; its computation is not trivial and for further details we refer to [74].

Computing music similarity

Most of the techniques for music similarity computation are based on frequency spectrum analysis, as this has shown very good results since the first MIREX context (a more detailed overview about MIREX will be given in Chapter 3). Specifically, MFCC are frequently used, after their introduction to MIR by Beth Logan in [34]. One of the first studies on music similarity function based on audio signal analysis was published by Logan and Salomon in [76] (2001), where they propose to compare songs by the means of the EMD used on *signature models*, which are the results of k-means clustering applied to the MFCC values of the songs themselves.
Another very important contribute to this field came from Aucouturier and Pachet in

[6] (2004), where they propose to describe music pieces using multiple Gaussian mixture models (*GMMs*) (computed by the means of an Expectation Maximization algorithm) that fit the MFCC vectors best. Similarity computation is then performed by computing the likelihood of samples from GMM_{song_a} given GMM_{song_b} and viceversa:

$$sim_{a,b} = \frac{p(a|b) + p(b|a)}{2} \tag{2.19}$$

In 2005, Mandel and Ellis published a new fast method for computing similarity, based on the approach of Aucouturier and Pachet [77]. Instead of computing mixtures of three Gaussians to fit the MFCC vectors, they propose to use Support Vector Machines (SVM) to represent a song as a single Gaussian distribution. To build this single Gaussian distribution, they use the first 20 MFCCs of the song: specifically, they use the 20x20 covariance matrix and 20-dimensional mean vector of their values over the entire song. The comparison of two songs modeled as single Gaussian distribution is then done by computing the Kullback-Leibler divergence. A great advantage of using single Gaussians is that the simple closed Eq. 2.18 can be used with it for computing the KL divergence: no closed form for the KL divergence of two mixtures would exist otherwise, and in that case we should compute very expensive approximations, including EMD or MonteCarlo sampling. This method has shown to be much several orders of magnitude faster to compute, with just a little decrease in accuracy.
Another interesting result comes from Pohle et al. in [60], in which they present a high-accuracy system based on rhythm descriptors: specifically, they extend the descriptor of fluctuation patterns (that measures periodicities of the loudness in various frequency bands) into proposed extensions *onset patterns* and *onset coefficients*. Such descriptors differ are novel in the computation of onsets: the authors propose to:

- Reduce the signal to the parts of increasing amplitude (i.e., likely onsets)
- Use semitone bands to detect onsets instead of fewer critical bands
- Use Hanning window and zero padding before detecting periodicities with FFT
- Represent periodicity in log scale instead of linear scale

This system ranked in the very first positions at MIREX from 2009 to 2014 in the audio music similarity task. Interestingly, there haven't been many researches targeted toward combining low and high-level features; to this end, interesting results have been achieved by Bogdanov et al. in [72], in which they combine timbral, rhythmic and semantic features, for this approach is more appropriate from the point of view of music cognition. Specifically, for measuring the BPM similarity of songs X and Y (with BPMs X_{BPM} and Y_{BPM}) the authors propose the formula:

$$d_{BPM}(X,Y) = \min_{i \in \mathbb{N}} \left(\alpha_{BPM}^{i-1} \left| \frac{max(X_{BPM}, Y_{BPM})}{min(X_{BPM}, Y_{BPM})} - i \right| \right) \tag{2.20}$$

where $\alpha_{BPM} \geq 1$. Eq. 2.20 is based on the assumption that songs with the same BPMs or multiples of the BPM are more similar than songs with non-multiple BPMs; for instance, the songs X and Y with X_{BPM} and Y_{BPM} should have a closer distance than the songs X and Z with $Z_{BPM} = 100$. Furthermore this formula is robust to the frequent problem of duplicating (or halfing) estimated tempo in tempo estimation algorithms. With regard to high-level descriptors, the authors apply SVMs to low-level descriptors to infer different groups of musical dimensions, such as genre and musical culture, moods and instruments, and rhythm and tempo. This hybrid system ranked in the first positions at MIREX 2009 in the audio music similarity task.

Notable studies on large datasets

When large music collections are used, performance of similarity computation algorithms become critic. Although the collection to be used by the system during its public use can't be considered large, the necessary decomposition of it into hundred of thousands excerpt to be analyzed just in few seconds makes performance a critical factor when designing and implementing an algorithm. Therefore, a deep look into studies where large collections were used was needed.

The first content-based music recommendation system working on large collections (over 200,000 songs) was published by Cano et al. in [54], 2005. The system presented on this work relies on rhythmic and timbre features, combined to form a music similarity feature vector. No special indexing technique was used.

The first music recommendation system for large datasets using Gaussian timbre features was proposed some months later by Roy et al. in [55]. In this work, they propose to use a Monte-Carlo approximation of the Kullback-Leibler (KL) divergence to measure music similarity. The method proposed is, in principle, similar to the one proposed by Schnitzer et al. in [53], which has been used on the development of this thesis work (see 5.2). To pre-filter their results, Roy et al. increase the sampling rate of the Monte-Carlo approximation. As the divergence values converge, they are able to reduce the number of possible nearest neighbors. This method has shown good performance, both in query time and results.

A different attempt of improving performance was proposed by Levy and Sandler in [56] where they use only diagonal covariance matrix instead of a full one to compute music similarity. While this has shown a ten-fold speedup compared to the full Kullback

Leibler divergence, the quality of this simpler similarity measure results in worse genre classification rates.

2.4 Conceptual differences between metadata and audio content information

The difference between metadata and audio content data concerns not only the way the data is extracted, but the nature of the data itself. Given that they are generally provided by humans, metadata are human readable, comprehensible and semantically highly related to music facets. On the other hand, audio content data is generally a collection of data, and sometimes it is really hard to give these number a musical meaning. For instance, despite their intensive use and good performance in similarity functions, MFCCs are still considered "hard-to-read". This gap between the content of audio content descriptors and their actual meaningfulness is called *semantic gap*, and many efforts have been taken into the direction of making this gap smaller [11]. This difficulty of translating them into relevant semantic descriptors implies the hidden difficulty of detecting how much room there is for improvement of content-based music similarity systems [5], and generally leads to lower performance of content-based approaches [10]. However, several problems are related to metadata as well, and they are different according to the type of metadata under consideration. Generally, main concerns about the use of metadata come from their questionable completeness and consistency: they are generally available only for a small subset of songs, leaving the long-tail almost totally unprovided of meaningful data [29]. In an automated process, data completeness and consistency are crucial, for the performance of the process may greatly decrease in presence of inconsistent or incomplete data. For instance, if metadata are used in an automatic playlist generator, lacking data about some songs will result in never playing those songs, while inconsistencies may lead to the preferred choice of certain songs, again with the result of always skipping the remaining part of the music catalogue. Moreover, if metadata are in the form of social annotation or collaborative filtering, this situation may lead to the situation where "the rich gets richer" [13]: the more a track is retrieved by a music information retrieval system, the more metadata may be generated by its usage and thus the easier it will be for this track to be retrieved again. Given that social tags are generally subjective and come from a very large users' base, they may greatly suffer from inconsistencies [29]; furthermore, the perception of music is again subjective and can even be influenced by the perception of other people [30], making the quality of these data even more questionable. Problems of consistencies could be solved with the use of expert annotations, but the actual size of digital catalogue of music makes this choice clearly unfeasible. Regarding collaborative filtering, one significant problem lies

in sparsity of negative feedback: negative ratings are uncommon and, more in general, not inserting a track into a playlist or simply not giving an opinion about it carries no useful information about the track itself.
On the other hand, relying on the audio signal makes the process objective and easily extendable by automation. However, extracting the features can be computationally costly [22]. Moreover, some features as popularity or release data may be useful for the playlist generation process, and they cannot be extracted from the audio signal [23].
It is thus clear that, despite the average results show that metadata-based systems perform better, the choice of one kind of representation over the other should come from the context and from the purpose of the system to be developed. The two typologies should be seen as complementary rather than as mutually exclusive: recently, efforts have been taken in order to integrate the advantages of both in new music information retrieval systems, with encouraging results.

In the next chapter we will introduce the problem of assessing the performance of a music similarity computation system. This problem is very relevant, as a good evaluation process can highlight advantages and defects of each system, therefore leading the research community into a well-defined path to improvement.

Chapter 3

Assessing the performance of a music similarity computation system

3.1 Evaluation conferences in MIR

Evaluation of a retrieval system is a fundamental task in order to achieve continuous improvements of it on the basis of the results obtained. Evaluation of MIR systems in generally based on test collections [78], following the Cranfield paradigm that is traditionally employed in Text IR [79] [4]. On the other hand, the Text IR has a long tradition of conferences devoted to the evalutation of information retrieval systems, such as Text REtrieval Conference (TREC) [80] and Conferenceand Labs of Evaluation Forum (CLEF) [81], where research teams interested in participating in a specific task can use data published by organizers and submit the output data of their own information retrieval system. Results of submitted data are then compared and evaluated by organizers, and during the actual conference results are discussed with participants, thus encouraging sharing of new promising techniques and main concernings.
No such evaluation conference exists in MIR [4]. In 2000, the International Conference of Music Information Retrieval (ISMIR) series of conferences started, as the premier forum for research on MIR. The first edition[1] was held at Plymouth, Massachusetts (USA), covering the following topics:

- Estimating similarity of melodies and polyphonic music
- Music representation and indexing

[1] http://ismir2000.ismir.net/

- Problems of recognizing music optically and/or via audio
- Routing and filtering for music
- Building up music databases
- Evaluation of music-IR systems
- Intellectual property rights issues
- User interfaces for music IR
- Issues related to musical styles and genres
- Language modeling for music
- User needs and expectations

However, until year 2004, MIR systems were evaluated with self-made test collections: each research group was using different documents, queries and measures [3]. The first step toward a common evaluation framework has been carried out by Music Technology Group[2] of Universitat Pompeu Fabra (Barcelona), which hosted ISMIR in that year. The evaluation framework was called *Audio Description Contest* and divided into six indipendent tasks[3]:

- *Genre Classification*: label an unknown song with one out of six possible music genres
- *Artist Identification*: identify one artist given three of his songs, after training the system with seven more songs
- *Rhythm Classification*: label audio signals with one out of eight rhythm classes (Samba, Slow Waltz, Viennese Waltz, Tango, Cha Cha, Rumba, Jive, Quickstep)
- *Tempo Induction*: induce the basic tempo (i.e. a scalar, in BPM) from audio signals
- *Melody Extraction*: main melody detection from polyphonic audio signal

In the first edition, participants had to submit their own algorithms instead of the output data. These algorithms would have been compiled and run by organizers, who would have finally published the results. There was general agreement on the benefit of doing so, but it was also clear that data on which the systems were tested should have

[2]`http://mtg.upf.edu/`
[3]`http://ismir2004.ismir.net/`

been published before, so that researchers could test their systems before submission and improve them between editions. This has instead been possible thanks to the many efforts of Dr. J.S. Downie, who organized several workshops on MIR evaluation (with the purpose of collecting ideas and needs of researchers in MIR) and finally started the International Music Information Retrieval System Evaluation Laboratory (IMIRSEL) project[4]. The aim of the project is to create an provide secure and easily accessible music collections for MIR evaluation. Furthermore, the role of participants was made more important, as they could propose a particular task, defining the final goal, providing the datasets for training and testing the results, and defining the measures by which the results would have been ranked. The first evaluation campaign based on IMIRSEL, called *Music Information Retrieval Evaluation eXchange* (MIREX). was organized in the year 2005 in London, and the results were presented and discussed at the ISMIR of the same year. MIREX is still based on an algorithm-to-data paradigm: participants submite the code or binaries for their systems and IMIRSEL runs them with pertinent datasets. The ISMIR-MIREX is being held every year, with an increasing amount of retrieval tasks performed: since 2005, over 1500 different runs have been evaluated for 22 different tasks, making it the premier evaluation forum in MIR research. The next conference is scheduled at Malaga, 26-30 October 2015[5].

3.2 Difficulties in the evaluation of MIR systems

Several hassles specific only to Music IR have arised since the birth of this research field. The most important difference with Text IR lies in the availability of data: while textual documents are readily available for example on Internet, music files are protected by copyrights, and the expenses related to the use of such files would make it pratically impossible to create publicly accessible collections [4]. The result has been that research teams acquired their private collections of audio files; they then test and evaluate their system on this specific private collection, hence reducing both the reproducibility and the validity of the research.

Moreover, music is inherently more difficult than text, as it is composed of several facets (for instance timbre, rhythm, lyrics, etc.); the result is that a music piece is still perceived as the same one even after alteration of some facets, such as pitch or lyrics.

Finally, the size of music files is several order of magnitude larger than the one of text files, with the result of collections requiring large storage space.

For all these reasons, the option of providing an entire public collection of music files to be used during evaluation is generally disregarded. The only viable alternative in

[4]`http://www.music-ir.org/evaluation/`

[5]`http://ismir2015.uma.es/`

many cases is to just provide a set of features computed by third parties, such as in the Latin Music Database [82] or the Million Song Dataset [83]. The problem of this approach is that, not being provided of a direct access to the multimedia files, research teams are constrained to the use of provided features, hence not giving any room for the exploration of new features extractable from signals.

3.3 Evaluation of automatically generated playlists

The coherence of the tracks is a typical quality criterion for playlists [33]. Therefore, maximizing similarities between consecutive elements is an obvious strategy to generate and to evaluate playlists. In general, *user satisfaction* should be considered the main criterion to assess the quality of an automatically generated playlist. However, many factors come into play to determine the user's satisfaction: for instance we should consider the extent to which the list matches its intended purpose, fulfills the characteristic desired by the user, or is in line with user's expectactions [17] [84]. Hence it is clear that the evaluation of automatically generated playlists could concern many different aspects; for this reason, several categories of evaluation have been studied. In [85], Mcfee and Lanckriet propose to organize evaluation approaches in three different categories: human evaluation, semantic cohesion, and sequence prediction. In [17], Bonnin and Jannach propose to generalize this approach into the subdivision of evaluation approaches in four more general categories: user studies, log analysis, objective measures and comparison with handcrafted playlists.

- *User studies.* Allow to determine the *perceived* quality of playlists with good confidence. In such studies, participants to the experiment listen to automatically generated playlist and submit surveys in order to leave a feedback about the perceived quality of the playlist. The main drawback of this kind of studies is that they are time consuming and expensive. For some experimental designs, participants are required to listen to the entire catalogue of music. Studies of this kind are generally based only on 10 to 20 participants; only few studies have involved a considerable amount of participants (for instance, in [86] Barrington et al. present the results of a study conducted over 185 subjects). Moreover, another drawback of this category of evaluation approach is that the results are difficult to reproduce, for the experiment is based on specific software application.

- *Log analysis.* Users' logs about listening and interaction are analyzed. Such logs contain information about how often each user listened to an automatically generated playlist. Furthermore, if the playback system implements a *"like/dislike"* feature, additional data regarding the enjoyability of the playlist for the users is provided. The main advantage of this technique is that it doesn't require direct participation of subjects to the evaluation. However, this approach requires a platform sharing information regarding listening behavior: such platforms are usually closed and thus they don't share data.

- *Objective measures.* With the term objective measures, Bonnin and Jannach refer to *measures that can be automatically computed for a given playlist and that try to approximate a user-perceived subjective quality* [17] [87]. Typical examples of subjective qualities are diversity or homogeneity of a playlist. Many ways for assessing the homogeneity of a playlist have been presented, such as the number of different artists or genres appearing in the playlist. Anyways, a single quality criterion might not be sufficient to assess the quality of a playlist [17]: determining, for example, the homogeneity alone might not lead to a good evaluation, especially since some researches have shown that determining diversity is at least as important as homogeneity [88] [89]. Other measures that have been explored are freshness, novelty of track, their familiarity with respect to a certain user, the consistency of the mood, and the smoothness of transitions [17].

- *Comparison with handcrafted playlists.* Estimates the ability of the algorithm to generate playlists similar to the ones generated by music enthusiasts. The advantage of this technique is that for it, two well known protocols can be used for this kind of evaluation: hit rates and average log-likelihood.

  1. Hit rates are frequently used in Information retrieval, and give the ratio between the number of times the system retrieves a "good" result over the number of overall tries. For applying this measure, the idea is to take a handcrafted playlist and "hide" some of the tracks. The playlist generation system is left to guess the hidden items. A limitation of this strategy is that is based on the assumption that the hidden items are the most relevant ones of the collection, for they are the one on which the system is tested. Clearly, other elements are just as important.

  2. Average log-likelihood (*All*) can be used to assess how likely a system is to generate the tracks of a given set of deemed positve playlists. This measure is computed as:

$$All(Train, Test) = \frac{1}{\|Test\|} \sum_{(h,t) \in Test} \log(P_{Train}(t|h)) \tag{3.1}$$

     where $P_{Train}(t|h)$ is the probability of observing t given h according to a model learned on $Train$. The possible values range from $-\infty$ to 0, therefore it only allows to compare results between each others without knowing if the best one is actually good. It can be seen as a complementary measure to hit rate.

Part II

Methodology

Chapter 4

Requirements and approach

4.1 Catalogue of music

The catalogue of provided music features 584 songs, for a total length of 91 hours, 43 minutes and 35 seconds. The average length of each song is 9 minutes and 25 seconds circa. This catalogue has been provided with metadata indicating only artist, year of release and title of each song. Furthermore, all of these works can be labelled as belonging to the electro-acoustic genre, which usually indicates very abstract and arrhythmic, for which it is difficult to provide semantic descriptors or tags. Given this feature of the music and the length of the entire catalogue, the possibility of manually annotating it with proper metadata has been soon disregarded. This collection of music has therefore represented a great chance for developing a system based on the latest findings on audio content analysis.

The catalogue features songs recorded over 40 years and coming from different sources (mainly open-reel tape, cassette, DAT and vinyl). These recordings were provided transcoded by us to CD-quality format and then transcoded into mp3 format at 192kbps.

