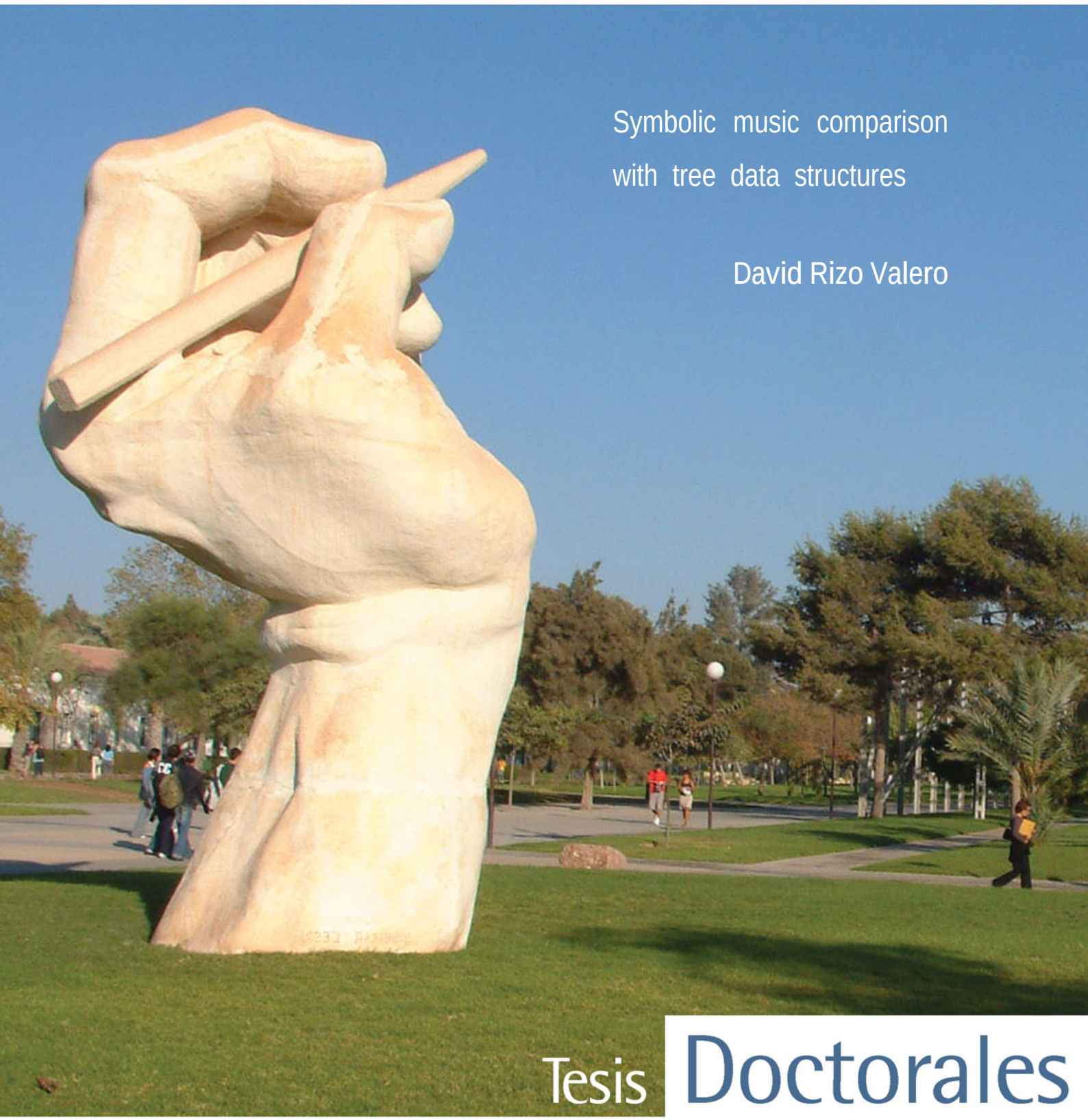
Universitat d'Alacant
Universidad de Alicante



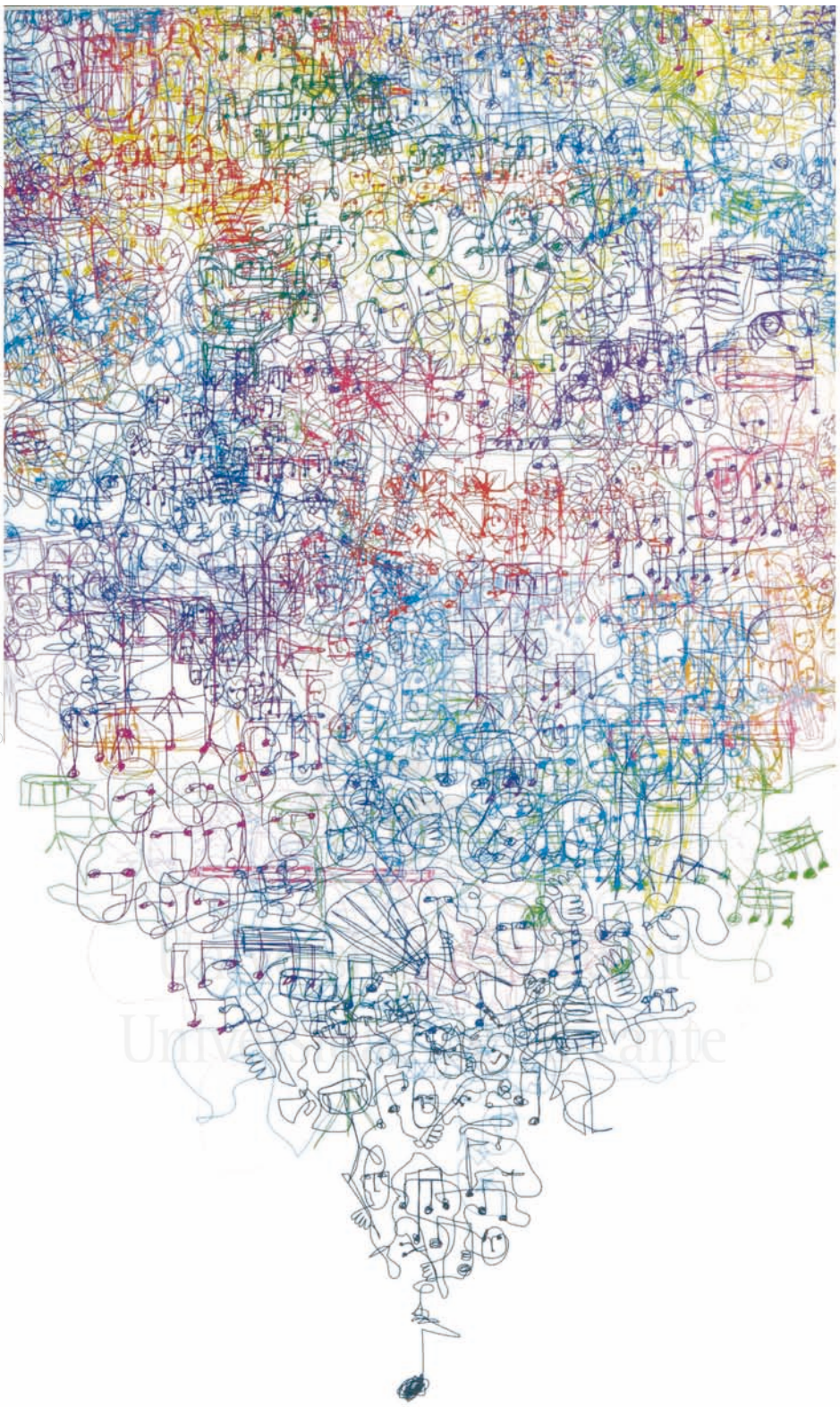
Symbolic music comparison with tree data structures

David Rizo Valero

Tesis Doctorales

www.eltallerdigital.com

UNIVERSIDAD de ALICANTE



Design by Henderson Bromsted for the Winston-Salem Symphony

Symbolic music comparison with tree data structures

Ph.D. Thesis by David Rizo Valero

Supervised by José Manuel Iñesta Quereda | 2010 Universidad de Alicante

Ph.D. Thesis

Symbolic music comparison with tree data structures

Author:
David Rizo Valero

Supervisor:
José M. Iñesta Quereda

Universitat d'Alacant
Universidad de Alicante
Departament de Llenguatges i Sistemes Informàtics
Departamento de Lenguajes y Sistemas Informáticos

August 2010

Para Gloria, Pablo y Quique

Agradecimientos

La elaboración de una tesis es un trabajo ilusionante y arduo, con momentos agradables y difíciles, pero que sobre todo es extenso en el tiempo. Una tarea de estas dimensiones no puede llevarse a cabo por una sola persona aislada. Sin la guía, apoyo científico, técnico, económico y moral de todas las personas e instituciones que a continuación detallo, esta tesis no habría salido adelante.

Quiero dar gracias especialmente al director de esta tesis, José Manuel Iñesta. Él ha hecho posible que se estableciera y consolidara nuestro grupo de investigación en informática musical, haciendo realidad mi sueño personal de investigar en este campo, ha participado activamente en todos y cada uno de los trabajos de investigación en los que he trabajado, ha facilitado la posibilidad de realizar las estancias de investigación en el extranjero, y sobre todo, me ha hecho creer que esta tesis tenía sentido.

La interacción con los miembros del Grupo de Investigación y Reconocimiento de Formas e Inteligencia Artificial del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante ha enriquecido el contenido de esta investigación. José Oncina, María Luisa Micó, Jorge Calera, Juan Ramón Rico, Rafael Carrasco y Paco Moreno han resuelto muchas de las dudas que se me han ido surgiendo durante este largo proceso. Debo agradecer el apoyo moral y mucho de lo que he aprendido a mis compañeros del Laboratorio de Informática Musical: Pierre León, junto a quien comencé esta andadura, Antonio Pertusa, Carlos Pérez-Sancho, Plácido R. Illescas, José Bernabeu, Tomás Pérez, Pedro Cruz, Tico Agoiz y Javier Sober quien ha hecho posible el corpus *COVERS*.

A las personas que me han acogido en los centros de investigación en los que he realizado estancias les quiero reconocer su trato y ayuda. A Amaury Habrard, Marc Sebban, Colin de la Higuera en la estancia en Saint-Étienne. A Kjell Lemström que me ayudó a extender el sistema a polifónico en Helsinki. A Carlos Agón, Karim Haddad y Jean Bresson que me facilitaron la programación para OpenMusic en el IRCAM de París.

Durante el proceso de elaboración de la tesis muchos investigadores han respondido a mis consultas y peticiones de trabajos y software: Rafael Ramírez, Maarten Grachten y Mohamed Sordo por el analizador de Narmour, Iman Suyoto por *fanimae*, a Daniel Mullensiefen por *SMILE*, Rainer Typke por su ayuda con los datos del *MIREX*, de nuevo Kjell Lemström y Niko Mikkilä por adaptar *C-Brahms* para mis necesidades, Alberto Pinto por el código fuente de su sistema basado en grafos, Tao Jiang por sus consejos sobre la comparación de árboles, Alan Marsden por aclarar ciertos aspectos sobre su sistema de análisis schenkeriano, Masatoshi Hamanaka por darme acceso a *FATTA*, su implementación del GTTM.

Agradezco la disponibilidad de Mrs. E. Merritt Vale, presidenta y CEO de la Winston-Salem Symphony al permitirme usar la ilustración de la portada realizado por la firma Henderson Bromsted Art.

El conjunto de publicaciones que soportan este trabajo de investigación ha sido financiado a lo largo de los años por los proyectos:

- DRIMS: Descripción y recuperación de información musical y sonora. Ministerio de Ciencia e Innovación. TIN2009-14247-C02-02
- MIPRCV: Multimodal Interaction in Pattern Recognition and Computer Vision. Ministerio de Educación y Ciencia (Consolider Ingenio 2010). CSD2007-00018
- Acc. Int. E-A: Clasificación de géneros musicales combinando descriptores de audio y simbólicos mediante un sistema de transcripción automática. Ministerio de Educación y Ciencia. HU2007-0025
- Pump Priming: Learning Stochastic Edit Distances from Structured Data. Application in Music Retrieval. PASCAL Network of Excellence. Unión Europea
- PROSEMUS: Procesamiento semántico de música digital. Ministerio de Educación y Ciencia. TIN2006-14932-C02-02.
- GV-JMI: Transferencia de tecnologías de categorización de textos a aplicaciones de recuperación de información en música digital. Generalitat Valenciana. GV04B-541
- TIRIG: Tratamiento, indexación y recuperación de información en grandes volúmenes de datos mediante técnicas de reconocimiento de formas: aplicaciones en descripción y clasificación de datos secuenciales. CICYT. 2003-08496-C04-04

Finalmente, el mayor agradecimiento va a mi familia, aquellos que no sólo se han sacrificado para que yo pueda gozar del tiempo necesario tras el trabajo y en época de vacaciones para hacer esto posible sino que también me han alentado para que llegara a buen puerto este proyecto: Gloria, mi mujer, de paciencia infinita, Pablo, que por iniciativa propia fue incluso capaz de soportar el dolor de una caída en la escuela de verano para que yo acabara unos experimentos durante el verano pasado, Quique, que se ha tenido que conformar con jugar un poco menos con su padre para que éste le dedicara el tiempo al ordenador, Victorino, Joaquina y Manolita, que han hecho las funciones de padre cuando he estado ausente.

A todos, gracias

Contents

List of Figures

List of Tables

1 Introduction

Nowadays, the availability of huge amount of music stored in digital format in public or personal collections creates problems for their organization and for locating information inside of them. Additionally, sometimes the name of the files in those collections or the meta-data describing them are inaccurate or even non-existent. This is why the user often needs to listen each file or visualize their score or other graphical representation, what makes even more inefficient the use of those collections.

It arises the necessity of tools able to classify and allow the user to look for music given its content. These tools belong to the pattern recognition domain in what is known as Music Information Retrieval (MIR).

MIR includes all disciplines that manipulate musical related information to give digital systems users answers to questions about the music itself or its organization, either through its content in a variety of formats or from the meta-data that describes it. Several classifications of those MIR tasks have been proposed so far from different points of view (Downie, 2003; Futrelle and Downie, 2003; Orio, 2006; Reiss and Sandler, 2002; Taheri-Panah and MacFarlane, 2004; Typke et al., 2005a; Uitdenbogerd, 2002). Roughly they can be grouped in the following categories:

- representation of music information and signals
- comparison and searching
- recommendation
- classification
- transcription from digital audio into musical scores
- musical information discovering from raw data
- user interfaces
- optical music recognition
- music social networks
- legal issues

- ethnomusicology

Our work is framed in the MIR domain, specifically in the representation of music information and its comparison. In this dissertation a new representation of music in symbolic format by means of trees is proposed along with the methods to evaluate the similarity between two songs using this representation.

The use of those comparison methods solves the problem of classifying music by its content, in particular finding duplicates or similar works and the search by musical content. The applications range from the study and analysis tasks in musicology to the detection of plagiarism, useful to protect copyrights in the music record industry. Some of them are listed just below:

Digital libraries

One common problem in personal, academic, or other kind of digital libraries is the presence of the same musical piece with slight differences such as distinct sample rates in digital audio, or duplicate tracks in MIDI files that make them different in file size. These little variations make impossible for tools that compare files at bit level like the Unix *diff* to identify them as equal. Other usual problem is the existence of live versions and covers of songs, that should be catalogued as such.

Song interrelation Some pieces refer to others including material from the latter in so-called *cites* or *allusions* (Sadie, 2000). Known examples might be the inclusion of some notes of the French national anthem into the Tchaikovsky's 1812 Overture or the allusions to Rossini's William Tell Overture from the 15th Shostakovich's symphony first movement. On the other hand, the variation form (e.g., Bach's Goldberg Variations), or the mere development of a theme throughout the movements of a symphonic work interrelate individual pieces as a part of a whole. For a digital library, it is very interesting to be able to automatically construct a graph of interrelations to allow a hyper-medial browsing of the musical database.

