**UNIVERSITÀ DEGLI STUDI DI PADOVA**

**DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE**

TESI DI LAUREA

# GRAPH BASED REPRESENTATION OF THE MUSIC SYMBOLIC LEVEL

***A MUSIC INFORMATION RETRIEVAL APPLICATION***

**Laureando**: *Federico Simonetta*

**Relatore**: *Prof. Antonio Rodà*

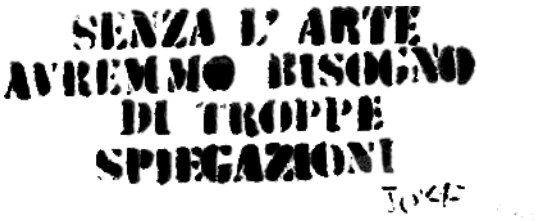
**Corso di Laurea Magistrale in Ingegneria Informatica**

9 aprile 2018

Anno Accademico 2017/2018

# Abstract

In this work, a novel music symbolic level representation system is described. The approach has been tested in two music information retrieval tasks, namely melodic similarity and genre classification. The proposed representation system is graph-based and could theoretically be able to represent music both from an horizontal (contrapuntal) and from a vertical (harmonic) point of view. It could also include relationships between internal variations of motifs and themes. Moreover, a new large dataset consisting of more than 5000 leadsheets is presented, with meta informations taken from different web databases, including author information, year of first performance, lyrics, various features, genre and stylistic tags.




*Che differenza pensate vi sia fra coloro che nella caverna di Platone contemplano le ombre e le immagini delle varie cose, senza desideri, paghi della propria condizione, e il sapiente che, uscito dalla caverna, vede le cose vere?*

e.r.

# Acknowledgments

Ringraziare è sempre una cosa difficile e ancora più difficile è farlo su un pezzo di carta stampata che verrà messo agli atti e poi dimenticato tra scaffali insieme ad altri ringraziamenti, perlopiù tutti uguali, senza affetti nè emozioni.

I ringraziamenti per il lavoro svolto vanno ovviamente al prof. Antonio Rodà, che mi ha pazientemente assistito nello svolgimento della tesi, e al prof. Nicola Orio, che ha suggerito preziosi consigli per l'articolo che ne è scaturito. In generale ringrazio chi mi ha aiutato nello svolgimento dei miei studi, sia informatici che musicali, tutti gli insegnanti, i docenti e i compagni di studio di tutti questi anni. Un ringraziamento particolare va a Gian-Luca che in questi ultimi mesi mi ha offerto l'unico momento in cui poter davvero tornare a immaginare, da un piccolo salotto sperduto nella periferia di Padova, un mondo vivo, fatto di emozioni e di colori, ma anche a tutti gli insegnanti che mi hanno guidato fino a qui, Maurizio, Ugo, Fabio, Franco e Patrizia e Sandro che nonostante le mie convinzioni materialistiche della vita sogno che stiano leggendo queste parole, sono state proprio le loro parole a farmi capire che la mia strada aveva qualcosa a che fare con la musica, gliene sarò sempre infinitamente grato.

Ci sono tutti gli amici, vecchi e nuovi, i parenti, gli FDP, il Ruz, studx, il coord e radio. Ognuno ha contribuito, seppur in minima parte, a questo lavoro. Avevo scritto dei ringraziamenti bellissimi ma poi eravate troppi e la commozione una montagna.

Oggi, mentre scrivo a Pavia, vorrei essere a Padova ma ieri mentre facevo le valigie a Padova avrei voluto essere a Pavia. La soluzione può essere una sola: inventare il teletrasporto.

28 febbraio 2018
Pavia

# Prefazione

In questo lavoro viene descritto un nuovo paradigma per la rappresentazione del livello simbolico musicale, orientato ad applicazioni pratiche nell'ambito dell'informatica musicale. Il progetto nasce da un articolo di quasi dieci anni fa' in cui Orio e Rodà descrivevano un nuovo approccio per effettuare riduzioni musicali in un modo computazionalmente sostenibile. Partendo da qui, ho approfondito l'attuale stato dell'arte per la rappresentazione dell'informazione simbolica della musica e ho sviluppato un paradigma teorico che vorrebbe riuscire ad includere gli approcci più interessanti in uso ad oggi e, contemporaneamente, a sorpassare la tradizionale dualità orizzontale-verticale – o armonia-contrappunto – per meglio rappresentare le informazioni musicali in modo consono ai più svariati tipi di applicazioni informatiche.

Nel primo capitolo viene fornita un'introduzione generale allo studio del livello simbolico della musica, nonché una proposta per un modello di comunicazione centrato sugli utenti, con la speranza che possa risultare utile alla comunità dell'informatica musicale e che la ricerca presti più attenzione in futuro alla nostra cognizione astratta della musica. Successivamente, introduco brevemente le rappresentazioni più usate per il livello simbolico, da quelli basati sui valori MIDI delle altezze – pitches – a quelli basati sulle teorie neo-riemanniane e sul Tonnetz, fino a quelli basati su grafi e riduzioni. Inoltre, ho anche tentato di fornire una semplice categorizzazione della grande varietà di sistemi esistenti in letteratura.

Nel secondo capitolo ho tentato di formalizzare il mio approccio capace di rappresentare relazioni complesse sia in musica monodica sia polifonica, giustificando le principali scelte effettuate e descrivendo le linee generali dell'algoritmo utilizzato negli esperimenti.

Nel terzo capitolo viene fornita la descrizione dettagliata del design del software e dell'implementazione. Vegono descritti i miglioramenti apportati all'algoritmo di segmentazione LBDM nonché le principali problematiche riscontrate durante l'implementazione. Probabilmente, chi volese continuare questo lavoro dovrebbe prestare attenzione a questo capitolo perché la grande eterogeneità degli algoritmi e degli strumenti utilizzati mi hanno impedito, in questa prima fase, di approfondire molti aspetti che generavano problemi.

Nel capitolo 4 viene descritto il dataset utilizzato e il procedimento impiegato per la sua costruzione, che ha portato ad un grande database di più di 5.000 canzoni basato su un vecchio archivio proveniente da Wikifonia. Ho raccolto varie informazioni tramite web mining e ho ripulito l'archivio da quei brani che sembravano più corrotti o inutilizzabili dall'algoritmo.

Infine nel quinto capitolo mostro i risultati e li discuto, fornendo alcuni suggerimenti per indicare quelle che mi sembrano le direzioni che potenzialmente possono apportare i maggiori miglioramenti all'approccio. Molte strade sono state aperte, ma poche sono state chiuse.

Spero che qualcuno in futuro sarà ancora interessato negli studi sul livello simbolico della musica.

# Preface

In this work I describe a new paradigm for the symbolic level of music representation to be used in practical music computing applications. The work was born from a paper of about ten years ago in which Orio and Rodà described a new approach to perform music reductions in a computationally sustainable way. Starting from here, I investigated the state-of-art of music symbolic representations and I developed a theoretical paradigm which aims to include the most interesting approach in use today and to overcome the horizontal-vertical duality to better represent musical informations for various music computing applications.

In chapter 1, a general introduction to the study of the music symbolic level is given, together with a proposal for a music communication model, user centered, in the hoping that it could be useful to the music computing community, and that researchers will give more attention to our abstract cognition of music in the near future. Then, I present an introductory explanation of the most used music symbolic level representations, from MIDI value based to *Tonnetz* based, until the ones based on graphs and reductions. Also, I tried to collect and categorize references to different types of representations.

In chapter 2, I try to make a formalization of my approach to represent complex relationships in monodic or polyphonic works, explaining the most basic choices and giving an outline of the algorithm used in the experiments.

In chapter 3, design and implementation details are given. I explain the improvements made to the *LBDM* algorithm used in the segmentation stage and I describe the main issues found during the implementation. Maybe, who would to continue this work should read carefully this chapter because the great heterogeneity of the algorithms and tools used in the work prevented me from studying further many aspects that generated problems.

In chapter 4, I describe the construction and the features of a new huge dataset for leadsheet elaboration, based on the old *Wikifonia* archive. I collected several informations through a web mining approach.

Finally, in chapter 5 results are described and discussed, together with some suggestion to whoever wants to continue this project. Many roads have been opened but few were closed. I hope that someone will still be interested in symbolic music studies.

# Contents

# 1

# Introduction

*Somewhere underneath, very deeply, there's a common place in our spirit where the beauty of mathematics and the beauty of music meet. But they don't meet on the level of algorithms or making music by calculation. It's much lower, much deeper – or much higher, you could say.*

G. Ligeti

## 1.1 Motivations for a new music symbolic representation paradigm

### 1.1.1 Physical vs symbolic level

Since first theoretical studies in computer science, computers were associated to the elaboration of pitch sounds, harmony and music composition (Menabrea and Lovelace, 1843). In the XX sec., many composers, scientists and musicologists proposed different models to formalize musical informations so that it could be processed by computers. In the last decades, researchers in this field of study have been mainly engaged in audio signal elaboration, mostly geared to enterprise applications. For instance, at January 2018, by querying "symbolic music" on Google Scholar and filtering out documents older than 2008 you get 148.000 results, while querying "music signal processing" with the same filter returns 356.000 results; a query "music score processing" returns 105.000 pages, while "audio processing" retrieves 1.230.000 documents.

In contrast to this trend, many studies have shown that humans have an abstract cognition of music (Deutsch, Thompson, Honing, and McAdams, 2013): we can recognize a music played on different instruments, with different timings, intensity changes, various metronome markings, tonalities, tunings, background noises and so on. This characteristic of our music cognition process seems to be culture and age invariant. Two key concepts in this approach

are important: *categorical perception* and *phonemic restoration* (Aiello, 1994; Schön, Akiva-Kabiri, and Vecchi, 2007). The former refers to the fact that we tend to perceive stimulus in categories; the quality of categories is not culture invariant – e.g. Western musical intervals differ from Indian śruti. The latter is the effect why we understand a musical tune also if we do not listen some of its notes. This type of study has underlined the importance of taking into account the abstraction level while modeling music cognition. Modern cognitive psychology suggests that the study of the music symbolic level may be very interesting for computational elaboration of music, also in the enterprise sector.

### 1.1.2 Music symbolic level: a definition

What is *music symbolic level* (MSL)? In simple terms, it is an abstract description of the audio signals, which encores the most significant details about what we call *music*. According to Vinet (2003)

> *the symbolic representation is content-aware and describes events in relation to formalized concepts of music (music theory), whereas the signal representation is a blind, content-unaware representation, thus adapted to transmit any, non-musical kind of sound, and even non-audible signals; even digitized through sampling, the signal representation appears as a continuous flow of information, both in time and amplitude, whereas the symbolic representation accounts for discrete events, both in time and in possible event states*

Actually, MSL is an attempt to represent the abstraction level that we use in listening to music and processing it in our brain. We can reasonably state that we like music *because* of the abstraction we apply to the physical stimulus that our auditory system receives. Addressing our studies to the MSL and its relationship to our cognition could really improve any application in which users opinion matters. As Widmer (2017) states, Music Information Retrieval (MIR) systems in the near future need to put attention on the user as an active listener. Doing this needs, iner alia, a model of our cognitive abstraction of music.

### 1.1.3 What does musicology think about it?

MSL is important not only in MIR tasks. For instance, it is used everyday by any musician in the world, sometime in the form of musical score, sometime not. Every music that we listen to is created using some form of MSL representation, commonly, but not necessarily, a musical score, often written or digitized.

Indeed, MSL is studied from very longer time than physical level: the most ancient notation testimony dates back to the I century CE (Mathiesen, Conomos, Leotsakos, Chianis, and Brandl, 2001) and it is known that Western musical notation has had a gradual evolution (Hiley and Szendrei, 2014). This remark forwarded music semiologist to track connections between musical style, practice, notation and social changes (Middleton, 1990; Meyer, 1989). This is a nice example of how musical scores may reflect human musical thinking change. Anyway, more than one musicologist propose to reduce the prominence of musical scores

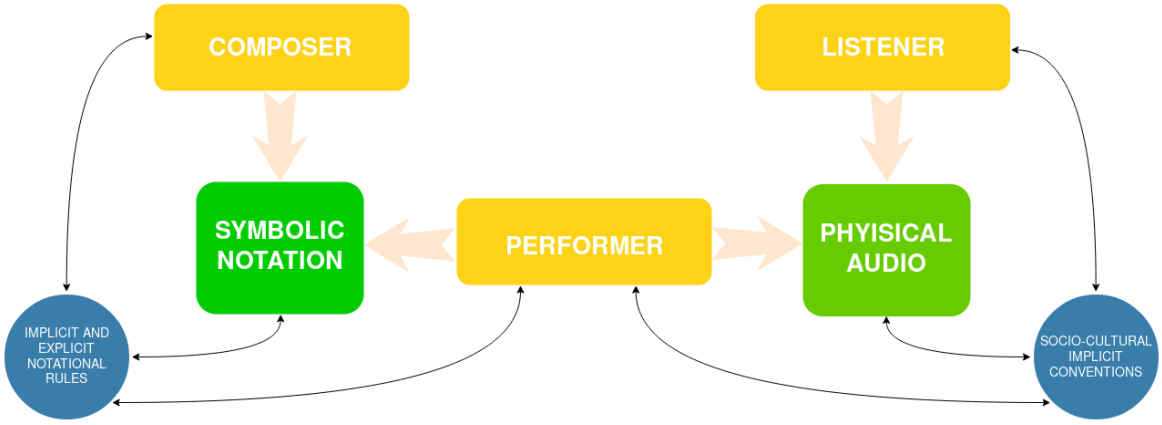



**Figure 1.1:** ***Giacobazzi's musical communication model****: listener is an active player of the communication process, together with performer and composer. Arrows start from a subject and point toward an object of a creative or cognitive process in which the subject uses actively some his own knowledge. Campbell and Heller (1981), Nattiez and Ellis (1989), and Molino, Underwood, and Ayrey (1990) can be thought as degenerate forms of this model.*

in future studies, but this is said with regard to the musicological studies of the formalist tradition (Fabbri, 2014) and we do not care.

To understand to what extent we should take into account the MSL in a practical application, we found useful proposing the music communication schema by Michele Giacobazzi[1]. He – starting from Campbell and Heller (1981), Nattiez and Ellis (1989), and Molino, Underwood, and Ayrey (1990) – proposed a model of music communication involving composer, listener, performer and MSL. It is shown in Figure 1.1.

According to Nattiez (1990), listener perception differs from listener to listener because he is an active player in the communication process: using his own codes, he understand "something" that is not necessarily what composer or performer coded in the message. Campbell and Heller (1981) proposed a different model in which performer appears to be an important figure in that he is in charge of decoding composer message through implicit notational code rules and to transform them in the physical audio level. Giacobazzi's model of Figure 1.1 merge these two different views. In contrast to other models proposed for MIR tasks, like the communication chain by Richard Moore (Serra, 2002), Richard Middleton's and Philip Tagg's models (Inskip, MacFarlane, and Rafferty, 2008; Tagg, 2012), Giacobazzi's model is focused on the actors of the communication, rather than on the medium, and expresses the idea of an active role by all players.

Actually, in some situations composer is at the same time listener and often performer too. Moreover, notation could not exist and, even if more rarely, neither the audio. Therefore, we think that square edges should be intended as fluid bounds.

[1] In report realized by Michele Giacobazzi while studying "Scienze e tecniche per la comunicazione musicale at the University of Milan, 2006, https://www.slideshare.net/Smirne/i-modelli-della-comunicazione-musicale. The report is available by contacting thesis authors.

### 1.1.4 Why music computing should care about it?

It is now easier to see why the study of listener perception only based on audio can just give a partial view of the communication process and of the internal cognitive process, as suggested by psychological studies. By example, a system which aims to understand the emotional response of the user, should also consider his abstract representation of the music and maybe its relations with the one of the performer and, maybe, of the composer. Studying performance should take into account the MSL, the listener *perception* and *reaction* (due to – but not only – *socio-cultural implicit conventions*), the composer's *implicit notation rules* and their performer understanding. A completely automatic system for algorithmic composition, instead, should take into account the whole process, and this is a very difficult challenge; this is why, very often, music composition algorithms are polarized on particular styles, objectives or situations decided by the developer choices.

Since at now we do not have a total understanding of the musical communication process, we need to restrict a little bit the scope of action of computational systems in the model. We could think that computational applications could look at it from left (composer) or from right (listener). We propose to subdivide today music computing applications in four macro-objectives, reducible to two areas:

- facilitation in *listening* to a musical work (user) and/or in its *execution* (performer):
  - Recommendation systems
  - Live concerts enhancement
  - Cultural heritage applications
  - Acoustic enhancement of listening
  - Automatic (and expressive) score execution
  - Automatic score transcription
  - Most of Music Information Retrieval tasks
- facilitation in *studying* (musicologist) and/or in *creating* (composer) music:
  - Musicological studies
  - Cultural heritage applications
  - Automatic composition
  - Educational systems
  - Facilitation of composers activity (notation software etc.)
  - Automatic arrangers (real-time or off-line)
  - Expressive score execution

In both the macro-areas – user oriented – a good MSL representation could bring many benefits: concerning the second area, MSL is the direct product of music creation activities and it is almost a must-have; while concerning the first area, MSL could enhance the cognitive models of our music understanding, so that user or performer reactions can be fully

elaborated.

Anyway, to our knowledge there is no MSL representation satisfactory for all the above tasks, therefore new general MSL models are needed. In this thesis we are going to describe a new MSL representation paradigm, which try to take into account most of the tasks named above.

## 1.2 Symbolic music representation: related works

### 1.2.1 Classification of Symbolic Music Representation Systems

Giving an extensive survey of the main *music symbolic representation systems* (from now on MSR) is quite difficult. For our purpose, we can divide MSRs in four broad groups based on their principal purpose:

- *File* representations like *MIDI* or *MusicXML*;
- *Cognitive* representations, which try to describe our music cognition in general terms, these include frameworks like *GTTM* (F. Lerdahl and Jackendoff, 1985), *Implication-Realisation* theory (Narmour, 1992; E Glenn Schellenberg, 1997) and others (e.g. Deutsch and Feroe (1981), Temperley (2007), Marsden (2000), and Camurri, Frixione, and Innocenti (1994)). An extensive description of some of these models is given in Wiggins, Pearce, Müllensiefen, et al. (2009);
- *Analytical* representations, which try to emphasize particular aspects of the theoretical underlying structures, such as classical, schenkerian, Meyer's and neo-Riemann's theories, the ones based on *pitch class sets* and others (e.g. Chew (2014), Tymoczko (2012), Callender, Quinn, and Tymoczko (2008), and Cohn (1997));
- *Computational* representations, whose objective is to bring the benefits of the previous to computational music analysis. Actually, some of the already named *cognitive* MSRs are thought to be *computational*, while other *analytical* and *cognitive* models have been implemented in computer algorithm versions. A good point to start can be found in Meredith (2016), but many others computational MSRs exist (e.g. Marsden (2005), Marsden (2010b), Widmer and Goebl (2004), and Chew (2014));

In the following we are going to focus on the *computational* MSRs, describing the most interesting to the purpose of a new MSR aimed at Music Information Retrieval (MIR) and computational musicology. Hence, we will just look at those extensively used in computational tasks or graph based. Also, we will limit our survey to those of which we found tangible results in some practical task (see subsection 1.1.4). Surely, finding a measure of the effectiveness of a MSR is a hard challenge, nevertheless it is a necessary information for practical applications design. In this thesis, our MSR will be tested with typical MIR tasks.

We will also describe the two major MSL file formats, MusicXML and Standard Midi File.

Anyway, a complete description of these models is very hard because often researchers refer to MSL and MSR with various different nomenclatures or do not stress this part of their work, so that it is difficult to find related papers.