4.2 Requirements

Despite its intended use as part of the exhibition *"Phonos, 40 anys de música electrònica a Barcelona"*, the software developed should feature good flexibility to different catalogues of music, in order to be exploited as a part of the research for the GiantSteps project. This has represented a strong requirement during the development, and has induced the adoption of several descriptors that may not be particularly meaningful for the Phonos catalogue of songs, but that extend the range of possible music catalogues in which the system performance could be satisfactory. Furthermore, as a

part of a research project, the system developed should be easily extendable in other research activities, hence a modular, coherent and well-document code is preferred.
The software is intended to be used at the exhibition through an interactive kiosk: it will be available to users as a link inside a more general *webpage* containing several information regarding Phonos history and music creation technologies and devices. In addition, it must fully support touch devices, provided that this will be the only way users will be able to interact with the application, specifically with some sliders that allow them to control the flow of music in regards to the year of release of recordings or some relevant and perceivable audio features.
All of these requirements have lead to the choice of developing a *web-application*. Anyways, the interactive kiosk to be used at the exhibition was not available during the development; furthermore, its technical specification was unknown. For these reasons, it was therefore decided to develop a *two-layers system* made of the interactive kiosk machine connected (by an Ethernet cable) to a server machine. The latter one is in charge of providing and executing all the complex functions required during the functioning of the system. This solution has been chosen also because of its flexibility: it could easily be extended into a client-server architecture for web-based remote access, making the system more accessible for further use or evaluation outside the exhibition.
An additional strong requirement for the system is to be able to react to the real-time interaction of users with the user interface. Computation times must hence be as low as possible, in order to avoid a notable and inconvenient delay between the user interaction and the effective perception of changes in the flow of audio.
Finally, given the substantial average length of the songs, the system should segment the songs into very short excerpts (from 2 seconds to around 30; the choice of this length should be available to users in a real-time fashion), in order to allow users to listen to as many works as possible during the visit at the exhibition and to more easily find tracks that fulfill their taste or personal requirements. It must then be found a way to properly segment the audio pieces and computing descriptors for each slice obtained. In order to achieve a better sense of *"flow of music"*, the computation of similarities should therefore be carried out between these short excerpts, instead of exploiting descriptors for the entire songs.

4.3 Design of the system

The requirements cited above have lead to the following choices for the design of the system. First, the computation of audio descriptors can be performed offline, because the catalogue of music on which the system will run is not subject to changes. It is therefore safe to compute descriptors prior to their public use. The performance of the system

will greatly benefit from this choice, given that the computation of audio descriptors for each excerpt of every song of the catalogue is the most computationally intensive step to be performed. The audio descriptors will be stored on the server machine.
Second, for the system to have low response times to user inputs, the computation of music similarity is being carried out on the server (also because the performance of the interactive kiosk machine are unknown, as already cited in 4.2), with proper music similarity algorithms. The flow of music is not supposed to require any human interaction, to the meaning that it will automatically generate a flow of music based on the computation of audio similarity also without an interaction of the user. Actually, the system always concatenates segments in a way that only very similar segments are consecutive elements of the playlist. The interaction of the user will eventually give a direction to this flow, according to the user's will and taste.
Third, the application running on the server machine will be in charge of collecting the user interaction with the web-application running on the interactive kiosk machine, and that will come in the form of *HTTP POST requests*. At each user interaction, the application running on the server machine deletes the current and already computed playlist and performs an audio similarity computation between the currently playing excerpt and a set of excerpts that fulfill all the requirements about music, that the user has imposed through the graphic user interface. One of the most similar excerpts is taken from the list and a new content-aware playlist starts being built above that.

4.4 Evaluation

For our main concerns regard the musicality of the output and its flow, we wanted to collect data about user listening experience while interacting with the app. We therefore decided to evaluate the performance the system with surveys compiled after a short (5 minutes) interaction with the software. Since the enjoyment of the musical output highly depends on the familiarity with this typology of music, we will attest the participant's familiarity with a specific question. The flow of the music depends also on the ability of the system to show short response times to user interaction, so that the user is not frustrated by the slow responsiveness; some questions of the survey will then try to establish the enjoyment in the use of the software.
We have therefore decided to collect the following data for each participant:

- Ease of use
- Understanding of GUI controllers' meaning
- Enjoyability of the musical output

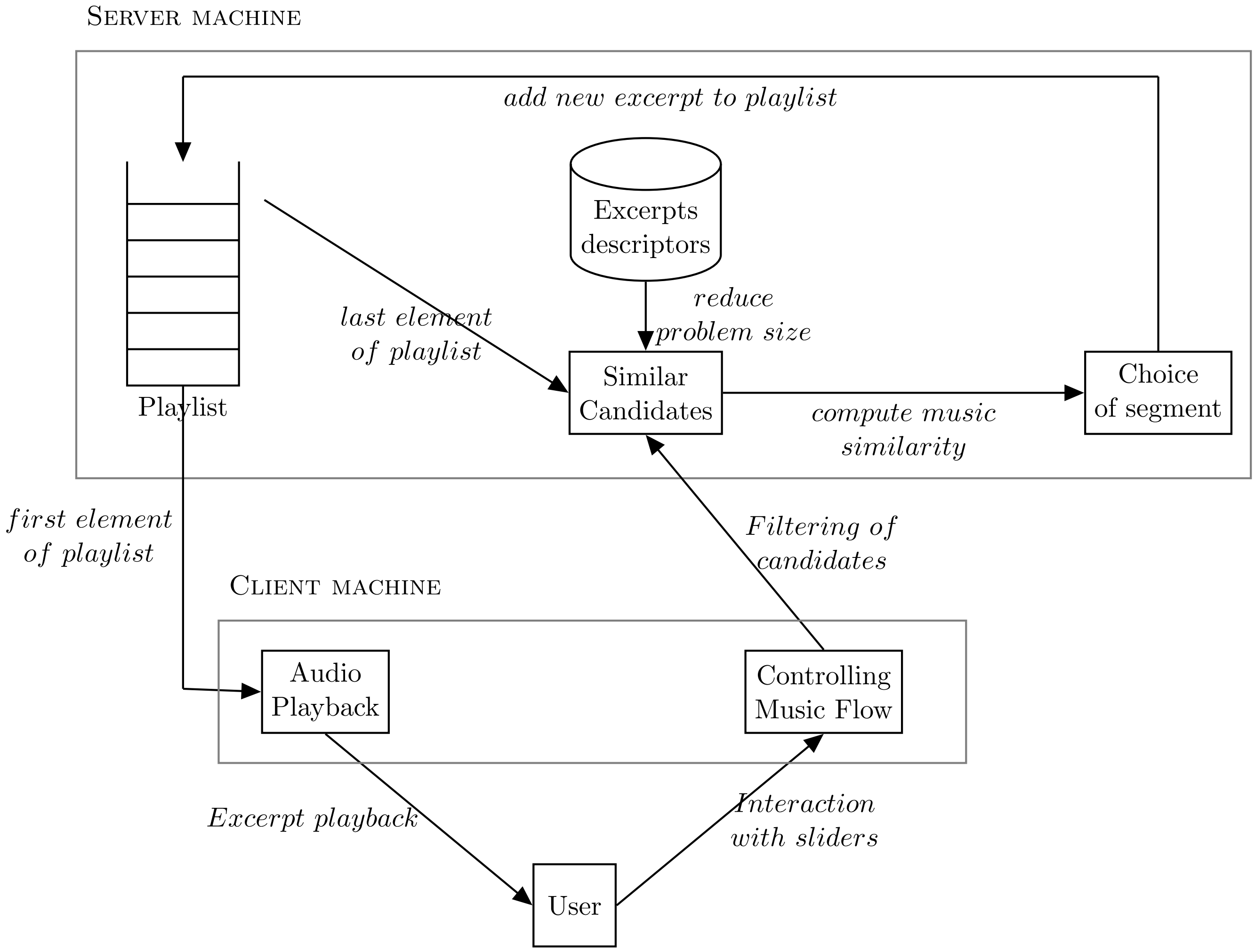

Figure 4.1: The implementation of the system.

- Encountered problems
- Familiarity with this kind of software
- Familiarity with the music styles the collection included

The participant is then asked if he thinks that the software provides a more enjoyable way of listening to music (compared to a full-track player) and if he would use it for exploring a catalogue of music.
Results for the surveys will be shown and discussed on Chapter 7, and the whole questionnaire is presented in Appendix D.

Chapter 5

Off-line computation of audio features

In order to achieve good performance, two very computationally intensive tasks of the system are performed off-line, and their output is then going to be used by the real-time application. These tasks consist of the computation of the audio content descriptors and of the building of a *fast-map*, a high dimensionality space in which each point correspond to an audio musical excerpt. This space is built in a fashion that guarantees that nearby points of this space correspond to very similar excerpts.

5.1 Audio content features extraction

Solving this problem has involved two very important choices: what audio content descriptors to use and what library or tool to use for computing them. Many factors have been taken into account for solving both of these problems.

- Among the features of the tools, flexibility has constituted the strictest requirement: an easy way to compute descriptors for each excerpt of every track is required, while many tools provide only ways of computing descriptors for the entire file. In this latter case, the file should manually split into *subfiles* (one for each segment), therefore implying a huge waste of memory. This has soon lead to the exclusion of *jMir*, for it doesn't fulfill this requirement.
- The tool should easily be callable by source code or bash scripts, and results of the analysis must be stored in output files.

- The computation of descriptors should be as fast as possible, given that the excerpts to be analyzed are in the order of tens of thousands.

- Last but not least, the tool must provide descriptors whose usefulness for this specific case study has been empirically verified during the development of the system.

All of these requirements lead to the choice of performing the audio analysis with Essentia and Echo Nest: the first for its speed, flexibility and reliability. Echo Nest has been used for some of its descriptors are not present or not as accurate in Essentia, and have shown a great usefulness during the development or granted by existing previous research.
Furthermore, both of the two libraries are offered in Python, allowing the entire analysis task to be written in a single programming language, therefore improving the code consistency and readability.
The schema for the extraction of the audio features is illustrated in figure 5.2.
At first, the user is required to give the path of the folder in which the audio files are

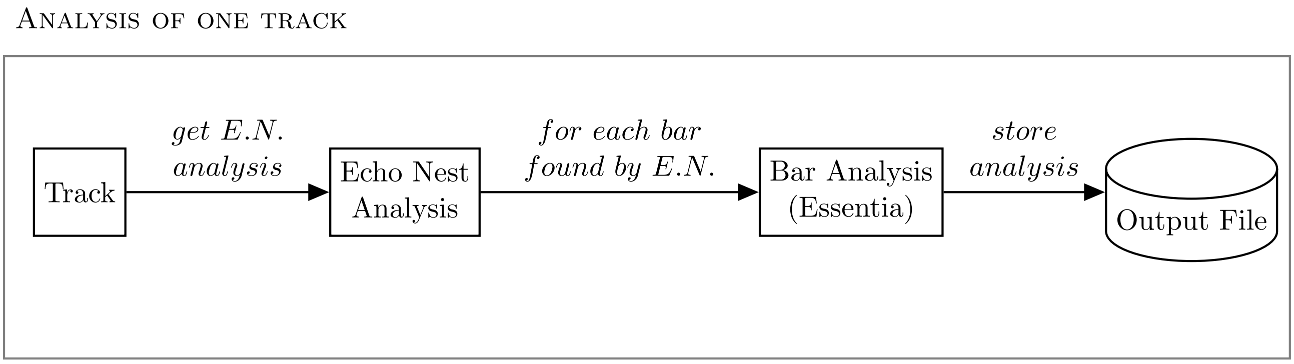

Figure 5.1: Schema for the extraction of audio features.

stored. The collection is entirely stored as .mp3 files with a sample rate of 44100Hz and a bitrate of 192kbps. The application then collects the path to all the .mp3 files in this folder, and mark them as to be analyzed if no previous analysis has be performed. An analysis of these files with Echo Nest (through Pyechonest) is performed, and we specifically use the following fields of the output of this analysis: *bars*, *BPM*, *loudness*, *HPCP* and *acousticness*. *Bars* give the starting and ending point of each bar detected and, although not particularly meaningful for the arrhythmic Phonos catalogue of music, have shown to perform well on the additional and more generic personal catalogue used during first stages of development; therefore, it was decided to use them in order to improve the flexibility of the system.
Segmentation of songs into excerpts is then performed, based on starting and ending

point of each bar. Then, we compute more specific descriptors with Essentia for these excerpts, with the following strategy:

- each excerpt is divided into frames, with a size of 2048 samples and a hop of 1024 samples. For each of these frames:
  - we apply an Hann windowing function
  - we apply the FFT algorithm provided by Essentia in order to get a spectral representation of the signal
  - we look for peaks in the spectrum, collecting their frequencies and magnitudes, and then we use them to compute the dissonance in the frame, with Essentia's algorithm `Dissonance`
  - an HFC onset function is computed on the spectrum, that will be used afterwards to compute the onset times
  - the MFCC bands and coefficients are computed with Essentia's algorithm `Mfcc`[1]
  - the energy in 27 Bark bands of the spectrum is computed

LOW LEVEL FEATURES EXTRACTION

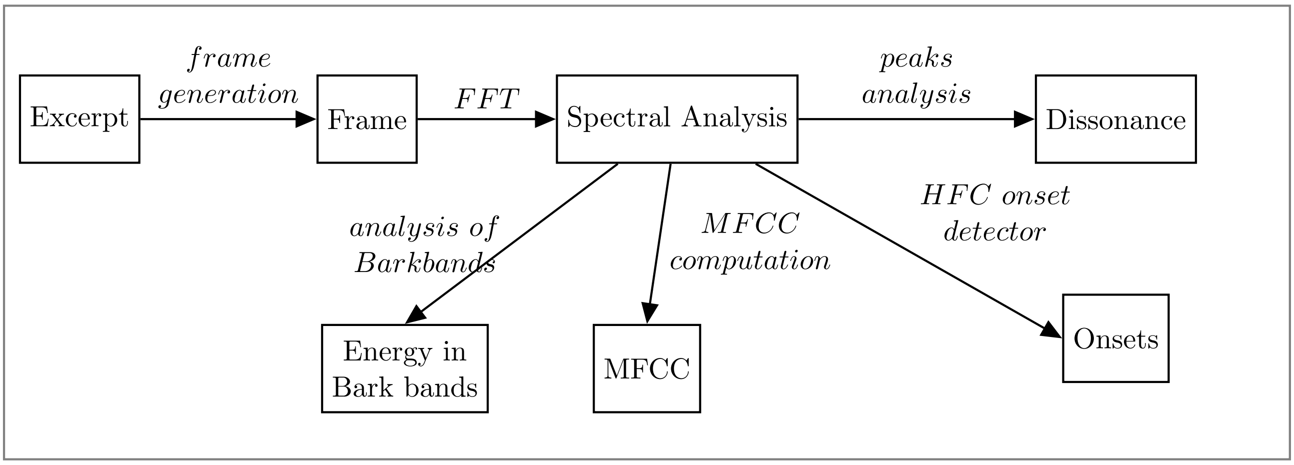

FIGURE 5.2: Schema for the extraction of low level audio features from excerpts.

- onset times in the excerpt are calculated, according to the onset function computed in each frame, and then onset rate is calculated with the formula:

$$OR_{excerpt} = \frac{Onsets_{excerpt}}{Length_{excerpt}} \tag{5.1}$$

[1]Essentia uses the MFCC-FB40 implementation, which decomposes the signal into 40 bands from 0 to 11000Hz, takes the log value of the spectrum energy in each mel band and finally applies a Discrete Cosine Trasform of the 40 bands down to 13 mel coefficients.

- dissonance in the excerpt is computed as a mean of the dissonance in each of its frames
- a single Gaussian model for the collected MFCC values is computed. Specifically, we collect its mean, covariance and inverse covariance. Mean is a 13 size vector, while covariance and inverse covariance are 13x13 matrices. The inverse covariance is stored in order to prevent having to compute it in the real-time application or during the fast map computation, therefore increasing the performance of both these stages. If a problem of ill-conditioned covariance matrices is encountered (i.e., a not positive semi-definite covariance matrix has been computed), only values of the diagonal of these problematic covariance matrices are used. This has allowed to avoid the presence of outliers when computing similarity, while still taking into account excerpts for which a covariance matrix of the MFCC values could not be correctly computed.
- based on the HPCP values computed by Echo Nest, we use Essentia's `Key Detector` to associate a key to each first and fourth beat of the bar. The reason why we keep values for these two particular beats is that this allows us to perform a more precise tonal comparison when trying to merge two excerpts in the real-time application: if the key of the first beat of the inspected excerpt is very different from the key of the fourth beat of the excerpt for which we're looking for similar pieces, the candidate is discarded.

This procedure is repeated for each excerpt, in order to get a deep description for all of them and perform more precise similarity computation in the real-time application. In addition, we store some additional level-song descriptors, specifically artist, title and year of release, and acousticness (computed with Echo Nest). Finally, for each song we create a corresponding JSON file in which we store all the descriptors computed.
The list of descriptors computed during this task is summarized in table 5.1.

Features	**Source**	**Level**	**Motivation**
Title, Artist, Year	Provided	Song-Level	Display more information about the current playing track in the GUI
Acousticness	Echo Nest	Song-Level	Give the user the chance to filter music in regards to its nature (acoustic or electronic music)
MFCC	Essentia	Bar-Level	Timbre similarity computation
BPM	Echo Nest	Bar-Level	Avoid consecutive excerpts with very different BPM

Onset Rate	Essentia	Bar-Level	Give the user the chance to filter music in regards to the presence of percussive elements
Dissonance	Essentia	Bar-Level	Give the user the chance to filter music in regards to the dissonance[2] of excerpts
Loudness	Echo Nest	Bar-Level	Give the user the chance to filter music in regards to its loudness
Bark Bands	Essentia	Bar-Level	Give the user the chance to filter music in regards to its "sparseness", i.e. the amount of mel bands with significant energy level
HPCP	Echo Nest	Beat-Level	Use them to compute key
Key	Essentia	Beat-Level	Use them to discard the possibility of having two consecutive dissonant excerpts in the playlist

Table 5.1: Descriptors computed by the offline application.

5.2 FastMap computation

The procedure just described for computing descriptors give us a 410 size vector for each excerpt, and a total number of 159239 excerpts.
In order to achieve good real-time performance when comparing these excerpts, a dimensionality reduction of these vectors is required. Furthermore, as seen in 2.3, the computation of Kullback-Leibler divergence, although showing very good results in capturing the timbre similarity, is a very intensive computational operation and therefore a simpler distance measure with comparable results is preferred.
These requirements were also faced by Schnitzer et al. in [53], who presented a filter-and-refine method to speed up nearest neighbor searches with the Kullback-Leibler divergence for multivariate Gaussians, yielding high recall values of 95-99% compared to a standard linear search. The original FastMap was proposed in 1995 by Faloutsos and Lin [59] for indexing and data-mining multimedia datasets. It was used for the first time for computationally heavy, non-metric measures and nearest neighbor retrieval in [57], for speeding up classification of handwritten digits. FastMap was used for the first time in MIR by Cano et al. in [58] in the attempt of reducing high dimensional music timbre similarity space into a 2-dimensional space. This was done not for speeding up

[2]During development, it has been empirically noticed that dissonance has a significant correlation to the perception of noise: the more an excerpt is perceived as noisy, the more it is dissonant.

classification, but rather for visualization purposes.
The idea behind the use of a FastMap for classification or computing similarities is to compute with the original distance measure $D()$ (computationally intensive) just a subset of all the distances, specifically the distances between each point and a subset of $2k$ points (the *"pivots"*); then, on the basis of these computed distances, each feature vector is mapped with a non-linear trasformation into a point of a k-dimension space, where a simpler distance measure can be applied, with a small decrement in accuracy.