Search One of the most prominent applications of the music similarity computation is the querying of a sequence of notes against a database of songs to obtain the pieces that most resemble the query, allowing to look for a song whose name is unknown. If the query is obtained by introducing a sequence of notes through a keyboard, by singing or humming the discipline is named with the generic name Query-By-Melody (QBM) (Doraisamy, 2004). In the case of humming, the discipline is named Query-By-Humming (QBH) (Ghias et al., 1995). If the query is a song in digital audio format given as an example, or it is obtained using Optical Music Recognition (OMR) (Bellini et al., 2007; Choudhury et al., 2000) it is denoted as Query-By-Example (QBE) (Tzanetakis and Cook, 1999).

Plagiarism detection

Music copyright infringement cases use to be founded on expert opinions on the similarity between the two works being litigated. In many cases one sequence of notes that is said to be the object of the copy occurs in so many pieces that cannot be taken as original and thus is not likely of being the subject of a plagiarism case. In order to allow lawyers to have this information, musical databases must be available with tools that automatically discover and measure the similarity between two works and the originality among the rest (Cronin, 2002). It is then necessary for this kind of systems to have a precise similarity measurement tool.

Composer Composers, in their creation process, elaborate some basic ideas that come to their minds by inspiration or from other means like an automated random process. The problem comes when this inspiration is not such, but it is a memory of another known song. A similarity computation tool in a music database can inform the composer that the idea is actually original, or at least, it is not found in the database.

Ethnomusicologists

Selfridge-Field (1998) poses the necessity of tools so that "music historians locate similar tunes distinguished by modular substitutions (as in the transmission of early chant), re-texting (as in masses that parody motets), consolidation of parts (as in lute or keyboard transcriptions of chansons), and elaborations (as in divisions, diminutions, and variations)".

Motive extraction

For visualizing huge sets of works one cannot plot on-screen the whole score of each work, nor play from start to end the pieces. Instead, one representative portion of each score uses to be shown or played. This little portion of music is usually named as *theme*, and most of the times is a repeated motive of the musical piece [1]. The automatic calculation of the motives requires a precise and flexible similarity measure to detect the possibly altered repetitions of the main material. The automated motive extraction is also useful to speed up some digital libraries tasks described above: it is faster to compare only the main material of two songs than look into the complete pieces.

[1] In (Selfridge-Field, 1998) the following terms are stated:

theme Random portion of the work that enjoys the greatest melodic importance (e.g. Parsons (1975), Barlow and Morgenstern (1978) catalogue.)

incipit Samples from start of a movement or a work (e.g. RISM (ris, 2005), LaRue (1988) symphony catalogue). Used mainly for folk-songs and early repertories, where themes maybe coincident with incipits.

The MIR community uses to categorize musical data into *digital audio*, *metadata*, *printed music*, and *symbolic representations*. The first one stands for all file formats that store recordings with the actual sound information after its digitization (see Fig. 1.2) in well known formats like *wav*, *aiff*, *ogg*, or *mp3*. Those files may contain any kind of sound, from the noise of a car to music, thus, if one wants to use musicological assumptions this musical information must be inferred from the raw data. The *metadata* has information referent to the song like performer, composer, and title. As *printed music* we understand any graphical representation of a musical score, sheet music that a person can read but for the computer is only a picture. Finally the *symbolic representations* include scores in digital formats that contain information about notes, clefs, staffs, etc. that must be rendered by a notation application in order to generate sheet music. Besides, this music format can be played either by a person (after being rendered) or by a machine. Examples of these kind of score formats are *MusicXML* (Good and Actor, 2003), *SCORE* (Selfridge-Field, 1997), *kern* (Huron, 2002), or *GUIDO* (Hoos et al., 1998). Music can be also represented by the recording of an interpretation in *symbolic format*. In those formats the operations that make music sound are recorded, like the onset and offset of notes, its volume, or the pressure on a keyboard key. The most extended format for this is the *Standard MIDI files (SMF)* (MID, 1996). See Figs. 1.3 and 1.4 for two visualizations of the same music performance information recorded in MIDI files.

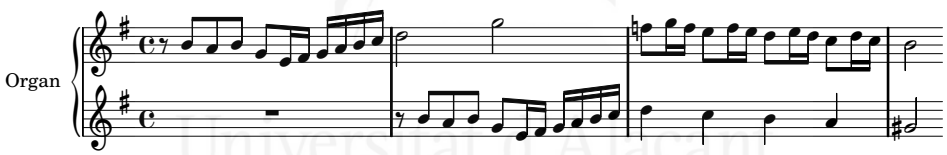


Figure 1.1: Juego de versos para Psalmodia de tono 3º, 5 Vs, by Juan Bautista Cabanilles (1644-1712), in common music notation

We are interested in the two *symbolic representations*, both the scores and the symbolic representation of played music, because we want to investigate into musical similarity using musical terms (notes, degrees, etc...), and symbolic data is the representation that is closest to that kind of information. In the score, the actual notes with possibly additional information can be found explicitly, whereas in digital audio (Pertusa and Iñesta, 2008; Plumbley et al., 2002) or after the use of an OMR (Bellini et al., 2007) they must be detected, possibly with mistakes. We want to leave aside timbre to be able to manage only notes and structures. It is worth to remark that transcription from audio and sheet music is being considerably improved and in the future the extraction of the music elements needed from those media will be as accurate as obtaining them from a symbolic source.

Nowadays, music similarity methods are coarsely divided into those that work with monophonic music and those able to process polyphonic music, directly or by converting it previously into monophonic.

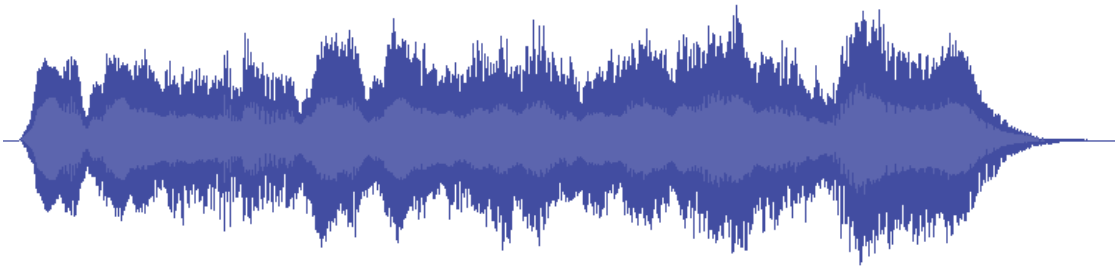


Figure 1.2: Raw audio corresponding to score in Fig. 1.1

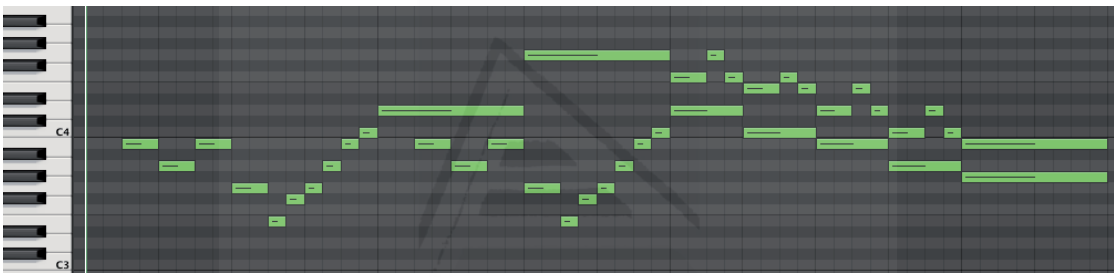


Figure 1.3: Piano roll extracted corresponding to score in Fig. 1.1

The New Grove Dictionary of Music and Musicians (Sadie, 2000) defines the term *polyphony* as: "A term used to designate various important categories in music: namely, music in more than one part, music in many parts, and the style in which all or several of the musical parts move to some extent independently. *Polyphonos* (many-voiced) and *polyphonia* occur in ancient Greek without any connotations of musical technique. After classical antiquity, forms of the adjective came into use in modern languages, designating both non-musical phenomena such as birdcalls, human speech and multiple echoes, and musical phenomena such as instrumental range and tonal variety, as well as the various tunes playable on an automatic musical device".

Thus, the term polyphony can be regarded from two different points of view: the simultaneity of notes or the concurrent voices. From the first point of view, the difference resides in the maximum number of notes played at a time, if this number is one, the work is monophonic (see Fig. 1.5), being polyphonic otherwise (see Fig. 1.1). From the other approach, the polyphony depends on the number of simultaneous voices, and not only in the number of simultaneous notes (Meudic, 2003; Pickens, 2004). For example there are numerous examples of several voices being produced simultaneously by one instrument with an only a note played at a time, like the Sonatas and Partitas for solo violin by J.S. Bach. Under this perspective, another kind of music can be introduced: the homophonic

```
DivisionType: PPQ
Resolution: 1024 ticks per beat

------------------------
Track 0:
------------------------
tick 0: SMTPE Offset: 0:0:0.0.0
tick 0: Time Signature: 4/4
tick 0: Key Signature: G major
tick 0: Set Tempo: 120.0 bpm
tick 16384: End of Track
------------------------
Track 1:
------------------------
tick 0: Meta event:  51 75 69 63 6B 54 69 6D 65 20 31
tick 0: Sequence/Track Name: Organ
tick 0: [C0 13] chnl 1: program change 19
tick 0: [B0 07 65] chnl 1: control change 7 value: 101
tick 0: [B0 0A 40] chnl 1: control change 10 value: 64
tick 512: [90 47 40] chnl 1: note On B4 veloc.: 64
tick 1024: [80 47 00] chnl 1: note Off B4 veloc.: 0
tick 1024: [90 45 40] chnl 1: note On A4 veloc.: 64
tick 1536: [80 45 00] chnl 1: note Off A4 veloc.: 0
tick 1536: [90 47 40] chnl 1: note On B4 veloc.: 64
tick 2048: [80 47 00] chnl 1: note Off B4 veloc.: 0
tick 2048: [90 43 40] chnl 1: note On G4 veloc.: 64
tick 2560: [80 43 00] chnl 1: note Off G4 veloc.: 0
tick 2560: [90 40 40] chnl 1: note On E4 veloc.: 64
tick 2816: [80 40 00] chnl 1: note Off E4 veloc.: 0
tick 2816: [90 42 40] chnl 1: note On F#4 veloc.: 64
tick 3072: [80 42 00] chnl 1: note Off F#4 veloc.: 0
tick 3072: [90 43 40] chnl 1: note On G4 veloc.: 64
tick 3328: [80 43 00] chnl 1: note Off G4 veloc.: 0
tick 3328: [90 45 40] chnl 1: note On A4 veloc.: 64
tick 3584: [80 45 00] chnl 1: note Off A4 veloc.: 0
tick 3584: [90 47 40] chnl 1: note On B4 veloc.: 64
tick 3840: [80 47 00] chnl 1: note Off B4 veloc.: 0
tick 3840: [90 48 40] chnl 1: note On C5 veloc.: 64
tick 4096: [80 48 00] chnl 1: note Off C5 veloc.: 0
tick 4096: [90 4A 40] chnl 1: note On D5 veloc.: 64
tick 4608: [90 47 40] chnl 1: note On B4 veloc.: 64
tick 5120: [80 47 00] chnl 1: note Off B4 veloc.: 0
tick 5120: [90 45 40] chnl 1: note On A4 veloc.: 64
tick 5632: [80 45 00] chnl 1: note Off A4 veloc.: 0
tick 5632: [90 47 40] chnl 1: note On B4 veloc.: 64
tick 6144: [80 4A 00] chnl 1: note Off D5 veloc.: 0
tick 6144: [90 4F 40] chnl 1: note On G5 veloc.: 64
tick 6144: [80 47 00] chnl 1: note Off B4 veloc.: 0
tick 6144: [90 43 40] chnl 1: note On G4 veloc.: 64
tick 6656: [80 43 00] chnl 1: note Off G4 veloc.: 0
tick 6656: [90 40 40] chnl 1: note On E4 veloc.: 64
tick 6912: [80 40 00] chnl 1: note Off E4 veloc.: 0
tick 6912: [90 42 40] chnl 1: note On F#4 veloc.: 64
tick 7168: [80 42 00] chnl 1: note Off F#4 veloc.: 0
tick 7168: [90 43 40] chnl 1: note On G4 veloc.: 64
tick 7424: [80 43 00] chnl 1: note Off G4 veloc.: 0
tick 7424: [90 45 40] chnl 1: note On A4 veloc.: 64
tick 7680: [80 45 00] chnl 1: note Off A4 veloc.: 0
tick 7680: [90 47 40] chnl 1: note On B4 veloc.: 64
tick 7936: [80 47 00] chnl 1: note Off B4 veloc.: 0
tick 7936: [90 48 40] chnl 1: note On C5 veloc.: 64
tick 8192: [80 4F 00] chnl 1: note Off G5 veloc.: 0
tick 8192: [80 48 00] chnl 1: note Off C5 veloc.: 0
tick 8192: [90 4D 40] chnl 1: note On F5 veloc.: 64
tick 8192: [90 4A 40] chnl 1: note On D5 veloc.: 64
tick 8704: [80 4D 00] chnl 1: note Off F5 veloc.: 0
tick 8704: [90 4F 40] chnl 1: note On G5 veloc.: 64
tick 8960: [80 4F 00] chnl 1: note Off G5 veloc.: 0
tick 8960: [90 4D 40] chnl 1: note On F5 veloc.: 64
tick 9216: [80 4D 00] chnl 1: note Off F5 veloc.: 0
tick 9216: [90 4C 40] chnl 1: note On E5 veloc.: 64
tick 9216: [80 4A 00] chnl 1: note Off D5 veloc.: 0
tick 9216: [90 48 40] chnl 1: note On C5 veloc.: 64
tick 9728: [80 4C 00] chnl 1: note Off E5 veloc.: 0
tick 9728: [90 4D 40] chnl 1: note On F5 veloc.: 64
tick 9984: [80 4D 00] chnl 1: note Off F5 veloc.: 0
tick 9984: [90 4C 40] chnl 1: note On E5 veloc.: 64
tick 10240: [80 4C 00] chnl 1: note Off E5 veloc.: 0
tick 10240: [90 4A 40] chnl 1: note On D5 veloc.: 64
tick 10240: [80 48 00] chnl 1: note Off C5 veloc.: 0
tick 10240: [90 47 40] chnl 1: note On B4 veloc.: 64
tick 10752: [80 4A 00] chnl 1: note Off D5 veloc.: 0
tick 10752: [90 4C 40] chnl 1: note On E5 veloc.: 64
tick 11008: [80 4C 00] chnl 1: note Off E5 veloc.: 0
tick 11008: [90 4A 40] chnl 1: note On D5 veloc.: 64
tick 11264: [80 4A 00] chnl 1: note Off D5 veloc.: 0
tick 11264: [90 48 40] chnl 1: note On C5 veloc.: 64
tick 11264: [80 47 00] chnl 1: note Off B4 veloc.: 0
tick 11264: [90 45 40] chnl 1: note On A4 veloc.: 64
tick 11776: [80 48 00] chnl 1: note Off C5 veloc.: 0
tick 11776: [90 4A 40] chnl 1: note On D5 veloc.: 64
tick 12032: [80 4A 00] chnl 1: note Off D5 veloc.: 0
tick 12032: [90 48 40] chnl 1: note On C5 veloc.: 64
tick 12288: [80 48 00] chnl 1: note Off C5 veloc.: 0
tick 12288: [80 45 00] chnl 1: note Off A4 veloc.: 0
tick 12288: [90 47 40] chnl 1: note On B4 veloc.: 64
tick 12288: [90 44 40] chnl 1: note On G#4 veloc.: 64
tick 14336: [80 47 00] chnl 1: note Off B4 veloc.: 0
tick 14336: [80 44 00] chnl 1: note Off G#4 veloc.: 0
tick 16384: End of Track
```

Figure 1.4: Text dump of a MIDI file corresponding to score in Fig. 1.1

music, where the song is composed of several monophonic voices that move together in parallel (see Fig. 1.6). For our task another polyphonic texture may be considered as singular: that conformed by a melody voice with one or more accompaniment voices.

During this dissertation, the first classification will be used since the automatic voice extraction from raw data is currently an open problem (Rafailidis et al., 2008). In any case, the representation and methods we propose are able to work with both monophonic and polyphonic music.