## 1.2.2 File formats for MSRs

### 1.2.2.1 Standard Midi File

Standard Midi File (SMF) is a file format standardized the first time in 1988 (MMA, 1996). It is the most widely used MSL file format and is thought to represent music symbolic level in live performance. In fact, MIDI was born as a communication protocol aiming to make easier the contemporaneous usage of different musical devices, such as in live concerts. The SMF adds the timing informations to the messages of the MIDI protocol. In SMF, the MSL is represented as a sequence of events, each characterized by a *time stamp*, representing the time interval that occurs from the preceding event. In addition, it is added the duration of a quarter note as unity of measure.

To our purpose, we just say that in MIDI protocol and in SMF note pitch is represented by a number ranging from 0 to 127 where pitch number 0 is the $C_{-1}$ in scientific pitch notation, pitch number 1 is $C\sharp_{-1} = D\flat_{-1}$, and so on. This means that SMF has no enharmonic recognition ability out-of-the-box and that it must be implemented trough some *pitch spelling* algorithm (most famous are Chew and Chen (2005), Meredith (2006), and Cambouropoulos (2003)). Moreover, SMF do not include high level informations such as musical structure, lyrics, music dynamics. It instead includes several controls – such as *velocity*, *volume*, *portamento* and others – intended as modifications of fundamentals sound events. Controls can be in binary or continuous value – ranging from 0 to 127. SMF is very suitable for storing performance executions.

### 1.2.2.2 MusicXML

MusicXML is a file format based on XML and born to create a score representation format with interoperability purpose (MakeMusic and Good, 2017). MusicXML is the state-of-art in portable music file format. It has definitely superseded old kern and musedata files and it is now a standard for music score representation.

MusicXML brings everything needed to represent any type of traditional musical scores and it is used in many advanced algorithms in MIR tasks. Unfortunately, MIREX (Music Information Retrieval Evaluation eXchange), the most important algorithmic challenge in MIR fields, uses only SMF in symbolic music tasks. This can be a restriction for algorithms that could take advantage from more abstract data.

MusicXML can represent informations about almost everything a classical music score may store, including meta informations, lyrics, chord annotations, dynamics, refrains, *da capo*, etc.

## 1.2.3 Common MSR

### 1.2.3.1 MSL features

The most straightforward way to represent music for computational purpose is of course by emulating other computer science fields and by extracting some relevant features from a discrete signal representing the music symbolic level. Probably, the spread of this approach among music computing applications is due to the fact that music listening and understanding is also accessible without deep background studies, while this is more difficult for

computer science; this leads many computer scientists to be keen on music and few musician to be interested in computers.

We think that this approach threatens to degrade the computer ability to understand music if it is not accompanied by a careful choice of the feature space. However, it is a fact that this method can give satisfactory results with faster times and easier learning curve. Another big limitation by using this approach is that it is difficult to represent features changes in time – see Widmer (2017) about the importance of music evolution in time.

Generally, these algorithms make use of the MIDI pitch notes and look for some relevant features to extract. A common method is by using pitch histograms (Tzanetakis, Ermolinskyi, and Cook, 2003). A nice collection of symbolic music features is in McKay and Fujinaga (2006).

#### 1.2.3.2 Interval and chord representation

As already said, most of the software that involves symbolic music use MIDI note number representation. Others use something similar but with pitch class information instead of the absolute number. Actually, they are not only based on MIDI pitch numbers, but they often use some more abstract representation, usually a relative interval. By example, only the first melody note is expressed in pitch number, while the others are expressed through the difference between their MIDI number and the preceding. This approach let the analysis to be transposition invariant, a quality very important in several tasks. Actually, in this way, algorithms are just chromatically invariant but not diatonically, that is the most common type of transposition in tonal music.

To this purpose, Cambouropoulos (1996) proposed the GPIR (General Pitch Interval Representation), a new representation paradigm in which each interval class is associated to an index. In this way, moving from $D\flat_4$ to $E_4$ is no more represented as a step of 4 but only of 2, because it is an augmented second: no matter about alterations, it counts only diatonic intervals. Another more classical approach is the *step-leap* representation which allows an *approximate search*: it consists in representing only most "significant" interval classes. Similar to this approach is also the so called *contour melody*.

Regarding harmonic description, not so many models for analysis and MIR were proposed. The majority of the algorithms which analyze music harmony adopt one of two possible approach:

- use the *Tonnetz* or some derivative form (see subsubsection 1.2.3.3)
- use a vector of notes in absolute or relative number (Cambouropoulos, 2016)

#### 1.2.3.3 Tonnetz and Spiral Array

*Tonnetz*, or *harmonic network*, is a geometric representation of musical pitch space, described the first time by Euler and adopted later by Arthur von Oettingen, Riemann and recently by neo-Riemannian theorists (Cohn, 1997) (see Figure 1.2). It has been used in MIR tasks especially for harmonic analysis (e.g. Harte, Sandler, and Gasser (2006) and Humphrey, Cho, and Bello (2012)).

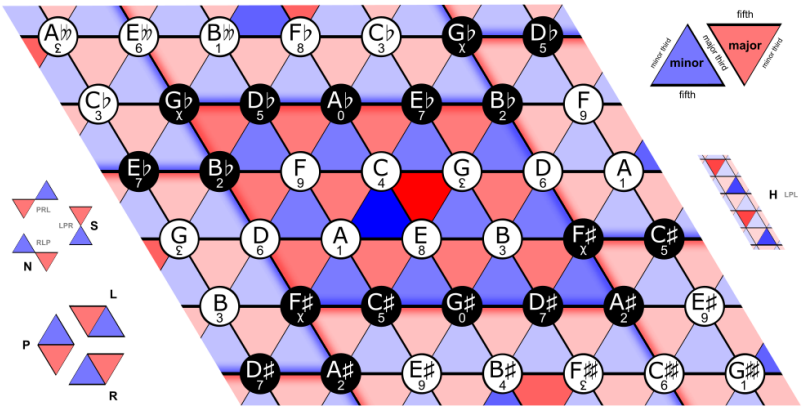



**Figure 1.2:** ***Tonnetz****: a representation of the Tonnetz space. Horizontal lines represent fifth circle, while triangles represent triads.*

Spiral Array is a *Tonnetz* based representation of music which removes some of its redundancy (see Figure 1.3). It was first proposed by Chew (2000) and then applied to several other tasks, such as key detection (Chew, 2002), music enjoyment (Chew and Francois, 2003) and pitch spelling (Chew and Chen, 2005). It has been recently defined in Chew, 2014. We are going to briefly describe it adapting our survey from *"The spiral array: An algorithm for determining key boundaries"* (Chew, 2002).

In Spiral Array, each pitch class is indexed by the number of perfect fifths separating it from an arbitrarily chosen reference pitch and it is placed in 3D space. Moving through the spiral of a quarter turn, coincides in incrementing the index. Four quarter turns places pitch classes related by major third interval in the same vertical line. Moreover, higher level musical entities (such as chords) are represented as convex combinations of pitches (lower level musical entities). Easy speaking, like a pitch is represented by a point, power chords can be represented as lines, three notes chords as 2D spaces, and so on, similarly to *Tonnentz*.

As *Tonnentz*, the Spiral Array is very fine in representing tonal harmonic relations. Whereas it has been used in several interesting applications (Chew, 2008), it becomes of quite difficult usage in non-tonal contexts and in tasks involving motivic analysis.

### 1.2.4 Graph based MSRs

MSR paradigms mentioned above have a weakness: they are not able to overcome the vertical-horizontal duality, resulting in representational systems more appropriate for motivic or contrapuntal analysis while others for harmonic studies.

A way to bypass this problem could be by using a graph based representation as we will explain in the next chapters. Therefore we are going to describe the most interesting graph based MSRs in literature.

#### 1.2.4.1 Hamanaka – Hirata – Tojo GTTM

Over the past fifteen years Hamanaka, Hirata and Tojo undertook in the implementation of the famous *Generative Theory of Tonal Music* (GTTM) (F. Lerdahl and Jackendoff, 1985), a cognitive model of tonal music language. F. Lerdahl and Jackendoff, starting from Gestalt theories and from Chomsky's influential writings, tried to describe music through generative

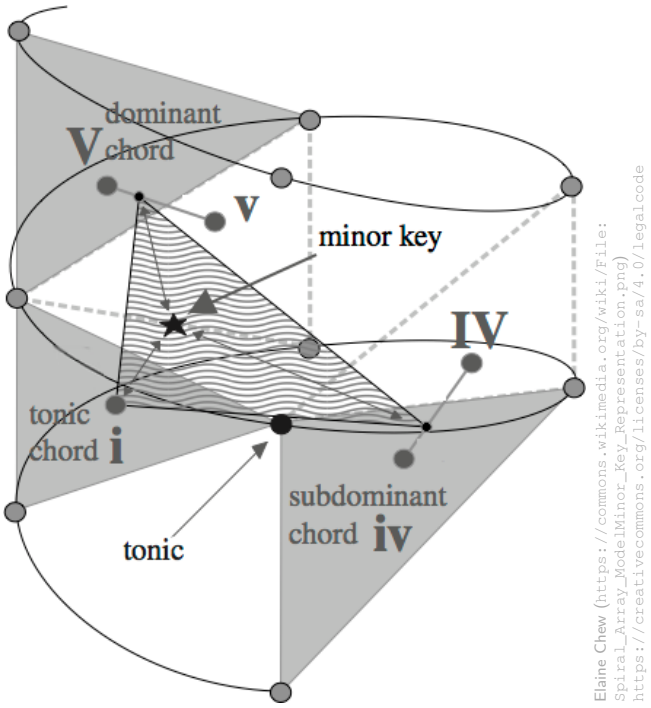



**Figure 1.3:** ***Spiral Array****: A representation of the Spiral Array with pitches, tonic and triads.*

grammars, stating at the end of the book that most of the system was "universal", that is idiom independent. We will shortly describe the work by Hamanaka, Hirata, and Tojo based on Hamanaka, Hirata, and Tojo (2016).

Basically, GTTM describes four different structural analysis: *metrical structure*, *time-span tree*, *prolongational tree* and *grouping structure*. Each of these analysis has two type of rules: *well-formedness rules* and *preference rules*. Through these rules, the GTTM deduce 4 different structure representations on the score:

- *Grouping structure*: gives a Gestalt based music description and says which segments are perceived more "unified" than others, identifying motifs, periods and sections;
- *Metrical structure*: describes the most important events in metrical sense according to strong and weak beats sequence (i.e. alike to the common metrical structure);
- *Time-span tree*: merging the previous two structure, GTTM retrieves a hierarchical structure, defining most relevant events, in a fashion resembling Schenkerian theories;
- *Prolongational tree*: while time-span tree determined the most relevant events based manly on metrical structure, this takes into account the alternation of tension and relaxation;

Since F. Lerdahl and Jackendoff did not specify an algorithm for constructing hierarchical structure but let several ambiguities in many crucial points, problems arise in its implementation. The GTTM implementation is quite complex and this is not the context to explain it further. However, Hamanaka, Hirata, and Tojo's algorithm has been applied both to similarity (Hirata, Tojo, and Hamanaka, 2013; Matsubara, Hirata, and Tojo, 2014) and educational tasks (Hamanaka, Hirata, and Tojo, 2008) with encouraging results.

#### 1.2.4.2 Marsden Schenkerian reductions

During the same period in which Hamanaka, Hirata, and Tojo worked to the GTTM implementation, Marsden worked to the implementation of an algorithm to perform a Schenkerian analysis over a short music extract. Actually, at now, it is still difficult to realize a formal Schenkerian analysis because of the computationally expansive cost of the operations (Marsden, 2011).

As for GTTM, Schenker did not explained his theory in an algorithmic way and it is hard to derive a formal procedure. Moreover, it must be said that Schenker did not intend his theory in a cognitive sense – cognitive psychology and Gestalt would have arrived later – but as a way to found a superior meaning that only genius composers are able to understand (Temperley, 2012). Schenkerian theory is based on the identification of structural relevant elements – which compose what he called *Ursatz* or *fundamental structure* – and on the assumption that all music (namely tonal music) can be seen as the development of an alternation of fundamental harmonic elements, i.e. tonic and dominant.

Aside from Schenkerian theoretical aspects, which have been criticized and at the same time extended by several musicologists, hierarchical reductions may could reveal deep relations, such as variations of a motif or a theme, that could enhance the computation of music similarity and therefore of many MIR tasks. By example, Marsden (2010a) tested his Schenkerian approach to similarity task, obtaining nice results. In any case, this algorithm still need careful insights.

#### 1.2.4.3 Rizo - Iñesta trees

An interesting graph based structure are the Rizo - Iñesta trees (Rizo Valero, Iñesta Quereda, and Moreno-Seco, 2003; Rizo Valero and Iñesta Quereda, 2002; Rizo Valero, Iñesta Quereda, and León, 2006; Rizo Valero, Iñesta Quereda, and Lemström, 2008; Rizo Valero, 2010).

They developed a MSR based mainly on the metric structure and on the bottom-up propagation of more relevant pitches according to some rule system, which vary according to the task. The level to which each note belongs is given by its duration: the longer the note is and the higher the level at which it is located.

Rizo Valero, Iñesta Quereda, and Lemström (2008) and Rizo Valero (2010) extended the same approach to polyphonic music too. This is done by building a separated tree for each voice and then merging them together with multiset labeling of pitch – in a way resembling the chord representation described in subsubsection 1.2.3.2.

The most interesting thing in Rizo - Iñesta trees is that they are specifically designed for MIR tasks, as the following others, with successful results. In particular, authors use tree topological similarity to compare different pieces of music.

#### 1.2.4.4 Pinto graphs

Pinto, Van Leuken, Demirci, Wiering, and Veltkamp (2007) proposed a graph based representation for melodic datasets. The basic idea was simple: an edge was build at every new note, following the melodic contour, leaving out the pitch class information. The graph was composed by 12 vertex – one for each note – and by $n - 1$ edges, where $n$ is the number of

notes in the melody; each edge is weighted by the occurrences of that interval.

Obviously, this approach is time independent and, although it had some benefits – it is invariant to transposition, retrogradation, inversion, etc. – it was limited to little number of melodies. Moreover, information that it stores are the same of a histogram (see subsubsection 1.2.3.2).

Pinto, Van Leuken, Demirci, Wiering, and Veltkamp used the Laplacian matrix representation, they computed the laplacian eigenvalues, sorted them by magnitude and used them as points in an euclidean space. Distance between two melodies is the euclidean distances between the laplacian spectra. They obtained nice results in a typical retrieval task.

#### 1.2.4.5 Orio - Rodà dataset graphs

Orio and Rodà (2009) suggested a different approach through which, resembling schenkerian and GTTM theories, was possible to represent an entire monophonic dataset as a graph. The algorithm is composed by four steps:

1. *Normalization*: the music is transformed in interval representation (see subsubsection 1.2.3.2) and undergoes to harmonic analysis;

2. *Reduction*: each melody is reduced by three rules until only one note remains (see below);

3. *Segmentation*: each melody is segmented and each segment inherits all the reductions;

4. *Encoding and merging*: segments and reductions are then coded in one of the method described in subsubsection 1.2.3.2; then, they are merged: since only one note remains after the reduction, more than one melody can be merged in the same graph;

At this point, melodic distance between two segments is computable as the minimum path between them – they could also have some mid reduction in common, hence minimum path do not necessarily pass through the single fundamental note.

The central part of Orio and Rodà's algorithm is the reduction algorithm, which creates a hierarchical structure similar to the one of Hamanaka, Hirata, and Tojo, Marsden and Rizo Valero and Iñesta Quereda. It uses some weight coefficients assigned to each note in the melody, then, by using a sliding window of two notes, the least heavy note according to the first coefficient is deleted in each window; if the two notes have the same weight, the second coefficient is used and so on. Namely, they used three coefficients that they called *harmonic*, *metric* and *melodic*. We propose a different but more explanatory nomenclature:

- *Functional*: a weight relative to the importance of the harmony of a note with respect to the tonality of the segment, representing the functional harmonic analysis – i.e. notes in a tonic or dominant harmony are heavier than notes in subdominant harmonies;

- *Metric*: a coefficient relative to the metric structure – see subsubsection 1.2.4.1;

- *Consonance*: a coefficient which rewards consonant notes and disadvantages dissonant ones;

A simple example of reduction procedure according to this algorithm is shown in figure fig. 3.1.

Results by this MSR were very encouraging, but the testing dataset was small and it should be submitted to a further analysis.

## 1.3 Proposals for a new paradigm

Merging all observations from the previous two sections, we suggest the following intentions and guidelines for a new MSR:

1. MSRs should be coherent with psychological discoveries;
2. It is important that MSRs are designed to be used in a context user centered, that is taking into account the active cognitive process of the final user;
3. MSRs should be useful for musicological analysis and creative activities;
4. Despite existing MSRs, it is of central importance that MSRs are able to represent music both from horizontal and vertical point of view; this would make them able to perform contrapuntal and harmonic analysis as well to better learn the texture;
5. MSRs with practical aims should also be tested with real world dataset, not only with prepared and cleaned corpora; although this is hard in the symbolic music context due to the lack of dataset and to copyright laws, this is an unavoidable passage, sooner or later;
6. Moreover, it would be nice if MSRs would provide support to standard encoding format for musical analysis, as suggested in Marsden and Rizo (2016) and Harley (2015);

# 2

# Formal definition and implementation choices

In this chapter we are going to define a new MSR paradigm. First, we will define a general structure that is algorithm independent. Then, we will derive two possible algorithms to create an MSR, one for polyphonic music and one for monophonic.

## 2.1 A general framework

### 2.1.1 Outline of a new MSR

An important feature that is missing in almost all the MSRs until now is the ability to provide a good contrapuntal description of the music texture. Instead, MSRs are quite good in detecting key changes or in harmonic analysis, as well as in describing some abstract knowledge of our music perception. Musicology and composers spend a lot of time to understand the complex relationships which characterize music also at a contrapuntal and motivic elaboration level. Moreover, encapsulating this type of structural features would pave the way for new strategies for genre classification, recommendation systems, expressive music performance and so on. Since we aim to design an MSR satisfactory for the most part of music computing applications, we think that this is an aspect that cannot be postponed anymore (see subsection 1.1.4).

The question which arise is: *how to represent music in its contrapuntal relationships, motifs, themes and variations?* Surely, our MSR should "know" the concept of motif – or theme, or subject, depending on the context. This knowledge could be extracted manually or by some automatic algorithm – this side will be analyzed in section 2.2. At now, we observe that a fine structural description of the music piece could be useful to represent it in a MSR. Also, each segment could be described with one of the already existing MSRs, so that its intrinsic harmonic informations could be stored as well. Instead, we could add informations about the mentioned relationships, improving the texture representation. Something similar has already been suggested by Marsden (2001), Marsden (2004), and Marsden (2005).

Another feature we aim to achieve is a cognitive based approach (see subsection 1.1.1). To this purpose, a new MSR should take into account at least:

- Transposition invariance
- Duration invariance
- Timing variation invariance – in case of performance base representations such as MIDI
- Intensity changes invariance
- Little variations invariance
- Pitch hierarchy – according to the cultural context
- Consonance-dissonance – according to the cultural context

Actually, from the above list, one should decide what is needed to his own purpose: by example, an expressive performance algorithm could be enhanced by the knowledge of a transposition – e.g. performers often create particular dynamics to emphasize a subject growing up from low frequencies to higher – while a recommendation system may does not want this information because people usually do not care about it – the main distinction here is between professional musicians and not musical trained people, for a review see E. Glenn Schellenberg and Weiss (2013). Maybe, to create a general purpose MSR, these relations could be introduced in a way which allow to take them into consideration or not. Note that some of the above items are also important for the contrapuntal texture discussed above.