For choosing the $2k$ pivot elements, the original FastMap [59] follows this strategy:

- k element $x_1^1, x_2^1, ..., x_k^1$ are randomly selected from the collection of feature vectors
- for each x_i^1, its corresponding most distant object x_i^2 according the original distance measure $D()$ is picked

Each vector of features x is then mapped into the point $(F_1(x), ..., F_k(x))$ of the new k-dimensional space, where $F_j(x)$ is computed with the formula:

$$F_j(x) = \frac{D(x, x_j^1)^2 + D(x_j^1, x_j^2)^2 - D(x, x_j^2)^2}{2D(x_j^1, x_j^2)} \tag{5.2}$$

In other words, the coordinate in the $j - th$ dimension of each point is determined by the pair (x_j^1, x_j^2), specifically by the original distance (computed with $D()$) of the point from both these pivots and the distance between the pivots themselves.
For our work, we have decided to use the Kullback-Leibler as the original distance function, computed for the multivariate normal distributions x_1 and x_2 with the Eq. 2.18, that for convenience we report again here:

$$KL(x_1, x_2) = \frac{1}{2}\left(tr(\Sigma_{x_2}^{-1}\Sigma_{x_1}) + (\mu_{x_2} - \mu_{x_1})^T\Sigma_{x_2}^{-1}(\mu_{x_2} - \mu_{x_1}) - k + ln\left(\frac{det\Sigma_{x_2}}{det\Sigma_{x_1}}\right)\right) \tag{5.3}$$

As it has been widely used (achieving good results in [14], [25], and [53]), we can be very confident on using it here too. Anyways, we must take into account several aspects.
As already seen in 2.3, the Kullback-Leibler cannot be intended as a pure distance measure, for it fails to be symmetric and to fulfill the triangle inequality. It can simply be made symmetric by considering the distance SKL (symmetric Kullback-Leibler) defined as:

$$SKL(x_1, x_2) = \frac{1}{2}KL(x_1, x_2) + \frac{1}{2}KL(x_2, x_1) \tag{5.4}$$

Regarding the triangle inequality, a proper solution is not that trivial. However, in [53] Schnitzer et al. have shown that rescaling the symmetric Kullback-Leibler divergence with the square root leads the new distance function to fulfill the triangle inequality in more than 99% of the cases. Therefore our original distance function $D()$ that we use in Eq. 5.2 is:

$$D(x_1, x_2) = \sqrt{SKL(x_1, x_2)} = \sqrt{\frac{1}{2}KL(x_1, x_2) + \frac{1}{2}KL(x_2, x_1)} \tag{5.5}$$

This procedure can be further improved by a small modification in the strategy for choosing pivots: once the pivot x_i^1 is randomly picked, we choose to pick the object lying at the distance media as x_i^2, i.e. the object at the index j=$\lfloor \frac{N}{2} \rfloor$ once all the distances from point x_i^1 are sorted. We have decided to use $k = 20$ (therefore having 20 pairs of pivots and a final 20-dimensional space) as this has allowed us to find a good balance between computational times and quality of the output the similarity computation.
The accuracy and performance of this procedure are well-documented in [53]. This technique constitutes the basis on which our system will perform the real-time similarity computation, with some additional tweak that will see in the Chapter 6.
The computed data is stored on a JSON file: for each point (corresponding to an excerpt), we store its coordinates in the new 20-dimensional space plus some additional descriptors that allow us to do a faster filtering in the real-time application, as we won't need to lookup to the original JSON descriptor file for each song just for retrieving the values of these descriptors. The list of features stored in the map for each point is shown in table 5.2.
During this stage, we additionaly save lists that associate each segment to the decade the song it has been extracted from has been produced. This will allow very fast filtering techniques on the real-time application when the user interacts with the sliders for selecting music according to the year of release.
The computational times of this stage are shown in table 5.4 and the configuration of the computer used in table 5.3.

Features	**Motivation**
Year, Artist, Title	Speed up access to information
Starting and ending point inside the track	Allows fast extraction of the excerpt from the entire audio signal
BPM, Key	Be faster when filtering out music with very different BPM or key
Acousticness, Loudness, Dissonance, Bark Bands, Onset Rate	Perform a fast filtering of database of excerpt when the user interacts with the GUI for controlling the musical output

Table 5.2: Features stored in the map for each point.

Laptop Model	Packard Bell EasyNote TS-11HR
CPU	Intel®Core^{TM}i5-2410M @ 2.50GHz
RAM	4GB DDR3 @ 1066MHz
Hard Disk Drive	5400rpm
OS	Linux Mint 17.1 "Rebecca" (64 bit)

Table 5.3: Hardware configuration of computer used during off-line descriptors computation.

Stage	Time required	Stats
Descriptors computation	04h 32m 25s	Minimum time for track: 00m 15s
		Maximum time for track: 00m 52s
		Average time for track: 00m 28s
FastMap computation	00h 47m 12s	Choosing pivots: 16m 43s
		Computing points coords: 30m 29s
Total	05h 19m 37s	

Table 5.4: Computational times for descriptors computation of a collection of 584 tracks, with a total length of 91 hours, 43 minutes and 35 seconds (the time for uploading these tracks to Echo Nest is not considered in these results).

The features collected and the FastMap computed over this stage will constitute the basis on which the real time computation of music similarity will be performed; this particular core of the system will be shown and discussed in next section.

Chapter 6

Real-time application development

The real-time application is based on a two-tier architecture, organized as follows:

- the server machine runs a Python Flask application, and it is responsible for generating playlists and audio
- the client displays an HTML web-page that collects user interactions and sends them to the server machine for realtime editing of playlists. Additionally, it receives audio streaming from the server.

Therefore, the realtime computation of music similarity happens on the server machine.

6.1 The server application

As already stated above, the server application is in charge of offering several features: it generates the playlist, sending audio and additional information to the client (such as *artist* and *title* of current playing piece, so that the client can display them for the user on the GUI). Additionally it has to generate audio, that will be streamed to the client in order for the user to listen to it through its own device. For generating the playlist, a realtime music similarity algorithm with very good performance must run on the server.
Many Python web frameworks are available; the most used ones are Django[1], Flask[2] and

[1]`https://www.djangoproject.com/`
[2]`http://flask.pocoo.org/`

Figure 6.1: Flask logo.

Pyramid[3]. This realtime server application has been based upon Flask framework, that is a lightweight web application framework written in Python and based on the WSGI toolkit[4] and Jinja2 template engine[5]. It is provided with a BSD license and, contrarily to Django and Pyramid, is aimed at small applications with simple requirements. Its first version was released in 2010 and it comes with a great usability, where a simple *"Hello World"* web-app can be written with only 7 lines of source code[6]. Web application framework are usually thought to be separated into several conceptual units called "apps", each one providing different functionalities to the system. Flask is intended to make really simple the development of a single app; many others may be added, but in this latter case Django and Pyramid may provide a better ease of use.

All of these factors have lead to the choice of this framework for our system: the web platform to develop is actually intended to be quite simple, displaying just the main GUI and a few more details and options for the user. Given that the application is meant to be offered just to one client at time, we decided to use the builtin server of Flask also on production; indeed, we considered a full deployment option (such as Apache or CGI) to be a waste of resources for this simple use case. The server application executes two parallel tasks: the generation of the playlist, based on realtime computation of music similarity, and the generation and streaming of this playlist to the client of the audio. It furthermore provides several methods that are handled by Flask routing techniques and invoked at specific interaction of the user with the client application; these methods have deep impact on the generation of the playlist and allow the user real-time control over this process.

[3] `http://www.pylonsproject.org/`

[4] A specification for universal communication between web servers and web applications or frameworks for Python programming language. Published on December 2003 by its author Phillip J. Eby, it has become a standard for Python web application development.

[5] `http://jinja.pocoo.org/docs/dev/`

[6] `http://flask.pocoo.org/docs/0.10/quickstart/#a-minimal-application`

6.1.1 Realtime computation of music similarity and playlist generation

As we mentioned, this computation is performed on the server machine, for the hardware configuration of the interactive kiosk has been unknown until the beginning of the exhibition, and might have not been able to achieve good performance with the software developed. The hardware configuration of the server machine is shown in table 6.1.

CPU	Intel®Core™2 Quad Processor Q6600 @ 2.40GHz
RAM	2GB DDR2 @ 800MHz
Hard Disk Drive	5400rpm
OS	Linux Mint 17.1 “Rebecca” (32bit)

Table 6.1: Hardware configuration of the server machine.

The task for generating the playlist follows a well-defined schema: at first, the FastMap computed as described in section 5.2 is loaded into memory; this process usually takes just few seconds. A random point of this map is pick, and will be used as the first excerpt of the playlist. This excerpt in then put inside the playlist, a Python dictionary whose keys are the position of the elements inside the playlist and the corresponding values are tuples containing several important aspects for the playback; the details of these tuples are shown in table 6.2.

URI of file	Song title	Song artist	Song Year	Starting time	Ending Time

Table 6.2: Information stored for each element of the playlist.

Once the first segment is picked, the application enters in a loop in which each iteration ends in adding a new excerpt to the playlist. The comparison of music similarity is always performed between all the candidate elements of the FastMap and the last element of the playlist. The procedure invoked in this loop can be summarized as follows:

1. If any user interaction with sliders or knobs has happened since the last iteration, delete the content of playlist. This allows users to immediately hear musical differences in the playlist as soon as they interact with the client application.

2. Delete already played elements from the playlist in order to avoid memory leaks

3. If we already have enough elements in the playlist, let the task *"sleep"* for one second and then go back to step one. This prevents the cpu from always working at full load, a behaviour that could cause serious overheating problems in a server machine running this application for several consecutive hours at the museum.

4. At this point, we get into the procedure for actually choosing the next excerpt to be inserted into the playlist. At first, a weighted queue according to the sliders for filtering by decades is created.

5. The entire map of excerpts is now filtered according to the current positions of sliders in the client application. If there is no excerpt fulfilling all the constraints imposed by the sliders, we only take the segments whose descriptors values fulfill less strict thresholds based on actual sliders values. If instead the amount of excerpts available after this filtering is over 500, a Monte Carlo sampling of them is performed, to bring the total number of candidates to 500. We experienced unsatisfying performance of the application during successive steps of the procedure (also due to a not particularly powerful configuration hardware of the server machine) with less aggressive sampling, and we noticed that with 500 candidate excerpts good results were still achieved. This value may be increased in more powerful devices.

6. Additional filtering is performed, based on the values of BPM and loudness of the candidates. Candidates who greatly differ on these values from the last element of the playlist are discarded. For judging similarity in terms of BPM, the Eq. 2.20 (with $\alpha_{BPM} = 1$) has been used, with a maximum distance of 3 allowed. The maximum discrepancy allowed in loudness is of 5dB. If no candidate excerpt fulfill this stage, the list of candidates before this filtering is restored.

7. At this point we finally choose the number of candidates in which we'll perform deeper analysis. This number, that we call $N_{Neighbors}$, is computed according to the following formula:

$$N_{Neighbors} = filter\_size * |FastMap| \tag{6.1}$$

   where $|FastMap|$ is the number of excerpts in the FastMap (i.e., the total number of excerpts in the catalogue), and $filter\_size$ is a value in $[0, 1]$. We empirically noted that a value of 0.1 for $filter\_size$ already gives good results, while allowing to achieve highly satisfactory computational times. We then select the $N_{Neighbors}$ nearest neighbor to the current element through an Euclidean distance on the 20-dimensional space.

8. We now compute the symmetric Kullback-Leibler distance between the last element of the playlist and all its neighbors. We do this specifically only if:

   - We have a margin of at least 5 seconds of playback in the current playlist after the current playing excerpt
   - The user has not interacted with the controllers of the client-application since the last iteration of the loop

   If any of this two conditions is not met, we don't compute the symmetric Kullback-Leibler distances but we rather choose the next element of the playlist on the basis of the euclidean distance on the 20-dimensional space. We do this because this stage could require several seconds (around 4 to 9 seconds on the server machine[7]) and the conditions for performing such a slow computation could not be met, resulting in a perception of a high-latency system. The second condition is used because, even if the playlist is emptied as soon as the user interacts with the controllers (but there still may be more than 5 seconds to play, if the current excerpt is very long), it doesn't make sense for us to perform computational intensive task for computing similarity when the user's will is actually to change the flow of the music by interacting with the controllers.
   Once all the distances are computed, we keep only the segments whose SKL distance from last element in the playlist is less than 20, a threshold that we empirically noticed to be quite selective in the quality of the output despite not being extremely selective in the amount of results. An excerpt from this list is finally randomly picked and put in the playlist. If the list is empty (or the computation of symmetric Kullback-Leibler couldn't be performed), the next excerpt of the playlist is randomly picked among the 10 nearest neighbors by the mean of the Euclidean distance.

The procedure described allows to choose the next element of the playlist with satisfying performance (see Chapter 7), although this may greatly vary with the condition; specifically, computational times become much longer when all the symmetric Kullback-Leibler distances are computed, but this generally leads to better musical results.
It may be useful to mention two further features of the application:

[7]This considerable amount of time is due not only to the complexity of the formula for computing the symmetric Kullback-Leibler distance, but also to the necessary access to JSON files, where the needed MFCC values are kept. We could not store them on primary memory, as the low amount of RAM in the server machine (2 GB) might have been problematic.

- When the user interacts with the slider for changing the length of the excerpt to be played, the procedure for computing similarity doesn't change. Longer segments are obtained by playing consecutive excerpts of the same song, and the procedure for computing similarity will look for similar excerpts to the last one in this queue of consecutive excerpts of the same song.
- The software provides options for managing the playlist generation in regards to repetition of songs or excerpts: specifically, the user can force the application of never picking two excerpts belonging to the same song unless a specific amount of different excerpts in the playlist has already put between them. We noticed that disabling this feature may greatly improve the quality of the musical flow (some loops between excerpts of the same song may be generated, creating a strong cohesion of the musical output; this behaviour is the same one proposed by *The Infinite Jukebox*[8]) but may annoy some users if they want to broadly explore the collection of music and would possibly like to avoid repetitions.

6.1.2 Audio generation and streaming

Everything we have seen so far allows to dinamically generate a content-aware playlist of excerpts. To allow the user to actually listen to this playlist we need to read the corresponding slices of the audio files and implement a streaming over the network of this audio content.

Feature	**Motivation**
Seek by millisecond	Perform very accurate extraction of excerpts from audio tracks, in order to perform beat synchronized track mixing
Audio Crossfade	Improve the audio "flow", making the transition between consecutive excerpts less abrupt
Programmable	Facilitate communication with the code for computing music similarity. Python preferred.
Streaming	Streaming over the network is required for the user to listen to the playlist.

TABLE 6.3: Requirements of the audio player.

[8] `infinitejuke.com/`

This is not a trivial task, for not many audio players on Linux provide the needed flexibility by the application. Specifically, it has been found no audio player on this platform that simultaneously provides all the needs reported on Table 6.3. Therefore, we decided to build our custom audio player, exploiting the very popular multimedia framework *GStreamer*.

GStreamer

GStreamer[9] is a free and open-source multimedia framework written in the C programming language, subject to the GNU Lesser General Public License (LGPL). It allows developers to modularly build multimedia applications with the use of *pipelines*, where lower-level units are connected; each unit has a specific purpose. It fully supports Linux, Android, iOS, Mac OS X and Windows, and offers bindings in several programming languages, Python included. The list of popular applications built upon this framework includes *Amarok*[10], *Banshee*[11], *Flumotion*[12], *Pitivi*[13], *QuodLibet*[14] and *RhythmBox*[15].
The main advantage in the use of this framework lies in its modularity: it offers many units (also called *plugins*) with media-handling features, including audio and video playback, recording, streaming and editing. The pipeline design serves as a base to create many different types of multimedia applications, for instance media players, video editors, and streaming media broadcasters.
It fulfills all the requirements of Table 6.3 and therefore we decided to use it for developing our custom audio player.

FIGURE 6.2: GStreamer logo.

[9] `http://gstreamer.freedesktop.org/`
[10] `https://amarok.kde.org/`
[11] `http://banshee.fm/`
[12] `http://www.fluendo.com/`
[13] `http://www.pitivi.org/`
[14] `https://code.google.com/p/quodlibet/`
[15] `https://wiki.gnome.org/Apps/Rhythmbox`

Audio player developed

Given that we want to smooth the transition between two consecutive excerpts, the use of a crossfade is preferred. This implies that two different audio players should be playing simultaneously when the crossfade is being performed. We solved this by creating a simple audio player (the custom bin shown in Figure 6.3) for each track that is then connected in a global pipeline (Figure 6.4) responsible for the audio synchronization of different custom bins and of the streaming over the network of the audio content.
The units used in the custom bin are explained in Table 6.4, while the ones used in the global pipeline are explained in Table 6.5.

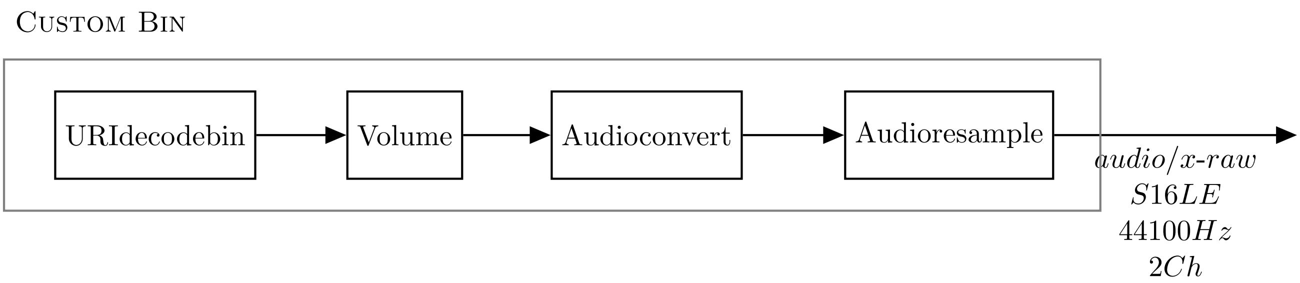

Figure 6.3: Custom audio bin, that corresponds to an audio player only responsible for the playback of a single excerpt.

Unit	**Input**[16]	**Output**[16]	**Motivation**
URIdecodebin	*mp*3 file	*audio/x-raw* $S32LE$, $2Ch$ $44100Hz$	Loads the raw audio content of a file by its location (URI)
Volume	*audio/x-raw* $S32LE$, $2Ch$ $44100Hz$	*audio/x-raw* $S32LE$, $2Ch$ $44100Hz$	Used in crossfades, allows fade in and fade out on single audio tracks
Audioconvert	*audio/x-raw* $S32LE$, $2Ch$ $44100Hz$	*audio/x-raw* $S16LE$, $2Ch$ $44100Hz$	Negotiates a raw audio format according to formats supported by its end and the format of the input
Audioresample	*audio/x-raw* $S16LE$, $2Ch$ $44100Hz$	*audio/x-raw* $S16LE$, $2Ch$ $44100Hz$	Needed by the adder to ensure that its input files will always be of the same type

Table 6.4: Elements of the custom bin.