The term *music similarity* is ambiguous or at least it can be judged from different points of view (Barthelemy and Bonardi, 2001; Byrd and Crawford, 2002; Hofmann-Engl, 2001; Novello et al., 2006; Selfridge-Field, 1998). It may refer to the resemblance between the melodic line of two musical fragments (Fig. 1.7), the similarity of their rhythmic patterns (Fig. 1.8), or even their harmonic coincidence (Fig. 1.9).

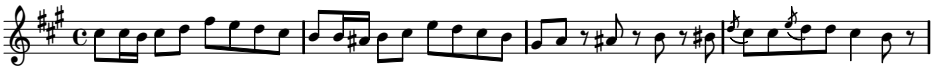


Figure 1.5: Monophonic melody. Beginning of Mozart's k331 Theme, Var VI

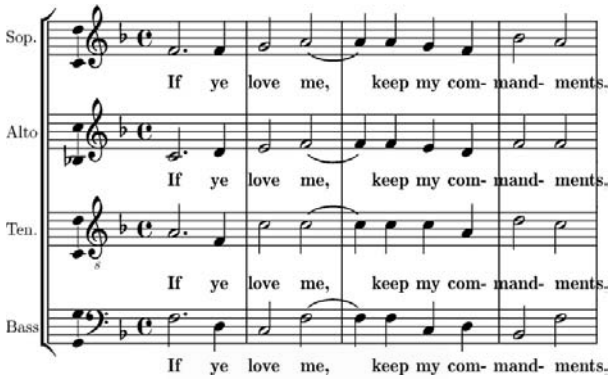


Figure 1.6: Homophonic excerpt

In order to disambiguate this concept for the span of this dissertation, we will consider as ground-truth that the most similar songs are different interpretations of the same song or those produced by the variation compositional form. This statement implies that the similarity is measured upon a trade-off between the melodic, rhythmic, and even the harmonic dimension of songs.

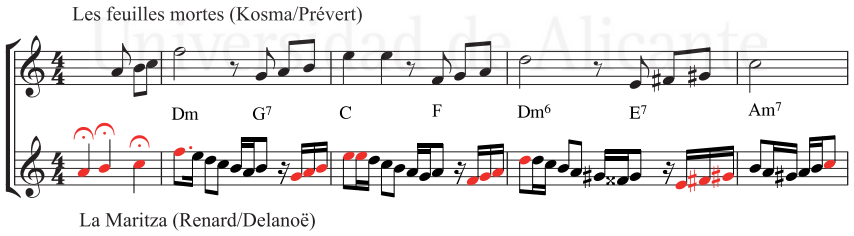


Figure 1.7: Manual transcriptions of excerpts of the two songs Les Feuilles Mortes (autumn leaves) and La Maritza. All the notes of the melody from Les Feuilles Mortes are also present in the Maritza's (red notes). The inserted black notes in La Maritza can be considered as ornaments (from (Robine et al., 2007b)).

In the literature (Barthelemy and Bonardi, 2001; Grachten et al., 2004; Rolland, 1998; Selfridge-Field, 1998) it has been widely accepted that the encoding of the music strongly influences the capability and quality of the comparison methods. Traditionally in MIR symbolic music has been represented by means of sets of tuple strings, where each

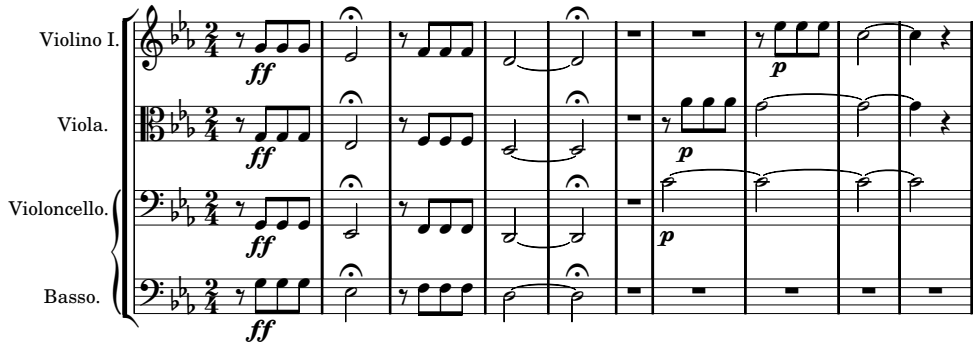


Figure 1.8: Excerpt of the Beethoven's Symphony No. 5 - Opus 67 (1st Movement). The rhythmic cell of the first two bars played at unison holds the main theme of the work.

tuple, in diverse ways, usually contains information on pitch, duration and onset time. Both the retrieval and the comparison have been mostly tackled with structural pattern matching techniques in strings. There are some other approaches, like the geometric one, which transforms the melody into a plot obtained tracing a line between the successive notes in the staves. This way, the melody comparison problem is converted into a geometric one. Other methods have been introduced in the last years like the representation by means of graphs or the comparison using statistical measures.

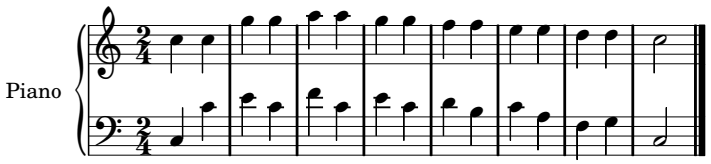
In this dissertation, we use a non-linear representation of melody: by means of trees that express the metric and rhythm of music in a natural way. The approach to tree construction is based on the fact that the different music notation figures are designed on a logarithmic scale: a whole note lasts twice a half note, whose length is the double of a quarter note, etc. This representation provides us with a richness of possibilities that the strings and geometric methods never will: implicit description of rhythm, more musical meaning and automatic emphasizing of relevant notes, for example. Furthermore, trees open a new representation power that allows to merge in the same single data structure all dimensions of music involved in musical similarity: pitch, rhythm, and harmony.

Outline of thesis

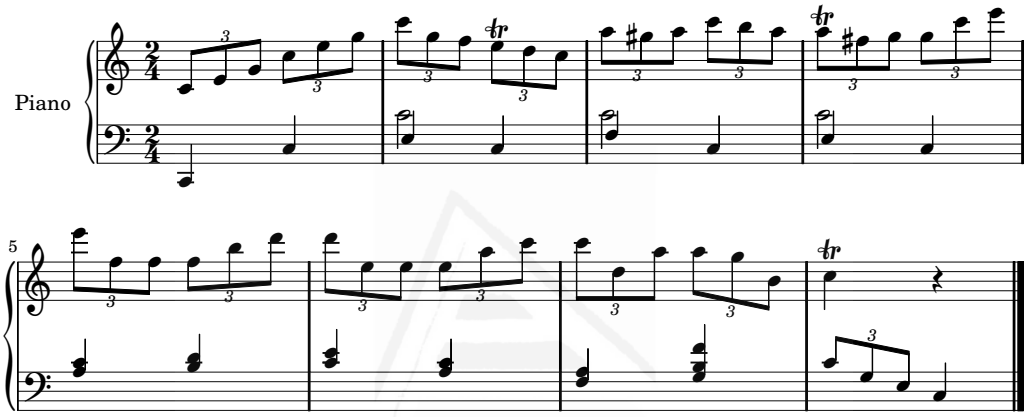
This manuscript is organized as follows:

- First, a background on music symbolic encoding methods and similarity computation systems available in the literature.
- Next, the current state of the art on successful uses of trees for music other than comparison.
- After that, our proposed method to encode and compare music.

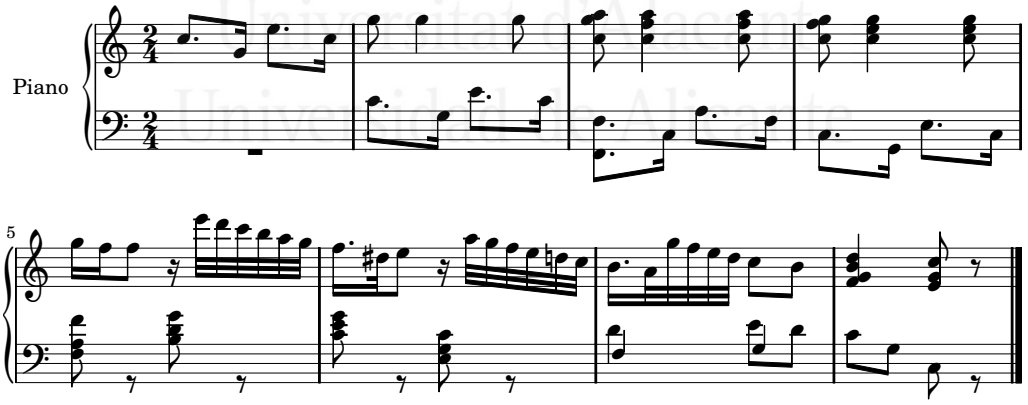
- The details about the experiments performed and their results.
- Future works, and some aspects related to this dissertation that have been tackled by the author in other publications but have not been included in this thesis.
- Finally, an appendix that complements some of the previous chapters can be found at the end of this volume.



(a) Theme



(b) Variation 3



(c) Variation 11

Figure 1.9: Sample variation excerpts, from the Mozart 'Twinkle, Twinkle, Little Star' composition. The similarity can be found through an harmonic analysis of the fragment (Pickens and Crawford, 2002).

2 Background

Musical works can be compared from different perspectives of description and representation. One may compare just song names using string comparison techniques. More detailed metadata (Pérez-García et al., 2009) can be also used as high level descriptors like author name, meter and key changes, or instrumentation. Finally, the music content itself can be used to check the similarity between songs. This thesis is focused on this later case, the comparison using just the musical content represented in a symbolic way.

Depending on the application domain, musical content is said to contain different attributes. In the psychoacoustics domain, Levitin (1999) uses pitch, rhythm, tempo, contour, timbre, loudness, and spatial location. In the *MIR* domain, Downie (1999) considers seven facets: pitch, temporal, harmonic, timbral, editorial, textural, and bibliographic. Other authors also include more elaborated features like thematic information obtained from the raw data (Hsu et al., 1998). For comparison based on musical content, the most used and directly available properties are pitch and rhythm. In this dissertation those two attributes will be used, also including in some cases harmonic information, so this background section will focus on state-of-the-art works that use these kind of data to compare music.

In the automated comparison of musical pieces in symbolic format there have been two works that can be considered milestone. The first one by Mongeau and Sankoff (1990) used string matching techniques (Wagner and Fischer, 1974) from computational biology adapted to the music comparison task. The second one, edited by Selfridge-Field (1998), reviewed in-depth the representation methods and comparison techniques at that date, showing that the string edit distance algorithms were the most commonly used methods to measure musical similarity, along with some early systems based on geometric representations (Maidín, 1998). Since then, many methods have been proposed to compare music content in symbolic format. However, the number of really new representations or algorithms introduced remains limited. The contribution of most works is rooted in the study of the high number of variations of those original techniques, both in the way of representing symbolically music and in the pattern matching algorithms used.

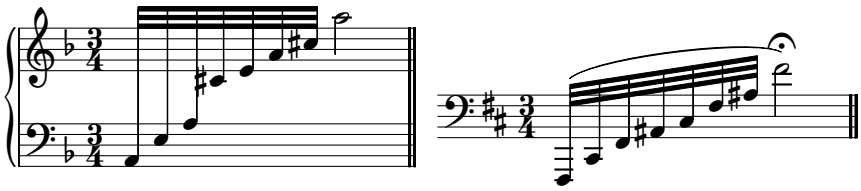
The way music is encoded determines the kind of searches that can be undertaken as stated in several works like (Selfridge-Field, 1998) or (Barthelemy and Bonardi, 2001). To the date of writing this manuscript, there are basically six trends in the representation and comparison of music in symbolic format:

- string edit distance and alignment methods using a variety of representations (Grachten et al., 2005; Lemström, 2000; Mongeau and Sankoff, 1990)
- n-gram algorithms (Doraisamy, 2004; Downie, 1999; Uitdenbogerd, 2002)
- graph encodings (Pinto and Tagliolato, 2008)
- statistical comparison measures (Bernabeu et al., 2009; Engelbrecht, 2002; Habrard et al., 2008; Pardo et al., 2004; Pickens, 2004)
- geometrical frameworks (Aloupis et al., 2006; Tanur, 2005; Typke, 2007; Ukkonen et al., 2003a; Wiggins et al., 2002)
- tree representations (Rizo et al., 2003, 2008)

In this section, the different ways of encoding pitch and rhythm for monophonic and polyphonic music will be presented first, then the particularities of each comparison system will be reviewed. The last paradigm, using trees for representing music, is the central topic of the current dissertation and will be covered through the rest of chapters.

2.1 Monophonic music

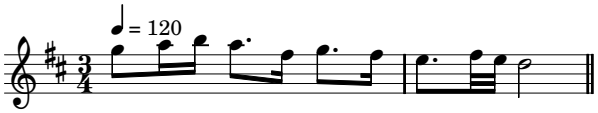
In the context of music comparison, a monody is a sequence of notes or rests represented here by two main properties: pitch and rhythm. In this section *monophony* only means *one note sounding at any time*. No consideration is made on whether the composer wanted to collapse several perceived musical lines in a monody as it happens with the Bach's *Partitas for violin solo*. This issue will be covered in further sections. Several different encodings for both properties have been proposed so far in the literature.



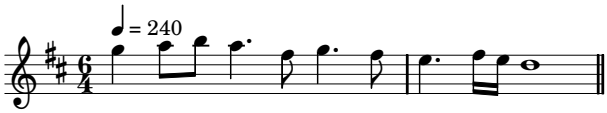
(a) (b)

Figure 2.1: First bar of "Rumores de la Caleta" from Isaac Albeniz (left), op. 71, and a transposition (right).

These encodings are not designed to describe thoroughly the music they depict, but to serve as adequate intermediate data for this comparison task. Besides, it is desirable that the intermediate representations hold some properties that ease the comparisons: *transposition invariance* and *tempo invariance* (Lemström, 2000). The former means that the transposition of a melody or its tonality change should not affect its representation (see Fig. 2.1), because the transposition has little effect on similarity



(a)



(b)

Figure 2.2: Different figurations for the bars 3 and 4 of the Manuel de Falla's "Sombrero de tres picos: danza de los vecinos".

perception (Uitdenbogerd and Zobel, 1999). The latter means that a change in the tempo or figuration of the song should not alter the representation (see Fig. 2.2). For the interested reader, a set-theoretic formalism for defining and classifying the various musical invariances can be found in (Lemström and Wiggins, 2009).

Some of the different representations that will be exposed have been approached with slight differences depending on the author. In all cases our own definition of the encoding that will be used later in the experiments will be stated.

2.1.1 Pitch encodings

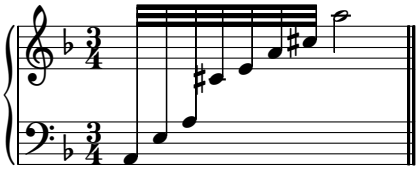
The different properties used for the symbols that encode the pitches in a melody can be *absolute*, if the property depends only on the represented note, or *relative*, if it is defined in terms of differences to their surrounding notes, usually the preceding one. Relative encodings make no sense in a polyphonic context as there may be more than one previous or next note to be selected as the reference note.

Next, some commonly used pitch properties are presented. In each case, the alphabet, Σ_p, applicable is enunciated. The pitch encodings presented below are illustrated in Fig. 2.3.

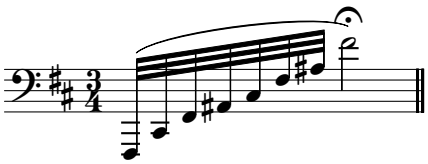
For each of the definitions of pitch representations given below, a function with the same name is implicitly defined from the set of pitches $\mathcal{P}$ to the representation alphabet. In general, given a pitch representation r, the function $p_r\colon \mathcal{P} \to \Sigma_{p_r}$ returns the pitch representation r of the given note.