Finally, we think that the time information is really the most significant feature of music since it permeates all its aspects. Removing it means losing too much crucial knowledge for an adequate understanding of music.

In the next paragraph we try to give a formalization of these concepts.

### 2.1.2 Formalization

Let start with some useful definition.

**Definition 1: Musical notes universe**

We call $\mathscr{N}$ the *universe of musical notes*. Each note is a set of four elements:

- *main name*, a value $\in \{ A, B, C, D, E, F, G \}$
- *alteration*, a value $\in \{ -2, -1, 0, +1, +2 \}$
- *octave*, a value $\in \mathbb{Z}$
- *duration*, a value $\in \mathbb{R}$ corresponding to the note duration expressed in quarters

**Definition 2: Segment**

We call *segment* an ordered sequence of notes $a_i : a_i \in \mathscr{N}$.

We remark that a note is not a vector because we do not know how to define a commutative sum between two notes. The same is for segments.

**Definition 3: Operations between segments**

Given a segment $a$ and a segment $b$, the following operations are defined:

- *membership*: $a \in b \Leftrightarrow \forall\ n_i \in a, n_i \in b$
- *equality*: $a = b \Leftrightarrow a \in b \wedge b \in a$
- *length*: $|a|$ is the number of notes in $a$
- *sum*: $a + b$ is a new segment given by juxtaposing $b$ to $a$
- *intersection*: $a \cap b$ is a new segment composed by the notes $n : n \in a \wedge n \in b$
- *union*: $a \cup b$ is a new segment given by the removal from $b$ of the starting items which are in common with the end of $a$ and by the juxtaposition of the remaining part of $b$ to $a$. Therefore $a \in a \cup b\ \ \wedge b \in a \cup b$. Also $a + b = a \cup b \Leftrightarrow a \cap b = \varnothing$ and $a \in b \rightarrow a \cup b = b$.
- *duration*: $d(a)$, the sum of the durations of the notes of segment $a$

**Definition 4: Segmentation of a segment**

We call *segmentation* of a segment $b \in \mathcal{N}^l : |b| = l$ a set $E \subset \mathcal{N}^{k \times h} : b \notin E$ of $h$ segments with length $k$ obtained through a function $e : \mathcal{N}^l \mapsto \mathcal{N}^{k \times h}$ called *segmentation function*.

**Definition 5: Parent function**

We call *parent function* a function $f : \mathcal{N}^k \mapsto \mathcal{N}^l$ which maps each segment $a$ into its *parent* segment $b$, so that $a \in e(b)$.

**Definition 6: Set of parent relations**

Given a segment $b$ and its segmentation $E = \{\, a_0, a_1, \ldots, a_h \,\}$ then $p(a_i) = b\ \forall i \leq h$. Therefore we can define the set of parent relations that is the set of ordered pairs $\{\, (a_0, b), (a_1, b), \ldots \mid i \leq h \,\}$. We call this set $\mathcal{P}$.

**Definition 7: Graph of a music piece**

Let $e$ be a segmentation function and $\mathcal{A}$ be the union of all the recursive segmentations obtained through $e$ starting from a music piece $\mathcal{S}$. Then $\mathcal{A}$ can be intended as the set of nodes of a graph in which the set of edges is $\mathcal{P}$. We call $G = (\mathcal{A}, \mathcal{P})$ *graph of the music piece* $\mathcal{S}$ . An example is shown in Figure 2.1.

Note that if a music piece $\mathcal{S}$ does not have pattern repetitions, then its graph is a tree, and it is the hierarchical structure of the piece. Since by definition a set requires all its items to be distinct, in general the graph of a music piece is not a tree. Instead, we can observe that the union of the segments in a segmentation set *is* the parent of the items. Therefore, at any point of the recursive segmentation procedure, we can reconstruct the initial segment. Also, the union of all the segments in the graph $G$ is exactly the music piece $\mathcal{S}$.

At now, we have generalized the typical hierarchical structure, without adding informations to it. Actually, we could always build a tree by considering each segment distinct based on its parent, so that at every segmentation a new set of segments is returned. Then, we could add edges to the graph $G$ to represent *equalities* relationships. These structure could then be used in some useful tasks, like formal analysis, genre detection, and so on.

In a similar way, we could define any complex relationship of which in subsection 2.1.1 by adding special edges to the graph. In this way we are representing contrapuntal relationships in a melody. For polyphonic materials, the same approach could work, by simply building separated graphs for each voice and then adding edges between each voice graph, or, maybe, by using the above formalism and keeping in each node of graph $G$ a list of pairs $(t, v)$, where $t$ is the time and $v$ is the voice in which the segment appears.

We still have to represent cognitive and harmonic informations. Concerning the first, we can store them in arcs between nodes – e.g. transposition, retrogradation, inversion, different levels of similarity – or by simply using one of the MSR described in section 1.2, depending on the aim. About the harmonic informations, we think that the most practical way is to use some of the MSR of which in section 1.2, such as the Spiral Array. Maybe, if a hierarchic MSR is chosen, other segments can be added, of which each one can have the same above mentioned relationships with all the other segments plus an edge representing the reduction. This, as suggested by Marsden (2010a) and Orio and Rodà (2009) could also

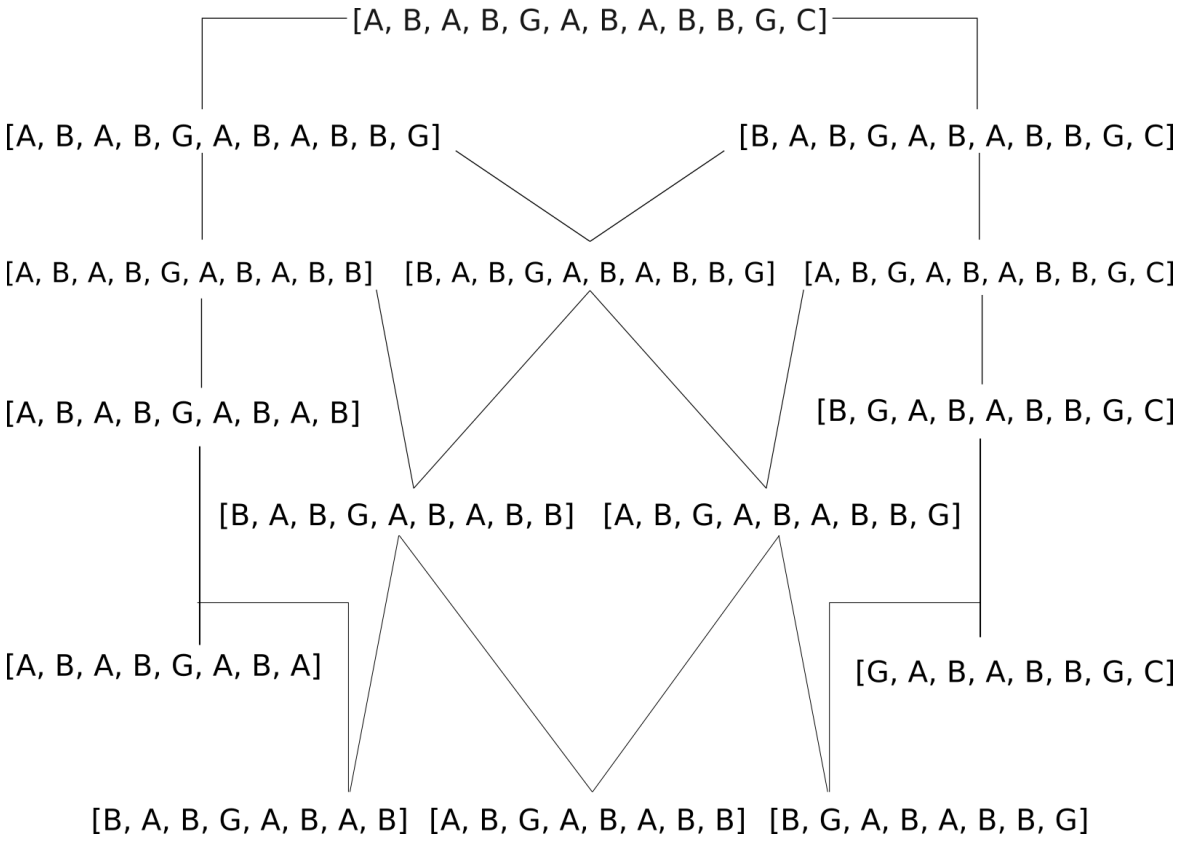



**Figure 2.1:** ***Recursive segmentation****: in this example, the* ***segmentation function*** *used was* $e(b) = a : |a| = |b| - 1$*, that is the extracted segments are the two subsequences with one note less than the parent. The results is a graph.*

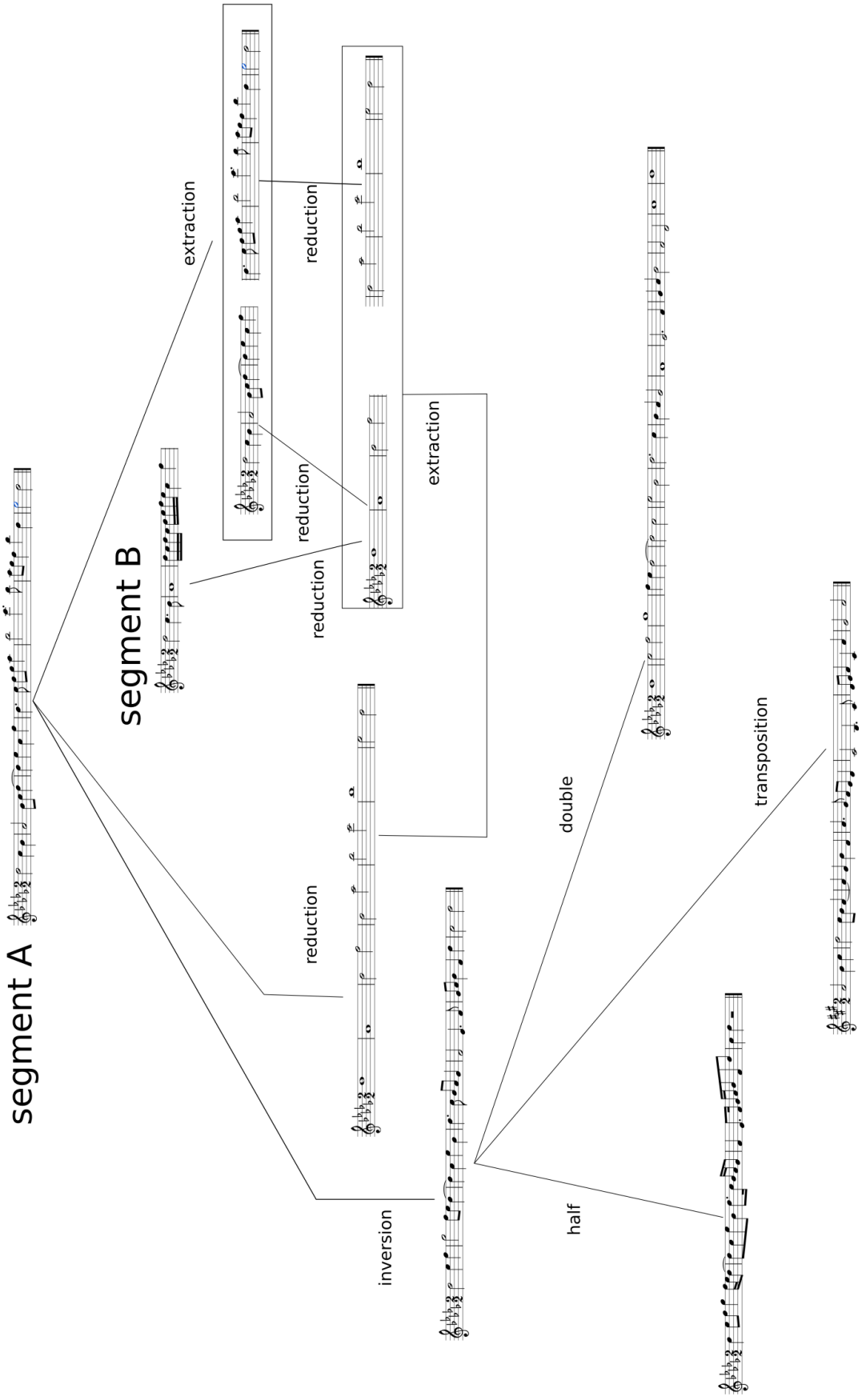



**Figure 2.2:** ***An example of graph****: each edge is annotated with the relative operation. Looking* ***parents*** *and* ***reductions*** *relationships, it is clear that this graph is not a tree. Also, segment B can be detected as variation of the first part of segment A thanks to the* ***reduction*** *operation.*

enhance the identification of similarity between segments. An example of graph produced with this approach is shown in Figure 2.2.

By using *breadth-first search* and *depth-first search*, it would be possible to find the maximum connected components of the graph according to one ore more type of edges, making the graph invariant to that particular operation. Also, graph theory offers many algorithms that may have significant meaning in music context: e.g. *centrality* measures could enhance the representativeness of a database entry in a *features-like* approach, *minimum spanning threes* could reveal hidden patterns and *neighborhood* functions may can help in similarity tasks.

Finally, the initial score could also be removed from the graph since its informations are all contained in the union of segments. According to the task, one could also decide the number of recursion of the segmentation function and/or to remove intermediate extracted segments maintaining just the lowest level. A map of the time succession of the segments and their superposition in polyphonic works could also be added to speed-up some polyphonic task – e.g. harmonic analysis using reductions and formal analysis.

We remark that this way of representing the musical texture may allow for a better embedding of musical informations without losing what was already captured by previous MSRs. In the following, we will discuss how to implement an MSR based on this framework, keeping effective aspects of existing MSRs.

## 2.2 A new MSR framework

In this paragraph we will explain a possible algorithm coherent with the graph based paradigm just described. Choices are performed with in mind a similarity task and a *query-by-excerpt* like algorithm in which given a generic segment, we aim to found the work from which it was extracted.

### 2.2.1 Melody representation

The first two challenges we meet in using the paradigm depicted in subsection 2.1.2 are the choice of *segmentation function* and *segment representation*. By now, in this project we have focused on these two problems and we were not dealt with pattern transformations such as transposition and inversion.

There are many segmentation algorithms. A good survey can be found in López and Volk (2012). Using hierarchical algorithms allows for choosing to store just longer or shorter segments, or maybe both of them. By using short segments, the MSR will give more weight to little motifs tending to lose structural informations if segments are very short – i.e. one or two notes. With long segments, instead, internal pattern repetitions or elaborations – such as motifs – may are not tracked. The best would be to represent all of them, which would need a hierarchical segmentation algorithm. To this purpose we considered to implement the *VM1* algorithm by Velarde, Meredith, and Weyde (2016), which outperformed other segmentation algorithms in the most of measures in MIREX 2017 competition[1], and which perform a multi-

[1]MIREX is an annual competition between different algorithms in the context of Music Information Retrieval. One task is *Discovery of Repeated Themes & Sections* and refers to the segmentation of symbolic

level segmentation. Since our aim is a *query-by-excerpt* like algorithm, maybe we could keep only the most useful segments length. However, since *VM1* is targeted to *repetition* identification, just before testing the software we changed to the more classical *LBDM* algorithm by Cambouropoulos (2001). In this algorithm the length of segments is in some way determined by a threshold which has to be tuned. As it will be explained in the next chapter, we introduced new parameters in the algorithm which allows to a better adaptive behavior of *LBDM*

About the structure to represent each segment, we could chose an interval based representation remembering the first absolute note of each segment. Maybe also *GPIR* or *step-leap* could be used (see subsubsection 1.2.3.2). In this way, segments would already be transposition invariant without losing absolute pitch information. Regarding the harmonic content, it is absent in melodies. Anyway, the Orio and Rodà (2009) algorithm may allows to a more accurate similarity measure provided that we know chord annotations. Since we build a dataset of leadsheets with chord annotations, we used the Orio and Rodà representation with a slightly changed melodic similarity algorithm.

Since we decided to use Orio and Rodà (2009) algorithm, another important choice we had to do were the rules to assign scores given the harmony chords. Recalling the cognitive basis we want to give to our MSR (see subsection 2.1.1), we based scores on works by C. Krumhansl and Cuddy (2010) and Fred Lerdahl and C. Krumhansl (2007).

Details of *LBDM* implementation, scores assignment rules and segments representation will be given in the next chapter.

In general, we propose the following algorithm:

**Monophonic Graph**

1. segmentation at minimum level (LBDM local maxima)
2. detection of most important segment (single-linkage clustering of LBDM local maxima)
3. reduction of each added segment until just one note remains
4. adding edges between all segments (including reductions)
5. comparing segments (including reductions) and adding edges (not reductions)

### 2.2.2 Polyphonic music

We also designed an algorithm to use the same representation based on Orio and Rodà (2009) in polyphonic music. However we still did not implement it.

The main issue with polyphonic material and Orio and Rodà approach is the harmonic evaluation from which they extract scores. Our idea is to retrieve firstly the most important

music. Results can be found at http://www.music-ir.org/mirex/wiki/2017:Discovery_of_Repeated_Themes_%26_Sections_Results

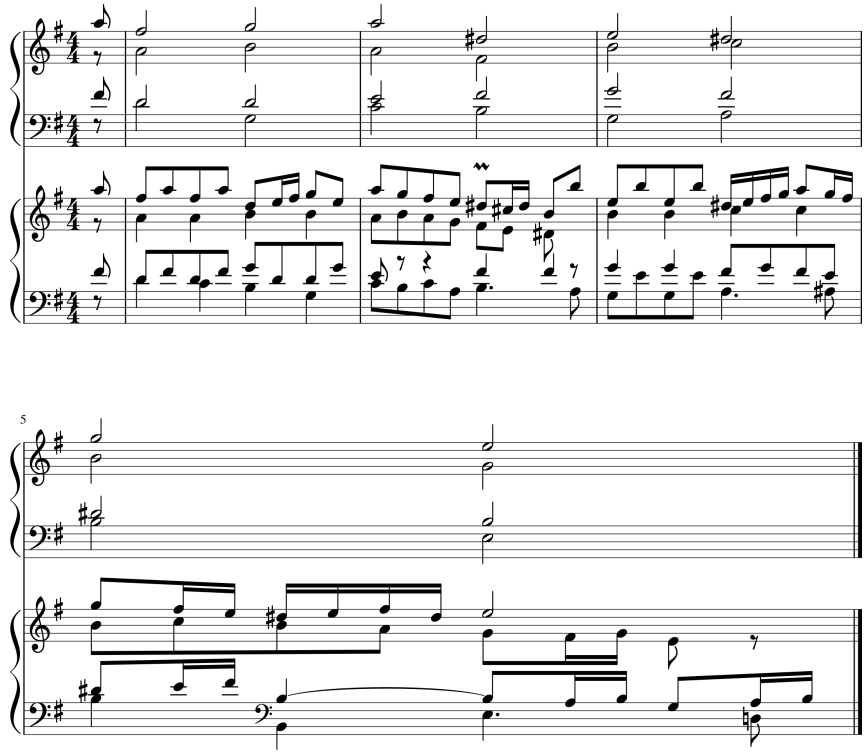


**Figure 2.3:** ***Polyphonic reduction****: an example of reduction obtained after 2-3 steps on an excerpt from Goldberg Variations, n.30, BWV 988. The reduction, contains quite all the harmonic material even though the texture is rather complex because of what De La Motte (1981) called "harmonic counterpoint"*

bass notes and secondly the scores of other voices notes. A possible approach could be therefore to compute exclusively the harmony basses notes – not fundamentals – based on the dissonance grade in respect to the other voices. Then, scores could be given to each note in each voice basing on the bass note.