[16]Values shown here are related to particular files of the Phonos catalogue of music used by the system, and they have been inserted just as examples. Their values may vary with different types of files.

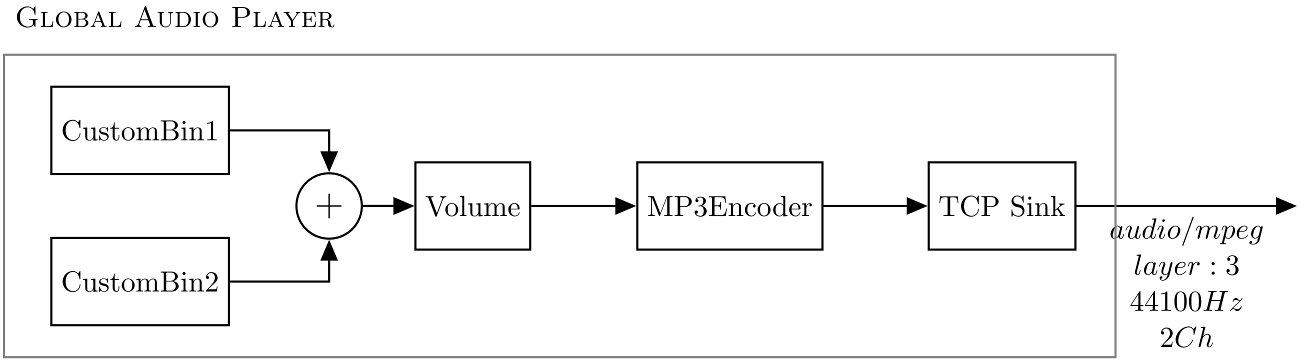

Figure 6.4: Schema for the audio player implemented.

Unit	Input[16]	Output[16]	Motivation
Adder	*audio/x-raw* $S16LE$, $2Ch$ $44100Hz$	*audio/x-raw* $S16LE$, $2Ch$ $44100Hz$	Mixes together samples coming from multiple audio streams, producing a single audio stream
Volume	*audio/x-raw* $S16LE$, $2Ch$ $44100Hz$	*audio/x-raw* $S16LE$, $2Ch$ $44100Hz$	Gives control over the global volume. This will be settable by the user on the client application GUI
MP3Encoder	*audio/x-raw* $S32LE$, $2Ch$ $44100Hz$	*audio/mpeg* $layer3$, $2Ch$ $44100Hz$	Converts the raw audio stream into an mpeg layer 3 stream
TCPSink	*audio/x-raw* $layer3$, $2Ch$ $44100Hz$		Provides streaming over the network of the mpeg audio content

Table 6.5: Elements of the pipeline.

The class responsible for handling the global audio player has access to the playlist generated by the algorithm explained in Section 6.1.1. It extracts the first element on this queue, creates a custom bin for it, performs the seeking[17] and plays it with an initial fade in, whose length is `CROSSFADE`[18]. `CROSSFADE` seconds before the end of the current excerpt, the algorithm extracts the next element on the playlist. If this is empty, we

[17]Seeking is actually performed on the URIdecodebin element.

[18]The default value is 0.8s, enough for creating a sense of music "flow". The user can edit this value through the client graphical user interface.

keep playing the current track until a new excerpt is inserted into the playlist. The algorithm then creates a new custom bin for this new excerpt, adds it to global pipeline, performs the seeking and starts the playback of this custom bin with a fade in. The seek sets the inpoint of the playback to the point `(start_point`[19] `- CROSSFADE)`, so to have a beat-level synchronization of music (see Figure 6.5): when the old excerpt reaches the end of its length (i.e. at the end of the crossfade, that also corresponds to the first beat of the next bar), the new one reaches the first beat of its corresponding bar[20]. These two beats are then played together. This aspect greatly improved the musicality of the output with the music collection used during development, while not being particularly relevant for the arrhythmic Phonos collection of music.
In order to prevent memory leaks, the old excerpt and its corresponding custom bin are both removed respectively from the playlist and from the global pipeline.

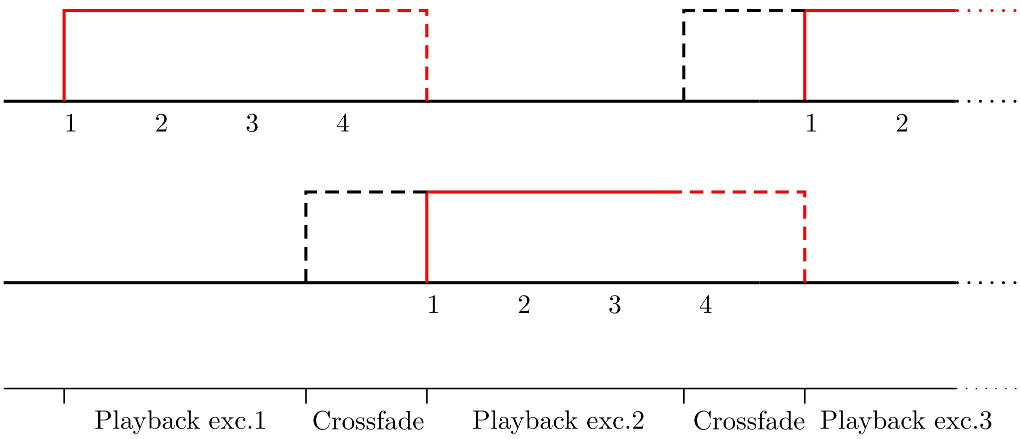

FIGURE 6.5: Handling of audio crossfades. The red rectangles indicated the content of the excerpt, and dashed lines indicate crossfades. Note that the playback involves more than just the excerpts' content: we use the portion of audio before it during the fade-in to achieve a beat-level synchronization. The indices indicate the number of the beats inside the excerpts.

The audio of the global pipeline is collected by the TCP sink, that is in charge of streaming this content over the TCP port 8070. This stream will be collected by the client application.

[19]By `start_point` we mean the starting point of the excerpt inside the track it belongs to.
[20]We recall that each excerpt corresponds to a bar.

6.2 The client application

The client application consists of a web-application hosted by the Flask application running on the server. To access it, the client needs to connect to the address `http://server_address:5000` on a browser. We entirely designed the graphical user interface of this application with the software *Adobe Photoshop CS6*[21], with the intention of providing a "metallic" looking (that could resemble of the analogue synthesizers used in early Phonos records) coupled with the presence of elements (sliders and knob controllers) whose purpose could be easily understood by users. This interface is shown in Figure 6.6.

FIGURE 6.6: Client application GUI.

This interface provides several ways for the user to control the music flow. Each time the user interacts with one of them, an HTTP post request is done from the client machine to the server, resulting in a change of the candidates for the playlist.
There are ten sliders: five of them are related to the year of release of the musical pieces, the other five are instead related to intrinsic characteristics of the music. In this way, the user has control both over the decade, and both over the type of music he wants to listen to. The motivation of this design choice is that we want to make the process of discovering music interactive while preserving ease of use. Furthermore, the suddivision of music into decades may be particularly useful in the use at the exhibition, since

[21] `http://www.adobe.com/products/photoshop.html`

visitors could be particularly interested in hearing the differences between the works belonging to just a particular era over the entire 40 years life of Phonos.
The five sliders for music features are:

- Loudness
- Noisiness: related to the dissonance of the signal
- Rhythm: higher values of the slider lead to excerpts with a high amount of onsets on high frequencies
- Density: higher values of the slider lead to excerpts where many Barkbands have a considerable amount of energy
- Acousticness: sets the ratio Acoustic/Electronic. Lower values of the slider mostly lead to purely electronic music.

The ranges of the internally managed sliders' values are dinamically generated during the computation of the FastMap: once the corresponding values for all the excerpts have been collected, these are sorted and we then pick the minimum, the maximum, and the first, second and third quartile for the values related to each descriptors. Therefore keeping the slider of the loudness at maximum will for instance lead to all the excerpts whose loudness value is between the third quartile and the maximum value of loudness of all excerpts. Step values for these five descriptors are calculated after the computation of the FastMap and kept in a separate JSON file.
The GUI additionally provides a set of presets for the values of these five sliders, a monitor for displaying information about the currently playing track, a slider for selecting the length of the audible segments (from 1 to 5 bars), and a knob for the volume (which controls the volume element of the global pipeline explained in Table 6.5).
By clicking on the button with a star on it, the user has the possibility of marking a track as favorite. The list of *"starred tracks"* is accessible on the second page of the GUI (shown in Figure 6.7), together with the list of the five last played track. The motivation behind this choice is to give the user the chance to keep track of the songs he has been finding interesting. At the exhibition, visitors may be particularly interesting in looking for more information about a track they like.
Furthermore, this interface is offered in three different languages: English, Spanish and Catalan. This has been done to increase the usability of the software at the exhibition, taking into account possible cultural differences.
The interface fully supports touch screen environments and is based on HTML5, CSS3 and Javascript. Many features of the jQuery library for Javascript are also used. The range sliders are based on *noUiSlider*[22], while the volume knob is based on *jQuery*

[22]`http://refreshless.com/nouislider/`

Knob[23]. The design of the graphical user interface has been directed toward the ease of use, for the visitors of the Museum may not be particularly confortable with the use of software or of tools related to music manipulation or playback.
Corcerning the reception of the audio streaming, many efforts have been done in order to achieve low-latency in the transmission of the multimedia content. Specifically, tries have involved the use of an icecast[24] server or specific GStreamer units to try to implement low-latency audio streaming directly accessible from the html5 page. None of these tries have fully worked: latency was always registered around 5 seconds, probably due to browser's buffering techiques. This performance was clearly unacceptable. Thus we decided to exploit the functionalities provided by VideoLAN VLC[25]: specifically, we wrote a daemon for the interactive kiosk that launches a hidden instance of VLC as soon as it detects a stream on the TCP port 8070 (generated by the TCPSink of GStreamer). This istance of VLC is then in charge of capturing and playing this stream of multimedia content. The main advantages of this choice are:

- Good latency (around 500ms)
- The user is completely unaware of this, for it is possible to start VLC in a daemon mode, thus without any sort of windows popping up.

Generally, for real-time web application, the use of web protocols RTP (Real Time Protocol) and RTSP (Real Time Streaming Protocol) is suggested, as this usually allows low latencies in multimedia streaming. The use of these protocols for this application have not been taken into account, for their use requires to be integrated into Adobe Flash[26] applications, which are generally discouraged as they introduce additional constraints and are usually not supported on touch devices. Furthermore, we had no experience with this particular programming language, and it could have not been feasible to develop the audio streaming in this language before the inauguration of the exhibition.

The performance of this real-time system will be analyzed in the following chapter.

[23] `http://anthonyterrien.com/knob/`
[24] `http://icecast.org/`
[25] `http://www.videolan.org/vlc/index.html`
[26] `http://get.adobe.com/it/flashplayer/`

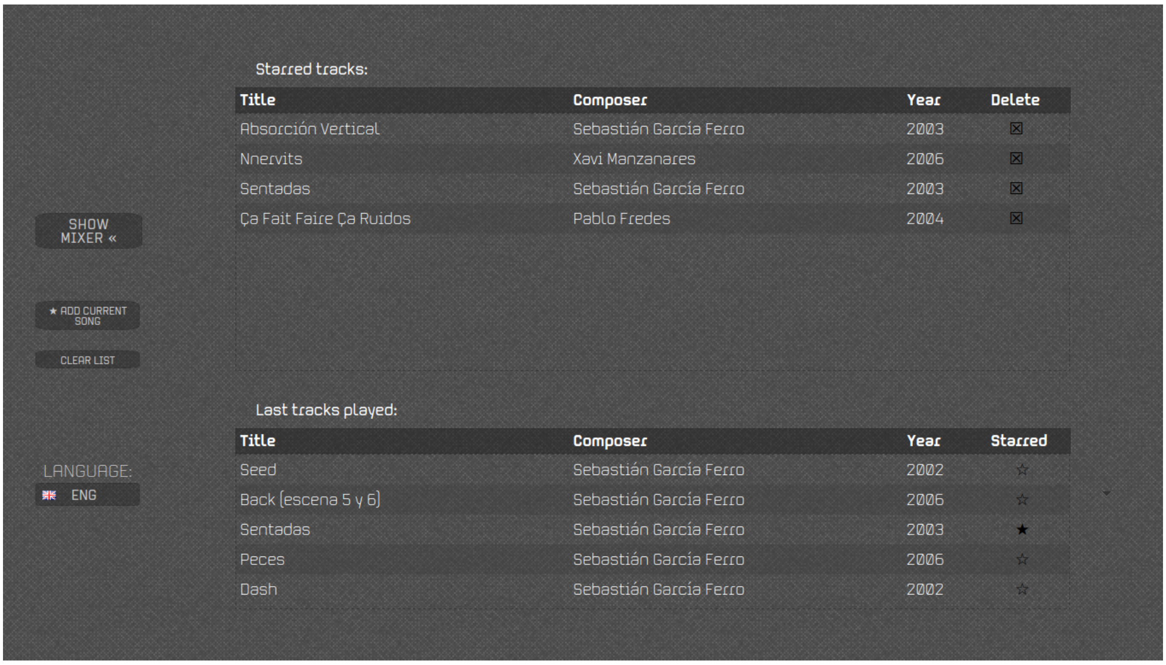

FIGURE 6.7: The second page of the client application GUI, providing information about favorite and last played tracks.

Part III

Results and Discussion

Chapter 7

Results, usage and future work

7.1 Performance

Performance has been the main concern in the development of the system. As already seen in previous chapters, many efforts have been made in order to achieve a good responsiveness to user input in the real time application. We made the clear choice of preferring low times in the offline computation of descriptors (reported in Table 5.4) for this has helped us in achieving good response times in the real time application. Response times were controlled in order to provide the fastest response while preserving the quality of the retrieval: when the user moves sliders, the playlist queue gets empty (which will result in temporary shorter computational times, due to the use of the least precise but fastest music similarity computation algorithm in order to get some new element into the playlist as soon as possible); contrastringly, when the user chooses long segments or does not interact with the sliders, the computational time may increase (for the system realizes that it has more time available for computing music similarity and then uses the most accurate algorithm[1]).
In other words, we have built an *adaptive* system, defined in [73] as a system that:

- Behaves different in different contexts given the same input, and
- Performs this adaptation in order to optimize the system's behaviour in the given context according to some predefined measure.

In order to analyze its performance, we decided to collect data about computational times of the real time application for the choice of 1000 consecutive excerpts, with

[1]We recall that the only difference between the two algorithms lies in the choice of the similarity function, as shown at the point 8 of Section 6.1.1

occasional interaction of the user. This is a reasonable analysis case, for it may be very similar to the real use of the system and also provides a good perspective on the computational times while using the most demanding algorithm of the system for computing music similarity. The hardware used in this analysis is the one reported in Table 5.3. Values of the global times required for choosing these 1000 consecutive excerpts are shown in Figure 7.1.

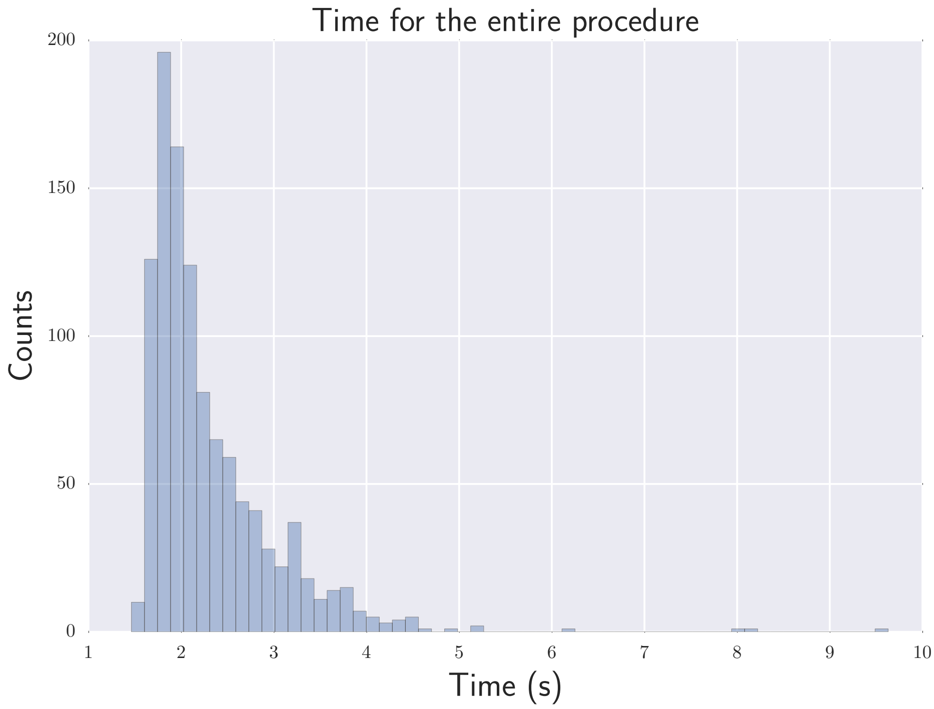

FIGURE 7.1: Global times for selecting next segment.

It can be seen that most of the times, the algorithm for choosing the next excerpt requires between 1.5 seconds and 3 seconds. The presence of some outliers above 5 seconds is due to particular conditions of the environment or of the operative system (such as some other process starting running in the background) and should not be considered meaningful for judging the performance of the algorithm itself.

We consider particularly valuable that the system is capable of adapting its responsiveness to the environment, while still getting good response times also with most intensive computations. As already stated, during this experiment user interactions were occasional, leading the system to use the most intensive (but more precise regarding to retrieval) variant of the algorithm 732 out of 1000 times.

We will now discuss the computational times required for individual steps of the procedure presented in Section 6.1.1. The first step consists of selecting, among all the excerpts of the Phonos catalogue of music, the ones that fulfill the current sliders' values. For this step has to compare several values of all the 159239 excerpts, it is the most demanding task of the algorithm, requiring around 1.6 seconds.