Common music notation pitches (*absolute*) (p_{cmn})

In this dissertation, only western tonal music for the common practice period is covered. As *common music notation* we understand the set of graphical elements that are used for writing this kind of music. In terms of pitch, the fundamental elements found in any score are the name of the notes, their accidental, and the octave: $\mathcal{P} = (\mathcal{N} \times \mathcal{A} \times \mathcal{O}) \cup \{\text{'s'}\}$, $\mathcal{N} \in \{C, D, E, F, G, A, B\}$, $\mathcal{A} \in \{\flat\flat, \flat, \natural, \sharp, \text{x}\}$, $\mathcal{O} \in [-2, 8] \subset \mathbb{Z}$. The 's' stands for "rest".



D minor



B minor

Pitch encoding	Song	Pitch Sequence
Common music notation (p_{cmn})	D minor	(A,♮,2) (E,♮,3) (A,♮,3) (C,♯,4) (E,♮,4) (A,♮,4) (C,♯,5) (A,♮,5)
	B minor	(F,♯,1) (C,♯,2) (F,♯,2) (A,♯,2) (C,♯,3) (F,♯,3) (A,♯,3) (F,♯,4)
Base-12 (p_{12})	D minor	9 4 9 1 4 9 1 9
	B minor	6 1 6 10 1 6 10 6
Base-21 (p_{21})	D minor	10 5 10 2 5 10 2 10
	B minor	7 2 7 11 2 7 11 7
Base-40 (p_{40})	D minor	32 15 32 4 15 32 4 32
	B minor	21 4 21 33 4 21 33 21
Absolute (p_{abs})	D minor	33 40 45 49 52 57 61 69
	B minor	18 25 30 34 37 42 46 54
Interval (p_{itv})	D minor	0 7 5 4 3 5 4 8
	B minor	0 7 5 4 3 5 4 8
Interval from tonic (p_{ift})	D minor	7 2 7 11 2 7 11 7
	B minor	7 2 7 11 2 7 11 7
Directed-modulo12 (p_{dm})	D minor	trqprqu
	B minor	trqprqu
Contour (p_c)	D minor	0 1 1 1 1 1 1 1
	B minor	0 1 1 1 1 1 1 1
HD-Contour (p_{hdc})	D minor	0 2 2 1 1 2 1 2
	B minor	0 2 2 1 1 2 1 2

Figure 2.3: Different pitch encodings for the first bar of "Rumores de la Caleta" and its transposition. Note the key signature to read the pitch class representation.

Definition 2.1.1 *The alphabet is* $\Sigma_{p_{cmn}} = (\mathcal{N} \times \mathcal{A} \times \mathcal{O}) \cup \{\text{'s'}\}$, $|\Sigma_{p_{cmn}}| = 351$ [1] *although in practice is usually more reduced. The range for piano is* $[\mathrm{A}_{-1}, \mathrm{C}_7]$, *which is enough for most cases.*

This encoding is the closest to that used in music written scores. However, it is necessary to perform a pitch spelling process (Meredith, 2006) if the source of the music is a MIDI stream or the result of a transcription.

Base-n pitch representations (*absolute*)

There are two properties the encoding of a note in the range of an octave should hold for being adequate for representing and handling music for analysis and composition, and to a lesser extent, for similarity computation. The *discrete accommodation of enharmonic pitches* property means that two notes with the same pitch but different spellings (e.g. $(E, \sharp, 3)$ and $(F, \flat, 3)$) should be given two different codes. The second one, the *interval invariance*, is held when given an interval, it leads to the same numerical value (obtained

[1] This figure has been obtained through the product of 10 octaves by 5 possible accidentals for each one of the seven note names plus the rest.

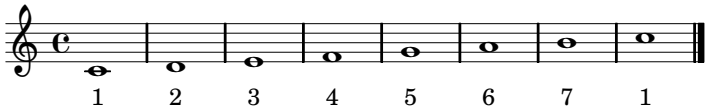


Figure 2.4: Base-7 representation of pitch.

through the subtraction of the numerical values of the represented notes) for any pair of notes separated by this interval.

Four different ways to encode a note from the common music notation into a single number have been studied focusing in those two properties in (Hewlett, 1992) and (Selfridge-Field, 1998), namely *base-7*, *base-12*, *base-21*, and *base-40*.

In the context of melodic comparison, the main advantage of working with a single number representing pitches instead of tuples is the speed-up of the symbol comparison computation.

Base-7 (p_7) It is the diatonic representation for the C major scale, an integer mapping to the note name $\mathcal{N}$ in $\mathcal{P}$ (See Fig. 2.4).

Definition 2.1.2 *The alphabet is* $\Sigma_{p_7} = \{i \in \mathbb{N} \mid 0 \leq i \leq 7\}$*, where* 0 *is used to encode rests.*

For the comparison task this representation has no advantage. Thus, it has not been used in any similarity computation system.

Base-12 and Interval from tonic (p_{12}, p_{ift}) This encoding has been also named as *folded pitch* or *name without octave* (Rizo et al., 2003), or more commonly, *pitch class*. In some cases it has been referred as *absolute modulo 12* as in (Gómez et al., 2007b; Hanna et al., 2008), where pitches are represented modulo 12, so that only one occurrence of the same note in different octaves is preserved (see Fig. 2.5).

The *pitch class* is just the interval formed by the note and C. If instead of using C as base note, the local tonality is used, this representation is converted in *interval from tonic* (see (Barthelemy and Bonardi, 2001; Habrard et al., 2008; Mongeau and Sankoff, 1990)), *interval from key note* (Uitdenbogerd and Zobel, 1999), *key relative* (Hanna and Ferraro, 2007), *interval from a reference note* or *numerical scale* in (Uitdenbogerd and Yap, 2003), in which case the note can be anyone, not only the tonic. Barthelemy and Bonardi (2001) encode the same melody with all possible tonalities using this interval from tonic representation in what the author names as *multireferenced descriptor*. Sometimes a symbol denoting the direction of the motion from the previous note is also added as in the *directed key relative* representation (Ferraro and Hanna, 2007).

This encoding has the advantage of being *octave invariant*, i.e, two melodies belonging to two different octaves will be equally encoded. However, it is not able to distinguish between enharmonic tones. On the other hand, it can be obtained directly

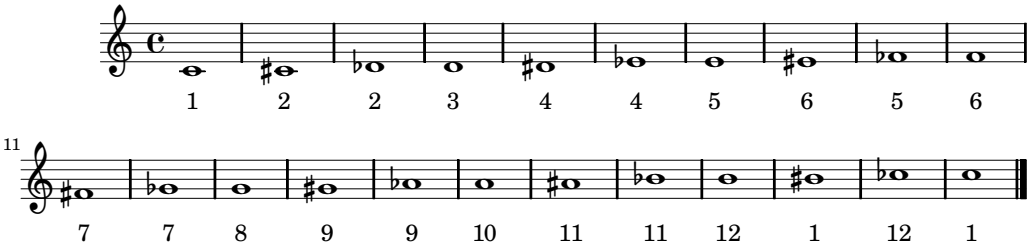


Figure 2.5: Base-12 representation of pitch

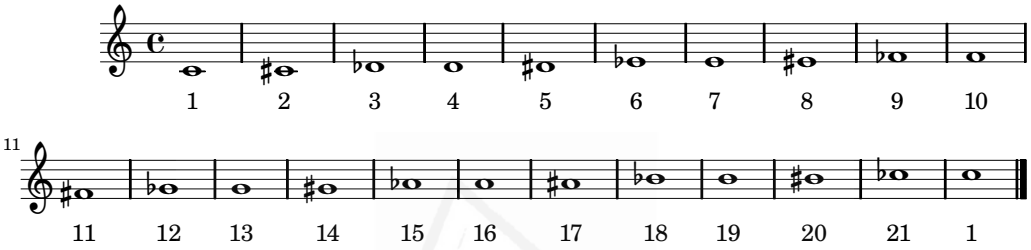


Figure 2.6: Base-21 representation of pitch

from the *absolute pitch* just by computing a modulo 12 on the original MIDI note and summing up 1 so that the rest can be encoded as a 0.

Definition 2.1.3 *The* pitch class p_{12} *encoding has an alphabet* $\Sigma_{\mathrm{p}_{12}} = \{i \in \mathbb{N} \mid -1 \leq i \leq 11\}$*, corresponding to the 12 semitones of the octave, from A to G, including flat and sharp notes, and the rest. C is encoded with a 0, C♯ with a 1, B with a 11, and the rest as a -1.*

Definition 2.1.4 *The* interval from tonic *($\mathrm{p}_{\mathrm{ift}}$) is defined as the difference in steps between a note and the tonic of the working tonality.* $|\Sigma_{\mathrm{p}_{\mathrm{ift}}}| = 13$*, that is, the 12 octave semitones plus a symbol for rests. The unison is represented by a 0 and rests by a* -1*.*

Base-21 (p_{21}) "The *base-21* system provides a sufficient number of numerals to differentiate each enharmonic tone within the range of single sharps and flats" (Selfridge-Field, 1998).

It solves the problem of the encoding of the dual naming of enharmonic tones into a unique integer number (see Fig. 2.6). However, as for the *base-7*, and *base-12*, it is not *interval invariant* either. In the case of the *base-21* representation, the minor third between C (represented by a 1) and E♭ (encoded with a 6) produces a value of 5 (i.e., $6 - 1$), while the same interval between E♭ and G♭ leads to a value of $12 - 6 = 6$.

Definition 2.1.5 *The alphabet is* $\Sigma_{\mathrm{p}_{21}} = \{i \in \mathbb{N} \mid 0 \leq i \leq 21\}$*. The number 0 is used for representing rests.*

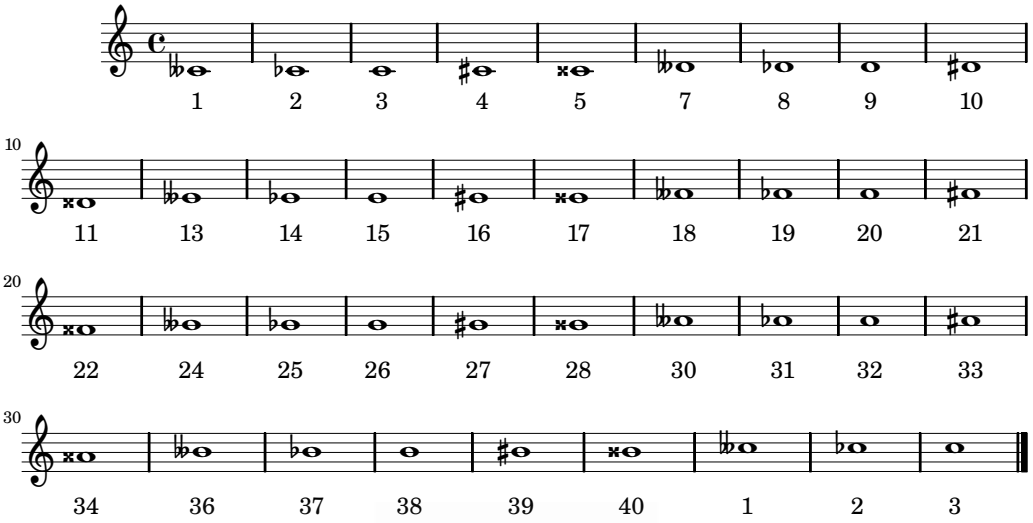


Figure 2.7: Base-40 representation of pitch

Base-40 (p_{40}) The *base-40* encoding solves the problem of the *interval invariance* property. The list of integers associated to each note are shown in Fig. 2.7. This representation is used as an intermediate representation in Humdrum (Huron, 2002) and in (Typke et al., 2003).

Definition 2.1.6 *The alphabet is* $\Sigma_{p_{40}} = \{i \in \mathbb{N} \mid 0 \leq i \leq +40\}$. *The number 0 is used for representing rests.*

p_{abs} Absolute pitch (*absolute*)

In this representation, pitch is encoded by an absolute number that distinguishes univocally each frequency of the scale. No distinction is made of enharmonically tones. Pitch is represented with its MIDI number. See Fig. 2.8 for a listing of the note numbers and Fig. 2.3 for an example.

This representation has been thoroughly used in similarity computation as a non-processed pitch encoding. In the literature, it has been mainly named as *absolute pitch* as in (Ferraro and Hanna, 2007; Gómez et al., 2007b; Neve and Orio, 2004; Uitdenbogerd, 2002), as *notes* (Grachten et al., 2002), or just referred as MIDI note numbers in (Lemström, 2000).

The main advantage of this encoding is its direct availability from the most extended symbolic representation, the MIDI files, and its easy computing from a score. However, it has the drawback of not holding the *transposition invariance* property.

Definition 2.1.7 *The alphabet is* $\Sigma_{p_{abs}} = \{i \in \mathbb{N} \mid 0 \leq i \leq 128\}$. *The number 128 is used for representing rests.*

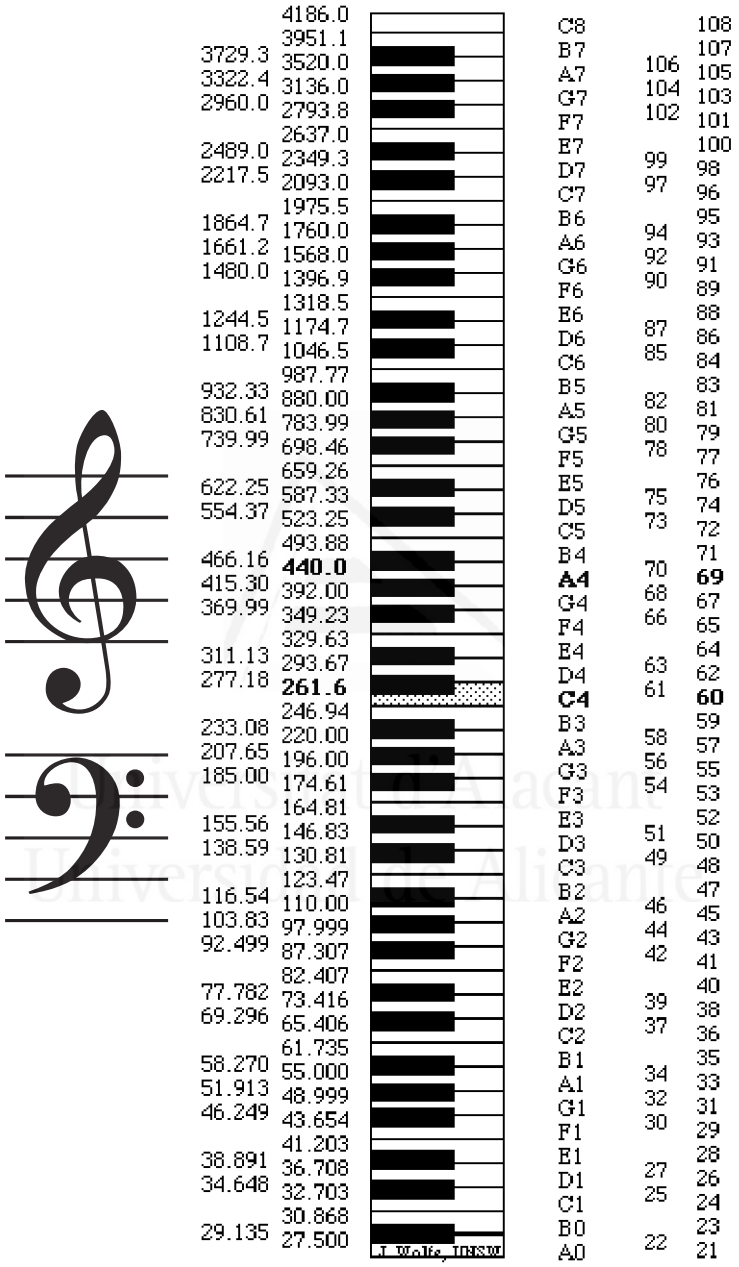


Figure 2.8: MIDI note numbers equivalences (From Joe Wolfe, University of South Wales - UNSW - http://www.phys.unsw.edu.au/jw/notes.html)

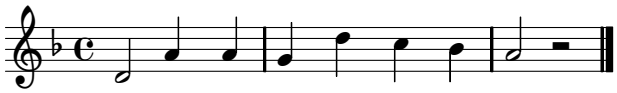


Figure 2.9: Hanson intervallic representation for two bars.