Once a three for each voice is computed, one could build a graph as explained in subsection 2.1.2. This representation would allow an harmonic and contrapuntal representation of the musical texture. Namely, harmonic representation would be captured by the reductions and their superpositions (see Figure 2.3), while the contrapuntal representation would come out from the segmentation algorithm and the added edges.

The algorithm would resemble this guideline:

**Polyphonic Graph**

1. segmentation of each voice
2. reduction of each segment
   a) score assignment to the bass
   b) score assignment to the other voices based on bass scores
   c) reduction
3. transformation identification between different segments - both involving reductions and original segments
4. restart from point 3 for each reduction with length > 1 added to the graph

Bass scores of which in item 2a may be computed with a measure which takes into account the dissonance in respect to the other voices. We propose the following one, which still must be tested:

$$\frac{1}{N-1}\left((1-\alpha)\sum_{n=1}^{N-1}\Theta_n+\alpha\sum_{n=1,\Theta_n<\beta}^{N-1}\Theta'_n\right)$$

Where:

- $n$ is the voice index, with $n=0$ the bass voice;
- $N$ is the total number of active voices in the onset instant of the bass note;
- $\Theta_n$ is the consonance score assigned to voice $n$, based on variable consonance classes to take into account different cultural contexts;
- $\Theta'_n$ is analogous to $\Theta_n$ but computed after that voice $n$ has changed note during the IOI (Inter Onset Interval, the time interval between two onsets) of the bass;
- $\alpha$ e $\beta$ are parameters of the model;

The first part of the formula computes the dissonance level in the onset instant, while the second part takes into account the dissonance level of eventual resolutions. Letting $\beta=1$, any variation in any voice will be considered – e.g. this could be useful for pedals – while letting $0<\beta<1$ the classical concept of resolution of a consonance on the strong beat is simulated. Finally, $\alpha$ allows to weight the contribute given by the resolution to the final score. The best would therefore be to find some rule for the automatic computation of $\alpha$ and $\beta$.

Moreover by changing $\Theta_k$ one could adapt the algorithm to different cultural contexts and different psychoacoustic models.

Lastly, one could decide to add other scores to the ones proposed by Orio and Rodà, i.e. by adding melodic accent weights (Parncutt, Bisesi, and Friberg, 2013).

# 3

# Implementation details

## 3.1 Overview of the algorithm

As described in section 2.2 we used the schema below. In the following sections we are going to describe details of each single point.

**Monophonic Graph**

1. segmentation at minimum level (LBDM local maxima)
2. detection of most important segment (single-linkage clustering of LBDM local maxima)
3. reduction of each added segment until just one note remains
4. adding edges between all segments (including reductions)
5. comparing segments (including reductions) and adding edges (not reductions)

### 3.1.1 Enhanced LBDM

We used the *Local Boundary Detection Model* presented the first time in Cambouropoulos (1996). We adopted the formulation in Cambouropoulos (2001). Anyway, we introduced some new parameters and procedures to make the algorithm more adaptive and able to manage different music contexts. However we did not evaluated in a strong way the improvements brought by these little changes, but we confined the evaluation to informal qualitative judgments.

In the following we are going to describe the *LBDM* algorithm in its original formulation and our alterations.

#### 3.1.1.1 Original formulation (Cambouropoulos, 2001)

*LBDM* is mainly focused on a cognitive approach based on the acoustic associated to notes. We think that this could be, in general, a limitation in that the theoretical meaning of a note – e.g. its function in the context –, that is also explicated by the conventional western notation, is in some way expression of the way we listen to music. In other words, a $B\sharp$ is different from a $C$ because $B\sharp$ includes the context information according to which we perceive a *tension* and the need of a *resolution*.

*LBDM* is based on MIDI note values, that is a limitation since we know that local boundary may occurs with more probability near and after a leading note. Also, it does not take into account context informations, such as harmony and chords. However, it has bee largely used in many projects and it is almost a *de facto* standard, together with other more complex models.

*LBDM* uses three different scores computed between intervals and re-evaluated two times, so that each final score takes into account 4 consecutive elements – notes or rests. Scores are computed based on *pitch intervals* – that is MIDI value difference – *inter onset intervals* – the time which occurs between two onset, rests excluded – and *rests intervals* – the time of a rest. This formulation is clearly thought to be used with MIDI files, where "rests" do not exist; in fact, the *inter onset interval* should actually be the duration of the last note. For the sake of simplicity we decided to maintain this formulation and we considered the timing intervals with reference to the quarter length which is 1.0, as defined by `music21` library which we used in the software. Actually, one could think to use other interval representations for pitch difference, as described in subsubsection 1.2.3.2.

Given these three scores for each interval, *LBDM* computes three local boundary profiles by assigning a score to each interval – but not the first one and the last one – according to the formula:

$$s_i = x_i \left( \frac{x_{i-1} + x_i}{x_{i-1} + x_i} + \frac{x_i - x_{i+1}}{x_i + x_{i+1}} \right)$$

where $x_i$ is the score – *pitch*, *interonset* or *rests* – assigned to the $i$-th interval.

Then, each score profile is normalized between 0 and 1 and then a final local boundary profile is computed by a weighted average of the three scores. The proposed weights were:

$$\begin{cases} w_{pitch} = 0.25 \\ w_{ioi} = 0.5 \\ w_{rests} = 0.25 \end{cases} \tag{3.1}$$

Once we have the local boundary, we can detect the local maxima and choose the ones that are over a given threshold.

Actually, as Cambouropoulos says, this is an incomplete model but it is nice in that it can be computed in a $2 \times O(n)$ and in that it is quite precise.

#### 3.1.1.2 Improvements

We introduced some new procedure to make *LBDM* stronger to stylistic variations.

First of all, we made the weights changing based on the frequency of rests. The idea is that in songs with many rests, phrases are mainly expressed through pauses and they may contain huge pitch intervals relevant to the boundary profile; on the contrary, songs with few rests, tend to express hiatus through pitch intervals. To formalize this concept we introduced a threshold $t = 0.12$ according to which a value $\alpha = 0.2$ is added to $w_{rests}$ and subtracted to $w_{pitch}$ or vice versa. Namely, if the number of rests – not their duration – over the total number of notes and rests is $< t$, then $\alpha$ is subtracted from $w_{rests}$ and added to $w_{pitch}$; if the ratio between the number of rests and total number of notes and rests is $> 1.5 \times t$, then the other way around.

Another improvement we made was to make the threshold to select local maxima completely adaptive. We adopted a *hierarchic* clustering approach due to the small number of one dimensional samples – local maxima in this case. The clustering was stopped when only two clusters remained and the one with the highest values was used to segment the song. The algorithm adopted was *single-linkage*. Nevertheless also other more classical clustering algorithms such as *k-means* could work by setting the two centers one at 0 and the other at 1.

The last addition we made was to force the algorithm to not found boundary in tuplets, but only before the first note or after the last note. Actually, this is not an exact rule, since in general, a music could be written with tuplets to not introduce metrical changes. Also, phrases could terminate between two different tuplets, but the rule we introduced forbid this. In the most part of cases, however, this prevent from erroneous detections.

#### 3.1.1.3 Suggestion for future improvements

Some enhancement could still be made to *LBDM*. By example, one could think to use Orio and Rodà (2009) scores to take into account the functional harmonic context. Also, one could use *GPIR* or other interval based scoring to take into account the theoretical meaning of the intervals.

### 3.1.2 Computing reductions

Reductions are the most original part of this framework. They allows to highlight the most relevant musical elements (Marsden, 2010b), to summarize music (Hamanaka, Hirata, and Tojo, 2008), to express a similarity measure (Orio and Rodà, 2009; Rizo Valero, Iñesta Quereda, and León, 2006) and to detect little variations of a theme (Marsden, 2010a). The main issue is the computability since neither Hamanaka, Hirata, and Tojo nor Marsden are very efficient from a computational point of view.

From this perspective, the work of Orio and Rodà (2009) seems to be more appropriate for a practical application which should work on very large databases. We extended the reduction algorithm described in subsubsection 1.2.4.5 and used by Orio and Rodà (2009).

Figure 3.1 shows a simple example of reduction procedure of a simple melody according to our new algorithm derived from Orio and Rodà (2009).

In the new algorithm windows are set in a more intelligent way: they are no more long as the double of minimum duration in the last reduction, but they are large as the duration just a level longer than the minimum duration in the metrical tree of the song meter. By

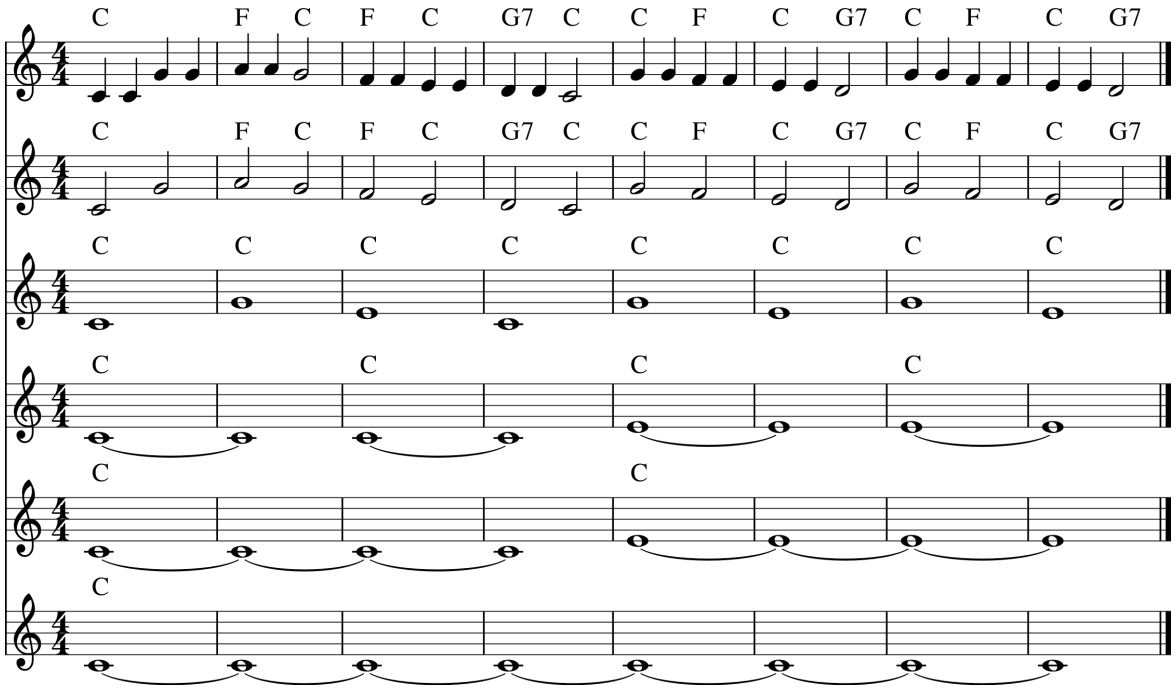



**Figure 3.1:** ***Monophonic reduction****: a simple example of reduction procedure of a melody until just one note remains. Note that the resulting note is the root of the tonality.*

example, in 3/8, if the minimum duration at a certain stage is the *eighth*, then the window will be long as three eighths. The key was to find a rule to delete one notes from a group of three and to assign the duration of the deleted note to the most important note of the two remaining. Once that just one note remained in each measure, all notes were reduced in duration to thirty-seconds and put in the same measure with time signature equal to $\frac{2 \times number\ of\ remaining\ notes\ or\ rests}{64}$. Then, the same algorithm was applied but, instead of padding with rests the semi-empty final windows, as we did in the first part, we deleted the last note – that is the last window since all notes had the same length; this was did because the algorithm used for padding was based on the original song measures and not on the computed windows, so that the rests that would have been added to obtain a complete ending measure would have been very long, making the computation really slow. Figure 3.2 explains this issue.

The new rule to decide which note must be deleted was made of two steps: first it deletes the note in the window with the lowest score, according to the rule precedence defined by Orio and Rodà; then it assigns the duration of the deleted note to the nearest note with the highest score and, in case of tie, to the previous note. Since we selected songs so that at most three notes per window appeared, in at most two rounds any group is reduced.

Following the same principle, we made the algorithm capable to work with any tuplets and metrics, also compound meters, but we tested it just with triplets and binary/ternary meters. It must be said that the new algorithm first parses the score in search of tuplets, put them in standalone measures and then reduces them to a single note in the original meter of the measure. After this, it continues with the normal reduction. We also made the algorithm able to manage ties. In general, the new version of the algorithm is able to manage more

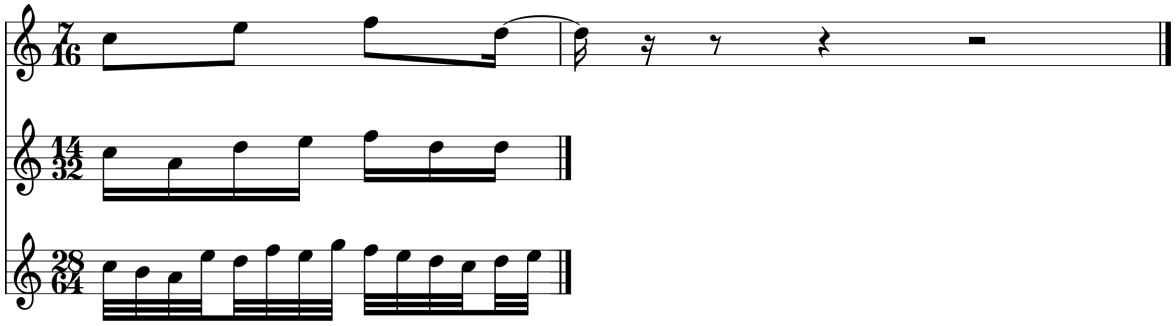


**Figure 3.2:** ***Padding last measure****: given a reduction containing one note per each measure, every note is converted to a thirty-seconds note; padding last measure with rest could need to double the total length of the score, making the computation really slow.*

complex cases coming from a noisy dataset as the one described in chapter 4.

About the scores, we were inspired by C. L. Krumhansl and Shepard (1979) and more recent works (Fred Lerdahl and C. Krumhansl, 2007; C. Krumhansl and Cuddy, 2010). Actually, few studies were conducted about a quantification of the dissonance produced by the superposition of two sounds. The following two tables show the scores we assigned to each harmonic interval and to each chord, that is the *consonance* score – Table 3.1 – and *functional* score – Table 3.2 – according to our nomenclature respectively – see subsubsection 1.2.4.5 for the nomenclature. *Functional* scores were computed based on the number of semitones which divide the root of the chord from the tonic, while *consonance* score is computed by using theoretical intervals between the melody note and the root of the chord. Metrical scores were assigned based on the theoretical metrical tree[1]. If one of the two notes was a rest, *functional* and *consonance* score was set to 0.5 – this value can be changed easily in the setting file.

We also tested only three score classes for both *consonance* and *functional* scores, since Orio and Rodà claim to have better results with this setting, but we did not obtained significant changes.

### 3.1.3 Putting all together: dataset representation

For each segment we computed the reduction. Note that time and key signatures are inherited by each child segment. Since the dataset we used was very noisy, with some score not compliant to the MusicXML standard, some segments were not reducible and we discarded them – anyway, the number of the discarded segments was small compared to the total.

We also added a weight to the graph branches following different strategies:

- add one for each note deletion in a reduction step
- consider each branch with weight 1

[1] Despite `music21` uses some misleading structure in the representation of the metrical tree – i.e. in 4/4 it considers the third quarter to be as strong as the second and the fourth – we attained to the usual western musical theory

**Table 3.1:** *Consonance scores*

| **Intervals** | **Scores** |
|---|---|
| perfect unison | 1.00 |
| perfect fifth | 0.75 |
| diminished fifth | 0.50 |
| major third | 0.75 |
| minor third | 0.75 |
| diminished third | 0.20 |
| perfect fourth | 0.30 |
| minor sixth | 0.40 |
| major sixth | 0.40 |
| minor second | 0.20 |
| major second | 0.20 |
| minor seventh | 0.50 |
| major seventh | 0.20 |
| diminished seventh | 0.50 |
| *rest* | 0.50 |

**Table 3.2:** *Functional scores*

| **Scale grade** | **Scores** |
|---|---|
| I | 1.0 |
| II♭ | 0.2 |
| II | 0.5 |
| III | 0.4 |
| IV | 0.8 |
| IV♯ | 0.3 |
| V | 0.9 |
| VI♭ | 0.1 |
| VI | 0.6 |
| VII | 0.7 |
| *rest* | 0.50 |

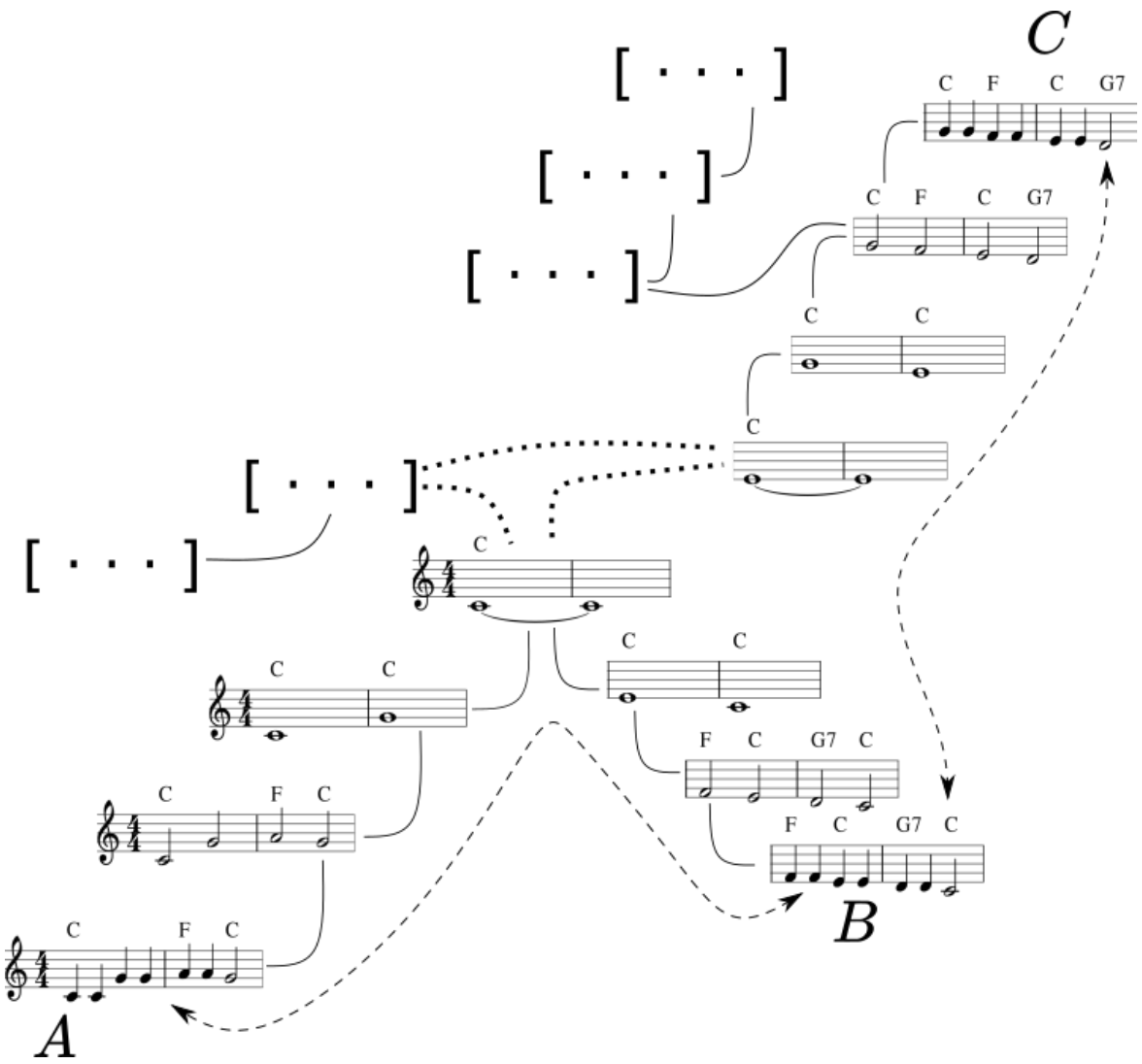



**Figure 3.3:** ***Dataset representation****: the distance between two segments – i.e. A and B – is computed as a path (dashed lines) in the graph representation of the dataset; if two segments – i.e. B and C – have a different top note in their own reduction tree, then another edge is added at retrieval time (dotted lines). Weights are not shown in the figure.*

- add a distance which takes into account the intrinsic differences between the note deleted and the note enlarged in duration

About the last point, distance used was the note-to-note distance described in section 3.2.