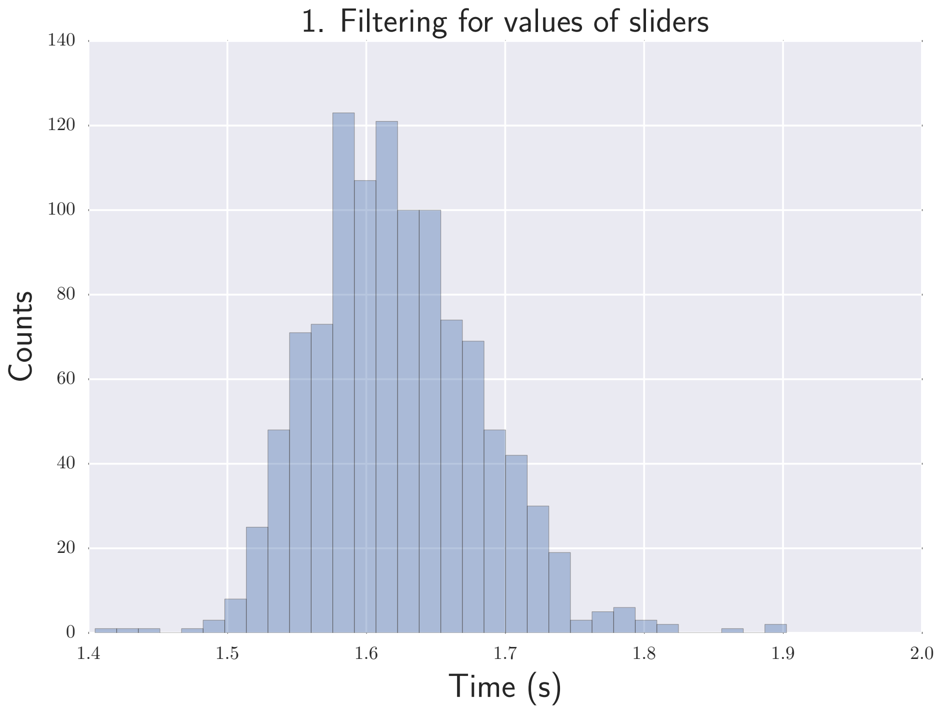

FIGURE 7.2: Time for performing the first step of the procedure: filtering of excerpts based on the current positions of sliders.

The next two steps consist of further filtering of candidate excerpts: at first a Monte Carlo sampling is performed to further reduce the size of the problem, and then we filter out candidate segments that would be generate a quite dissonant result if mixed with the previous excerpt of the playlist. It can be seen the both these operations are much faster than the first one, although the Monte Carlo sampling still requires a tiny considerable time (around 0.02 seconds).

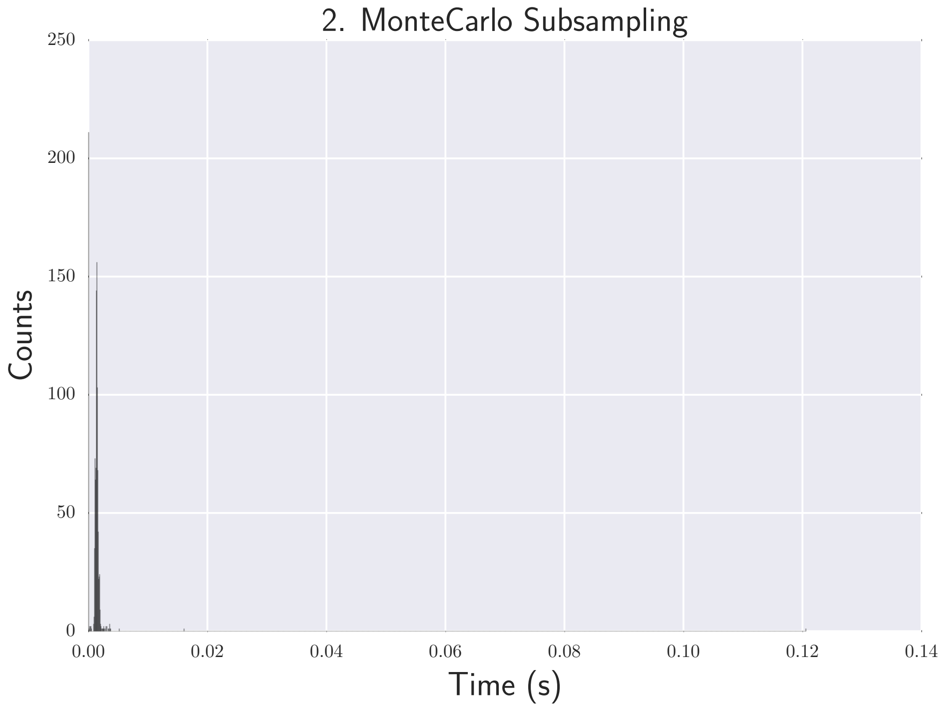

Figure 7.3: Time for performing Monte Carlo sampling.

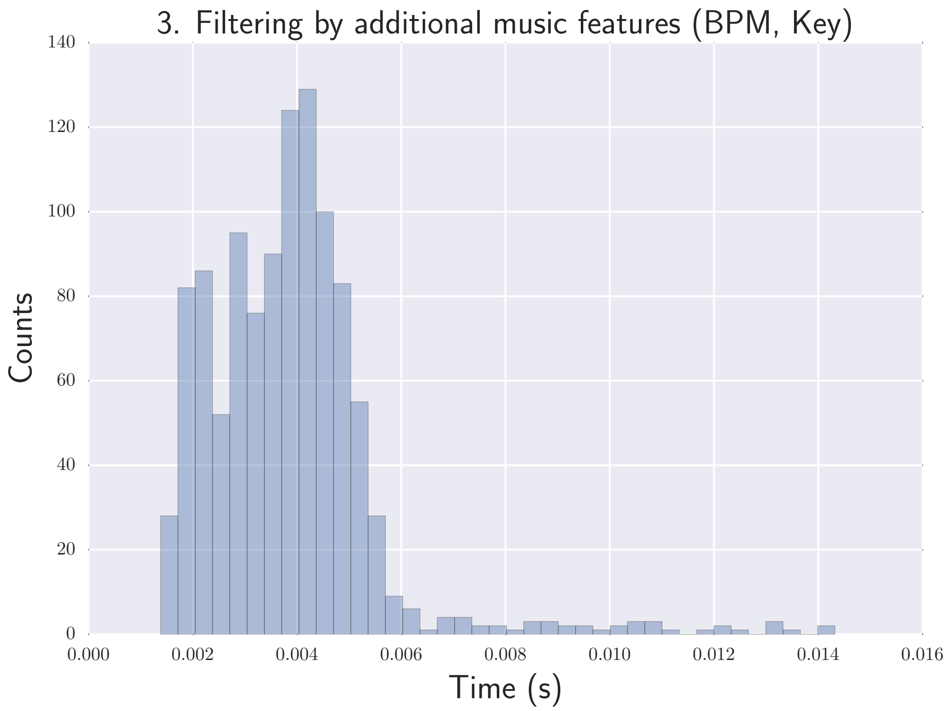

Figure 7.4: Time for filtering music according to musicality with current excerpt (in regards of BPM and key).

At this point, a context-aware distance function is applied to select the next excerpt of the playlist among all the remaining candidates. According to the context, a simple Euclidean distance or a more complex Kullback-Leibler divergence may be computed, and from the following graphs we can see the difference in the computational times of the two:

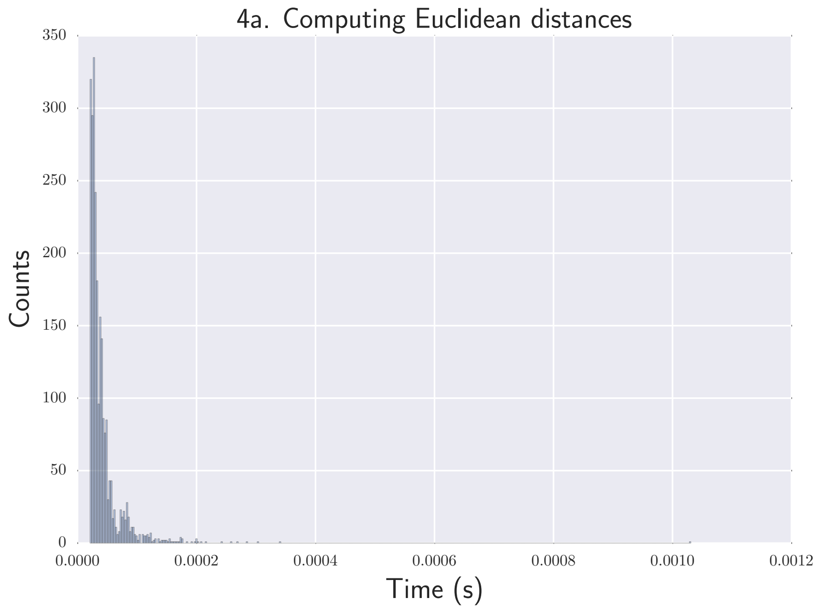

FIGURE 7.5: Time for computing euclidean distance between two 20D points.

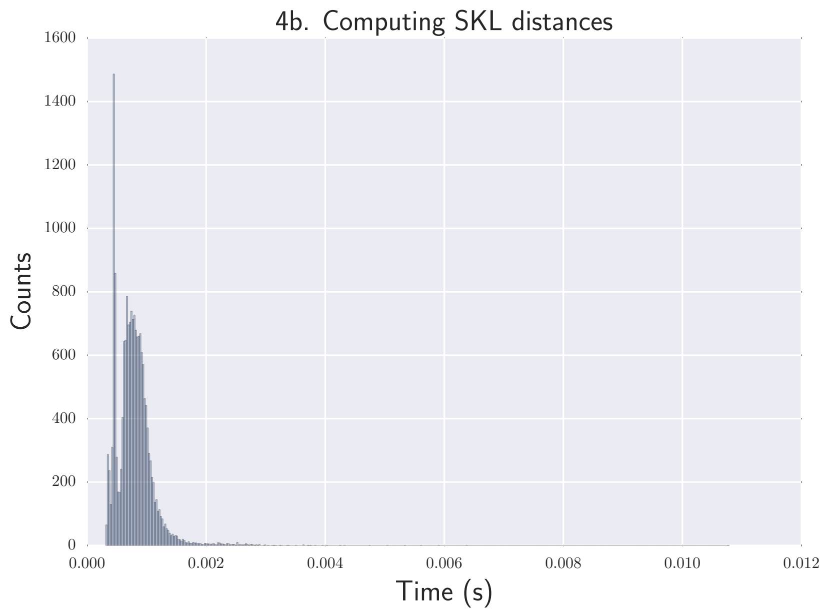

FIGURE 7.6: Time for computing symmetric Kullback-Leibler divergence between the single Gaussian distributions of the MFCC values of two excerpts.

The global times for computing the distance between all the remaining candidates and the last excerpt of the playlist are shown in Figure 7.7. It is important to remark that the values of the single Gaussian distributions are stored on JSON files; therefore the computation of KL divergence requires the access to these files. The time required for accessing and parsing these files is shown in Figure 7.8.

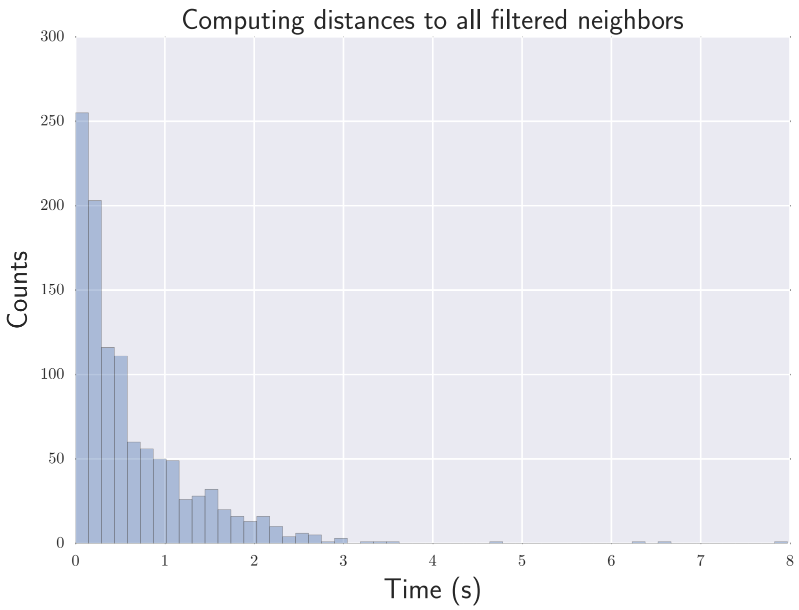

FIGURE 7.7: Time for computing distances from all filtered segments.

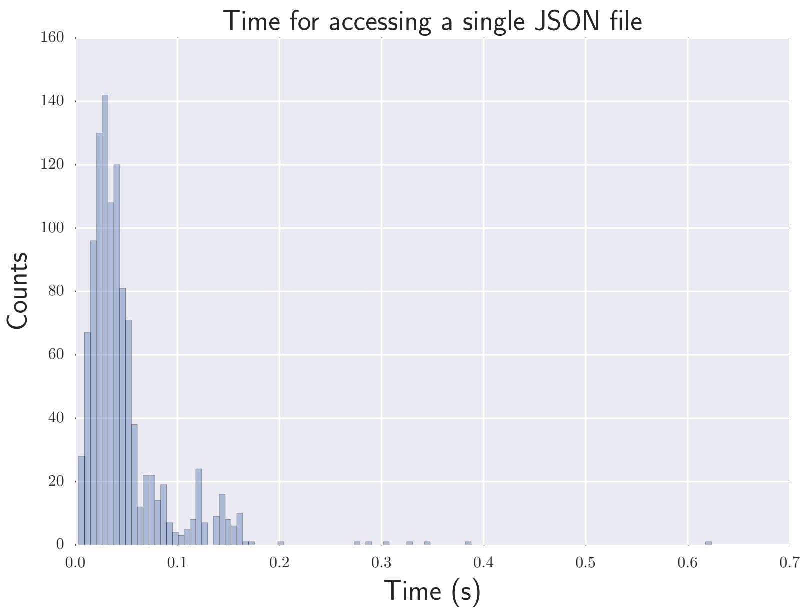

FIGURE 7.8: Time for accessing and parsing a JSON file.

Looking at these graphs, several details emerge:

- Without any doubt, the most demanding task is the filtering of candidates on the base of sliders value. This is due to the fact that this is step acting on the highest number of excerpts. This filtering is based on values that are stored on RAM and therefore is not sensibly slowed down by the time for accessing these values.
- Random sampling, having possibly to act on a very large collection of excerpt, is one of the longest tasks.
- Once all the filtering steps are done, computing all the similarity distances generally requires around one second.
- Computing symmetric Kullback-Leibler distance is around 10 times slower then computing Euclidean distance.
- Time for accessing and parsing JSON file is not negligible and is actually 100 times longer than computing the symmetric Kullback-Leibler distance.

7.2 User evaluation

As explained in Section 4.4, we have decided to perform the evaluation of the system with learn-play-discuss sessions, followed by the compilation of a survey. Specifically, the evaluation sessions are organized as follows:

- The subject of the experiment is introduced to the purpose of the application, without explaining any details about the interaction or the functioning;
- The subject is given 5 minutes to freely interact with the application (playing with the Phonos collection of music), asking for clarification about the use if necessary;
- The subject is finally given the chance to ask about more the functioning of the system;
- The subject answers the survey (presented in Appendix D) with specific questions about ease of usage, enjoyment of musical output, familiarity with the music and with this kind of software, and any problems.

19 subjects took part to this evaluation, recruited from members of the Music Technology Group of Universitat Pompeu Fabra, Barcelona. The subjects were all male, aged between 24 and 34.

The application was generally evaluated very easy to use, with 13 subjects considering it extremely easy:

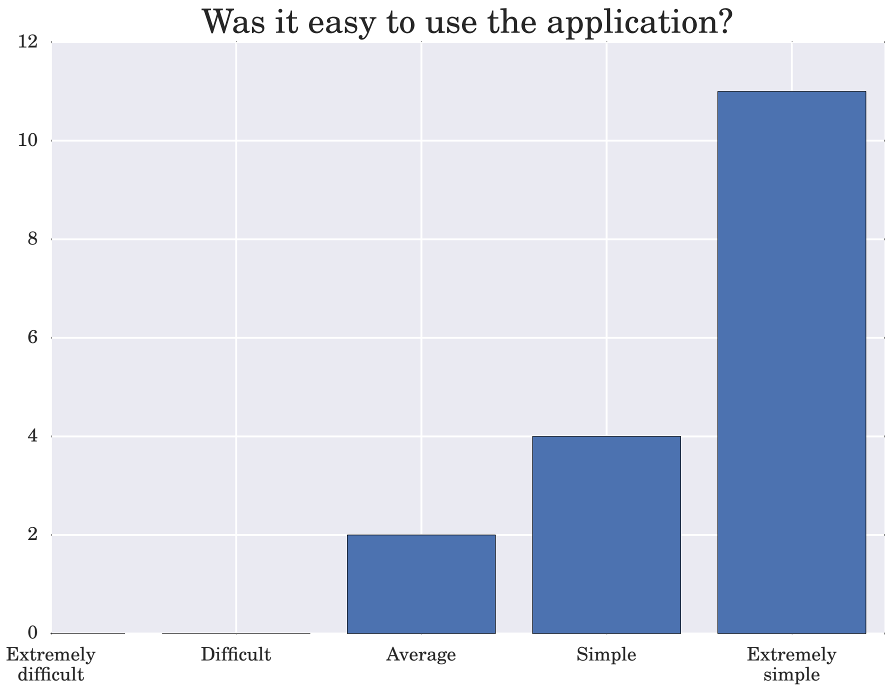

FIGURE 7.9: Evaluation results: ease of use.

despite many of them not having used any similar software before:

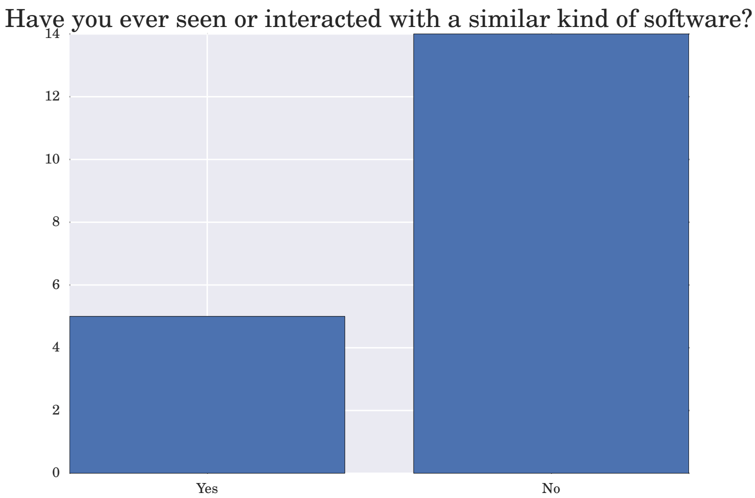

FIGURE 7.10: Evaluation results: familiarity with software.