Interval and directed-modulo-12 (*relative*) (p_{itv}, p_{dm})

A way to accomplish the *transposition invariance* of a pitch representation is the use of intervals between successive notes expressed in terms of number of steps.

This encoding has been widely used in the literature in a number of names and variants. Some works establish several levels in which the difference between two successive notes can be quantized, being the semitone unit just a possibility. This is the case of the *Delta Pitch (DP)* (Orio, 2005) and one of the *abstraction levels* (Grachten et al., 2002).

Some interval representations include a sign that encodes the motion direction, for example Grachten et al. (2004) and Ferraro and Hanna (2007) name it as *directional intervals* or *signed intervals*.

In other works the size of the interval is limited to a maximum value or clipped with a modulo function. For example, in (Orio, 2005) and in (Meek and Birmingham, 2002) the *interval mod 12* coding is just a clipping of the maximum interval to an octave, e.g, "the interval from D3 to G4 (perfect 11th, or 17 semitones) would be mapped to the interval of a perfect 4th (5 semitones)". In other cases, instead, when the maximum value is reached a special value or function is applied (Dannenberg et al., 2003; Pardo and Birmingham, 2002b; Uitdenbogerd and Zobel, 1999). In works like (Orio, 2005), it is suggested that the different levels of interval quantization may be accompanied by other pitch properties as the local tonality or the relative frequency of pitches.

The interval is usually obtained from the previous note, as in (Uitdenbogerd and Yap, 2003; Uitdenbogerd and Zobel, 1999) or in (Neve and Orio, 2004) where it is denoted by *Pitch Interval in semitones with the previous note (PIT)*, but it may be calculated relative to the next note.

The intervals have been sometimes mapped to characters in an ASCII alphabet in order to take advantage of text retrieval tools. Doraisamy (2004) maps the most frequent intervals into ASCII chars using a sigmoid function. The same scheme is used by Uitdenbogerd (2007) in her *directed modulo-12* representation (see p_{dm} in Fig. 2.3). A similar approach is followed by Serrano in his implementation of the *Hanson Intervalic system* (Serrano and Iñesta, 2006a), where the intervals are represented with the symbols d-s-n-m-p-t of the Hanson system (Hanson, 1960) plus the five vowels a-e-i-o-u to form the intervals of two octaves and a half (See Fig. 2.9).

Definition 2.1.8 *The* interval *is defined as the difference in semitones between a note and the preceding one. Theoretically,* $\Sigma_{p_{itv}} = \{i \in \mathbb{Z} \mid -127 \leq i \leq +127\}$*, but in practice large intervals seldom appear, so the limiting approach is followed:* $\Sigma_{p_{itv}} = \{i \in$

$\mathbb{Z} \mid -24 \leq i \leq +24\}$, *being any other larger value assigned to the extremal values. This way,* $|\Sigma_{\text{pitv}}| = 49$. *Rests are encoded also here with a 0.*

Pitch contour (*relative*) (p_c)

This is the coarser quantization of an interval: it encodes the motion direction of the pitch by three symbols indicating unison, ascending, or descending (e.g.,{'U', 'A', 'D'}). In (Uitdenbogerd and Zobel, 1999) it is stated that contour can be more important than exact pitch for remembering a song, although exact pitch intervals help listeners to distinguish between melodies. However, the lower representation precision requires the length of the sequences to be longer for their recognition.

This representation was introduced by Parsons (1975), where a catalogue of themes was compiled with an index built on the contour representation of them. It has been widely used, with a variety of symbols denoting the three possibilities of motion, in QBH systems (Ghias et al., 1995) and in QBE from text or OMR (Bainbridge et al., 1999), because this representation is more robust against transcription errors and easier to remember or reproduce for an average user. Maybe for this reason, it has also been used to be compared with other pitch encoding schemes or abstractions levels (Barthelemy and Bonardi, 2001; Dowling, 1978; Ferraro and Hanna, 2007; Grachten et al., 2002, 2004; Kim et al., 2000; Lemström and Perttu, 2000; Müllensiefen and Frieler, 2004a; Uitdenbogerd and Yap, 2003; Uitdenbogerd and Zobel, 1999).

Definition 2.1.9 $\Sigma_{\text{p}_\text{c}} = \{-1, 0, +1\}$; *'+1', if* $\text{p}_\text{abs}(n_i) > \text{p}_\text{abs}(n_{i-1})$, *'−1' if* $\text{p}_\text{abs}(n_i) < \text{p}_\text{abs}(n_{i-1})$, *and '0' otherwise. As for the other relative pitch properties, for the first note in the sequence it is not defined.* $|\Sigma_{\text{p}_\text{c}}| = 3$. *Rests are coded also with a '0'.*

High definition contour (*relative*) (p_hdc)

Sometimes the *contour* representation is too general and leads to many false positives in the melodic retrieval task, and the *interval* too fine producing false negatives. In the context of the number of interval quantization levels, this representation can be positioned between both representations as a trade-off. It has been also referred as *extended contour* (Uitdenbogerd and Yap, 2003). In that work, the threshold that divides the levels is questioned and in the work of Downie (1998) a more in-depth study of it can be found paying a special attention to on the optimal point of division of each level.

Other variations of the previous definition can be found in the literature. In (Barthelemy and Bonardi, 2001) the *qualified contour* is introduced, encoding conjunct or disjunct degrees with direction, i.e., the levels of the contour are 'd' for disjunct, 'c' for conjunct, plus the sign indicating the motion direction. In (Müllensiefen and Frieler, 2004a) the interval range is divided into up to nine divisions in what they name as *fuzzification*, borrowing this term from the fuzzy logic just to give concept names to the numerical magnitudes of the intervals. (see Table 2.1). Finally, Lemström and Laine (1998) introduce a real fuzzy division of the interval space in what they denote as *Quantized, Partially Overlapping Intervals (QPI)*.

Table 2.1: Interval classes used in (Müllensiefen and Frieler, 2004a)

Class	Intervals	Name
-4	< 7	Big leap down
-3	-7, -6, -5	Leap down
-2	-4, -3	Big step down
-1	-2, -1	Step down
0	0	Same
1	1, 2	Step up
2	3, 4	Big step up
3	5, 6, 7	Leap up
4	> 7	Big leap up

Definition 2.1.10 *Same as contours (Def. 2.1.9) but it also includes '+2' and '−2' if the pitch difference exceeds 4 semitones.* $\Sigma_{\text{Phdc}} = \{-2, -1, 0, +1, +2\}$. *Rests are encoded with a 0.*

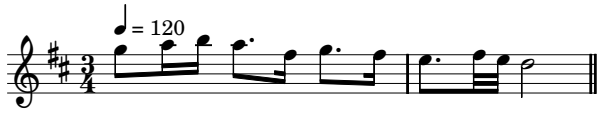
2.1.2 Rhythm encodings

By *rhythm* all properties related to the temporal dimension of notes are considered, being the onset time and the duration of notes the most commonly used. Other rhythm related features have been seldom used, like for example the stress related to the note position in the bar.

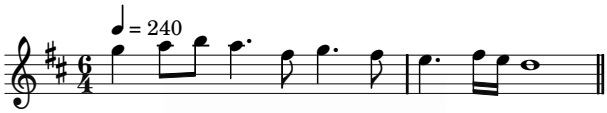
In many of the symbolic music comparison methods so far, rhythm is not present or just ignored. This is why there is a lower number of rhythm encodings than pitch representations in the literature. Studies where the rhythm has been left out can be found in (Uitdenbogerd and Yap, 2003). On doing the contrary, the work presented by Pardo and Birmingham (2002b) only encodes rhythm. Some works have used no information about duration but only the position of each note in a sequence, using a dummy symbol (e.g., a dash) where a rest is found (Barthelemy and Bonardi, 2001). Although the convenience of using rhythm is not widely established in the literature, there are works that report better results when using rhythm combined with pitch (Ferraro and Hanna, 2007). In fact, as indicated by (Barthelemy and Bonardi, 2001), the music dictionary of Barlow and Morgenstern (1978) shows that music retrieval based on pitch information only leads to results with typically too many false matches. See (Selfridge-Field, 1998), page 27 for such absurd matches.

The different rhythm encodings presented below (denoted by alphabets Σ_r) are illustrated in Fig. 2.10.

For each of the definitions of rhythm representations given below, a function with the same name gets implicitly defined. In general, given a rhythm representation s, the function $r_s \colon \mathcal{P} \to \Sigma_{r_s}$ returns the rhythm representation s of the given note.



(a)



(b)

Rhythm encoding	Song	Representation
Absolute time (r_{tabs})	♩= 120	$0\ \frac{1}{2}\ \frac{3}{4}\ 1\ \frac{7}{4}\ 2\ \frac{11}{4}\ 3\ \frac{15}{4}\ \frac{31}{8}\ 4$
	♩= 240	$0\ 1\ \frac{3}{2}\ 2\ \frac{7}{2}\ 4\ \frac{11}{2}\ 6\ \frac{15}{2}\ \frac{31}{4}\ 8$
Absolute duration (r_{dabs})	♩= 120	$\frac{1}{2}\ \frac{1}{4}\ \frac{1}{4}\ \frac{3}{4}\ \frac{1}{4}\ \frac{3}{4}\ \frac{1}{4}\ \frac{3}{4}\ \frac{1}{8}\ \frac{1}{8}\ 2$
	♩= 240	$1\ \frac{1}{2}\ \frac{1}{2}\ \frac{3}{2}\ \frac{1}{2}\ \frac{3}{2}\ \frac{1}{2}\ \frac{3}{2}\ \frac{1}{4}\ \frac{1}{4}\ 4$
Contour (r_c)	♩= 120	$0\ -1\ 0\ 1\ -1\ 1\ -1\ 1\ -1\ 0\ 1$
	♩= 240	$0\ -1\ 0\ 1\ -1\ 1\ -1\ 1\ -1\ 0\ 1$
High Definition Contour (r_{hdc})	♩= 120	$0\ -1\ 0\ 2\ -2\ 2\ -2\ 2\ -2\ 0\ 2$
	♩= 240	$0\ -1\ 0\ 2\ -2\ 2\ -2\ 2\ -2\ 0\ 2$
IOI (r_{ioi})	♩= 120	$\frac{1}{2}\ \frac{1}{4}\ \frac{1}{4}\ \frac{3}{4}\ \frac{1}{4}\ \frac{3}{4}\ \frac{1}{4}\ \frac{3}{4}\ \frac{1}{8}\ \frac{1}{8}\ 2$
	♩= 240	$1\ \frac{1}{2}\ \frac{1}{2}\ \frac{3}{2}\ \frac{1}{2}\ \frac{3}{2}\ \frac{1}{2}\ \frac{3}{2}\ \frac{1}{4}\ \frac{1}{4}\ 4$
IOR (r_{ior})	♩= 120	$2\ 1\ \frac{1}{3}\ 3\ \frac{1}{3}\ 3\ \frac{1}{3}\ 6\ 1\ \frac{1}{16}\ 0$
	♩= 240	$2\ 1\ \frac{1}{3}\ 3\ \frac{1}{3}\ 3\ \frac{1}{3}\ 6\ 1\ \frac{1}{16}\ 0$
Accent (r_{acc})	♩= 120	$1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0$
	♩= 240	$1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0$

(c)

Figure 2.10: Different rhythm encodings for two different figurations of the same excerpt.

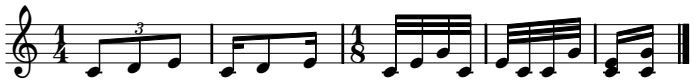


Figure 2.11: Side effects of quantization using sixteenth quantization (from (Clausen et al., 2000)): first bar is quantized at 16th as the second, third and fourth bars are quantized equal as shown in fifth bar.

Absolute rhythm encoding (*absolute*) (r_{tabs}, r_{dabs})

The simplest and most common way to encode rhythm is to use the note onset time and its ending or its duration. In some cases, only the onset is used, in other cases only the duration. Anyway, the onset time of a note can be obtained just by summing up the duration of all preceding notes.

The main issue to deal with here is the duration resolution, i.e., the number of subdivisions a beat can be split in. If the songs are stored in SMF files, the resolution is found as a meta-data in the file, and duration is expressed in terms of onset and offset times in ticks, where a tick is a unit of resolution and is given in ticks per beat. The duration is computed through the subtraction of offset minus onset. This problem has been avoided in (Doraisamy, 2004) by the use of milliseconds. However, this method is too tight to the tempo of the piece.

The resolution may be referred also as quantization, because notes that do not fit in the grid defined by the given resolution must be quantized. The most commonly used quantization figure is the 16th (Clausen et al., 2000; Hanna and Ferraro, 2007; Mongeau and Sankoff, 1990; Uitdenbogerd and Yap, 2003). However, this quantization may introduce problems after the encoding of the durations as shown in Fig. 2.11.

One special case of absolute encoding is that of the *relative to a standard duration* representation (Selfridge-Field, 1998; Uitdenbogerd and Yap, 2003), where real numbers are allowed, representing the relative duration of a note given a reference note, for example the quarter note. In that case, a 8th dotted note would be encoded with the real 0.75. The main advantage of this absolute encoding is that it does not suffer from the quantization problems [2].

The main disadvantage of any absolute duration representation is that it does not hold the *tempo invariance*, when tempo is modified by the change of figures as can be seen in the two scores in Fig. 2.10.

Definition 2.1.11 *Given a note n_i, let $t_{ON}(n_i)$ be its onset time, and $t_{OFF}(n_i)$ its offset time. Let the resolution $q \in \mathbb{N}$ be the duration of a single beat. The absolute time is the standardized time t_{ON}/q. Strictly speaking, $\Sigma_{r_{tabs}} \in \mathbb{R}$ is in practice a limited set of durations appear depending on the quantization used.*

[2] This affirmation is true only in theory. The float number implementations in any programming language have a precision, and finally this is also a quantization level

Definition 2.1.12 *The absolute duration of a note n is defined as $t_{OFF}(n_i) - t_{ON}(n_i)$ expressed as multiples or fractions of the beat duration or resolution q. Strictly speaking, $\Sigma_{\mathrm{r_{dabs}}} \in \mathbb{R}$ is in practice a limited set of durations appear depending on the quantization used.*

Tied notes. Tied notes may appear for a number of reasons, for example if their source is a written score. There are three ways of handling this situation: just ignore the ties and consider the tied notes as individual ones, by merging all tied notes summing up their durations, and finally encoding the tie with a special symbol, say for example '^', that may accompany the rhythm symbol in Σ_r.

In some cases, notes are splitted in tied notes of duration equal to the quantization, as the *time grids* in (Hanna et al., 2008) with a limitation to 6 consecutive notes for implementation issues, or the necessary subdivision imposed by the tree representation we propose (Rizo et al., 2003). In (Müllensiefen and Frieler, 2004a) the duration is encoded in what they name *rhythmical weightings*: every note in the melody is substituted by d times the pitch of the melody, being d the duration of that note. Thus, duration is encoded (in this case without the need of that special tie symbol) with the multiple sequential occurrences of pitch. This same encoding is used in (Tanur, 2005) with the name *repeated unit time steps*.

Rhythm contour (*relative*) ($\mathrm{r_c}$)

The same method to solve the *transposition invariance* in pitch can be followed to overcome *tempo invariance*: the use of encodings relative to the surrounding notes. The rhythm contour, introduced in works like (Neve and Orio, 2004) and (Uitdenbogerd and Yap, 2003) with the name *duration relative to previous duration*, encodes the duration of a note as longer, equal, or shorter than the previous (or next) note.

Definition 2.1.13 $\Sigma_{\mathrm{r_c}} = \{-1, 0, +1\}$; *'+1' if the* $\mathrm{r_{dabs}}(n_i) > \mathrm{r_{dabs}}(n_{i-1})$, *'−1' if* $\mathrm{r_{dabs}}(n_i) < \mathrm{r_{dabs}}(n_{i-1})$, *and '0' otherwise. For the first note in the sequence it is not defined.*

High definition contour ($\mathrm{r_{hdc}}$) The same way in which several levels of pitch contour were established, the rhythm contours can be grouped in several levels, like in (Frieler, 2004; Müllensiefen and Frieler, 2004a), where duration contours are categorized to five duration classes, represented as *gaussified* values.