All segments were uniquely associated to a song and each song was associated to a list of segment reductions. Each segment reduction contained the branch weights to be summed up to compute the distance between two segments. On the top level only one note remained and an edge is added at retrieval time between top-levels, weighting through note-to-note distances. From now on, we will call *internal* edges the ones added at indexing time – they could be weighted or not – and *external* edges the ones added at retrieval time – always weighted.

The resulting is a graph like the one in fig. 3.3 in which we can compute distances between two segments as path in the graph.

## 3.2 Distances

We developed a number of distances, but we did not tested all them in our Symbolic Music Representation since they were too many. Distances were computed with a block model as depicted in figure 3.4.

There are a number of settings that can be used. The following apply to *note-to-note* distance and can be used in *segment-to-segment* distance by simply setting the *distance frequency* described below.

- **duration weight**: one can decide to weight each distance with the duration of the less long note
- **distance type**: this could have a different value, namely:
  - *semantic*: uses the mean of our functional, consonance and metrical scores; optionally, this value can be weighted with the *estrada* distance (Rocher, Robine, Hanna, and Desainte-Catherine, 2010) of the harmonic context of the two notes. Since scores take into account the harmonic context, these are also transposition-invariant. It uses note based representation.
  - *boolean*: it assigns 1 if intervals are different, 0 if they are equals; intervals are obviously transposition invariant. It uses interval based representation.
  - *fuzzy*: this returns a value depending on the interval representation and included between 0 and 1; as before, this is transposition invariant. It uses interval based representation.
  - *MIDI*: it normalizes the two segments to the their average midi value and returns the midi distance as in Velarde, Meredith, and Weyde (2016); normalization makes this measure transposition invariant. It uses note based representation.
- **interval representation**: *fuzzy* and *boolean* distance need an interval based representation; duration of intervals is the duration of the first note; we developed:
  - *music21*: uses theoretical intervals; with *fuzzy* distance, the difference can be based on the chromatic difference or on our *consonance* scores
  - *GPIR*: see subsubsection 1.2.3.2
  - *step-leap*: see subsubsection 1.2.3.2
  - *contour*: see subsubsection 1.2.3.2
- **distance frequency**: distance can be computed:
  - at each new onset in any of the two voices
  - with a fixed sampling, such as at each thirty-seconds note duration

If we consider three different values for chord weights in *semantic* distance and three different values for fixed sampling in *distance frequency*, we obtain 78 different distance measures.

## 3.3 Tools

In this section we will briefly describe the tools used to develop and test the algorithm.

### 3.3.1 `Python 3.6`

We chose to use `Python` as programming language. Python is a modern multi-paradigm interpreted strongly and dynamically typed programming language which is becoming more and more used in scientific contexts due to its efficiency and to the predisposition to integrate pure `C` library. Moreover, the library `music21` is written in `Python` and offers the ability to open ad elaborate various different symbolic music formats, MusicXML included, without loosing information. This was useful to implement our reduction algorithm described in subsection 3.1.2 which is based on chords annotations. More used tools, like `Matlab` and `midi-toolbox` were not suitable to our purposes.

One of the major pros of `Python` is that it can be applied in an incredible variety of fields, from big data analysis to machine learning and computer vision, from web development to end user GUI applications, from distributed systems to signal processing, numerical calculus and so on. This makes `Python` more desirable than `Matlab`, which instead is exclusively targeted to just scientific research. On the other hand, `Python` forced us to deal with some issues that, anyway, we think also `Matlab` would have compelled us to overcome.

Actually, our experience with `Python` was nice and we liked the way `Python` gives the ability to change the programming paradigm in different sections of the code. By example, our code is mainly object-oriented but the distance module, which is function based. Also, `Python` allows these functions to became methods of future complex objects thanks to the *binding* mechanism.

The main issues we found were about *serialization* and *parallel computing*.

### 3.3.2 `music21`

`music21` is a `Python` library developed at MIT by Michael Scott Cuthbert and collaborators (Cuthbert and Ariza, 2010). The main reason which forwarded us to use `music21` was the complete ability to open MusicXML files and the number of ready-to-use musicological procedures available.

`music21` is based on a tree music representation implemented through a *composite* pattern (Gamma, Helm, Johnson, and Vlissides, 1995), where a `stream.Stream` – i.e. a score – can contain other `stream.Stream` objects – i.e. some instrument parts or some measures. Actually, `music21` is a still in development project which under many perspective should be improved.

First of all, it is almost completely written in pure `Python` code, which makes it very slow.

Second, it includes only algorithms coming from quite old papers and music computing community is not helping in this – we said in the incipit of the thesis that symbolic music is very little studied.

Third, we think that some choices were non intuitive – i.e. see note about 4/4 subdivision in subsection 3.1.2 or bugs in chapter 4 due to the non saving of some attributes in the top-level of the *composite* pattern– and little suitable for musical analysis.

Fourth, we found that some methods and functions integrates little between them, by example we had to use measures as windows and no suitable function was provided to process the score in handy windows, even if this possibility is considered with a *windowed* module.

In general we have found in `music21` an interesting project which should be supported by the scientific community. It is born in field poorly studied where collaborations and reuse of code between different projects is of fundamental importance to achieve the best results.

## 3.4 Software design

### 3.4.1 Introduction to the design

Our implementation is mainly base on two objects: *SegmentReduction*, which represent the reduction of a segment, and *StreamReductor*. *SegmentReduction* contains a key referred to the song to which it belong and an ordered list of *StreamReductor*. This last one contains a `music21` *stream.Stream* and is able to produce a new *StreamReductor* containing the *stream.Stream* object representing its reduction. After it computes its own reduction, it also contains the distance between itself and the new reduction. All the implementation is based on *stream.Stream* objects, which is a superclass for measures, scores, parts and so on. In this way, *SegmentReduction* and *StreamReductor* should also work with other types of *stream.Stream*. This is in some way similar to the *builder* design pattern (Gamma, Helm, Johnson, and Vlissides, 1995).

An important approach we adopted during the development was the usage of *proxy* functions to achieve different behaviors with the same function based on global or setting variables. By example, *StreamReductor* has some methods which parse all notes of a stream or a single measure or a single note; the first depend upon the latter. These perform a variety of tasks that must be applied in different point of the code but always with a different combination. This is implemented through *proxy* functions to assure that each task is performed always in the same way. Another example is the distance module – see section 3.2. For the same purposes we also made a large usage of `IntEnum` from the standard library.

Scores are computed each one by a specific class which implements a completely abstract class *Scorers*. Tuplets and ties are managed by apposite classes, while the reduction rule described in subsection 3.1.2 was moved to a standalone class.

Finally, settings are in a standalone module and are simple variables. Of sure, this is not the best option from a security point of view, but it is the most straightforward for research purposes.

### 3.4.2 Implementation issues

`Python` does not allow to serialize every type of data. Moreover, `music21` uses *WeakReferences* objects that `Python` standard library is not able to serialize. Luckily, there is `dill`, which is a library to serialize even complex `Python` but it is very slow.

*Serialization* is of particular importance when trying to parallelize the computation, since `Python` interpreter forbid the parallel execution of threads: it set a lock on a global variable

– called GIL for *Global Interpreter Lock* – and each thread must wait for the GIL to start its execution. This shall facilitate the development of pure `C` libraries, upon which `Python` is strongly based, and allows the developer to focus on single threading performances of the `Python` interpreter, that is, in fact, very efficient. There is a lot of discussion about the GIL in the `Python` community because it makes harder introducing parallel computing. The only way to take advantage of multiple processors is by using multiple processing with multiple instances of the interpreter. Moreover, `Python` standard libraries can manage in the shared memory only the `C` standard types – e.g. `int`, `char`, `double`, etc. – ad this makes more difficult to share complex `Python` objects, which must be serialized and copied in each private memory of each parallel process. It must be said that also `Matlab` makes difficult to use parallel threads.

Anyway, we ended up by developing a custom `pickle` module which uses `music21` functions to serialize `music21` objects. This allowed us to easily save the dataset representation to file. Surely, the best would have been the ability to use multiple threads; this could be achieved in the future by using `Jython` – a compiler for JVM bytecode, so no GIL at runtime –, which at now only supports `Python 2.7`, or by using `Cython` – a `Python` implementation which supports pure `C` statements and can disable GIL but this would need to create a wrapper from `music21` objects to pure `C` structures that can be altered in `with no GIL` blocks of `Cython`.

# 4

# Dataset

Since we needed music sheets with chord annotations, we build a new large dataset starting from a hold Wikifonia archive. We, by using `discogs.com` and `secondhandsongs.com`, added meta tags, lyrics, genres and various features to each score and filtered what our algorithm could not parse. The resulting dataset included more than 5.000 leadsheets. We also extracted a public domain dataset of about 550 scores.

In the following paragraphs we will describe all the dataset.

## 4.1 2018 EWLD

### 4.1.1 What is EWLD

EWLD (*Enhanced Wikifonia Leadsheet Dataset*) is a music leadsheet dataset coming with a lot of metadata about composers, works, lyrics and features. It is designed for musicological and research purposes.

The following table resumes the dataset.

| Feature | Number |
|---|---|
| Compositions with first performance date | 2031 |
| Composition with annotated language | 3290 |
| Compositions with lyrics | 4940 |
| Compositions with annotated features | 5151 |
| Composition with annotated genres | 3489 |
| | |
| Composers | 2804 |
| Composers with death/birth date | 2188 |
| Composers with nationality | 2472 |
| | |
| Traditional songs | 57 |
| Public Domain scores | 568 |

| Feature | Number |
|---|---|
| Compositions with annotated composer | 3154 |
| Compositions by USA authors | 2713 |
| Compositions by UK authors | 434 |
| Compositions by Italian authors | 36 |
| Compositions by French authors | 76 |
| Compositions by Canadian authors | 67 |
| [etc.. ] | [. . . ] |
| *Total number of leadsheets* | *5151* |

### 4.1.2 What we have added

This dataset comes from the old Wikifonia archive that is available on the web. We just filtered its files to get only the ones compatible with algorithm descripted in the previous chapters that is a graph-based representation of musical scores aimed at computational music analysis, computational musicology and music information retrieval.

Moreover we added some notions taken from `secondhandsongs.com` and `discogs.com`, such as the correct title, authors, year of composition, authors' year of birth and death and nationality, language, musical genres and styles. Also, we added a lot of features relative to each music score computed through `music21` (Cuthbert and Ariza, 2010) and we separeted the lyrics where available.

The dataset is composed by one directory per author or combination of authors and one directory per work, containing the leadsheet a text file with lyrics and a `.csv` file with extracted features. Moreover, a little SQLite database contains all metatags such as year of the first performance, authors, geres and so on.

A `python3` script which creates the database is available.

### 4.1.3 Database creation

The database was extracted with a `python` $\geq$ `3.6` script from the old Wikifonia dataset. From this a public domain subset was extracted. You can regenerate it by simply launching the script with this command:

```
python3 EWLDcreation.py -d <path-to-wikifonia>
```

You will need a stable internet connection and some `python` libraries, that you can install using the following command:

```
sudo pip3 install discogs_client music21\
requests argparse csv json operator os sys\
traceback zipfile sqlite3 collections typing\
time
```

Propably, most of them are already installed in your `python3` distribution.

If internet connection goes down, you will find some file in a directory called `exception_dir`. Delete it and rerun the script a few times without changing anything. If exceptions still persist, please, contact us.

Note that the file `db_creation.sql` **must** be in the working directory. It contains the SQL script needed to create the initial tables.

If you want, you can also extract a subset containing only public domain scores by simply running this command from the same directory in which you find this README:

```
python3 OpenEWLDcreation.py
```

### 4.1.4 Dataset organization

The database is organized as follows:

- In the directory called `dataset` you will find a directory for each composer and within it a directory for each score by that composer (or combination of composers). You will also find a directory called `[Unknown]` for all scores without a recognized composer.
- In the directory `except_dir`, if it exist there will be scores and logs relative to files that cannot be parsed correctly
- In the file `EWLD.db` you will find a `SQLite3` database which contains all metatags and file path

Note that all scores are filtered and edited so that they have the following properties:

- Only one key signature and only one time signature – but they may not be there - see subsection 4.2.1
- No `strong` modulations –see paragraph about `tonality` field in the `features` table of the db
- Any triplet is allowed but no other tuplet, no nested and not between different measures – *N.B. We thought to have removed tuplets with notes of different values, but actually they could still be there, see line 276 of the creation script*
- the tonality is expressed in the MusicXML
- chords symbol (harmony) are always annotated in the MusicXML
- only one voice is present in the score, no secondary voices or chords are allowed
- no multiple white measures at beginning or at the end of the segment

### 4.1.5 Database structure

The database `EWLD.db` is a SQLite3 database. You can use any software to read it, for example SQLiteStudio which is simple, tiny and portable. With it you can also extract XML, HTML, JSON, SQL and PDF files.

It contains 6 tables, each described in the following paragraphs.

Each entry represents a work, as intended in `secondhandsongs.com`. A work is a music composition: it can have several recordings by different performers, but the music opera is only one.

Actually, because of the noisiness of the beginning dataset, you could also find entries representing derived works. Most of them should have the authors marked as '[Unknown]'.

Fields are:

- *id*: unique integer
- *title*: if author is not 'Unknown' it is the title given by `secondhandsongs.com`, otherwise it is taken from the original score metadata or from the filename
- *first_performance_date*: if available, the date of the first performance of the song, as given by `secondhandsongs.com`. Usually, it should be similar to the composition date
- *language*: language as given by `secondhandsongs.com` or as detected by music21
- *path_lyrics*: path to the txt file containing the lyrics (it could be empty)
- *path_leadsheet*: path to the compressed MusicXML file containing the leadsheet

**`authors` table** This represents the authors. Fields are:

- *commonName*: the name of the author as given by `secondhandsongs.com` (the real author name is not available through their API at now; homonimy are diversified with ending `[1]`, `[2]`, etc.
- *birth*: date of birth if available (from `secondhandsongs.com`)
- *death*: date of death if available (from `secondhandsongs.com`)
- *nationality* nationality if available (from `secondhandsongs.com`)

**`features` table** Each entry describes a work from a musical point of view.

- *id*: the id of the work
- *metric*: metric signature as stated in the original score **N.B. This is affected by a bug, sometime 4/4 could appear but it could be wrong, see subsection 4.2.1**
- *tonality*: the tonality as detected by *krumhansl-schmuckler* algorithm – to avoid erroneous notation given by wikifonia users. If the certainty computed by [music21] is not enough high – namely >= 0.9 –, the one provided in the score is used. This prevents by erroneouses key signatures provided by Wikifonia users. **N.B. a bug in saving score made key detection useless, see subsection 4.2.1**
- *incipit_type*: a string, it can be 'anacrusi', 'acefalo' o 'tetico'
- *has_triplets*: a boolean that is true if there are triplets
- *features_path*: path to the csv file containing features computed by music21. Each row is a different feature. You can find a feature given its index, e.g.:

  - with `features.base.getIndex('name')` you get the index of a certain feature, that is equal to the row index in the csv file;
  - with `[x.id for x in features.extractorsById('id')]` you can get the list of features id in the same order as in the `csv` file;
- read more here.

**`work_author` table** This table is needed to join `works` table and `author` table

**`work_genres` and `work_styles` tables** These tables give to each work a style and genre classification in a 2D space with a fuzzy approach. By example, a work genre could be identified by a 2D vector ('rock', 'pop'). Each entry of these tables represent an entry of the vector. Also, each entry of the vector is associated with a `occurrences` field that models the certainty of that genre for that work. A good way to infer genre and/or style of a work may could be by checking if one of the two vector entries has more than twice occurrences of the other. If yes, you could consider it as genre or style of the work, otherwise you can represent its genre/style using two coordinates.

These tables create association between a work and genres derived by `discogs.com` following this procedure:

- For each work creates a query made by *`title composer1 composer2`* `etc.`
- Consider the top-10 releases ordered by year returned by `discogs.com`:
- Takes histogram of genres and styles, following `discogs` categorization, that is users based
- For each work, memorize the two most common genres and the two most common styles

## 4.2 Discussion on dataset issues

### 4.2.1 What should be added

Despite our work, many enhancement can still be apported to the database. First of all, some bugs, due to a not complete documentation of `music21` and to non intuitive functioning of some classes, should be solved before the next recreation of the database.

Also, we got access to datadumps of *discogs* and *secondhandsongs* – *discogs* let datadumps freely available over the network, while *secondhandsongs* kindly sent us as Christmas gift simply by email. Therefore, one who need to recreate the database could made the creation script network indepedent.

Other improvements could concern the language and lyrics since we did not spend a lot of time on it: often Wikifonia users introduced dynamics or technique text in the lyric field or did not used a proper format for syllabication so that lyrics file at now are very dirty. Also one could use semantic information to add interesting features to each score.

Summarizing, the following is a list of possible improvements.

1. Lyrics semantic tags (maybe using gensim or polyglot library)
2. Delete path to empty lyrics files
3. Add other features
4. Use additional source info (`google.com`, `musixmatch.com`, etc.)
5. Add path to authors directory
6. Switch to extraction from data dumps no connection required
7. Add language detection (polyglot library)
8. Correct measure numbering after removing empty measures
9. Solve bug on key and time signatures setting: no signature is saved if it is setted in highest object hierarchy of the *composite* pattern
10. Solve bug on time signatures: method getTimeSignatures() returns '4/4' by default
11. ...

### 4.2.2 Related works

To our knowledge two main datasets containing leadsheets exist. One is coming from *wikifonia.org* a collaborative website closed in 2013 due to copyright law. The other one is the *LSDB* coming from *Sony Computer Science Laboratory* (Pachet and Suzda, 2013). This last database is the larger in our knowledge – about 8.000 songs – but it is mainly focused on jazz music and its content seems to be particularly cured. The first one, instead, is more noisy and needs more robust algorithms.

Actually, there are other two interesting datasets that are folk-tuned oriented: one is the *Nottingham Folk Music Database*, which contains more than 1200 leadsheets of British and American folk tunes in ABC format and was originally created by Eric Foxley[1]. The second one is the *Essen folksong collection* and do not contains leadsheets but just more than 6.000 folk melodies from all over the world – that is it lacks of chord annotations; it is distributed in the *kern collection* and in the `music21` corpora as well.

Our effort was intended to build a new rich dataset of more than 5.000 leadsheets that can be merged to the *LSDB* and *Nottingham Folk Music Database* to create an even more large dataset. Anyway, *EWLD* seems to be the more heterogeneous and noisy of the three, hence it is the most suitable to MIR challenges.