The meaning of the sliders was generally understood, and the output was evaluated mainly enjoyable and "flowing", despite general lack of familiarity with the music genre:

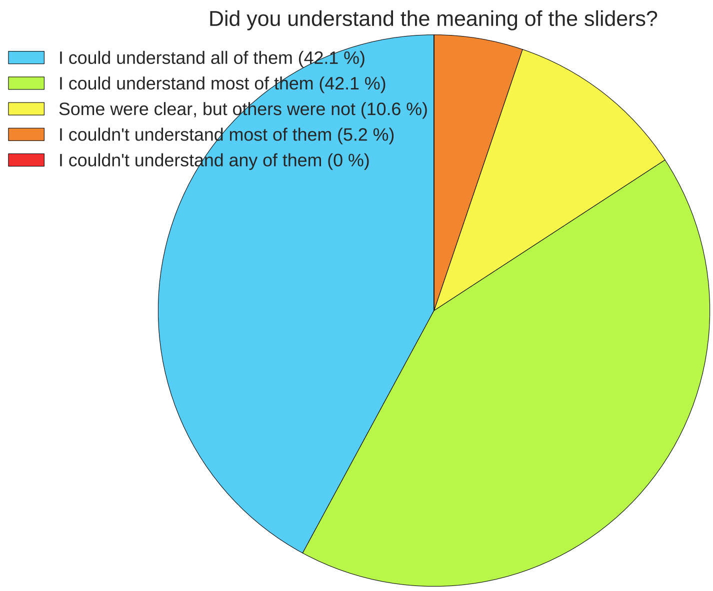

FIGURE 7.11: Evaluation results: understanding of sliders' meaning.

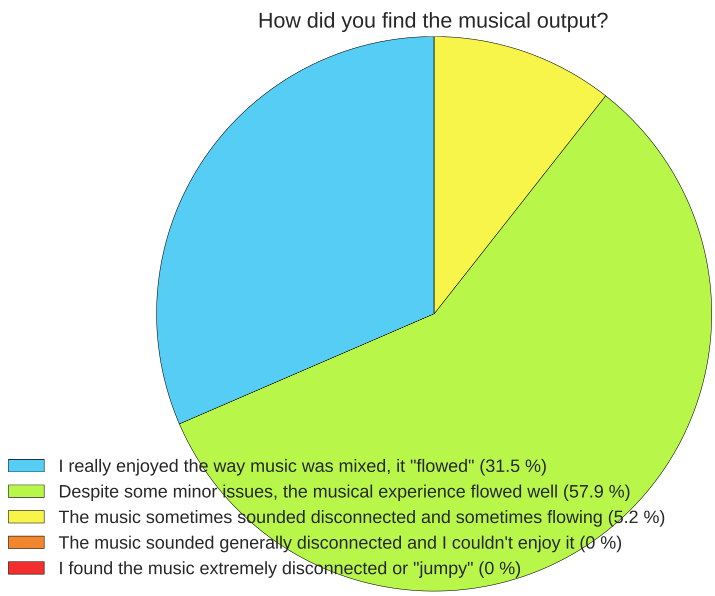

FIGURE 7.12: Evaluation results: enjoyability of musical output.

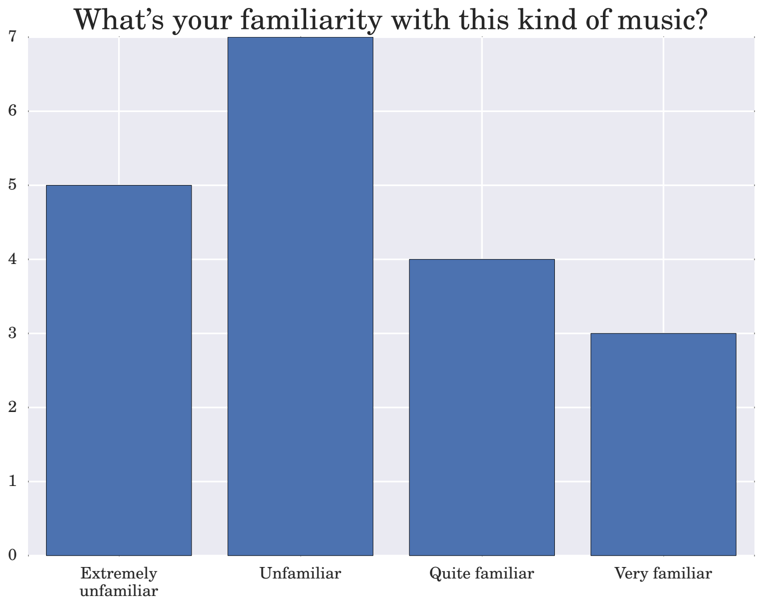

FIGURE 7.13: Evaluation results: familiarity to the electroacoustic genre of music.

Regarding its effective usefulness, the system was generally evaluated as useful for exploring a collection of music:

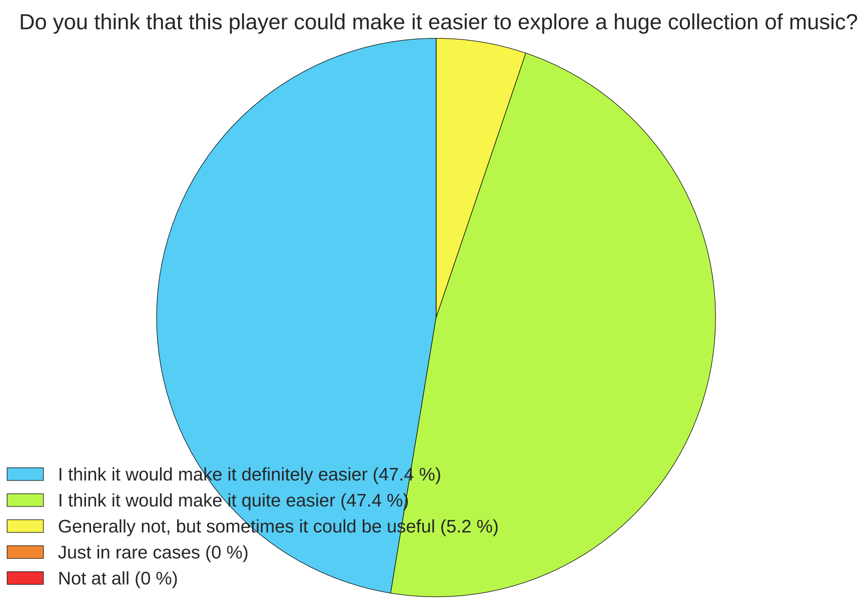

FIGURE 7.14: Evaluation results: usefulness of the software for exploring a collection of music.

while not necessarily constituting a new alternative to a traditional full-track player:

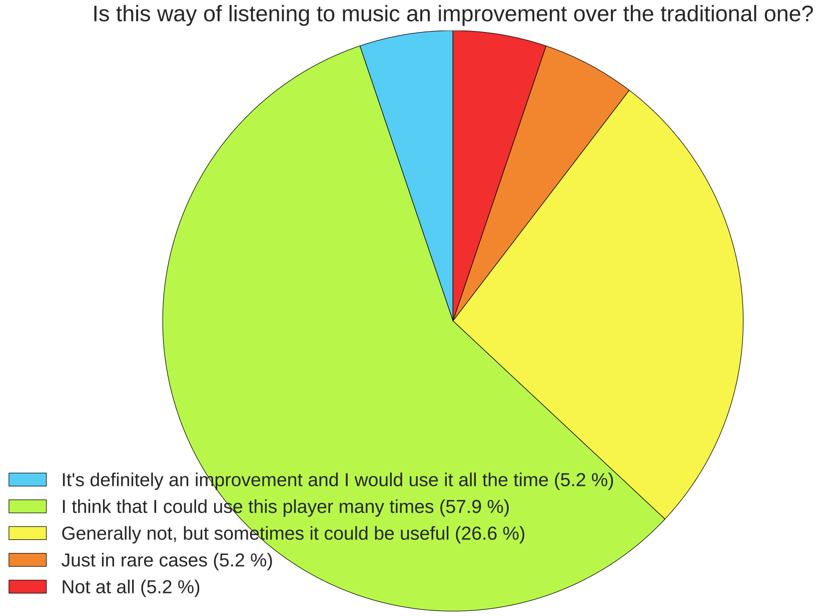

FIGURE 7.15: Evaluation results: comparison of enjoyability in regards to a standard full-track player.

Overall, evaluation results can be considered very satisfactory: the application has succeeded both its purposes of generating an enjoyable flow of music and yet to preserve ease of use. Subjects further commented that, although generally not fully understanding the meaning of all the sliders, they were surprised to see how much the flow of music changed at the interaction with sliders. Moreover, it is interesting to observe that, despite the subjects are generally very used to music technology software, most of them were totally unfamiliar with the software developed. This may be considered as a proof that the software developed actually succeeds in its purpose of providing a new way to approach music playlist generation.

7.3 Usage at exhibition

The inauguration of the exhibition has been on December 18th 2014, at Museu de la Musica, Barcelona. Many people have interacted with the system in order to explore

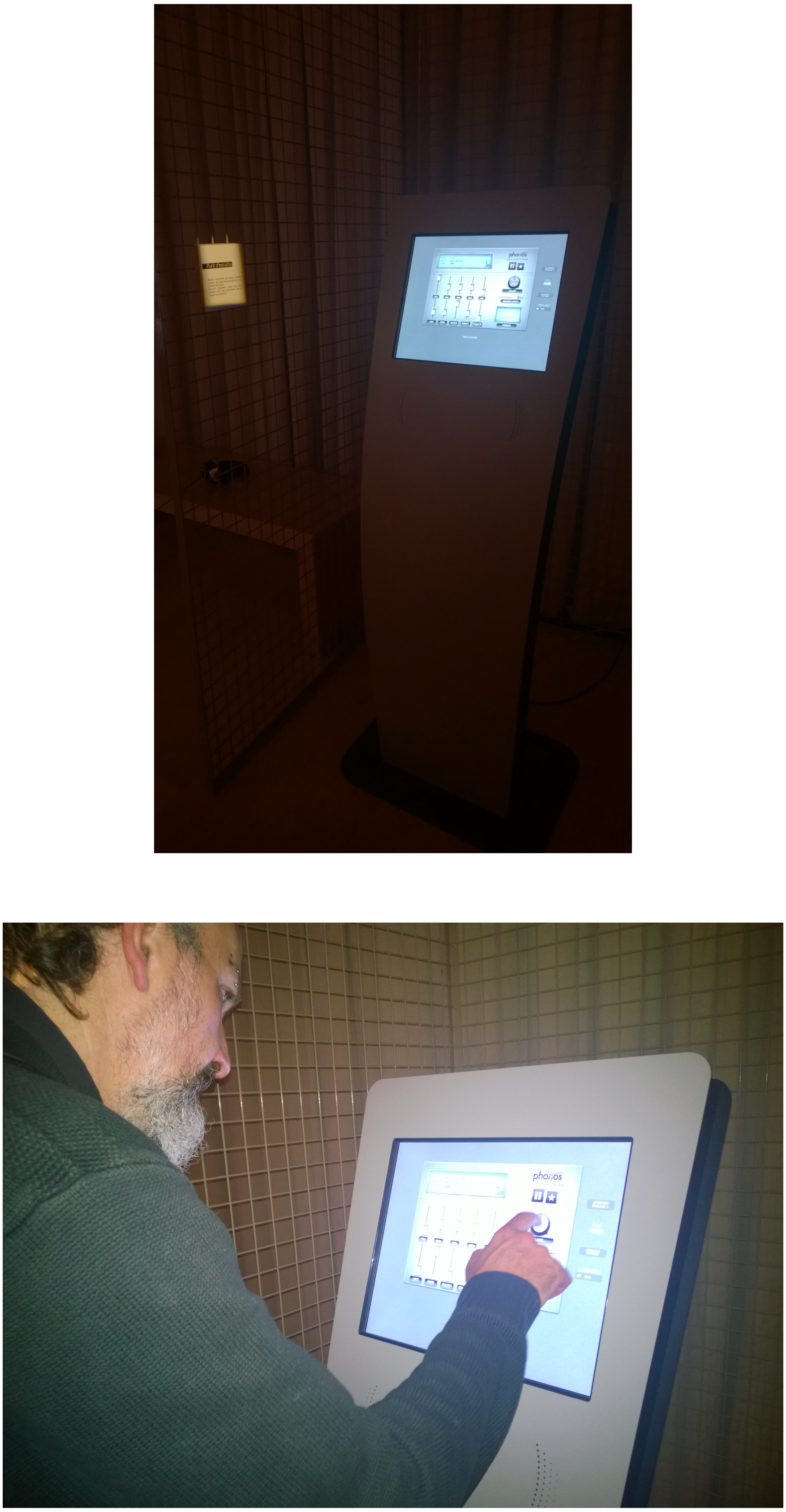

FIGURE 7.16: Use of the interactive kiosk at the exhibition.

the Phonos catalogue of music. The system hasn't incurred in any problem. At the time of the writing (February 2015), it's still daily used by several visitors at the Museum. The interactive kiosk will be dismissed at the end of the exhibition, on late September 2015.

7.4 Results obtained by the study

The main result achieved by the study was the exploitation of latest MIR findings for the development of a system that could easily be used by people not related to the research field and, more in general, not accustomed to the use of software.
This is a further proof that MIR technologies may be extremely useful in a wide range of applications, the most of which linked to common daily life situations. The software integrates not only different descriptors, but also different tools to extract them (Essentia and Echo Nest) in order to maximize the output, something that has rarely been done before.
Another contribute of this study lies in the integration of different researches into a single system: a study of of latest findings has been conducted in order to find what results have been achieved and could have been useful for our purposes. Despite being influenced by other solutions, ours constitutes an original way of solving the problem, for the algorithm we developed offer several new ideas; these are mainly due to the requirement of developing a low-latency system. Furthermore, the requirement of mixing together tracks (instead of just building a playlist of songs to be played one after another) has lead to the choice of implementing some personal musical knowledge in order to discard mixes that would have been perceived highly contrasting. This knowledge is especially related to the field of music composition and perception.

7.5 Future work

Despite having successfully reached its main goals, there is a lot of room for improving the system.
At first, the use of JSON files should be discarded in favour of much faster database tables, for instance PostgreSQL or MySQL. As seen in 7.1, accessing and parsing JSON files is one of the slowest operations of the algorithm (almost 100 times longer than computing the symmetric Kullback-Leibler distance). Implementing a database should allow to use the more computationally intensive variant of the algorithm more frequently

and to make the sampling less aggressive, therefore leading to generally better results.
The computation of music similarity could also be improved and use more sophisticated techniques, such as Fluctuation Patterns, that have shown very good results in similar systems [60].
Furthermore, the development of a web application imposes several limitations (such as general low performances and high latency on audio streaming) that could easily be solved in a native mobile application for tablets or smartphones.
The source code for the application is entirely available at `https://github.com/giuband/Phonos-Music-Explorer`, so that many users can contribute in making it better.
Once the above cited aspects are refined, the development of the system could follow two different paths.

1. The system could be improved in its use for music discovery. For instance the user interface could implement some way of letting the user explore his current position inside the map of excerpts, in order to give a more clear idea about the music of the catalogue. New descriptors could be used and some of them could also be inherited from metadata or machine learning processes.

2. The system could additionaly be integrated into a more creative environment for making music. It could be used for the automatic generation of recommendations while in the process of composing music. For instance, it could suggest the user of using a particular excerpt at some point of his composition to improve the quality of the work. It could also be used as the only source to compose music, providing the ability of automatically composing music made of excerpts while the user gives a direction to this flow, according to his creative intent.

The use suggested in 2. is particularly interesting, for such an application would perfectly fit the vision embraced by the GiantSteps project and provide a new system of producing music, making this amazing creative task accessible at anyone, indipendently from the skill. The process of making music could therefore overthrow its innate boundaries, leading to a world where the creation of art arises from the purest intent of contributing to the world cultural heritage, in spite of lack of limited technical knowledge, economic unavailability and physical impediments.

Appendix A

List of Essentia descriptors

As of November 2014, the features provided by Essentia 2.0.1 are:

Category	Subcategory	Name
Low-level	Barkbands	Values
		Kurtosis
		Skewness
		Spread
	Pitch	Value
		Instantaneous confidence
		Salience
	Spectral	Centroid
		Complexity
		Crest
		Decrease
		Energy
		Energyband high
		Energyband low
		Energyband middle high
		Energyband middle low
		Flatness db
		Flux

		Kurtosis
		Rms
		Rolloff
		Skewness
		Spread
		Strongpeak
	Other	Average loudness
		Dissonance
		Hfc
		Mfcc
		Sccoeffs
		Scvalleys
		Silence rate 30dB
		Silence rate 30dB
		Silence rate 60dB
		Zerocrossingrate
Rhythm	Beats	Position
		Loudness
		Loudness band ratio
	BPM	Value
		Estimates
		Intervals
	First peak	BPM
		Spread
		Weight
	Onset	Onset Rate
		Onset Times
	Second peak	BPM
		Spread

		Weight
Sfx	Pitch	After max to before max energy ratio
		Centroid
		Max to total
		Min to total
	Other	Inharmonicity
		Oddtoeven harmonic energy ratio
		Tristimulus
Tonal	Chords	Changes rate
		Histogram
		Key
		Number rate
		Progression
		Scale
		Strength
		HPCP
	Key	Value
		Scale
		Strength
		Thpcp
	Tuning	Diatonic strength
		Equal tempered deviation
		Frequency
		Nontempered energy ratio

Table A.1: List of features computable with Essentia.

Appendix B

List of Echo Nest Features

Category	Subcategory	Name
Meta		Timestamp
		Duration seconds
		Audio MD5
		Analysis time
		Num samples
		Album
		Decoder version
		Sample rate
		Title
		Duration
		Sample md5
		Decoder
		Artist
		Id
		Window seconds
		Genre
		Analysis sample rate
		Analyzer version
		Bitrate

		Md5
		Analysis channels
Structure	Segments	Start
		Duration
		Confidence
		Loudness start
		Loudness max
		Loudness max time
		Pitch 01
		Pitch 02
		Pitch 03
		Pitch 04
		Pitch 05
		Pitch 06
		Pitch 07
		Pitch 08
		Pitch 09
		Pitch 10
		Pitch 11
		Pitch 12
		Timbre 01
		Timbre 02
		Timbre 03
		Timbre 04
		Timbre 05
		Timbre 06
		Timbre 07
		Timbre 08
		Timbre 09
		Timbre 10

		Timbre 11
		Timbre 12
	Sections	Start
		Duration
		Confidence
		Mode
		Mode confidence
		Key
		Key confidence
		Tempo
		Tempo confidence
		Time signature
		Time signature confidence
		Loudness
Rhythm	Bars	Start
		Confidence
		Duration
	Beats	Start
		Confidence
		Duration
	Tatums	Start
		Confidence
		Duration
Desc		Danceability
		Speechiness
		End of fade in
		Start of fade out
		Liveness
		Acousticness
		Valence

	Energy
	Loudness
	Tempo
	Tempo confidence
	Time signature
	Time signature confidence
	Mode
	Mode confidence
	Key
	Key confidence

Table B.1: List of audio features provided by Echo Nest.