Definition 2.1.14

$$\Sigma_{\mathrm{r_{hdc}}} = \{-2, -1, 0, +1, +2\};$$
$$\begin{cases} \text{`}+2' & \text{if } \mathrm{r_{dabs}}(n_i) \geq 2 \cdot \mathrm{r_{dabs}}(n_{i-1}) \\ \text{`}+1' & \text{if } \mathrm{r_{dabs}}(n_i) > \mathrm{r_{dabs}}(n_{i-1}) \wedge \mathrm{r_{dabs}}(n_i) < 2 \cdot \mathrm{r_{dabs}}(n_{i-1}) \\ \text{`}\ 0' & \text{if } \mathrm{r_{dabs}}(n_i) = \mathrm{r_{dabs}}(n_{i-1}) \\ \text{`}-1' & \text{if } \mathrm{r_{dabs}}(n_i) \leq 2 \cdot \mathrm{r_{dabs}}(n_{i-1}) \\ \text{`}-2' & \text{if } \mathrm{r_{dabs}}(n_i) < \mathrm{r_{dabs}}(n_{i-1}) \wedge \mathrm{r_{dabs}}(n_i) > 2 \cdot \mathrm{r_{dabs}}(n_{i-1}) \end{cases} \tag{2.1}$$

For the first note in the sequence it is not defined.

Interonset Interval (IOI) (*relative*) ($\mathrm{r_{ioi}}$)

Equivalent to intervals in the pitch domain, the Inter-Onset Interval (IOI) encoding represents the interval of a note from the previous (or to the next) note (Grachten et al., 2004; Orio, 2005). Again, the value can be encoded in any of the resolutions or quantizations introduced previously in the absolute rhythm encoding, or standardized using a reference figure (Neve and Orio, 2004).

Definition 2.1.15 *The time lapse from the ith note onset to that of the next;* $IOI_i = (t_{ON}(i+1) - t_{ON}(i))/q$, *expressed the same way as in for* $\mathrm{r_{dabs}}$. *For the last note, it is defined as its duration* ($\mathrm{r_{dabs}}$). *The same situation about enumerability described for* $\Sigma_{r_{abs}}$ *is applicable to* $\Sigma_{\mathrm{r_{ioi}}}$. *Note that rests disappear for this property.*

Interonset Interval Ratio (IOR) (*relative*) (r_{ior})

This representation solves definitely the absolute duration encoding IOI by means of the use of ratios (Doraisamy, 2004). This is why sometimes it has also been named as *Inter-Onset Interval Ratio (IORatio)* in (Pardo and Birmingham, 2002b) or just *Inter-Onset interval Ratio (IORr)* in (Cambouropoulos et al., 2005; Dannenberg et al., 2003).

In some works this encoding is post-processed by mapping the Inter-Onset interval Ratio (IOR) values space in some equivalence classes. In (Pardo and Birmingham, 2002b) and (Pardo et al., 2004) the IOR is discretized into five log-spaced classes, in (Neve and Orio, 2004) a possible normalization or quantization is suggested.

Definition 2.1.16 *The ratio between successive IOI is defined as* $IOR_i = \frac{IOI_i}{IOI_{i+1}}$. *The same situation about enumerability described for* $\Sigma_{\mathrm{r_{dabs}}}$ *is applicable to* $\Sigma_{\mathrm{r_{ior}}}$. *It is not defined for the last note.*

Position in measure (*absolute*) (r_{pim})

If the meter of the song is present [3], the onset can be encoded by indicating its position relative to the bar start time (Barthelemy and Bonardi, 2001; Clausen et al., 2000). In (Typke et al., 2003), the time coordinate is calculated as the sum of the measures preceding the note plus the note's position within this measure.

This may be considered a special case of absolute encoding on onsets, where instead of using the absolute time from the beginning of the piece, the bar number and the time from the beginning of the bar are used, and thus, it has the same advantages and problems than any absolute encoding. However, if only the relative time is compared, it is a completely different encoding.

Accent (r_{acc})

Accent may be considered as a rhythm property because the stress of notes depends, among other things, on its position in strong or weak beats of the bar, besides other properties, like changes on contour direction, or large pitch intervals (Uitdenbogerd and Zobel, 1999).

Several works that study the importance of stress in music identification can be found (Huron and Royal, 1996; Jones, 1987; Tekman, 1997; Thomassen, 1982), with studies on the accent structure from a perceptual point of view. In (Selfridge-Field, 1998), an *accented-note model* is depicted. However, few works include this property in their codings for similarity computation (Müllensiefen and Frieler, 2004a; Robine et al., 2007a; Uitdenbogerd and Yap, 2003).

The same way Maidín (1998) gives different weights for the similarity computation to notes depending on their metrical stress, we will consider only the stress depending on the position of the note in the bar.

2.1.3 Melodic analysis

Depending on their task in the melody with respect to harmony, individual notes can be classified as *harmonic* (H) or *Non-Harmonic Tones (NHTs)*. The harmonic tones are tagged as 'h'. For the NHT, different tags are given based on the type of ornament: appoggiatura (ap), passing tone (p), neighbor (n), etc, and they are computed using information of meter, pulse, duration, and pitch interval information. $\Sigma_m = \{h, ap, p, n\}$.

Some of the rules to compute those tags (Illescas et al., 2007) use the functions defined below and are specified in Table 2.2. Mainly, the "harmonicity" of a note is determined by its pitch interval and relative duration with respect to its surrounding notes, and its metrical accent.

Definition 2.1.17 *strong*(n_i) *For quaternary meters a note is strong when its onset is located exactly on the first or third beat of the measure. In ternary meters, it is strong*

[3] it may be calculated (Eck and Casagrande, 2005; Frieler, 2004; Meudic, 2002b; Temperley and Sleator, 1999)

if and only if it onsets on the first beat. For the compound meters, this function can be computed from these two situations. The alphabet for this property is $\Sigma_{r_{acc}} = \mathbf{Bool}$.

Definition 2.1.18 $ratio(n_i) = \frac{\mathrm{r_{dabs}}(n_i)}{\mathrm{r_{dabs}}(n_{i-1})} \times \frac{\mathrm{r_{dabs}}(n_i)}{\mathrm{r_{dabs}}(n_{i+1})}$ *This ratio function determines the importance in terms of duration of a note in the context of its surrounding notes, such that* $ratio(n_i) \geq 1$ *means that the note seems to be harmonic, and* $ratio(n_i) < 1$ *suggests that the note can be a NHT.*

Definition 2.1.19 p_{nitv} *The next interval is the pitch interval* p_{itv} *between a note* n_i *and its successor.* $p_{nitv} = \mathrm{p_{itv}}(n_{i+1})$

NHT type	rules
appoggiatura ('A')	$strong \ \wedge \ \mathrm{p_{itv}} = 0 \wedge \ nextI \in \{-1, -2, +1\}$
passing tone ('P')	$(\neg strong) \wedge (ratio \leq 1) \wedge$ $((\mathrm{p_{itv}} \in \{-1, -2\}) \wedge \ (p_{nitv} \in \{+1, +2\}))$ $\vee \ ((\mathrm{p_{itv}} \in \{+1, +2\}) \wedge \ (p_{nitv} \in \{-1, -2\}))$
neighbor tone ('N')	$(\neg strong) \wedge (ratio \leq 1) \wedge$ $((\mathrm{p_{itv}} \in \{+1, +2\} \ \wedge \ p_{nitv} \in \{+1, +2\})$ $\vee \ (\mathrm{p_{itv}} \in \{-1, -2\} \ \wedge \ p_{nitv} \in \{-1, -2\}))$

Table 2.2: Summary of melodic rules for a note n_i. The parameter n_i for the functions has been omitted for clarity (from (Illescas et al., 2007)).

2.1.4 String representations of monophonic music

As stated in the introduction of this section, string representations have been the most used approach to encode music for comparison so far. The number of ways to encode notes by combining pitch and rhythm representations is high and it will be examined in the experiments section. Now, following the different representation encodings just presented, a string representation framework for monodies will be formally defined.

Strings are defined over a finite alphabet Σ and the empty symbol is denoted by $\lambda \notin \Sigma$. In this work, we use lower-case for symbols of $\Sigma \cup \{\lambda\}$ while upper-case are applied to strings built using symbols in Σ. Given a string X, the reference to the i-th individual symbol in a sequence will be denoted as X_i. The extraction of a substring from the position i to j included will be denoted as $X[i..j]$.

In string representations, note pitch and rhythm are coded with explicit symbols. For representing a melody as a string, symbols from the pitch description alphabet, Σ_{p}, and from that of rhythm, Σ_{r} are combined such that for $S \in \Sigma^*$, $S = \sigma_1\sigma_2...\sigma_{|S|}$, $\Sigma = \Sigma_{\mathrm{p}} \times \Sigma_{\mathrm{r}}$. Any of the alphabets for pitch and rhythm described in the previous section may be used. This code is said to be *coupled*. The pair for a note can only be formed when both dimensions are defined for it.

When both dimensions are handled independently, the representation is said to be *decoupled* or splitted. For decoupled string representations, $\Sigma = \Sigma_p \cup \Sigma_r$, being $\sigma_{2i-1} \in \Sigma_p$ and $\sigma_{2i} \in \Sigma_r$; $i = 1, 2, ..., \frac{|s|}{2}$. Similarly as before, the symbols for a note are included in the string only if both dimensions are defined for it.

The main sense of a decoupled representation is that it allows the comparing method to find a rhythm similarity in part of the song, and a pitch coincidence in other part. For coupled encoding, the similarities must be found in pitch and rhythm simultaneously for the whole sequences.

The symbols can be enriched by the combination of several encodings of the same dimension, i.e. pitch or rhythm. For example, *interval from tonic* can be joined with *pitch contour* information as done in (Ferraro and Hanna, 2007). Regarding rhythm, a combination of *IOI* and accent may be used. For example, in (Robine et al., 2007b), the accent is used in the string distance by giving more importance to notes located on strong beats of the bar, and in the distance by Typke et al. (2003), this property along with the note position from the beginning of the score is also included in the weighting of points. Additionally, the melodic analysis can be added to the tuple representing each note. For coupled representations, all those possibilities can be expressed as $\Sigma = \Sigma_p^+ \times \Sigma_r^+ \times \Sigma_m$.

A more sophisticated model of simultaneous note properties and transformations from the raw data has been proposed recently by Conklin and Anagnostopoulou (2001) in their *multiple viewpoints system*.

Rests

Depending on the source of the symbolic representation to be encoded, rests are explicitly represented or just skipped. If the source is an interpretation where only onsets and offsets are represented (e.g. MIDI notes in a SMF), rests are not explicitly present and they must either be induced or just omitted in the different encodings detailed above. If the files represent score information where rests are explicitly represented, they should be encoded.

In (Mongeau and Sankoff, 1990) rests are indicated with a dummy symbol as a pitch. In the relative representations (intervals and the different contours), usually they are encoded as the unison. In (Uitdenbogerd and Zobel, 1999) a study on the effects of including or omitting rests in the encoding can be found.

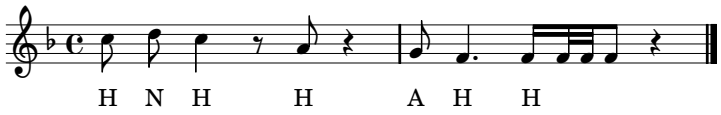
Note that for all pitch encodings in section 2.1.1, rests have been given a special code.

Harmonic tones

From music theory, it is well known that melody ornaments use to appear as NHT like *appogiaturas*, neighbor or passing tones. It is supposed that for comparing music, if notes with a NHT tag are removed, a large part of ornaments will be eliminated, and two works that differ only in the ornamentation will resemble more this way. One possible drawback of this is that those ornaments may be the main motive of the songs. To the

best of our knowledge, apart from our tree codings, only one similarity computation work has dealt with this hypothesis (Robine et al., 2007a).

For the illustration of the different string representations that are detailed in this section, a simple melody has been displayed in figure 2.12 and coded in terms of these pitch and duration properties [4].



Coupled	No Rests	No NHT	String representation using p_{abs} and r_{dabs}
			60 $\frac{1}{2}$ 62 $\frac{1}{2}$ 60 1 128 $\frac{1}{2}$ 57 $\frac{1}{2}$ 128 1 55 $\frac{1}{2}$ 53 $\frac{3}{2}$ 53 $\frac{1}{4}$ 53 $\frac{1}{8}$ 53 $\frac{1}{8}$ 53 $\frac{1}{2}$ 128 1
		X	60 $\frac{1}{2}$ 60 1 128 $\frac{1}{2}$ 57 $\frac{1}{2}$ 128 1 53 $\frac{3}{2}$ 53 $\frac{1}{4}$ 53 $\frac{1}{8}$ 53 $\frac{1}{8}$ 53 $\frac{1}{2}$ 128 1
	X		60 $\frac{1}{2}$ 62 $\frac{1}{2}$ 60 1 57 $\frac{1}{2}$ 55 $\frac{1}{2}$ 53 $\frac{3}{2}$ 53 $\frac{1}{4}$ 53 $\frac{1}{8}$ 53 $\frac{1}{8}$ 53 $\frac{1}{2}$
	X	X	60 $\frac{1}{2}$ 60 1 57 $\frac{1}{2}$ 53 $\frac{3}{2}$ 53 $\frac{1}{4}$ 53 $\frac{1}{8}$ 53 $\frac{1}{8}$ 53 $\frac{1}{2}$
X			[60,$\frac{1}{2}$] [62,$\frac{1}{2}$] [60,1] [128,$\frac{1}{2}$] [57,$\frac{1}{2}$] [128,1] [55,$\frac{1}{2}$] [53,$\frac{3}{2}$] [53,$\frac{1}{4}$] [53,$\frac{1}{8}$] [53,$\frac{1}{8}$] [53,$\frac{1}{2}$] [128,1]
X		X	[60,$\frac{1}{2}$] [60,1] [128,$\frac{1}{2}$] [57,$\frac{1}{2}$] [128,1] [53,$\frac{3}{2}$] [53,$\frac{1}{4}$] [53,$\frac{1}{8}$] [53,$\frac{1}{8}$] [53,$\frac{1}{2}$] [128,1]
X	X		[60,$\frac{1}{2}$] [62,$\frac{1}{2}$] [60,1] [57,$\frac{1}{2}$] [55,$\frac{1}{2}$] [53,$\frac{3}{2}$] [53,$\frac{1}{4}$] [53,$\frac{1}{8}$] [53,$\frac{1}{8}$] [53,$\frac{1}{2}$]
X	X	X	[60,$\frac{1}{2}$] [60,1] [57,$\frac{1}{2}$] [53,$\frac{3}{2}$] [53,$\frac{1}{4}$] [53,$\frac{1}{8}$] [53,$\frac{1}{8}$] [53,$\frac{1}{2}$]

Figure 2.12: Simple example of melody and how it is represented in terms of the different options regarding coupling, rests, and harmonic tones.

String matching algorithms

The similarity between two songs encoded in any string representation is measured with a string matching algorithm. For equal length sequences an euclidean or a hamming distance may be used (Mäkinen et al., 2003; Rolland, 1998). However this is not usually the case as songs use to be of different lengths, being the classical edit distance more suitable for the music comparison task.

The classical edit distance between two strings, also called the *Levenshtein distance*, is the minimal cost to transform one input string into an output one by edit operations. Three kinds of basic edit operations can be used in the transformation process: the insertion of a symbol, the deletion of a symbol, and the substitution of a symbol by another one. To each operation, a so-called *edit cost* c: $\Sigma \times \Sigma \rightarrow \mathbb{R}$ is assigned that can be defined using a costs matrix. Suppose, for example, that strings are built from an alphabet of two letters a and b. The costs for computing an edit distance can be represented by a 3×3 matrix $\mathbf{C}$ where the rows describe the input alphabet and the columns the output one. Therefore, $c(a, b) = \mathbf{C}_{ab}$ denotes the cost of applying the edit operation (a, b) where a is an input symbol while b belongs to the output alphabet, that in our case they are always the same; If $a = \lambda$, the operation denotes an insertion; If $b = \lambda$ the operation is a deletion. Note that the operation (λ, λ) is not allowed.