### 4.2.3 List of authors and other statistics

The following are some figure that show the *EWLD* dataset content distribution over the years and between different genres. Note that following results take into account just about the 60% of the total dataset (see table in subsection 4.1.1)

To evaluate the distribution of the dataset content over the time, we evaluated five different plots:

[1] It is available in ABC format at `https://ifdo.ca/~seymour/nottingham/nottingham.html`, while the original database in a old ASCII format at `http://www.chezfred.org.uk/freds/music/`

- Figure 4.1 and Figure 4.2 show the number of authors who dead in a given year. From here it is clear that the great part of the authors lived in first 90 year of XX sec;
- Figure 4.3, Figure 4.4 and Figure 4.5 show the distribution of works over the years. It is self-evident that the dataset content is distributed in the 40 years between 1930 and 1970;

About the distribution of genres, instead, this dataset is polarized versus *jazz* standards, like *LSDB* (see subsection 4.2.2), but we show in three different plots that also *pop* and *rock* music is quite good represented. The main issue here concerned the meaning of the genres attributes added in the dataset. We had from 0 to 2 genres associations for each work, and for each association we had a *certainty*-like measure given by the genre occurrences in the *discogs* database – which regarded *releases* returned by a query that not necessarily retrieved entries related to the work at hand. Also, in *discogs*, genres tag are attributed by the users. We performed three different evaluations:

- In Figure 4.6 we normalized the genre occurrences of each work and we added up all them for each genre – both the most common and the other;
- In Figure 4.7 we did not normalized occurrences; instead, we intended each work to be associated to a genre only and only if it had just one genre association or it had one genre with the double of the occurrences than the other, otherwise the work was not took into account;
- In Figure 4.8 we intended each work to be in the genre of the most common tag; we discarded only those works for which there was a tie between two associated genres;

From this little experiment we can argue that normalization is not so much important if we do not care about certainty relationships of the two detected genres. Instead, we think that the most accurate measure is the second one.

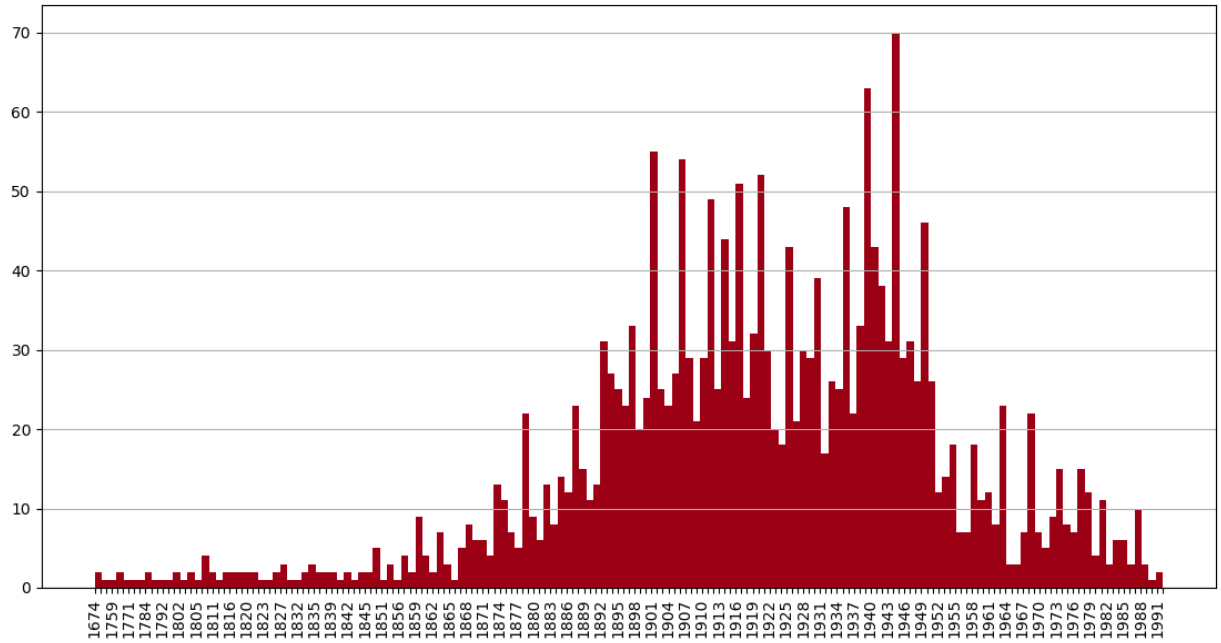



**Figure 4.1:** *Authors birth distribution over the years in absolute value*

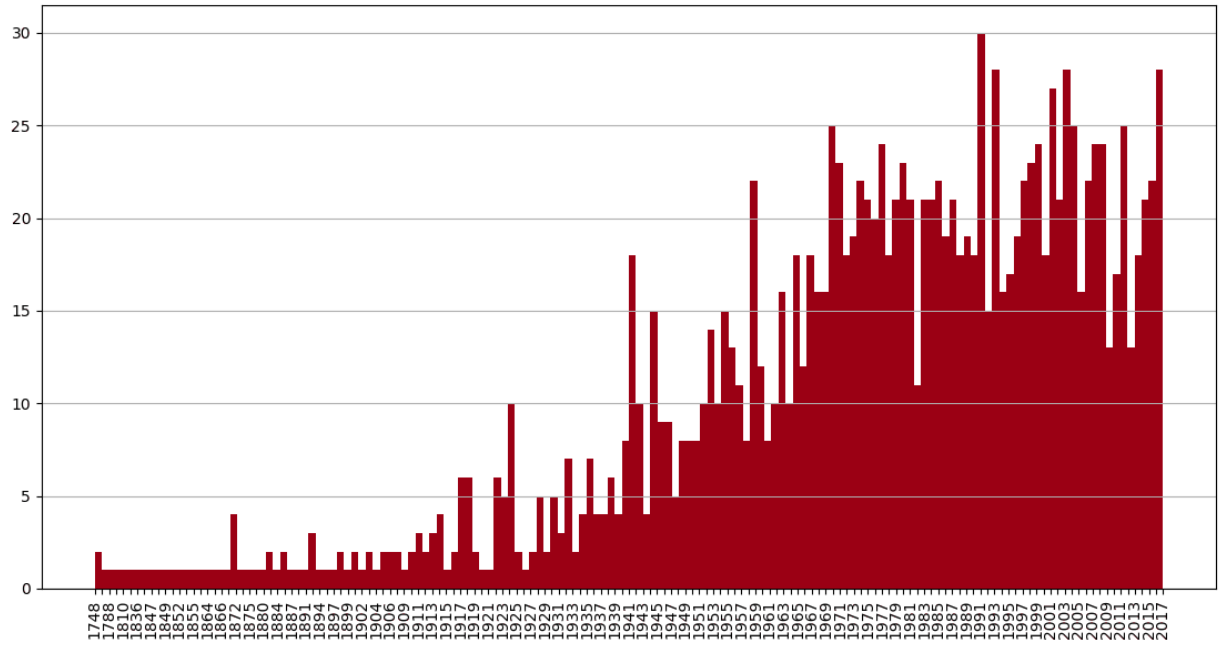



**Figure 4.2:** *Authors death distribution over the years in absolute value*

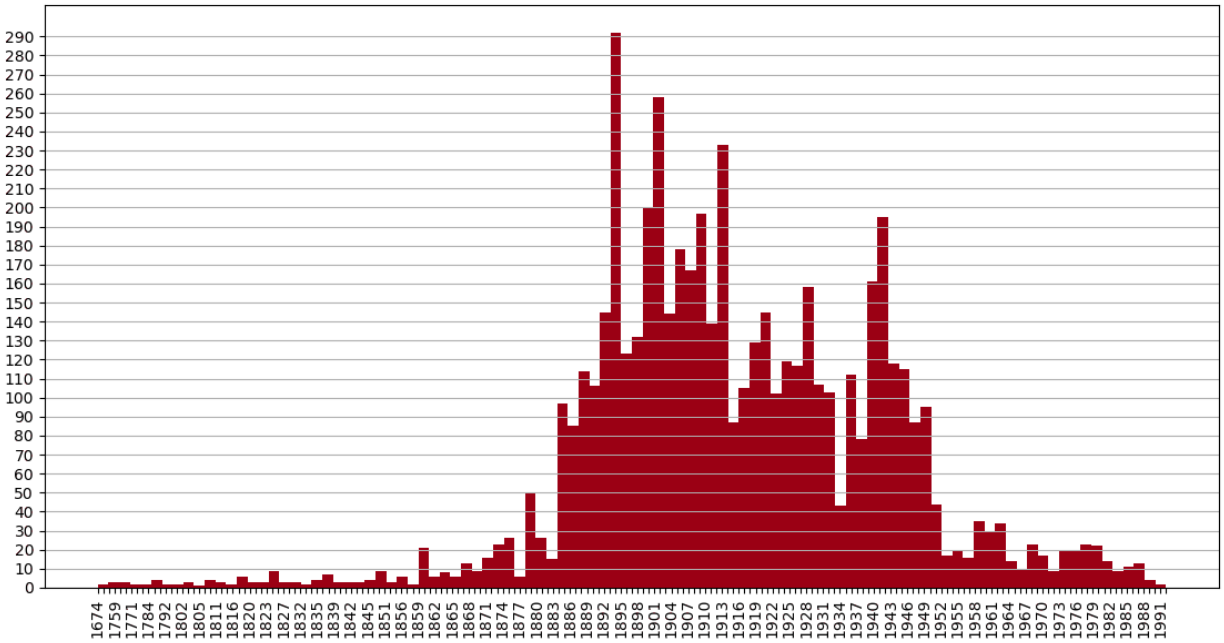



**Figure 4.3:** *Absolute number of works whose composer born in a given year*

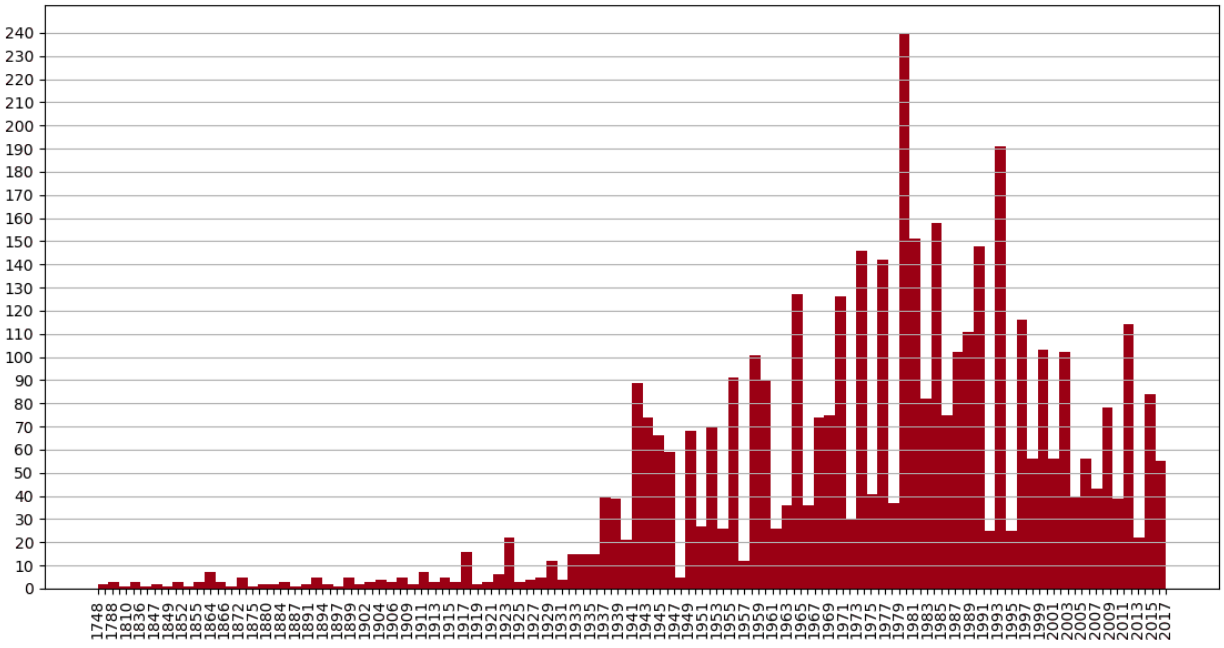



**Figure 4.4:** *Absolute number of works whose composer dead in a given year*

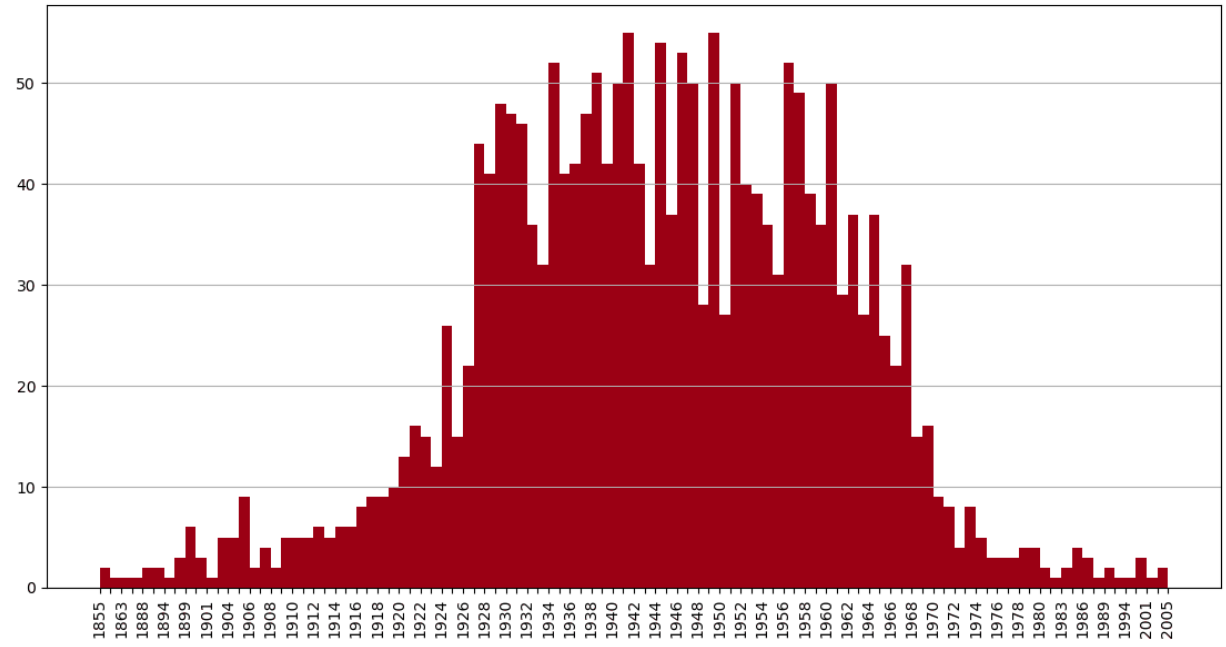



**Figure 4.5:** *Absolute number of works whose first performance date was in a given year*

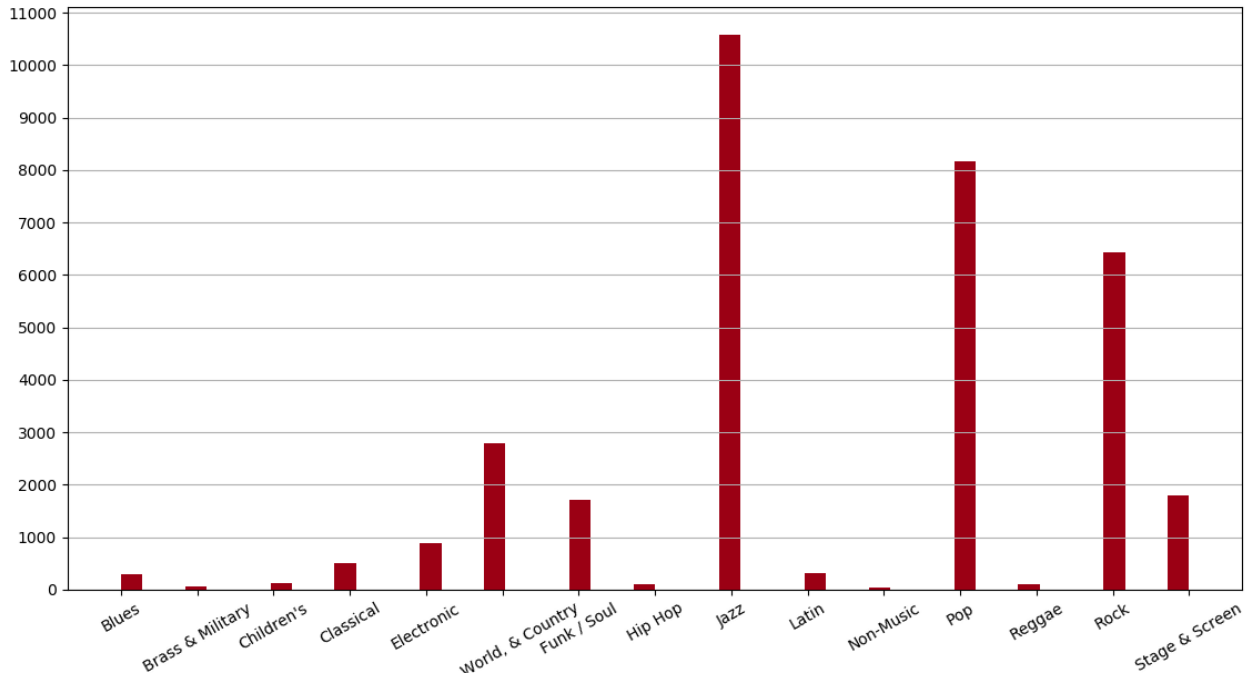



**Figure 4.6:** *The absolute number of occurrences of each genre in the dataset after having normalized the total number of occurrences of every work in the genre table to 100*

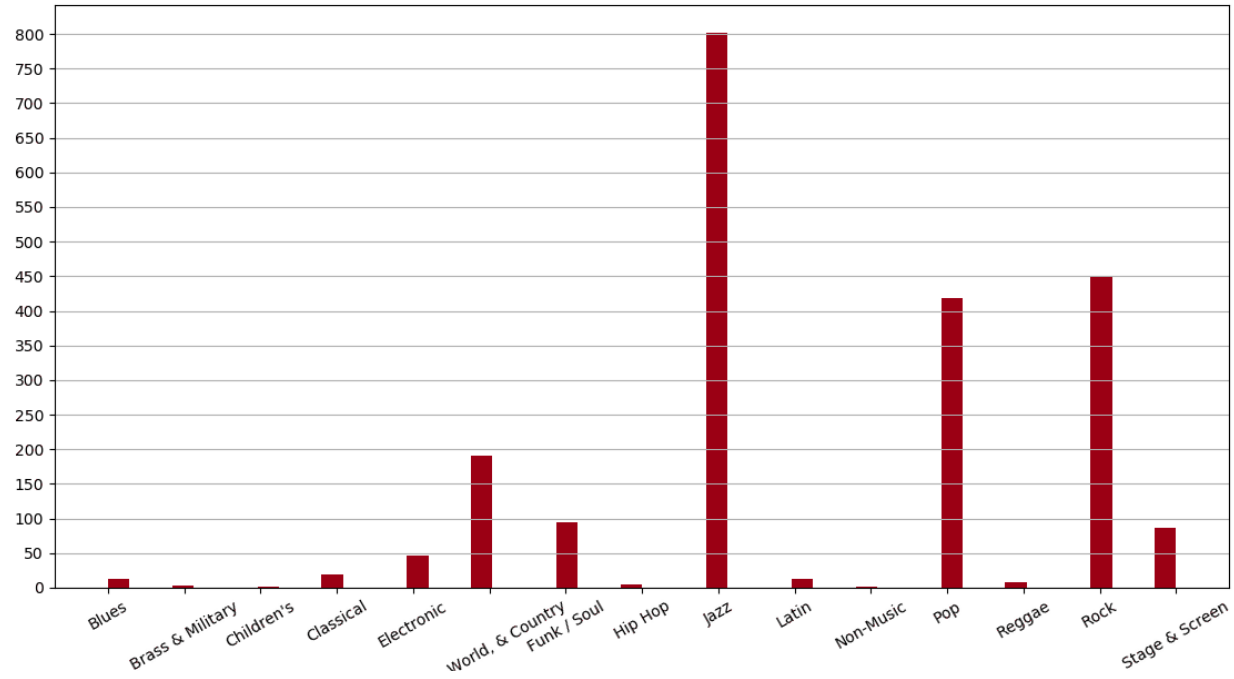



**Figure 4.7:** *The absolute number of works for each genre where we considered as genre of each work the one with more occurrences and where we discarded works which had 2 genres associated and the first genre was not 2 times the second genre*