Appendix C

Phonos: list of songs

The musical pieces to be used during the *“Phonos, 40 anys de música electrònica a Barcelona”* exhibition at Museu de la Musica (L’Auditori, Carrer de Lepant, 150, 08013 Barcelona) are:

Artist	**Title**	**Year**
Alain Perón	De Dos Para Uno	1996
	Los Edictos	1998
Albert Llanas	Nexus	1999
	Formants	2004
Alejandro Martínez	Monoleg	N.A.
	Helesponto	1982
	Tazir	1984
	Crisálida	1987
	Machina animata	1987
	Canción de Otoño	1989
	Homenaje L.Nono	1990
	Música Palimpesto	1991
	Vaciando el hueco	1996
Alex Arteaga	Témenos	2006
Alex Geell	Panales	2010
Alex Sanjurjo	Fluir	2003

Alexandra Gardner	Ayehli	2002
	Snapdragon	2002
	New Skin	2003
	Onice	2003
	Luminoso	2004
	Tourmaline	2004
Alexandre Marino	Apparatus, Experimentalis	2008
	Apparatus, Musical	2008
Andrés Lewin-Richter	Joc - Eventos	1976
	Joc - Fondo	1976
	Acción 2 - 1	1978
	Acción 2 - 2	1978
	Acción 2 - 3	1978
	Acción 2 - 4	1978
	Giravolt	1978
	El Paraiso	1979
	El Viento I - 1	1979
	El Viento I - 2	1979
	El Viento I - 3	1979
	El Viento II	1979
	Reacciones I II	1979
	Secuencia IV	1979
	Baschetiada	1980
	El Viento III	1980
	El Viento IV	1980
	Reacciones III	1980
	Wagler Walricci	1981
	Actualidad discográfica	1982
	Sones	1982

6 Songs	1983
Quorum	1983
Secuencia V	1983
Secuencia VI	1983
Tinell	1983
Cogida	1984
In memoriam Manuel Valls	1984
Isaac el Cec	1984
Juegos	1985
Musica electroacústica	1985
Solars Vortices	1985
Desfigurat	1986
Diálogos	1987
Secuencia VII	1987
Homenaje a Zinovieff	1988
Secuencia VIII	1988
Verra la Morte	1988
Verra la Morte 1	1988
Verra la Morte 2	1988
Verra la Morte 3	1988
Verra la Morte 4	1988
Verra la Morte 5	1988
Verra la Morte 6	1988
Verra la Morte 7	1988
Verra la Morte 8	1988
99 Golpes	1989
Ben avra questa donna cor di ghiacio	1989
Secuencia IX	1989
Strings	1989
Brossiana	1990

Donne Fiori	1990
Fragmento (a Nono)	1990
Frullato	1990
Ludus Basiliensis	1991
Reacciones IV	1991
Secuencia X	1991
Radio 2	1996
Sarangi	1999
Configuraciones	2000
Constelaciones	2000
Figuras	2000
Resonancias	2000
Secuencia XI	2001
Secuencia XII	2001
Dreams	2002
Ludus Allavarium	2002
Platjes	2002
Secuencia XIII	2002
Signals	2002
Viso di Primavera	2002
Fantasia	2003
Juego de Acordeón	2003
Meisoh No Ne	2003
Melodias	2003
Metálica	2003
Omaggio a Berio: sequenza per tuba	2003
Secuencia XIV	2003
Essay on Trombone	2004
Fragments	2004

	Secuencia XV	2004
	Arssonxx.rne	2005
	Fluxus es zen?	2005
	Interacciones	2006
	On ”Freesound” Water	2006
	Secuencia XVI	2006
	For Harry	2007
	Retales	2007
	Sombras	2007
	Soplos	2007
	Sospiri	2007
	Friendship Quartet	2008
	Homenaje a Pierre Schaeffer	2008
	Makeup	2008
	Schaeffer granulado	2008
	Aire	2009
	Génesis	2009
	Homenaje a Varese	2009
	Memento	2009
	Paseo BCN	2009
	Sancta Maria	2009
	Slapring	2009
	Spring	2009
	Imagenes	2010
	Secuencia XVIII Fagot	2010
	Multifonia	2011
	Campanas para una celebracion	2012
	Multifonia III	2012
	Secuencia XIX	2014
Anna Bofill	Espai Sonor	N.A.

	Trio para Violin y Cinta	N.A.
Ariadna Alsina	Sinapsis	2006
	Reconstrucció	2011
	Vels Vitris	2012
Ariadna Alsina & David Salleras	Contramarea	2009
Arturo Moya	La Música Que Había en Mis Objetos	1996
	Estampas de Caza 1	2000
	Estampas de Caza 2	2000
	Estampas de Caza 4	2000
	Estampas de Caza 5	2000
Arturo Palaudaria	Estate quieto Voltaire	N.A.
	Adolescencia y Estrella	1980
	Escudellers	1981
	Piamo	1984
	Toda la Memoria de un Hombre	1987
	El Destino de las Cosas	1988
	La Luz de los Sueños	1989
	Boule de Feu	1990
	Paréntesis militar	1990
	El Juicio Estético Universal	1991
	Moverse en el Tiempo	1997
Aurelio Edler-Copes	Women in Process	2013
Cadavers	Exquisits	2003
Carlos Lupprián	Latido	1995
	Agugagá	1996
	Naturaleza Muerta	1997
Claudio Nervi	Improvisación con Oratio Trio	2010
Claudio Zulian	Valent La Notte	N.A.

	El Libro de los Excesos	1983
	San Claudio Vive Solo	1985
	Sexo y Politica	1987
	I Quattro Continenti	1989
	Por de Ser Set	1989
	Sueños Ecléctricos	1989
	Variazione Angelica	1990
	2 Escenas de Macbeth - 1	1991
	2 Escenas de Macbeth - 2 Ruidos	1991
Concha Trallero	Armonias 1	1980
	Armonias 2	1980
	Armonías Sonoras 1	1980
	Armonías Sonoras 2	1980
Cristián López	Leftraru, Viajero Ensoñado - El Río de la Vida	2005
	Leftraru, Viajero Ensoñado - Espírutu Azul	2005
	Leftraru, Viajero Ensoñado - Interludio	2005
	Leftraru, Viajero Ensoñado - Piedra Solitaria	2005
	Leftraru, Viajero Ensoñado - Relámpago Azul	2005
Relief II	Cristián Morales-Ossio	2001
Daniel Domínguez Teruel	TRTPS	2010
	SKTHN	2012
	Study I	2013
	Study II	2013
	Study V	2013
Daniel Rios Aranda	Say It	1987
	Erial	1990
Danilo Vidotti	Sueños	2008
Danio Catanuto	Psicofonias Urbanas 1	2010
	Psicofonias Urbanas 2	2010
Darío Cortés	Formantes	1998

David Dalmazzo	Pulsajes	2010
David Padros	Confluencies	1985
Diego Dall'Osto	Caosmofonia	1998
Doénado, el Ur	Kinoko Tabí	1988
	Pedicoj en la Arena del Pamir	1989
	Zalody	1990
	Yñé do zalod	1991
	A Sensu Contrario	1992
	Blordt Prelar	1992
	Kzadzak	1994
Edgar Barroso	Tu Mateix	2004
	Dux	2005
	Tau	2005
	Tu Soplo Que Transporta	2005
	IOD	2006
	CYT	2007
Edson Zampronha	Mármore	2001
	Mármore 1	2001
	Mármore 2	2001
	Mármore 3	2001
Eduard Resina	Read my LISP	1991
	L'Esquizofrènia Dels Sons	1993
	Aca Amaron	2001
	L'Anna-crusa	2002
Eduardo Polonio	Espai Sonor	1976
Eduardo Reck Miranda	Requiem per una Veu Perduda	1997
Elsa Justel	Midi de Sable	2000
Enrique Marín	Elementos Constantes, Hechos Variables	2002
	Transiciones de Fase	2007

Ensamble Crumble y ReacTable	Untitled 1	2006
	Untitled 2	2006
	Untitled 3	2006
	Untitled 4	2006
	Untitled 5	2006
	Untitled 6	2006
	Untitled 7	2006
FMOL Trio	Untitled 1	2001
	Untitled 2	2001
	Untitled 3	2001
Felipe Pérez Santiago	CampoSanto	2004
	Encandilado	2007
	Hunger FM	2009
	Hurt	2009
	Ishmael	2009
	Miuk	2009
	Post War	2009
	Tacto	2009
	War-Post War	2009
	Pronto Desapareceremos	2012
Fernando Jobke	Ecos 1	2008
	Ecos 2	2008
	Ecos 3	2008
	Ecos 4	2008
	Ecos 5	2008
	Ecos 6	2008
Félix Luque & Ricardo Gadea	Cuerpos Sensibles	2005

Félix Luque & Thomas Charveriat	The Machine Manifesto	2004
Gabriel Brncic	Batucada Amenazante	1970
	El Túnel (a Ernesto Sabato)	1970
	Agua 1	1971
	Agua 2	1971
	Agua 3	1971
	Cielo	1980
	Destierro	1980
	Chile Fértil Provincia	1983
	Concert Gothique	1985
	Operas Rotas	1985
	Clarinen Tres	1986
	Clarinen Tres	1986
	Desêtre a Oscar Masotta	1986
	Triunfo Para las Madres	1986
	Aria y Pasacalle	1987
	Ese Mar	1987
	Música de cámara	1987
	Historia de Dos Ciudades	1988
	Alegrias	1989
	Composición de 1989 a Eduardo Polonio	1989
	Dulcian Concert	1989
	ariaciones sobre Sonatas e Interludios	1989
	Adagio-Scherzo	1990
	Vade Retro a Luigi Nono	1990
	Dos Esbozos Para Antiguos Instrumentos Electrónicos	1994
	...Que No Desorganitza Cap Murmuri	1995

	Constanza	1996
	Claro-Oscuro	1998
	Meng	1998
	Clarinet Concert	1999
	Coreutica	1999
	Ergon-Rondeau	2000
	A Joan Miró	2001
	Alto-Concert II	2001
	Bass clarinet-Concert for Harry Sparnaay	2003
	Son(ru)idos I	2003
	Son(ru)idos II	2003
	La Casa del Viento 1	2006
	La Casa del Viento 2	2006
Gaspar Lukacs Esguep	Pregoneros de Barcelona	2002
Germán Brull Moreno	Sin título	2004
	Sin título	2004
Graciela Muñoz Farida	Arboleda	2011
	Fragmentos de un Arbol	2011
	Lo Que No Das Te Lo Quitas	2011
	Viento Sur	2011
Graeme Truslove	Piece for Guitar and Tape	2001
Graham Coleman	Improvisation	2007
	Improvisation	2007
Guillermo Eisner	Guitarrísticamente	2007
Igor Bimsbergen	Duo Para Siete	1996
	Luis y Marylin	1998
Ismael Sanoja & Kai Kraatz	Free What	2006
	Free What 1	2006
	Free What 2	2006
	Free What 3	2006

	Free What 4	2006
	Free What 5	2006
	Free What 6	2006
	Free What 7	2006
	Free What 8	2006
Jan Schacher	Traumtäntze	2000
Javier Navarrete	Preludios	1976
Jelena Vico	Almogavers	2008
	Brithm	2008
	Mrzbw	2008
	Pangea	2008
	Zeno	2008
	Zitar	2008
Jep Nuix	Gallinària	1980
	Doble Peça de Lletres i Sons	1981
	Tres Canons de Noces	1981
	Ad Valorem	1984
	Halterofilia 1	1984
	Serenata Nocturna	1985
	L'Inizio	1986
	Dit a Dit, Pas a Pas	1988
	Asirara	1989
	Monoleg	1989
	Trialeg	1989
	His Master's Voice	1990
	Improvisació per a tubs	1990
	Pensant en Nono	1990
	Percuflu	1990
	Atentament	1992

	Stack	1995
Joan Bagés i Rubí	Intersections-BouleWav 2.0	2006
Joan Josep Ordinas & Claudio Zulian	Al Tranquilodromo	1981
Joan Sanmarti	Passadis	2001
	Reflexos Improvisacxiones Asistidas por Ordenador	1997
	Xtrapolució 4	1998
Jordi Rossinyol	Ricercare a 5	1986
	Objectes Trobats a la Platja	1987
	Ocellots	1988
	Mòbils Inquiets i Altres Equivocs	1989
	Prosper Laberint Intermitent	1990
	Variaciones guit	1990
	Concert Mestis	1997
	Ecliptic	2004
Jorge Sad	El Doble Bandoneón	1998
	La Ida Hacia Abajo de la Tierra de la Tarde	1999
Josep Maria Guix	Landscape	2010
	Landscape	2010
	Landscape	2010
Josep Maria Mestres Quadreny	Oxo	1963
	Peça per a Serra Mecanica	1963
	Homenaje a Galileo	1965
	Trois Cánones en Hommage à Galilea	1968
	Aronada	1972
	El Teler de Teresa Codina	1973
	Song for Jane Manning	1973
	Espai Sonor	1976
	Espai Sonor	1976

	Quina	1979
	Cánones a Galileo	1989
José Manuel Berenguer	El Pensamiento Que Se Trabaja Hacia la Luz	N.A.
	Spira	N.A.
	Montardo	1983
	A Florats	1984
	La Logica de la Sorpresa	1984
	El Ponent Excesiu	1985
	La Perla Estranya	1985
	La Relojeria del Tío Paco	1985
	Música en la Noche	1985
	Quartet Ambar	1986
	Color	1987
Juan Antonio Moreno	Polifonía de Colores	1984
	Preludio III a Lluis Callejo	1988
	Nono Está Aqui	1990
	Buenhache	1991
Lina Bautista	G-Gems	N.A.
	Bombyx Mori	2010
	Encélado	2011
Linda Antas	A River From the Walls	1999
	Sueño sin palabras	2001
Lisos-Estriados	Untitled	2001
Llorenç Balsach	Carota i Caramel	1976
	Espais residuals (Espai I)	1976
	L'assassi Bagliatti	1977
	El Cant de les Arteries	1979
Lluis Callejo	Caleidoscopi	N.A.
	Dibuixos	1981
	Estructures 6502	1982

	Paisatges	1983
	Tèxtils	1984
	A Pitàgores en do	1985
	A Pitàgores en re	1985
	Espai Sonor	2003
	Stokos IV	2003
Luis Caruana	La Triste Herida de Margot	2001
	Por Tus Pliegues Transita la Pena	2001
Marcelo DeMatei & Carlos Smith	Animales Divinos	2003
Mario Peña y Lillo	Petit Estudi	N.A.
	Beso	2013
	El Contorno de sus Ojos	2013
	Esencia	2013
	He Perdido la Apuesta	2013
	Youkali	2013
Mario Verandi	Figuras Negras	1992
	Flamencas	1995
	Faces and Intensities	1996
	Frèquences de Barcelone	1997
	Mu	1997
Matthew Burtner	Mists	1996
	Fern	1997
	Incantation S4	1997
	Split Voices	1997
	Glass Phase	1998
	Portals of Distortion	1998
	Delta 1	2000
Mauricio Valdés	Duo	2002

	Popan II	2008
Mercè Capdevila	Gramatges	1983
	Baobab	1985
	Nu	1990
	Alegries de Comèdia	1991
	Mercuri	1991
	Fons de Mar	2000
	Pols	2000
	Puente	2000
	A Chillida	2009
Miquel Jordà	Time Machine	2000
Nadine Kroher	La Máquina, el Humano y el Olivo	2013
	Mixed Signals	2014
Neil Harbisson	Concierto Sonocromático	2011
Oliver Rappoport	Catarsis III	2009
Oriol Graus	Laberint Mutant II	1987
	Miradaclosa IV	1987
	I despres...	1990
	La Solitud de l'Origen	1990
	La conseqüència	1990
	La intuïció	1990
	Oketus	1990
	Diferents Formes de Dir - T'Ho	1991
	La Tolerancia	1993
	El Laberint de l'Esperança	2000
	Paisatge Interior	2010
Oscar Martin	Black Nature	2012
	Black Nature	2012
Pablo Fredes	Fer et Defer	N.A.
	Historia del Vinilo	N.A.

	Trama	N.A.
	Las Nenias del Sonido	2002
	Ça Fait Faire Ça Ruidos	2004
	El Círculo de Cero	2009
	sX-off-on	2009
	Azu Gemma Torralbo	2011
	Son-ethos (Sueños en el Sueño)	2011
	Son-file	2011
	iO	2011
	on_off Gemma Torralbo	2011
	Cero Roce Sostenuto	2012
Pedro Barboza	Estratos	2001
	Estratos	2001
	La fila de Ocata	2001
	inTENSIONtres	2004
Ramon Humet	Mantra I	2005
Rebecka Biro	1	2005
	2	2005
Ricardo Arias	Daffodil for Peter Billings	N.A.
Ricardo Arias & Carlos Gómez	Improvisación	2009
Ricardo Arias & Roberto García	Sol Sonoro 1	2008
	Sol Sonoro 2	2008
Roger Costa	Je Suis l'Autre	2012
Ross Bencina	off ICMC2005	2005
	off ICMC2005	2005
Sanjay Fernandes	Simple Math	2010
Sebastián García Ferro	Ella Era Todo - Escribir Sobre Piel	N.A.
	Ella Era Todo - Yang	N.A.

Europa 1 - Piano	N.A.
Europa 2 - Crescendo	N.A.
Europa 3 - Bosque	N.A.
Europa 4 - Vibracion	N.A.
Europa 5 - Noise Delay Long	N.A.
Europa 6 - Piano	N.A.
Equs	2001
Noise	2001
Pulso	2001
Afro Dero	2002
Ceratti	2002
Dash	2002
Seed	2002
Shadow	2002
Silla	2002
Absorción Vertical	2003
Bosa	2003
Drugs	2003
Etheric	2003
Fiesta	2003
Final	2003
Huellas	2003
Huellas Intro	2003
Mistrius	2003
Nervio	2003
Rebotes	2003
Rhesus	2003
Sentadas	2003
Solo Caro	2003
Trio	2003

Viaje Transparente	2003
Vacio y Multitud 1	2004
Vacio y Multitud 2	2004
Bajo el Agua	2005
Caidas	2005
Come Home	2005
Flotar	2005
Sumergir	2005
Back (escena 1)	2006
Back (escena 3)	2006
Back (escena 5 y 6)	2006
Gatos	2006
Mandrös	2006
Modified - Intro	2006
Peces	2006
Caras Jazzie End	2007
Clock	2007
Corn	2007
Despertar	2007
Fork	2007
Mañana	2007
Mediodia	2007
Metting	2007
Noche	2007
Pointing	2007
Sueños	2007
Tarde	2007
Vaiven Parte 1	2007
Vaiven Parte 2	2007

	Travellers 1	2008
	Travellers 2	2008
	Travellers 3	2008
Sebastián Jara Bunster	La Lámpara	2010
Sergi Jordá	For Eric	2001
Sergio Naddei	Big Bang	2011
	Rock Memories	2011
	The Fly	2011
	Windows	2012
	Almost New Places	2013
	Almost New Spaces	2013
	Through Memories 1	2013
	Through Memories 2	2013
	Through Memories 3	2013
	Through Memories 4	2013
	Through Memories 5	2013
	Reactable	2014
Sergio Poblete	Actions	1998
Sáez,Ignacio	Místicos I Phonos Fund.Miro	1987
	El Riu Fosc	1988
	Horizonte Encadenado	1990
Teruyoshi Kamiya	For Fernando Riera	1996
	Dance of Stone	1998
Thomas Charveriat & Félix Luque	The Machine Manifesto	2004
Tim Schmele	Hemispherical Glitch Study	2013
	Neurospaces	2013
	Waiting	2013
Trino Zurita & Teresa Carrasco	Seguiriyas	2013

Xavi Manzanares	Doll_sa_caustika	2006
	Errortunnel	2006
	H2O	2006
	Massiva	2006
	Nnervits	2006
	Nuvols	2006
	Openspaceinvaders	2006
	Plastiknazzxs	2006
	R4gg4gg4r	2006
	Rezzaka	2006
	Segmentationfault0100	2006
	Segmentationfault1001a	2006
	Segmentationfault1001b	2006
	Standbykut	2006
	Stirofoammentre	2006
	Tripikx	2006
Xavier Maristany	East Cocker	1984
	Remember Me	1999

Table C.1: Phonos catalogue to be used during the exhibition *"Phonos, 40 anys de música electrònica a Barcelona"*.