[4] This musical excerpt does not have other musical meaning than that of illustrating the pitch and duration representations

An example of matrix is shown in Table 2.3, where the substitution costs is 2 for different symbols and 0 for equal symbols. The insertions and deletions, usually denoted as *indel* costs, are assigned a 1.

Table 2.3: Matrix of edit costs. $\mathbf{C}_{ab} = 2$ means that the cost of changing the symbol a in the input string into a b in the output one is 2.

c	λ	a	b
λ	-	1	1
a	1	0	2
b	1	2	0

Definition 2.1.20 *An edit script $e = e_1 \cdots e_n$ is a sequence of edit operations $e_i = (a_i, b_i) \in (\Sigma \cup \{\lambda\}) \times (\Sigma \cup \{\lambda\})$ allowing the transformation of a string X into a string Y. The cost of an edit script $\pi(e)$ is the sum of the costs of the edit operations involved in the script: $\pi(e) = \sum_{i=1}^{n} c(e_i)$.*

Definition 2.1.21 *Let $E(X, Y)$ be the set of all the scripts that enable the emission of Y given X, the edit distance between X and Y is defined by: $d(X, Y) = \min_{e \in E(X,Y)} \pi(e)$.*

For example, according to the costs in Table 2.3, the cost of a possible edit script between two strings a and ab is:

$$\pi((a, a)(\lambda, b)) = \pi((\lambda, a)(a, b)) = 3.$$

The computation of such an edit distance can be done in quadratic time using dynamic programming techniques (see recurrence in Eq. 2.2).

$$\begin{aligned} d_{0,0} &= 0 \\ d_{i,j} &= \min \begin{cases} d_{i-1,j} + c(X_i, \lambda) \\ d_{i,j-1} + c(\lambda, Y_j) \\ d_{i-1,j-1} + c(X_i, Y_j) \end{cases} \\ d(X, Y) &= d_{|X|,|Y|} \end{aligned} \tag{2.2}$$

The difference among works is the different processing of the costs $c\colon \Sigma \times \Sigma \to \mathbb{R}$, some limitations to the value returned by the recurrence formula, and several optimizations on the distance computation algorithm.

Global alignment. The *string edit distance* or *global alignment* (Needleman and Wunsch, 1970) should be used for similar length sequences because it aims to transform a whole sequence in another. Thus, it seems to be useful to compare motives and incipits,

but not to perform a short query to a whole song. It has been used in works such as (Ferraro and Hanna, 2007; Uitdenbogerd, 2002), where a number of different costs for the edit operations can be found. In some cases, only positive values are given, while in others a match between symbols is rewarded with negative values, and mismatch and *indel* operations are penalized with positive values. The *Levenshtein* or *unit cost* distance is a case where the *indel* operations have a unit cost, the cost of substitution is 0 if the compared symbols are equal, and 1 elsewhere.

Following this kind of alignment, Lemström (2000) uses bit-parallelism techniques to speed up process while respecting the transposition invariance.

Dynamic time warping. Some works (Birmingham et al., 2001; Hu and Dannenberg, 2002; Pardo et al., 2004; Tsai et al., 2005) use the so-called *Dynamic Time Warping (DTW)*, borrowed from the speech recognition field that finds an optimal match between two sequences of feature vectors which allows for compressed and stretched regions of the sequence. To compare strings X and Y, algorithm starts by building the *local cost matrix* $\mathbf{L} \in \mathbb{R}^{|X|\times|Y|}$ representing all pairwise distances between X and Y. Once the local cost matrix is built, the algorithm finds the optimal alignment path or *optimal warping path* which runs through the lowest cost areas on the cost matrix. The computation of this alignment path is done by dynamic programming with the recurrence in Eq. 2.3.

$$\begin{aligned} &c(i,j) = \mathbf{L}_{ij} \\ &d_{1,j} = \textstyle\sum_{k=1}^{j} c(X_1, Y_k), \quad 1 \le j \le |Y| \\ &d_{i,1} = \textstyle\sum_{k=1}^{i} c(X_i, Y_1), \quad 1 \le i \le |X| \\ &d_{i,j} = \min\{d_{i-1,j-1}, d_{i-1,j}, d_{i,j-1}\} + c(X_i, Y_j), \quad 1 \le i \le |X|, 1 \le j \le |Y| \end{aligned} \tag{2.3}$$

Local alignment. The *local alignment* (Smith and Waterman, 1981) is a variation of the *string edit distance* where, instead of computing the similarity between two whole strings, it looks for regions that are similar in the two sequences compared. To this end, instead of computing the distance with *min*, the similarity value is obtained with *max*, the match (i.e. $c(X_i, Y_j)$ when $X_i = Y_j$) is rewarded with +2, and *indel* and mismatch is penalized with −1 (see Eq. 2.4). Additionally, to avoid penalizing prefixes or suffixes and thus find a local alignment, the *indel* operations for prefixes and suffixes are not penalized (see the 0 value in the recurrence).

$$\begin{aligned} &d_{0,0} = 0 \\ &d_{i,j} = max \begin{cases} 0 \\ d_{i-1,j} + c(X_i, \lambda) \\ d_{i,j-1} + c(\lambda, Y_j) \\ d_{i-1,j-1} + c(X_i, Y_j) \end{cases} \end{aligned} \tag{2.4}$$

Many of the works that use this string matching variant try to find queries in whole song databases, being the local alignment more adequate than the global. See works (Ferraro and Hanna, 2007; Uitdenbogerd and Zobel, 1999) and *MusicBLAST* (Kilian and

Hoos, 2004) using *BLAST*, a faster version of the (Smith and Waterman, 1981) algorithm by the use of heuristics.

Longest common subsequence (LCS). In some cases, what has been used is the number of coincident symbols in the two compared strings, only rewarding matched symbols (See Eq. 2.5). Works such as (Lemström et al., 2005; Mäkinen et al., 2003) use this algorithm.

$$
\begin{aligned}
& d_{0,0} = 0 \\
& d_{i,j} = max \begin{cases} d_{i-1,j} \\ d_{i,j-1} \\ d_{i-1,j-1} + 1, & \text{if } X_i = Y_j \text{ and } i,j \geq 1 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{2.5}
$$

Mongeau and Sankoff variant. As cited in the introduction of this chapter, an important work in the use of strings for similarity computation was (Mongeau and Sankoff, 1990). Its main contribution are: first, the computing of the substitution cost according to musical rules based on the consonance of intervals, and second, the introduction of two new musical edit operations: *fragmentation* and *consolidation*, trying to model the division of a long note by several shorter notes. These operations leave the recurrence in Eq. 2.2 as the one in Eq. 2.6.

$$
\begin{aligned}
& d_{0,0} = 0 \\
& d_{i,j} = min \begin{cases} d_{i-1,j} + c(X_i, \lambda) \\ d_{i,j-1} + c(\lambda, Y_j) \\ d_{i-1,j-1} + c(X_i, Y_j) \\ d_{i-1,j-k} + c_f(X_i, Y[j-k+1..j]), & 2 \leq k \leq j \text{ (fragmentation)} \\ d_{i-k,j-1} + c_c(X[i-k+1..i], Y_j), & 2 \leq k \leq i \text{ (consolidation)} \end{cases}
\end{aligned}
\tag{2.6}
$$

Consolidation and fragmentation require the definition of two functions to compute their costs $c_f \colon \Sigma \times \Sigma^+$ for of fragmentation and $c_c \colon \Sigma^+ \times \Sigma$ for consolidation. In practice, those operations should only permit fragmentations and consolidations of symbols of Σ representing the same pitch.

The work of Mongeau and Sankoff has been extended or partially adopted in works other such as (Dannenberg and Hu, 2004; Grachten et al., 2004; Kadota et al., 2001; Tsai et al., 2005) with new distances between symbols (Grachten et al., 2005), and both the substitution costs tuned (Gómez et al., 2007a).

δ, γ, and ($\delta-\gamma$) matching. A different way to measure the approximate similarity between two strings is the use of the so-called δ, γ, and ($\delta-\gamma$) matching algorithms (Clifford and Iliopoulos, 2004; Crochemore et al., 2001). These algorithms compute the similarity

by looking at the distance between the individual notes, represented by single symbols in a string. To that end, a distance must be defined between individual symbols. Being defined the subtraction operation between symbols of the alphabet $-: \Sigma \times \Sigma \to \mathbb{Z}$, two symbols $a, b \in \Sigma$ are said to be δ-approximate, denoted by $a \overset{\delta}{=} b$, if and only if

$$|a - b| \leq \delta$$

Two strings X and Y are δ-approximate, denoted by $X \overset{\delta}{=} Y$, if and only if

$$|X| = |Y|, \text{ and } X_i \overset{\delta}{=} Y_i, \forall i \in \{1, ..., |X|\} \tag{2.7}$$

For a given integer γ, two strings X and Y are said to be γ-approximate, denoted by $X \overset{\gamma}{=} Y$, if and only if

$$|X| = |Y|, \text{ and } \sum_{i=1}^{|X|} |X_i - Y_i| \leq \gamma \tag{2.8}$$

The relation $\overset{\delta}{=}$ establishes a maximum difference between individual symbols and the operation $\overset{\gamma}{=}$ bounds the accumulated distance. So, two strings X and Y are said to be $(\delta - \gamma)$-approximate, denoted by $X \overset{\delta,\gamma}{=} Y$, if and only if conditions 2.7 and 2.8 are held.

Obviously, this similarity measure is not useful for the majority of cases in music comparison for the equal length restriction. One needs to use a measure able to work with different length sequences. This can be accomplished using the *δ-approximate pattern matching*, that is designed to look a pattern inside a whole sequence. It is defined as follows: given two strings X and Y, compute all positions j of Y such that

$$X \overset{\delta}{=} Y[j..j + m - 1] \tag{2.9}$$

Similarly, the *(δ, γ)-approximate pattern matching*, is obtained computing all positions such that

$$X \overset{\delta,\gamma}{=} Y[j..j + m - 1] \tag{2.10}$$

Different works have proposed methods to compute these approximate similarity measures adapted to music, such as δ-Tuned-Boyer-Moore, δ-Skip-Search, and δ-Maximal-Shift algorithms (Crochemore et al., 2001). The work by Clifford and Iliopoulos (2004) proposes to allow *gaps* in the comparison, allowing the skipping of notes, what makes possible the comparison of different length strings, and thus, of two whole songs, instead of a query pattern query against a whole length song, as happened in the previous δ-approximate schemes.

The *α-bounded δ-approximate pattern matching* (or (δ, α)-approximate matching) and *α-bounded (δ, γ)-approximate pattern matching* are variants of the just defined schemes where a maximum number α of notes can be skipped, i.e., the size of gaps must be lesser or equal than α.

Comparison of string representations of monodies.

The similarity of two monodies m_a and m_b, with string representations A and B is obtained by using any of the string matching techniques used in the previous section.

For applying those algorithms, the edit costs between individual symbols in the strings (Sec. 2.1) must be defined, or in some cases, just operators that judge whether a symbol can be considered equal to other one. When only a dimension of music is represented, either pitch or duration alone, a single cost function must be defined, $c_p \colon \Sigma_p \to \mathbb{R}$ for pitch and $c_r \colon \Sigma_r \to \mathbb{R}$ for rhythm. The problem arises when both dimensions are represented. In that case the edit cost for pitch and for duration must be computed separately to be combined later. Following the approach introduced by Mongeau and Sankoff (1990), the costs are put together with a linear combination. For a coupled scheme it is defined as expressed in Def. 2.1.22.

Definition 2.1.22 *Let $p(n_i)$ denote the pitch dimension of a note n_i, and $r(n_i)$ its rhythm component, for an edit operation $e_i = (a_i, b_i) \in (\Sigma_\mathrm{p} \times \Sigma_\mathrm{r} \cup \{\lambda\}) \times (\Sigma_\mathrm{p} \times \Sigma_\mathrm{r} \cup \{\lambda\})$, its cost c is defined as the linear combination $c(e_i) = (1-k) \cdot c_p(p(a_i), p(b_i)) \cdot k \cdot c_r(r(a_i), r(b_i))$, where $k \in \mathbb{R}$ is a constant $0 \leq k \leq 1$.*

In the case of using a decoupled representation, it makes no sense to compute the cost of replacing a pitch with a duration symbol. In order to incorporate this restriction in the cost computation, instead of using single cost matrix, two of them will be used, one for the pitches and another for the durations, and the cost of changing a pitch symbol with a duration symbol will be given a ∞ cost (see Def. 2.1.23).

Definition 2.1.23 *For an edit operation $e_i = (a_i, b_i) \in (\Sigma_\mathrm{p} \cup \Sigma_\mathrm{r} \cup \{\lambda\}) \times (\Sigma_\mathrm{p} \cup \Sigma_\mathrm{r} \cup \{\lambda\})$, its cost c is defined as:*

$$c(a_i, b_j) = \begin{cases} c_p(a_i, b_i) & \text{if } a_i \in \Sigma_p \text{ and } b_j \in \Sigma_p \\ c_r(a_i, b_i), & \text{if } a_i \in \Sigma_r \text{ and } b_j \in \Sigma_r \\ \infty & \text{otherwise} \end{cases}$$

The computation of c_p and c_r can be accomplished in a number of ways. One may compare literals, returning 0 for equal symbols and a constant greater than 0 for different literals. Other option is to use a substitution matrix like the one presented in Table 2.3. The values in the matrix may be fixed according to some musical criteria, like those used in (Hanna and Ferraro, 2007; Mongeau and Sankoff, 1990), where the consonance of the intervals to be compared determines the substitution cost, i.e., a fifth is more consonant than a second. Other authors use probabilities of a symbol to be substituted by other (Hu et al., 2002) (expressed as the negative logarithm of the probability of the interval defined by the two pitches $p_i - p_j$), or as we introduced in (Habrard et al., 2008). All the costs can be learned using probabilistic edit distances instead of classical ones. Finally, another option is to define a distance function between alphabet symbols. This is the case usually applied to compare durations. For example,

the $c_{\mathrm{r_{ioi}}}(a, b)$ may be simply defined as $|a-b|$ where a and b are IOI, or more sophisticated, like in (Hu et al., 2002), where the pitch distance between pitches pj and pi is defined as $\min((p_j - p_i)\ mod\ 12, 12 - (p_j - p_i)\ mod\ 12)$, where the possible inversion of the interval defined by both pitches is taken into account. However, in this case, the authors report best results for the probabilistic approach when dealing with QBH inputs.

Normalization and standardization. One thing to deal with is the normalization of measures in order to avoid making the results dependent on the lengths on the compared works. Two common strategies have been applied: the division of the distance or similarity measure by the maximum of the lengths of the sequences (Müllensiefen and Frieler, 2004a), and by the sum of the lengths of the sequences (Serrà and Gomez, 2008) because that is the longest possible path. In order to choose an option, one should divide by the maximum value the distance can produce.

Another issue is the fact that some measures yield a similarity value, while others give a distance. In order to be able to compare them, and eventually, to combine them, they must output the same kind of information. We have decided to use similarity values. In order to transform the normalized distance d into a similarity value we just return $-d$.

Transposition invariance As introduced earlier, the *transposition invariance* is an important property for the comparison methods to be taken into account. Some methods that compare strings use transposition invariance representations of pitch, making the used string matching algorithm transparent to that property. In other cases other non-relative representations that do not hold the *transposition invariance* have been used, so the string matching algorithm must solve this situation. Works like (Allali et al., 2007; Lemström, 2000; Lemström et al., 2005; Mäkinen et al., 2003) for standard edit distance and string alignment algorithms, and (Cantone et al., 2005a,b) for the (δ, α)-approximate matching problem introduce some approaches for the transposition invariant similarity computations. There are also works that use a brute-force solution to find the best transposition (Birmingham et al., 2001; Gómez et al., 2007b). A more elaborated approach is that of Lemström and Ukkonen (Lemström and Ukkonen, 2000) where both the interval representation and the absolute are taken into account to use the best of both: the substitution cost is 0 if the $\mathrm{p_{abs}}$ of the compared notes are equal, when those $\mathrm{p_{abs}}$ are different, their $\mathrm{p_{itv}}$ are compared.

Implication / Realization model.