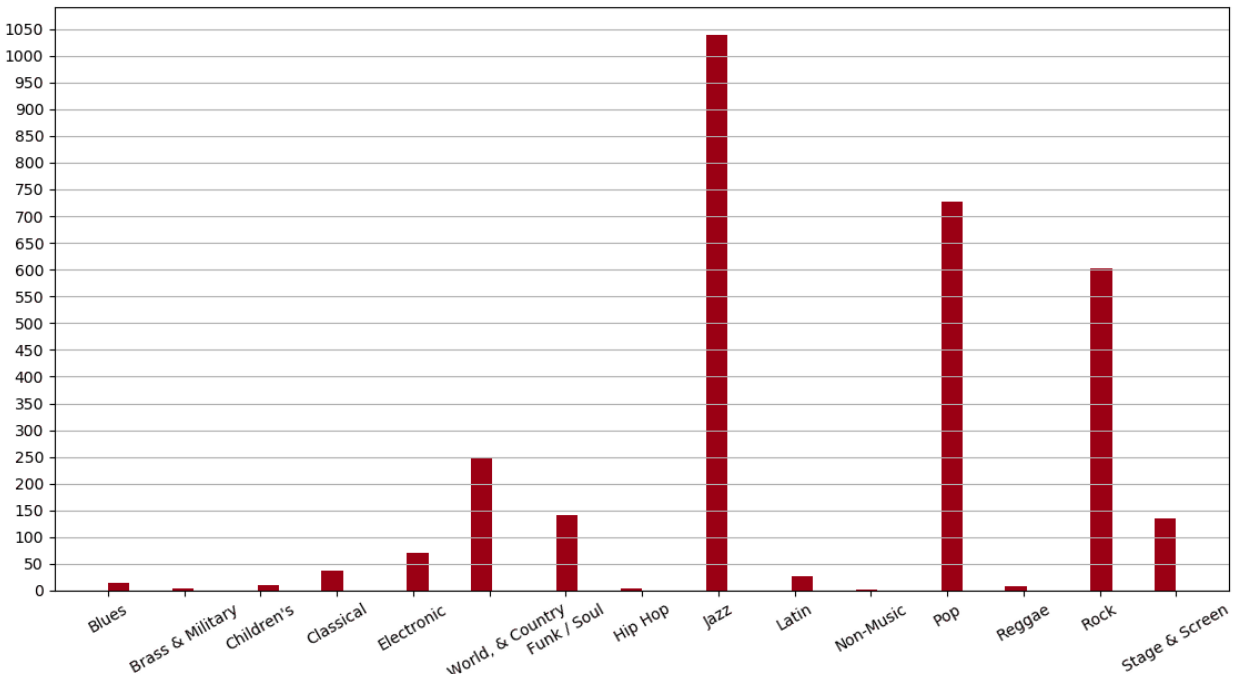



**Figure 4.8:** *The absolute number of works for each genre where we considered as genre of each work the one with more occurrences, without discarding nothing*

# 5

# Evaluating the MSR

As stated in section 1.3, we think that a good music symbolic representation system should be evaluated on a real world dataset. To this purpose we build the *EWLD* dataset. In the following sections we are going to describe the experiments we made and results we obtained. Lastly, we will give some observations on what the future developments could be.

## 5.1 Experiments

We performed four different experiments, combining two different measures and two different querying systems. In all these experiments, we tested our MSR in a similarity based task in which the distance $d$ between two segments is the sum of the branch weights in the reductions of the two segments until the top level – internal edges –, plus the *note-to-note* distance between the two top-level notes or intervals – external edges. Moreover, if the top level is the same for the two compared segments, the distance is computed starting from the highest level in which they differs. If they are equal in all the levels, their distance is 0. An demonstrative image is shown i fig. 3.3. Then, the distance $\delta$ between a query segment $q$ and a document $c$ is defined as the average distance between $q$ and all segments belonging to $c$. In formula:

$$\delta(c, q) = \frac{1}{N} \sum_i d(c_i, q), \;\; c_i \in c$$

This is the same approach used by Orio and Rodà (2009). Actually, they used some normalization factors and a *similarity* measure $s$ defined as

$$s(c, q) = \frac{1}{1 + \frac{\delta(c,q)}{\alpha}}, \quad \alpha = \frac{1}{N-1} \sum_{j=1,\; j \neq i}^{N} \Delta(c, k_i)$$

where $\Delta$ is the distance between $c$ and each other document in the collection $k_i$. Anyway, $s$ has a one-to-one correspondence with $\delta$, so that, to our purposes, it is an extra computation.

Since our aim was to test the representativeness of the MSR, we computed the top-20 similar documents in a *naive* way: we made a simple linear search on all documents in

the collection to compute an array of about 4800 dimensions, where each dimension is the distance between the query segment and a document; then, we sorted it with *quick sort* and we evaluated measures described in subsection 5.1.1. Surely, better algorithms can be implemented, for a review see Abbasifard, Ghahremani, and Naderi (2014). Distance between a segment and a whole document was computed as the average distance between the segment and all segments in the document.

We explored the space of distances described in section 3.2: each distance function is used as weight function for the branches of the model. Namely, we tested the following – see section 3.2 for an explanation of the words used below:

1. no duration weight in distance computing
2. just onset frequency
3. model weighted with semantic function
   - experiments by using semantic distance with chord weight = 0.0
   - experiments by using semantic distance with chord weight = 0.5
4. model weighted with MIDI function
5. model with all branch weights set to 1
   - experiments by using boolean function
     - `music21` interval representation
     - GPIR representation
     - step-leap representation
     - contour representation
   - experiments by using fuzzy function
     - `music21` interval representation chromatic difference
     - `music21` interval representation scores difference
     - GPIR representation
     - step-leap representation
     - contour representation

Experiments relative to item 5 were made without internal edge weights due to computational reasons. Anyway, we then checked that with internal weighted edges results did not change significantly. We explored other parameters without success. For a complete discussion, see sections 5.2 and 5.3.

### 5.1.1 Parent song and genre classification

Two of the four experiments had the objective to identify the song to which a segment belonged. The other two experiments, instead, aimed to identify only the *musical genre* of the parent song of the query segment. The first type of experiments is based on the assumption that two similar segments have a similar reduction, so that their distance is

low. The second type, instead is based on the assumption according to which the stylistic similarity is captured in the semantic distance, which is able to measure also the harmonic context.

In the *parent song detection* task we evaluated the following measures:

- **Precision at 20 (P-20)**: the number of queries upon the total (400) in which the parent document of the query segment appeared in the top-20 documents retrieved
- **Mean Reciprocal Rank (MRR)**: if the parent document of the query segment was retrieved in the top-20 documents, we stored its reciprocal rank; then, computed the mean over all samples stored
- **Mean Normalized Inverted Rank (MNIR)**: if the parent document of the query segment was retrieved in the top-20 documents, we stored its normalized and negated rank; then, we computed the mean over all samples stored

While the first two are typical measures in information retrieval (Croft, Metzler, and Strohman, 2010), the third one was added to have a linear evaluation of the mean rank with a "high is better" definition. In formula:

$$MNIR = \frac{1}{N}\sum_{i=1}^{N}\frac{20 - r_i}{20}$$

$$MRR = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{r_i}$$

Note that from $MNIR$ we can always compute the mean rank:

$$MNIR = \frac{1}{20N}\left[\sum_{i=1}^{N}20 - \sum_{i=1}^{N}r_i\right] = 1 - \frac{1}{20N}\sum_{i=1}^{N}r_i \implies MR = 20(1 - MNIR)$$

where $MR$ is the mean rank.

In the *genre classification* task, instead, we used the `discogs.com` info collected in *EWLD* – see chapter 4. Namely, we defined the genre of a song as the most common detected genre in *EWLD*. Then we balanced the dataset by randomly choosing 200 songs from four different genres – jazz, rock, pop and folk. We computed the **mean genre ratio**, i.e. the number of top-20 songs having the same genre of the parent song from which the query segment was extracted divided by the number of queries times 20, and the **precision** in genre classification, where the genre was detected as most common genre in the top-20 retrieved songs; we also tried to detect the segment genre as the one with highest *average precision*, where the average precision of a genre was defined as the usual average precision (Croft, Metzler, and Strohman, 2010) in which documents of that genre were considered to be relevant. Anyway results with average precision were poor.

**Table 5.1:** *Experiments nomenclature*

| | **indexed segments** | **random segments** |
|---|---|---|
| **parent song detection** | A1 | B1 |
| **genre classification** | A2 | B2 |

### 5.1.2 Indexed and random segments

We queried the system with 400 queries. In two experiments, these were collected taking 400 segments already reduced one per song extracted in a pseudo-randomly way. In the other two experiments, instead, each segment was built taking one or two consecutive random segments from a song pseudo-randomly extracted; then, if only one segment was extracted, we built a sub-segment of that, otherwise, if two segments were extracted, we joined them and in 50% of the cases we extracted a sub-segment, in the other 50% of cases we used the joined segment. Note that sub-segments were extracted taking from 1 to 4 measures, so that if the segment extracted was long just one measure, then the sub-segment was identical to its parent.

Table 5.1 resume the experiments. In the following sections, we will use the nomenclature of this table to refer to the various experiments.

## 5.2 Evaluation

### 5.2.1 Results

We built three different models for experiments of both type `1` and `2`, as described in section 5.1. In experiments `A1` and `B1`, we discarded songs without time signature – see bug in section subsection 4.2.1 – so that we used 4859 songs of the 5151 songs in the *EWLD* database; moreover, we discarded some segments that our system was not able to reduce, mainly due to erroneous file formats: we used 29517 segments and discarded 57. In experiments `A2` and `B2`, we discarded $X$ songs and 8 segments, keeping $X$ songs and about 4300 segments; a detailed view is given in Table 5.2.

We used the cluster *Blade* of the *Department of Information Engineering* of the *University of Padua* with 8 processes. The cluster is shared between different users, so that the evaluation time strongly depended upon the load of the node to which our job was assigned. Each one wrote to file a part of the dataset model, by using our custom `pickle` module and by compressing files with `lzma` compression; sizes of each chunk were quite equals. In Table 5.3 we show the cumulative file size for each model, the maximum RAM usage and the time needed to build the model.

Results are showed in tables 5.4 to 5.7.

Since *MIDI* and *semantic* functions outperformed other type of weights, we tested these two with other combinations too. More precisely, we observed how results varied by introducing a duration weight to increase the importance of long note deletion, by stopping the reduction at a note per measure – we were not convinced of the rule we used to reduce multiple measures, see subsection 3.1.2 – and the original order of scores during the deletion – Orio and Rodà used *functional, metrical* and *melodic* in this order, we instead used *functional, melodic* and *metrical*. In no case these new settings gave better results while in time

**Table 5.2:** *Models creation statistics*

| | **statistics** | | | |
|---|---|---|---|---|
| | songs used | songs skipped | segments used | segments skipped |
| **parent song detection** | | | | |
| semantic weights | 4859 | 292 | 29517 | 57 |
| MIDI weights | 4859 | 292 | 29517 | 57 |
| 1 per branch | 4859 | 292 | 29517 | 57 |
| **genre classification** | | | | |
| semantic weights | 758 | 42 | 4289 | 8 |
| MIDI weights | 761 | 39 | 4277 | 12 |
| 1 per branch | 759 | 41 | 4543 | 13 |

**Table 5.3:** *Models creation resources*

| | **resources** | | |
|---|---|---|---|
| | time | max RAM | file model size |
| **parent song detection** | | | |
| semantic weights | 1h 41m 42s | 75 GB | 98 MB |
| MIDI weights | 1h 32m 14s | 75 GB | 98 MB |
| 1 per branch | 1h 27m 54s | 75 GB | 98 MB |
| **genre classification** | | | |
| semantic weights | 17m 39s | 9 GB | 16 MB |
| MIDI weights | 16h 52m[1] | 9 GB | 16 MB |
| 1 per branch | 18m 03s | 9 GB | 16 MB |

**Table 5.4:** *Experiment A1: time is intended as averaged over all queries*

| **WEIGHTS TYPE** | | **TIME** | **P-20** | **MNIR** | **MRR** |
|---|---|---|---|---|---|
| semantic, no chords weight | | 111 s | 0.56 | 0.91 | 0.71 |
| semantic, chord weight = 0.5 | | 115 s | 0.45 | 0.91 | 0.75 |
| MIDI | | 192 s | 0.54 | 0.89 | 0.71 |
| **WEIGHTS TYPE** | **INTERVAL REPR** | | | | |
| boolean | music21 | 296 s | 0.235 | 0.48 | 0.13 |
| | GPIR | 266 s | 0.21 | 0.44 | 0.11 |
| | step-leap | 184 s | 0.19 | 0.44 | 0.12 |
| | contour | 189 s | 0.24 | 0.51 | 0.13 |
| fuzzy | music21, no chromatic | 177 s | 0.20 | 0.51 | 0.14 |
| | music21, chromatic | 166 s | 0.25 | 0.45 | 0.12 |
| | GPIR | 160 s | 0.215 | 0.51 | 0.14 |
| | step-leap | 150 s | 0.21 | 0.44 | 0.11 |
| | contour | 154 s | 0.175 | 0.515 | 0.13 |

**Table 5.5:** *Experiment B1: time is intended as averaged over all queries*

| **WEIGHTS TYPE** | | **TIME** | **P-20** | **MNIR** | **MRR** |
|---|---|---|---|---|---|
| semantic, no chords weight | | 102 s | 0.07 | 0.9 | 0.57 |
| semantic, chord weight = 0.5 | | 123 s | 0.04 | 0.8 | 0.49 |
| MIDI | | 98 s | 0.06 | 0.8 | 0.58 |
| **WEIGHTS TYPE** | **INTERVAL REPR** | | | | |
| boolean | music21 | 142 s | 0.03 | 0.5 | 0.11 |
| | GPIR | 117 s | 0.03 | 0.5 | 0.22 |
| | step-leap | 114 s | 0.03 | 0.6 | 0.16 |
| | contour | 110 s | 0.02 | 0.6 | 0.19 |
| fuzzy | music21, no chromatic | 104 s | 0.03 | 0.5 | 0.21 |
| | music21, chromatic | 111 s | 0.03 | 0.6 | 0.18 |
| | GPIR | 138 s | 0.02 | 0.5 | 0.17 |
| | step-leap | 121 s | 0.02 | 0.5 | 0.20 |
| | contour | 128 s | 0.03 | 0.6 | 0.23 |

**Table 5.6:** *Experiment A2: time is intended as averaged over all queries*

| **WEIGHTS TYPE** | | **TIME** | **GENRE RATIO** | **PREC** |
|---|---|---|---|---|
| semantic, no chords weight | | 41 s | 0.27 | 0.41 |
| semantic, chord weight = 0.5 | | 39 s | 0.27 | 0.365 |
| MIDI | | 37 s | 0.27 | 0.35 |
| **WEIGHTS TYPE** | **INTERVAL REPR** | | | |
| boolean | music21 | 42 s | 0.26 | 0.36 |
| | GPIR | 41 s | 0.26 | 0.33 |
| | step-leap | 44 s | 0.28 | 0.33 |
| | contour | 37 s | 0.27 | 0.33 |
| fuzzy | music21, chromatic | 43 s | 0.27 | 0.35 |
| | music21, no chromatic | 41 s | 0.27 | 0.33 |
| | GPIR | 41 s | 0.25 | 0.28 |
| | step-leap | 44 s | 0.27 | 0.34 |
| | contour | 43 s | 0.26 | 0.32 |

**Table 5.7:** *Experiment B2: time is intended as averaged over all queries*

| **WEIGHTS TYPE** | | **TIME** | **GENRE RATIO** | **PREC** |
|---|---|---|---|---|
| semantic, no chords weight | | 24 s | 0.25 | 0.26 |
| semantic, chord weight = 0.5 | | 27 s | 0.25 | 0.25 |
| MIDI | | 24 s | 0.26 | 0.27 |
| **WEIGHTS TYPE** | **INTERVAL REPR** | | | |
| boolean | music21 | 32 s | 0.25 | 0.26 |
| | GPIR | 29 s | 0.25 | 0.24 |
| | step-leap | 29 s | 0.25 | 0.25 |
| | contour | 31 s | 0.25 | 0.25 |
| fuzzy | music21, chromatic | 31 s | 0.24 | 0.23 |
| | music21, no chromatic | 30 s | 0.26 | 0.25 |
| | GPIR | 27 s | 0.25 | 0.25 |
| | step-leap | 24 s | 0.25 | 0.26 |
| | contour | 32 s | 0.25 | 0.27 |

**Table 5.8:** *Experiment A1 with varied parameters*

| WEIGHTS TYPE | Variation | P-20 | MNIR | MRR |
|---|---|---|---|---|
| Semantic | duration weight | 0.41 | 0.89 | 0.71 |
| | one note per measure | 0.18 | 0.78 | 0.54 |
| | new score order | 0.51 | 0.90 | 0.73 |
| MIDI | duration weight | 0.36 | 0.84 | 0.61 |
| | one note per measure | 0.16 | 0.72 | 0.48 |
| | new score order | 0.53 | 0.91 | 0.70 |

**Table 5.9:** *Experiment B1 with varied parameters*

| DISTANCE TYPE | Variation | GENRE RATIO | PREC |
|---|---|---|---|
| Semantic | duration weight | 0.27 | 0.25 |
| | one note per measure | 0.28 | 0.28 |
| | new score order | 0.27 | 0.38 |
| MIDI | duration weight | 0.27 | 0.24 |
| | one note per measure | 0.27 | 0.27 |
| | new score order | 0.28 | 0.38 |

resources no significant changes were obtained. Results are summarized in tables 5.8 and 5.9

We also tried to use only three score classes because Orio and Rodà claim this setting gives better results but we obtained no significant changes.

Finally, we checked that with interval based representations – for which we initially used not weighted internal edges and *boolean* and *fuzzy* functions for external edges – results did not change when using the same weights for internal and external edges.

### 5.2.2 Discussion

First of all it must be said that our evaluation is a feasibility evaluation of an MSR based on principles described in subsection 2.1.1. We did not perform any formal evaluation with relevance judgements and our results should be taken as indicative of the goodness of this approach. Surely, it can be improved in several ways. We will discuss about this in section 5.3.

Aside from that, results show that note based representations better fit our system in similarity tasks. Also, the reduction procedure founded on context based decisions seems to be effective, particularly with note-to-note distances which take into account the context of the note and not only the pitch. This comes out both from experiments A1 and A2, where the *semantic* weights overcomes the others. Also, the very poor results from experiments B1 ad B2 – which involved random queries – highlight the importance of the segmentation stage. We performed a non overlapping segmentation which obviously makes the system quite unusable with inter-segments queries. Maybe, in an application to discover music, one could look for a structure which is genre invariant; in this sense the normalized MIDI weights taken from Velarde, Meredith, and Weyde (2016) could be useful because in experiment A2 it retrieves 1/4 of songs from a genre and the other 3/4 from others three genres. Anyway, the distribution over genres should be checked.