Appendix D

Questionnaire used for evaluation

Question	Possible answers
Was it easy to use the application?	1 (extremely difficult)
	2 (generally difficult)
	3 (average)
	4 (generally simple)
	5 (extremely simple)
Did you understand the meaning of the sliders (e.g.: the sliders for setting the desired loudness)?	I couldn't understand any of them
	I couldn't understand most of them
	Some were clear, but others were not
	I could understand most of them
	I could understand all of them
How did you find the musical output?	I found the music extremely disconnected or "jumpy"
	The music sounded generally disconnected and I couldn't enjoy it
	The music sometimes sounded disconnected and sometimes flowing
	Despite some minor issues, the musical experience flowed well
	I really enjoyed the way music was mixed, it "flowed"

Have you ever seen or interacted with a similar type of software?	Yes
	No
Please rate the familiarity you felt with the music that was played (i.e., how similar or far was the played music with the music you are used to listen to):	Very dissimilar
	Quite dissimilar
	Familiar
	Very familiar
Do you think that this way of listening to music is an improvement over the traditional full-track player?	Not at all
	Just in rare cases
	Generally not, but I can see its usefulness
	I think that I could use this player many times
	It's definitely an improvement and I would use it all the time
Do you think that this player could make it easier to explore a huge collection of music?	Not at all, it just makes it harder
	Just in rare cases
	Generally not, but sometimes it could be useful
	I think it would make it quite easier
	I think it would make it definitely easier

TABLE D.1: Questionnaire used for evaluation of the developed system.

Bibliography

[1] A. Li-chun Wang, and Th Floor Block F. *An industrial-strength audio search algorithm*. Proceedings of the 4 th International Conference on Music Information Retrieval, 2003.

[2] J.S. Downie. *The Scientific Evaluation of Music Information Retrieval Systems: Foundations and Future*. Computer Music Journal, 28:12-23, 2004.

[3] N. Orio. *Music Retrieval: A Tutorial and Review*. Foundations and Trends®in Information Retrieval, 1(1):1-90, 2006.

[4] M. Schedl, E. Gómez, and J. Urbano. *Music Information Retrieval: Recent Developments and Applications*. Foundations and Trends®in Information Retrieval, 8(2-3):127-261, 2014.

[5] J.J. Aucouturier, and F. Pachet. *Improving Timbre Similarity: How High is the Sky?*. Journal of Negative Results in Speech and Audio Sciences, 1(1):1-13, 2004.

[6] J.J. Aucouturier, and F. Pachet. *Music similarity measures: What's the use?*. ISMIR, 2002.

[7] X. Serra, M. Magas, E. Benetos, M. Chudy, S. Dixon, A. Flexer, E. Gómez, F. Gouyon, P. Herrera, S. Jorda, O. Paytuvi, G. Peeters, J. Schlüter, H. Vinet, and G. Widmer. *Roadmap for Music Information ReSearch*. Geoffroy Peeters (editor), Creative Commons BY NC ND 3.0 license, 2013.

[8] M.E.P. Davies, P. Hamel, K. Yoshii and M. Goto. *AutoMashUpper: Automatic Creation of Multi-Song Music Mashups*. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 22(12):1726-1737, 2014.

[9] G. Szymanski. *Pandora, or, a never-ending box of musical delights*. Music Reference Services Quarterly, 12(1):21-22, 2009.

[10] M. Slaney. *Web-scale multimedia analysis: Does content matter?*. IEEE Multimedia, 18(2):12-15, 2011.

[11] J.J. Aucouturier. *Sounds like teen spirit: Computational insights into the grounding of everyday musical terms.* Language, Evolution and the Brain. Frontiers in Linguistics, 35-64, 2009.

[12] S.J. Green, P. Lamere, J. Alexander, F. Maillet, S. Kirk, J. Holt, J. Bourque, and X.W. Mak. *Generating transparent, steerable recommendations from textual descriptions of items.* ACM Conference on Recommender Systems (RecSys'09), 281-284, 2009.

[13] D. Bogdanov, and X. Serra. *From music similarity to music recommendation: Computational approaches based on audio features and metadata.* PhD Thesis, Universitat Pompeu Fabra, 2013.

[14] D. Schnitzer. *Mirage – High-Performance Music Similarity Computation and Automatic Playlist Generation.* Master's Thesis, Vienna University of Technology, 2007.

[15] L. Barrington, D. Turnbull, D. Torres, and G. Lanckriet. *Semantic similarity for music retrieval.* Music Information Retrieval Evaluation Exchange (MIREX), 2007.

[16] K. West, and P. Lamere. *A model-based approach to constructing music similarity functions.* EURASIP Journal on Advances in Signal Processing, 149-149, 2007.

[17] G. Bonnin, and D. Jannach. *Automated Generation of Music Playlists: Survey and Experiments.* ACM Computing Surveys, 47(2), Article 26, 2014.

[18] R. Dias, and M. J. Fonseca. *MuVis: An application for interactive exploration of large music collections.* Proceedings of MM, 1043-1046, 2010.

[19] J.S. Downie. *Music information retrieval.* Annual Review of Information Science and Technology, 37(1):295-340, 2003.

[20] M. Grachten, M. Schedl, T. Pohle, and G. Widmer. *The ISMIR cloud: A decade of ISMIR conferences at your fingertips.* Proceedings of ISMIR, 63-68, 2009.

[21] M. Dopler, M. Schedl, T. Pohle, and P. Knees. *Accessing music collections via representative cluster prototypes in a hierarchical organization scheme.* Proceedings of ISMIR, 179-184, 2009.

[22] I. Schmädecke, and H. Blume. *High performance hardware architectures for automated music classification.* Algorithms from and for Nature and Life, 539-547, 2013.

[23] O. Celma, and X. Serra. *FOAFing the music: Bridging the semantic gap in music recommendation.* Web Semantics: Science, Services and Agents on the World Wide Web, 6(4):250-256, 2008.

[24] F. Pachet, P. Roy, and D. Cazaly. *A combinatorial approach to content-based music selection*. Multimedia, 7(1):44-51, 2000.

[25] D. Bogdanov, and P. Herrera. *How much metadata do we need in music recommendation? A subjective evaluation using preference sets*. WProceedings of ISMIR, 97-102, 2011.

[26] N. Aizenberg, Y. Koren, and O. Somekh. *Build your own music recommender by modeling internet radio streams*. Proceedings of WWW, 1-10, 2012.

[27] R. Van Gulik, and F. Vignoli. *Visual playlist generation on the artist map*. Proceedings of ISMIR, 520-523, 2005.

[28] F. Coelho, J. Devezas, and C. Ribeiro. *Large-scale crossmedia retrieval for playlist generation and song discovery*. Proceedings of OAIR, 61-64, 2013.

[29] O. Celma. *Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Springer, 2010.

[30] J. McDermott. *Auditory preferences and aesthetics: Music, voices, and everyday sounds*. Neuroscience of Performance and Choice. Elsevier, 227-256, 2012.

[31] A. Germain, and J. Chakareski. *Spotify me: Facebook-assisted automatic playlist generation*. Proceedings of MMSP, 25-28, 2013.

[32] D. Hauger, J. Kepler, M. Schedl, A. Kosir, and M. Tkalcic. *The million musical tweets dataset: What can we learn from microblogs*. Proceedings of ISMIR, 189-194, 2013.

[33] B. Logan. *Music recommendation from song sets*. Proceedings of ISMIR, 425–428, 2004.

[34] B. Logan. *Mel frequency cepstral coefficients for music modeling*. Proceedings of ISMIR, 2000.

[35] F. Vignoli, and S. Pauws. *A music retrieval system based on user driven similarity and its evaluation*. Proceedings of ISMIR, 272-279, 2005.

[36] P. Knees, T. Pohle, M. Schedl, and G. Widmer. *Combining audio-based similarity with Web-based data to accelerate automatic music playlist generation*. Proceedings of MIR, 147-154, 2006.

[37] D. Jannach, M. Zanker, M. Ge, and M. Gröning. *Recommender systems in computer science and information systems — A landscape of research*. Proceedings of EC-Web, 76-87, 2012.

[38] N.H. Liu, S.W. Lai, C.Y. Chen, and S.J. Hsieh. *Adaptive music recommendation based on user behavior in time slot.* Computer Science and Network Security, 9(2):219-227, 2009.

[39] S. McNee, J. Riedl, and J. Konstan. *Being accurate is not enough: how accuracy metrics have hurt recommender systems.* CHI'06 extended abstracts on Human Factors in Computing Systems, p.1001, 2006.

[40] G. Jawaheer, M. Szomszor, and P. Kostkova. *Comparison of implicit and explicit feedback from an online music recommendation system.* Int. Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec'10), p.47-51, 2010.

[41] B. Whitman, and S. Lawrence. *Inferring Descriptions and Similarity for Music from Community Metadata.* Proceedings of the 2002 International Computer Music Conference (ICMC 2002), p.591-598, 2012.

[42] S. Dixon *Onset Detection Revised.* Proc. of the 9th Int. Conference on Digital Audio Effects (DAFx'06), p.133-137, 2006.

[43] J.P. Bello, C. Duxbury, M. Davies, and M. Sandler. *On the use of phase and energy for musical onset detection in the complex domain.* Signal Processing Letters, IEEE, 11(6):553-556, 2004.

[44] P. Masri. *Computer Modeling of Sound for Transformation and Synthesis of Musical Signal.* Ph.D. dissertation, Univ. of Bristol, 1996.

[45] J.P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M. B. Sandler. *A tutorial on onset detection in music signals.* IEEE Transactions on Speech and Audio Processing, 13(5):1035-1047, 2005.

[46] S. Böck, and G. Widmer. *Maximum filter vibrato suppression for onset detection.* Proceedings of the 16th International Conference on Digital Audio Effects (DAFx-13), Maynooth, Ireland, 2013.

[47] N. Degara, E.A. Rua, A. Pena, S. Torres-Guijarro, M.E. Davies, and M.D. Plumbley. *Reliability-informed beat tracking of musical signals.* IEEE Transactions on Audio, Speech, and Language Processing, 20(1):290-301, 2012.

[48] T. Fujishima. *Realtime chord recognition of musical sound: A system using Common Lisp Music.* Proceedings of the International Computer Music Conference, Beijing. 1999.

[49] E. Gómez. *Tonal Description of Polyphonic Audio for Music Content Processing.* INFORMS Journal on Computing, 18(3):294-304, 2006.

[50] J. Bonada. *Automatic technique in frequency domain for near-lossless time-scale modification of audio.* Proceedings of International Computer Music Conference, 396-399, 2000.

[51] G. Cabral, J.P. Briot, and F. Pachet. *Impact of distance in pitch class profile computation.* Proceedings of the Brazilian Symposium on Computer Music, 319-324, 2005.

[52] D. Bogdanov, N. Wack, E. Gómez, S. Gulati, P. Herrera, O. Mayor, G. Roma, J. Salamon, J. Zapata, and X. Serra. *ESSENTIA: an open-source library for sound and music analysis.* Proceedings of the 21st ACM international conference on Multimedia, 855-858, 2013.

[53] D. Schnitzer, A. Flexer, and G. Widmer *A fast audio similarity retrieval method for millions of music tracks.* Multimedia Tools and Applications, 58(1):23-40, 2012.

[54] P. Cano, M. Kaltenbrunner, and N. Wack. *An industrial-strength content-based music recommendation system.* Proceedings of the 28th annual internation ACM SIGIR conference on research and development in information retrieval, 673-673, 2005.

[55] P. Roy, J.J. Aucouturier, F. Pachet, and A. Beurivé. *Exploiting the Tradeoff Between Precision and Cpu-Time to Speed Up Nearest Neighbor Search.* ISMIR, 230-237, 2005.

[56] M. Levy, and M. Sandler. *Lightweight measures for timbral similarity of musical audio.* Proceedings of the 1st ACM workshop on Audio and music computing multimedia, 27-36, 2006.

[57] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. *Boostmap: A method for efficient approximate similarity rankings.* Proceedings of the 2004 IEEE Computer Society Conference, 2:II-268, 2004.

[58] P. Cano, M. Kaltenbrunner, F. Gouyon, and E. Batlle. *On the use of FastMap for Audio Retrieval and Browsing.* ISMIR, 2002

[59] C. Faloutsos, and K.I. Lin. *FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets.* Proceedings of the 1995 ACM SIGMOD international conference on management of data, 24(2):163-174, 1995.

[60] T. Pohle, D. Schnitzer, M. Schedl, P. Knees, and G. Widmer. *On Rhythm and General Music Similarity.* ISMIR, p. 525-530, 2009.

[61] T. Li, M. Ogihara, and G. Tzanetakis. *Music Data Mining.* CRC Press, p. 95, 2011.

[62] M. Schedl. *nowplaying Madonna: a large-scale evaluation on estimating similarities between music artists and between movies from microblogs.* Information retrieval, 15(3-4):183-217, 2012.

[63] F. Pachet, G. Westermann, and D. Laigre. *Musical data mining for electronic music distribution.* Web Delivering of Music, Proceedings, First International Conference, p.101-106, 2001.

[64] M. Zadel, and I. Fujinaga. *Web Services for Music Information Retrieval.* Proceedings of the 5th International Symposium on Music Information Retrieval (ISMIR 2004), 2004.

[65] W.W. Cohen, and W. Fan. *Web-Collaborative Filtering: Recommending Music by Crawling the Web.* WWW9 / Computer Networks, 33(1-6):685-698, 2000.

[66] E. Zangerle, W. Gassler, and G. Specht. *Exploiting Twitter's Collective Knowledge for Music Recommendations.* Proceedings of the 21st International World Wide Web Conference: Making Sense of Microposts, p.14-17, 2012.

[67] M. Schedl, D. Hauger, and J. Urbano. *Harvesting microblogs for contextual music similarity estimation - a co-occurence-based framework.* Multimedia Systems, 2013.

[68] M. Schedl, T. Pohle, P. Knees, and G. Widmer. *Exploring the Music Similarity Space on the Web.* ACM Transactions on Information Systems, 29(3), 2011.

[69] S. Davis, and P. Mermelstein. *Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences.* IEEE Transactions on Acoustics, Speech and Signal Processing, 28(4):357-366, 1980.

[70] M.E.P. Davies, and M.D. Plumbey. *Context-dependent beat tracking of musical audio.* IEEE Transactions on Audio, Speech, and Language Processing, 15(3):1009-1020, 2007.

[71] *Automatic audio segmentation using a measure of audio novelty.* IEEE International Conference on Multimedia and Expo, 1:452-455, 2000.

[72] D. Bogdanov, J. Serrà, N. Wack, P. Herrera, and X. Serra. *Unifying low-level and high-level music similarity measures.* IEEE Transactions on Multimedia, 13(4):687-701, 2011.

[73] S. Stober. *Adaptive methods for user-centered organization of music collections.* Doctoral dissertation, Magdeburg, Universitat, Diss., 2011.

[74] Y. Rubner, C. Tomasi, and L.J. Guibas. *The earth mover's distance as a metric for image retrieval.* International journal of computer vision, 40(2):99-121, 2000.

[75] H. Ling, and K. Okada. *An efficient earth mover's distance algorithm for robust histogram comparison.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 29(5):840-853, 2007.

[76] B. Logan, and A. Salomon. *A Music Similarity Function Based on Signal Analysis.* ICME, 2001.

[77] M.I. Mandel, and D.P. Ellis. *Song-level features and support vector machines for music classification.* ISMIR 2005: 6th International Conference on Music Information Retrieval: Proceedings: Variation 2: Queen Mary, University of London Goldsmiths College, p. 594-599, 2005.

[78] M. Sanderson. *Test collection based evaluation of information retrieval systems.* Foundations and Trends in Information Retrieval, 4(4):247–375, 2010.

[79] D. K. Harman. *Information retrieval evaluation.* Synthesis Lectures on Information Concepts, Retrieval, and Services, 3(2):1–119, 2011.

[80] E. M. Voorhees and D. K. Harman. *TREC: Experiment and Evaluation in Information Retrieval.* MIT Press, 2005.

[81] M. Braschler and C. Peters. *Cross-language evaluation forum: Objectives, results, achievements.* Information retrieval, 7(1-2):7–31, 2004.

[82] C. N. Silla Jr, A. L. Koerich, and C. A. Kaestner. *The latin music database.* Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008), p.451–456, 2008.

[83] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere. *The million song dataset.* Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011), 2011.

[84] B. Fields. *Contextualize Your Listening: The Playlist as Recommendation Engine.* Ph.D. Dissertation. Department of Computing Goldsmiths, University of London, London, 2011.

[85] B. McFee, and G.R.G. Lanckriet. *The natural language of playlists.* Proceedings of ISMIR:537–542, 2011.

[86] L. Barrington, R. Oda, and G.R.G. Lanckriet. *Smarter than genius? Human evaluation of music recommender systems.* Proceedings of ISMIR, 357–362, 2009.

[87] P. Cremonesi, F. Garzotto, S. Negro, A.V. Papadopoulos, and R. Turrin. *Looking for "good" recommendations: A comparative evaluation of recommender systems.* Proceedings of INTERACT, 152–168, 2011.

[88] M. Slaney and W. White. *Measuring playlist diversity for recommendation systems.* Proceedings of AMCMM, 77–82. 2006.

[89] J.H. Lee, B. Bare, and G. Meek. *How similar is too similar? Exploring users' perceptions of similarity in playlist evaluation.* Proceedings of ISMIR, 109–114, 2011.