It is very probable that results could be enhanced by introducing some topological structure according to which one can choose a subset of segments to compare, by example by clustering

according to the top-level note.

Another variable which has not been checked was the influence of the re-evaluation of scores: in our experiments we gave each note a score and it persisted until the last reduction; instead, a re-evaluation of scores at each reduction could change results.

However, the *frequency* at which notes were compared to compute distances and weights seemed to be not so much important but only empirical evaluation has been made about this. Moreover, we did not normalize weights over the three reduction because normalization would have been the computation of the average distance produced by the deletion of a note during a reduction procedure; but if a segment is long, more notes will be deleted and its final reduction should be more *distant*, that is less *similar*, to its original representation; by normalizing we would lose this information.

Also, since stopping reduction to one note per measure instead of to one note for all the segment gives low results, we can argue that weighted edges are a fine way to express intrinsic similarity, better than computing distances between longer segments to weight the external edges added at retrieval time.

Moreover, results in Orio and Rodà (2009) showed that *LBDM* algorithm was the worst between those tested. We can expect a certain improvement by adjusting the segmentation stage.

Finally, the *chord weight* had the opposite effect to what we wanted – i.e. a better context and stylistic measure. Anyway, it may be the *Estrada* distance to not fit well with our reduction procedure and other chord distances could enhance results.

We think that the most important lesson we can learn from these results is the importance of the musical context, included functional harmonic context. We think that this aspect should be taken into account more often and to this purpose it is needed to also use rich symbolic music format such as *MusicXML*, and not just *MIDI*.

## 5.3 Future development

As said repeatedly, this work present a first stage development of a MSR. There are many and many enhancement that can be applied. We will list below the ones that in our opinion could bring the greatest advancements.

- **Better segmentation algorithm**: we used a slightly changed version of *LBDM*, but we did not evaluated how much these changes improved the algorithm; also, we introduced many parameters that should be tested and evaluated. Moreover, other segmentation algorithms exist, and maybe these could turn out to be better than LBDM to our purposes. To a *query-by-excerpt* purpose, it would be of very help using an overlapping segmentation algorithm, e.g. by shifting all segments of a certain amount of time. Moreover, our results let suppose that harmonic context could very enhance segmentation algorithms. Also, Orio and Rodà (2009) showed that other segmentation algorithm performed better than *LBDM*.

- **Better search**: we adopted the simplest algorithm for neighbors search that one could think; many other algorithms exist, most of them based on some data structure. In our case, a topological description of the dataset graph could help, by example one could

think to set a threshold and to compare query segments only with segments whose distance is under threshold at the top level or to cluster segments on the top level.

- **Better scores and rule deletion**: other types of scores could be implemented and values could be evaluated based on cognitive studies or on their frequency in the literature. Moreover, Orio and Rodà obtained best results with only three classes for *functional* and *consonance* score. The rule deletion could be improved too, since, for example, we deleted last note instead of padding from a certain point on, see subsection 3.1.2 for an explanation of the issue.
- **Adding edges between song segments**: segments could also be compared by using some topological similarity measure referred to their relation with other segments in the song; by example, adding edges referred to transposition, inversion or retrogrades relations could help in the genre classification.
- **Other distances**: we developed many distances in the scope of this project, but they are limited in efficiency and typologies.

From an implementation point of view, we think that a major improvement could come from switching to `Jython` or `Cython` and by implementing a wrapper to store `music21` objects through arrays of basic `C` types. This would enhance the storage to file, the RAM usage and the multi-threading computations, impacting both on speed and memory usage.

# Bibliography


[1] Luigi Federico Menabrea and Ada King Countess of Lovelace. *Sketch of the Analytical Engine Invented by Charles Babbage, Esq.* Richard and John E. Taylor, 1843, p. 694 (cit. on p. 1).

[2] Carol L Krumhansl and Roger N Shepard. "Quantification of the hierarchy of tonal functions within a diatonic context." In: *Journal of experimental psychology: Human Perception and Performance* 5.4 (1979), p. 579 (cit. on p. 27).

[3] Warren Campbell and Jack Heller. "Psychomusicology & psycholinguistics: Parallel paths or separate ways." In: *Psychomusicology: A Journal of Research in Music Cognition* 1.2 (1981), p. 3 (cit. on p. 3).

[4] D. De La Motte. *Kontrapunkt: ein Lese- und Arbeitsbuch*. 1981 (cit. on p. 20).

[5] Diana Deutsch and John Feroe. "The internal representation of pitch sequences in tonal music." In: *Psychological review* 88.6 (1981), p. 503 (cit. on p. 5).

[6] F. Lerdahl and R.S. Jackendoff. *A Generative Theory of Tonal Music*. MIT Press series on cognitive theory and mental representation. MIT Press, 1985 (cit. on pp. 5, 8, 9).

[7] L.B. Meyer. *Style and Music: Theory, History, and Ideology*. Studies in the criticism and theory of music. University of Chicago Press, 1989 (cit. on p. 2).

[8] Jean-Jacques Nattiez and Katharine Ellis. "Reflections on the Development of Semiology in Music". In: *Music Analysis* 8.1/2 (1989), pp. 21–75 (cit. on p. 3).

[9] R. Middleton. *Studying Popular Music*. McGraw-Hill Education, 1990 (cit. on p. 2).

[10] Jean Molino, JA Underwood, and Craig Ayrey. "Musical fact and the semiology of music". In: *Music analysis* (1990), pp. 105–156 (cit. on p. 3).

[11] Jean-Jacques Nattiez. *Music and discourse: Toward a semiology of music*. Princeton University Press, 1990 (cit. on p. 3).

[12] Eugene Narmour. *The analysis and cognition of melodic complexity: The implication-realization model*. University of Chicago Press, 1992 (cit. on p. 5).

[13] Rita Aiello. "Music and Language: Parallels and Contrasts". In: *Musical Perceptions*. Ed. by Rita Aiello and John A. Sloboda. Oxford University Press, 1994, pp. 40–63 (cit. on p. 2).

[14] Antonio Camurri, Marcello Frixione, and Carlo Innocenti. "A Cognitive Model and a Knowledge Representation System for Music and Multimedia". In: *Journal of New Music Research* 23.4 (1994), pp. 317–347 (cit. on p. 5).

[15] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995 (cit. on pp. 31, 32).

[16] Emilios Cambouropoulos. "A general pitch interval representation: Theory and applications". In: *Journal of New Music Research* 25.3 (1996), pp. 231–251 (cit. on pp. 7, 23).

[17] MIDI Manufacturers Association MMA. "The complete MIDI 1.0 detailed specification version 96.1". In: *La Habra: MIDI Manufacturers Association* (1996) (cit. on p. 6).

[18] Richard Cohn. "Neo-riemannian operations, parsimonious trichords, and their" tonnetz" representations". In: *Journal of Music Theory* 41.1 (1997), pp. 1–66 (cit. on pp. 5, 7).

[19] E Glenn Schellenberg. "Simplifying the implication-realization model of melodic expectancy". In: *Music Perception: An Interdisciplinary Journal* 14.3 (1997), pp. 295–318 (cit. on p. 5).

[20] Elaine Chew. "Towards a mathematical model of tonality". PhD thesis. Massachusetts Institute of Technology, 2000 (cit. on p. 8).

[21] Alan Marsden. *Music, intelligence and artificiality.* 2000 (cit. on p. 5).

[22] Emilios Cambouropoulos. "The Local Boundary Detection Model (LBDM) and its Application in the Study of Expressive Timing." In: *ICMC*. 2001 (cit. on pp. 19, 23, 24).

[23] Alan Marsden. "Representing melodic patterns as networks of elaborations". In: *Computers and the Humanities* 35.1 (2001), pp. 37–54 (cit. on p. 13).

[24] Thomas J. Mathiesen, Dimitri Conomos, George Leotsakos, Sotirios Chianis, and Rudolph M. Brandl. *Greece*. 2001 (cit. on p. 2).

[25] Elaine Chew. "The spiral array: An algorithm for determining key boundaries". In: *ICMAI*. Springer. 2002, pp. 18–31 (cit. on p. 8).

[26] David Rizo Valero and José Manuel Iñesta Quereda. "Tree-structured representation of melodies for comparison and retrieval". In: *Int. Workshop on Pattern Recognition in the Information Society, PRIS 2002*. 2002, pp. 140–155 (cit. on pp. 10, 11).

[27] Xavier Serra. "The Musical Communication Chain and its Modeling". In: *Mathematics and Music : A Diderot Mathematical Forum*. Ed. by Gerard Assayag, Hans Georg Feichtinger, and Jose Francisco Rodrigues. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 243–255 (cit. on p. 3).

[28] Emilios Cambouropoulos. "Pitch spelling: A computational model". In: *Music Perception: An Interdisciplinary Journal* 20.4 (2003), pp. 411–429 (cit. on p. 6).

[29] Elaine Chew and Alexandre R.J. Francois. "MuSA.RT: Music on the Spiral Array. Realtime". In: *Proceedings of the Eleventh ACM International Conference on Multimedia*. MULTIMEDIA '03. Berkeley, CA, USA: ACM, 2003, pp. 448–449 (cit. on p. 8).

[30] David Rizo Valero, José Manuel Iñesta Quereda, and Francisco Moreno-Seco. "Tree-structured representation of musical information". In: *Pattern Recognition and Image Analysis* (2003), pp. 838–846 (cit. on p. 10).

[31] George Tzanetakis, Andrey Ermolinskyi, and Perry Cook. "Pitch histograms in audio and symbolic music information retrieval". In: *Journal of New Music Research* 32.2 (2003), pp. 143–152 (cit. on p. 7).

[32] Hugues Vinet. "The representation levels of music information". In: *International Symposium on Computer Music Modeling and Retrieval*. Springer. 2003, pp. 193–209 (cit. on p. 2).

[33] Alan Marsden. "Extending a network-of-elaborations representation to polyphonic music: Schenker and species counterpoint." In: *Sound and Music Computing'04* (2004) (cit. on p. 13).

[34] Gerhard Widmer and Werner Goebl. "Computational Models of Expressive Music Performance: The State of the Art". In: *Journal of New Music Research* 33.3 (2004), pp. 203–216 (cit. on p. 5).

[35] Elaine Chew and Yun-Ching Chen. "Real-time pitch spelling using the spiral array". In: *Computer Music Journal* 29.2 (2005), pp. 61–76 (cit. on pp. 6, 8).

[36] Alan Marsden. "Generative structural representation of tonal music". In: *Journal of New Music Research* 34.4 (2005), pp. 409–428 (cit. on pp. 5, 13).

[37] Christopher Harte, Mark Sandler, and Martin Gasser. "Detecting harmonic change in musical audio". In: *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*. ACM. 2006, pp. 21–26 (cit. on p. 7).

[38] Cory McKay and Ichiro Fujinaga. "jSymbolic: A Feature Extractor for MIDI Files." In: *ICMC*. 2006 (cit. on p. 7).

[39] David Meredith. "The ps13 pitch spelling algorithm". In: *Journal of New Music Research* 35.2 (2006), pp. 121–159 (cit. on p. 6).

[40] David Rizo Valero, José Manuel Iñesta Quereda, and Pedro J Ponce de León. "Tree Model of Symbolic Music for Tonality Guessing." In: *Artificial Intelligence and Applications*. Vol. 2006. 2006, pp. 299–304 (cit. on pp. 10, 25).

[41] Fred Lerdahl and Carol L. Krumhansl. "Modeling tonal tension". In: *Music Perception: An Interdisciplinary Journal* 24.4 (2007), pp. 329–366 (cit. on pp. 19, 27).

[42] Alberto Pinto, Reinier H Van Leuken, M Fatih Demirci, Frans Wiering, and Remco C Veltkamp. "Indexing music collections through graph spectra". In: *Proc. of the 8th International Conference on Music Information Retrieval (ISMIR'07)*. 2007, pp. 153–156 (cit. on pp. 10, 11).

[43] D. Schön, L. Akiva-Kabiri, and T. Vecchi. *Psicologia della musica*. Bussole: Psicologia. Carocci, 2007 (cit. on p. 2).

[44] David Temperley. *Music and probability*. Mit Press, 2007 (cit. on p. 5).

[45] Clifton Callender, Ian Quinn, and Dmitri Tymoczko. "Generalized Voice-Leading Spaces". In: *Science* 320.5874 (2008), pp. 346–348 (cit. on p. 5).

[46] Elaine Chew. "Music and operations research-the perfect match". In: *OR/MS Today* 35 (2008), pp. 26–31 (cit. on p. 8).

[47] Masatoshi Hamanaka, Keiji Hirata, and Satoshi Tojo. "Melody Morphing Method Based on GTTM." In: *ICMC*. 2008 (cit. on pp. 9, 25).

[48] Charles Inskip, Andrew MacFarlane, and Pauline Rafferty. “Meaning, communication, music: towards a revised communication model”. In: *Journal of Documentation* 64.5 (2008), pp. 687–706 (cit. on p. 3).

[49] David Rizo Valero, José Manuel Iñesta Quereda, and Kjell Lemström. “Tree Representation in Combined Polyphonic Music Comparison.” In: *CMMR*. Springer. 2008, pp. 177–195 (cit. on p. 10).

[50] Nicola Orio and Antonio Rodà. “A Measure of Melodic Similarity Based on a Graph Representation of the Music Structure”. In: (2009) (cit. on pp. 11, 16, 19, 21, 25–27, 47, 50, 53–55).

[51] Geraint A Wiggins, Marcus T Pearce, Daniel Müllensiefen, et al. “Computational modeling of music cognition and musical creativity”. In: *The Oxford Handbook of Computer Music*. Ed. by Roger T. Rean. Oxford University Press, 2009 (cit. on p. 5).

[52] W Bruce Croft, Donald Metzler, and Trevor Strohman. *Search engines: Information retrieval in practice*. Vol. 283. Addison-Wesley Reading, 2010. Chap. 8.4.3 (cit. on p. 49).

[53] Michael Scott Cuthbert and Christopher Ariza. “music21: A toolkit for computer-aided musicology and symbolic music data”. In: (2010) (cit. on pp. 31, 36).

[54] Carol L. Krumhansl and Lola L. Cuddy. “A Theory of Tonal Hierarchies in Music”. In: *Music Perception*. Ed. by Mari Riess Jones, Richard R. Fay, and Arthur N. Popper. New York, NY: Springer New York, 2010, pp. 51–87 (cit. on pp. 19, 27).

[55] Alan Marsden. “Recognition of variations using automatic Schenkerian reduction.” In: *International Society for Music Information Retrieval (ISMIR 2010)* (2010) (cit. on pp. 10, 16, 25).

[56] Alan Marsden. “Schenkerian Analysis by Computer: A Proof of Concept”. In: *Journal of New Music Research* 39.3 (2010), pp. 269–289 (cit. on pp. 5, 11, 25).

[57] David Rizo Valero. “Symbolic music comparison with tree data structures”. PhD thesis. 2010 (cit. on p. 10).

[58] Thomas Rocher, Matthias Robine, Pierre Hanna, and Myriam Desainte-Catherine. “A Survey Of Chord Distances With Comparison For Chord Analysis.” In: *ICMC*. 2010 (cit. on p. 30).

[59] Alan Marsden. “Software for Schenkerian analysis”. In: *International Computer Music Conference*. 2011 (cit. on p. 10).

[60] Eric J Humphrey, Taemin Cho, and Juan P Bello. “Learning a robust tonnetz-space transform for automatic chord recognition”. In: *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE. 2012, pp. 453–456 (cit. on p. 7).

[61] Marcelo Rodrguez López and Anja Volk. “Automatic Segmentation of Symbolic Music Encodings: A Survey”. In: (2012) (cit. on p. 18).

[62] P. Tagg. *Music’s Meanings: A Modern Musicology for Non-musos*. Mass Media Music Scholars’ Press, 2012 (cit. on p. 3).

[63] David Temperley. “Computational models of music cognition”. In: *The psychology of music* (2012), pp. 327–368 (cit. on p. 10).

[64] Dmitri Tymoczko. “The generalized tonnetz”. In: *Journal of Music Theory* 56.1 (2012), pp. 1–52 (cit. on p. 5).

[65] Diana Deutsch, William F Thompson, Henkjan Honing, and Stephen McAdams. *Psychology of music*. Elsevier, 2013. Chap. 2, 4, 6, 7, 9 (cit. on p. 1).

[66] Keiji Hirata, Satoshi Tojo, and Masatoshi Hamanaka. “Cognitive Similarity grounded by tree distance from the analysis of K. 265/300e”. In: *International Symposium on Computer Music Modeling and Retrieval*. Springer. 2013, pp. 589–605 (cit. on p. 9).

[67] François Pachet and Jeff Suzda. “A Comprehensive Online Database of Machine-Readable Lead-Sheets for Jazz Standards.” In: 2013 (cit. on p. 40).

[68] Richard Parncutt, Erica Bisesi, and Anders Friberg. “A Preliminary Computational Model of Immanent Accent Salience in Tonal Music”. In: *Proceedings of the Sound and Music Computing Conference*. 2013, pp. 335–340 (cit. on p. 21).

[69] E. Glenn Schellenberg and Michael W. Weiss. “12 - Music and Cognitive Abilities”. In: *The Psychology of Music (Third Edition)*. Ed. by Diana Deutsch. Third Edition. Academic Press, 2013, pp. 499–550 (cit. on p. 14).

[70] Mohammad Reza Abbasifard, Bijan Ghahremani, and Hassan Naderi. “A survey on nearest neighbor search methods”. In: *International Journal of Computer Applications* 95.25 (2014) (cit. on p. 48).

[71] Elaine Chew. “Mathematical and computational modeling of tonality”. In: *AMC* 10 (2014), p. 12 (cit. on pp. 5, 8).

[72] Franco Fabbri. “Suoni e segni. Un resoconto”. In: *Musica/Realtà* 103 (2014) (cit. on p. 3).

[73] David Hiley and Janka Szendrei. *Notation*. 2014 (cit. on p. 2).

[74] Masaki Matsubara, Keiji Hirata, and Satoshi Tojo. “Distance in Pitch Sensitive Timespan Tree”. In: (2014) (cit. on p. 9).

[75] Nicholas Harley. “An ontology for abstract, hierarchical music representation”. In: *Demo at the 16th International Society for Music Information Retrieval Conference (ISMIR 2015), Malaga, Spain*. 2015 (cit. on p. 12).

[76] Emilios Cambouropoulos. “The Harmonic Musical Surface and Two Novel Chord Representation Schemes”. In: *Computational Music Analysis*. Ed. by David Meredith. Cham: Springer International Publishing, 2016, pp. 31–56 (cit. on p. 7).

[77] Masatoshi Hamanaka, Keiji Hirata, and Satoshi Tojo. *Implementing Methods for Analysing Music Based on Lerdahl and Jackendoff's Generative Theory of Tonal Music*. 2016 (cit. on pp. 9–11).

[78] Alan Marsden and David Rizo. “An MEI module proposal for hierarchical analysis”. In: *Music Encoding Conference*. 2016 (cit. on p. 12).

[79] David Meredith. *Computational music analysis*. Springer, 2016 (cit. on p. 5).

[80] Gissel Velarde, David Meredith, and Tillman Weyde. *A Wavelet-Based Approach to Pattern Discovery in Melodies.* 2016 (cit. on pp. 18, 30, 53).

[81] MakeMusic and Micheal Good. "MusicXML". In: (2017) (cit. on p. 6).

[82] Gerhard Widmer. "Getting closer to the essence of music: The con espressione manifesto". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 8.2 (2017), p. 19 (cit. on pp. 2, 7).