This model, first introduced by Narmour (1990) and then used for comparing music in (Grachten et al., 2002, 2004, 2005), builds a string using an intermediate representation that tries to capture the sequence of realizations of the user expectations while listening the melody. It is supposed that two similar melodies produce the same intermediate model of expectations so they can be successfully used to compare music.

Given an interval, listeners use to expect a second one of size and direction that depend on the former two. This is what Narmour (1990) defines as *implication*. The

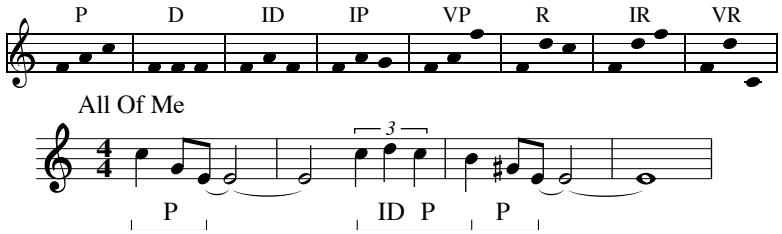


Figure 2.13: Top: Eight of the basic structures of the *I/R* model. Bottom: First measures of All of Me, annotated with *I/R* structures (from (Grachten et al., 2004)).

level of satisfaction that the second interval causes to the listener's expectation is named as *realization*. The different sequences of three notes are represented in eight so-named *Implication / Realization Narmour model (I/R)* structures. See Fig. 2.13 for illustrations of the *I/R* structures and an analysis of a melody excerpt. These structures can be extended with some further properties as the melodic direction of the pattern, the amount of overlap between consecutive *I/R* structures, and the number of notes spanned.

Having represented two melodies as a string of *I/R* structures, any string matching technique can be used just by defining a distance between the two *I/R* strings (Grachten et al., 2005).

2.1.5 *n*-grams

Given a string of symbols encoded in any of the previously exposed pitch and/or rhythm representations, one may group sequences of length n to constitute elements of a set of a so-called n-grams (see Fig. 2.14). The set of all n-grams extracted from a given melody forms a language, the language that describes that melody. It is supposed that the languages of n-grams of two similar songs are also similar. This hypothesis is the one used by the approaches that use n-grams to compare music. They use the techniques previously used in *text information retrieval* in order to accomplish this language comparison (Doraisamy, 2001, 2004; Doraisamy and Rüger, 2002, 2004; Downie, 1999; Melucci and Orio, 2000; Pickens, 2000; Uitdenbogerd and Zobel, 1999).

The main differences among n-gram based systems are the size of the n-grams, the windowing process to extract the n-gram (sliding or not), how they encode the pitch and rhythm properties into ASCII characters [5], and finally the algorithms to evaluate the similarity between n-gram languages. Most of the n-gram systems compare the sets of n-grams collected for each song to evaluate similarity. Other approach is the use of the statistical n-grams models, i.e., the probability of an n-gram is modeled using the probability of its $(n-1)$-gram: $P(x_i|x_{i-1}x_{i-2}...x_{i-n})$. To our best knowledge, only one work (Ling and Sharp, 2004) uses it for music similarity computation, where a Katz's

[5] this is not strictly necessary but eases the used of available software used for text information retrieval, e.g. Hillewaere et al. (2009) use directly a pair of pitch and rhythm

Representation	Actual value
String	trqprqu
1-grams	{p, q, r, t, u}
2-grams	{pr, qp, qu, rq, tr}
3-grams	{prq, qpr, rqp, rqu, trq}
4-grams	{prqu, qprq, rqpr, trqp}
5-grams	{qprqu, rqprq, trqpr}
6-grams	{rqprqu, trqprq}
7-grams	{trqprqu}

Figure 2.14: A string in *directed-modulo12* (from Fig. 2.3) and the corresponding n-gram sets.

backing-off model is used to smooth the probability estimation and a multi-class Support Vector Machine (SVM) to classify.

Uitdenbogerd (2002); Uitdenbogerd and Zobel (2002); Uitdenbogerd et al. (2006) state that the best n-gram size ranges from four to seven. Shorter sizes lead to high recall and low precision, longer sizes produce better precision but with the cost of a lower recall. In order to overcome this trade-off problem, Orio (2005) proposes to combine different systems, with the data fusion technique as described in Lee (Lee, 1997), built using different sizes and properties, leading to better precision than when using a single size. In the probabilistic approach, the longer the n-grams are, the bigger the training corpora needs to be. Ling and Sharp (2004) report an optimal size of 2 or 3 for music comparison.

For the non-probabilistic approach, all used similarity measures use formulas based on frequencies of occurrences of the different n-grams in the compared songs. One of them, named *count distinct measure* (Uitdenbogerd, 2002) counts the number of n-grams that occur in both strings (Def. 2.1.24).

Definition 2.1.24 *Let τ_A and τ_B be the set of n-grams of strings A and B, respectively. The similarity between both strings can be defined as:*

$$S(A, B) = |\tau_A \cap \tau_B|$$

Another commonly used way to compare n-gram sets is the *TF-IDF* family of similarity measures based on *Term Frequency (TF)* (relative to the occurrences of a term in both songs to be compared) and the *Inverse Document Frequency (IDF)* (referent the presence of the term also in the rest of the corpus). These measures try to give more weight to those terms that are more discriminant or less common in the corpus.

Another similarity measure borrowed from the text information is that of the *cosine* and some variants as the *Kaszkiel* and *pivoted cosine*, that Serrano and Iñesta (2006b)

report to yield better results. In that work the *hansonian* representation is used (see Fig. 2.9 in page 19), that can be regarded as a special case of transposition invariant bigram.

2.1.6 Geometric modeling

First explored by Maidín (1998), geometric representations of music have been widely explored in the literature (Aloupis et al., 2006; Lemström and Pienimaki, 2007; Lemström et al., 2008; Lubiw and Tanur, 2004; Romming and Selfridge-Field, 2007; Ukkonen et al., 2003a,b), mainly in the polyphonic context, as they will be reviewed below (see Sect. 2.2).

Maidín represents a monody as a graphic representation of the pitch-duration contour. In a cartesian plot, pitch is represented in the y-axis and time in the x-axis in something similar to a piano-roll sheet where the notes are connected with lines (see Fig. 2.15). The similarity between two melodies is the area between the lines that represent the contour of each melody (see Fig. 2.16).

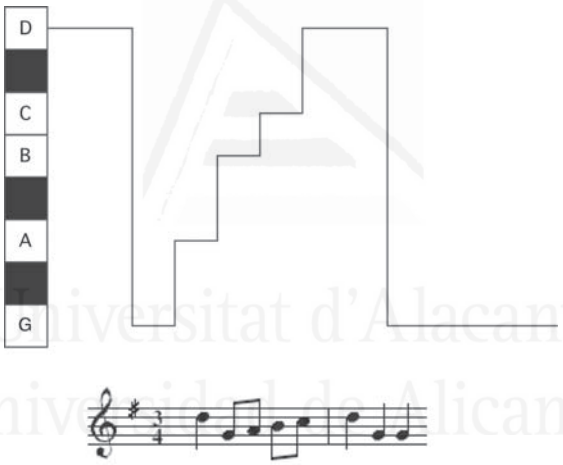


Figure 2.15: Geometric representation of music (from (Aloupis et al., 2006)).

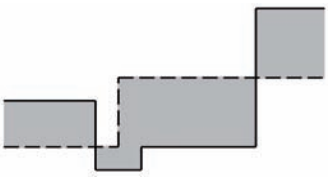


Figure 2.16: Geometric comparison of music (from (Tanur, 2005)).

In order to solve the *transposition invariance* and *tempo invariance*, Aloupis et al. (2006) propose an algorithm to minimize the area between line contours (see Fig. 2.17) by permitting the shift of horizontal and vertical dimensions.

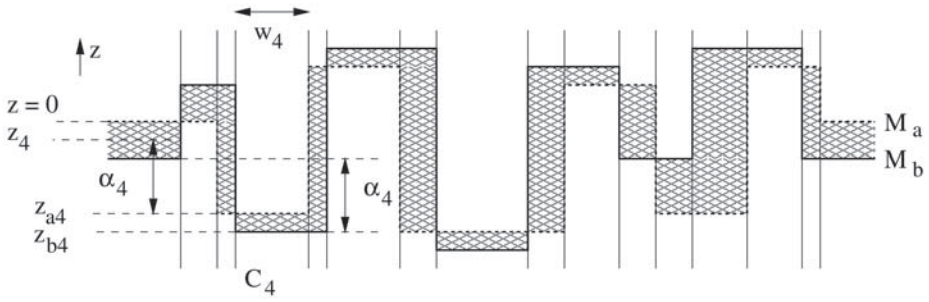


Figure 2.17: Minimization of the area between contours to find best *transposition* and *tempo invariance* (from (Aloupis et al., 2006)).

The main differences among methods can be found in the units used to map the pitch property in the y axis, and time in the x axis, i.e., the quantization used for both characteristics, and in the way the graphical representations of the melodies are compared.

Earth Mover Distance

In his model, Typke (2007) puts notes in the aforementioned cartesian plot, where duration of notes is represented by the weight or radius of the points in the plot (see Fig. 2.18). Having two pieces represented this way, their similarity is obtained by measuring the effort it takes to transform one weighted point set into the other in a transportation distance so-called *Earth Mover Distance (EMD)*. The transposition invariance is obtained as in (Maidín, 1998) by moving the two melodies where the distance is minimum.

2.1.7 Statistical measures

Engelbrecht (2002) introduces a different way to encode music, modeling a melody as defined in Def. 2.1.25. In this case, instead of editing one string trying to reach or align to the other, what is measured is how similar they are using some statistical measures like correlations. Thus, this technique is closer to geometrical methods than to string ones.

Definition 2.1.25 *A (mathematical) melody is a quadruple* (T, p, l, P) *consisting of sequence of onset times* $T \subset \mathbb{Z}$*, a pitch progression function* $p\colon T \to \mathbb{Z}$*, a loudness progression function* $l\colon T \to \mathbb{Z}$ *and a probability density* $P\colon T \to [0, 1]$*.*

In the original work (Engelbrecht, 2002), rhythm and loudness are considered dependent, and ignored.

In order to compare two melodies or quadruples the same statistical measures have been applied. Namely, correlation, central moments, and entropy-based measures.

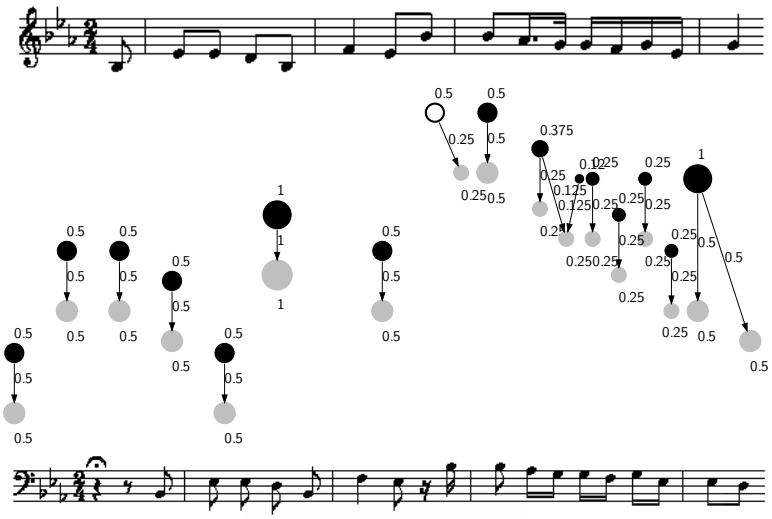


Figure 2.18: Two pieces of music and their representations as weighted point sets, along with the optimum flow that turns one of the point sets into the other. To make the two point sets easier to distinguish, they are shifted apart (top score is represented by black points, bottom score by gray points). For the actual distance calculation, they are positioned on top of one another (from (Typke et al., 2005b)).

2.1.8 Graph representations

In (Pinto et al., 2007), signatures of monodies are created using directed graphs from the pitch profile. In their graph, each pitch class is associated with a node, and each melody transition from a pitch class to another with a label. The label represents the frequency of the interval defined by the two pitch classes the edge connects (see Fig. 2.19). The graph represents also the interval from the last note of the melody to the first one.

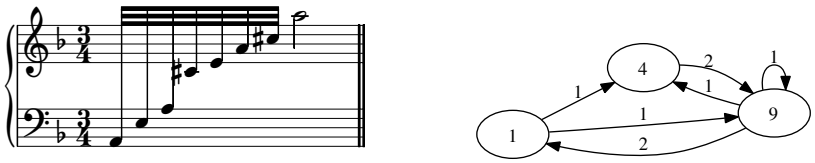
Let M be a melodic sequence of length $m = |M|$ defined by the sequence of pitch classes $\{\mathrm{p}_{12}(j)\}_{1 \leq j \leq m}$, the graph G is defined with vertex set $V_G = \mathbb{Z}_{12}$ and edge set whose elements are the edges a_j between vertices $v_i \in V_G$ such that

$$a_j = \begin{cases} v_j \to v_{j+1} & \text{for every couple } (\mathrm{p}_{12}(j), \mathrm{p}_{12}(j+1)) \subseteq M \\ v_m \to v_1 & \text{for the couple } (\mathrm{p}_{12}(m), \mathrm{p}_{12}(1)) \end{cases}$$

The arrow $a_m : v_m \to p_1$ does not represent an actual interval in the melody but it has been added for symmetry reasons.

The graph can be expressed as an adjacency matrix $\mathbf{A}(G)$, where the entry (i, j) represents the number of oriented edges from vertex i to vertex j. The laplacian matrix $\mathbf{L}(G)$ computed as $\mathbf{D}(G) - \mathbf{A}(G)$, being $\mathbf{D}(G)$ the degree diagonal matrix where entry (i, i) represents the sum of the cardinalities of outgoing edges of vertex i. The laplacian matrix has more representational power than the adjacency matrix, in terms of resulting in fewer cospectral or isospectral graphs. Two graphs are called cospectral if they

have the same eigenvalues. Thus, in order to measure the similarity of two graphs the eigenvalues of their laplacian matrices are computed, and then compared with an euclidean distance. The advantage of this method as noted by the author is that the measure is invariant to column shifts and then make the distance invariant to pitch transposition.



(a) (b)

Figure 2.19: Graph representation of a score

2.1.9 Hidden Markov Models

Two main works use Hidden Markov Model (HMM) (Rabiner, 1989) to encode and compare music: (Pardo et al., 2004) and (Pickens, 2004). The former uses note properties to build the HMM. The latter, instead, use chords as the source to construct the models, and will be reviewed below within the polyphonic music section (Section 2.2).

In the work of Pardo et al. (2004) and similarly in (Durey and Clements, 2001), states represent duples $\Sigma_{\text{p}_{\text{itv}}} \times \Sigma_{\text{r}_{\text{dior}}}$, where $\Sigma_{\text{r}_{\text{dior}}}$ is just a discretization of the $\Sigma_{\text{r}_{\text{ior}}}$ representation. All states are likely to be start state and ending states. Directed edges between states contain transition probabilities (see Fig. 2.20 and 2.21 with only the non-zero probability edges plotted). Until here, this model resembles somewhat the graphs proposed by Pinto reviewed above in Sec. 2.1.8. What really makes the systems different is the way melodies are compared. It is performed by encoding a target song as a HMM. The query song is treated as an observation sequence, and the similarity value is obtained from the likelihood of the HMM to generate this observed sequence. The observations are encoded also with duples $\Sigma_{\text{p}_{\text{itv}}} \times \Sigma_{\text{r}_{\text{ior}}}$.

The probability estimation of observations is performed by a standard counting process: "given a state s, the probability of observation o_i can be estimated by counting how often o_i is seen in state s, compared to the total number of times s is encountered" (see Eq. 2.11).

$$P(o_i|s) = \frac{\text{count}(o_i, s)}{\sum_{j=1}^{|\Sigma_{\text{p}_{\text{itv}}}| \cdot |\Sigma_{r_{dior}}|} \text{count}(o_j, s)} \tag{2.11}$$

They found that in order to estimate the probabilities of the paired observations and the hidden states would require a huge corpus. In order to make it more tractable they assume conditional independence between pitch interval and IOR:

$$P(o|s) = P(\text{p}_{\text{itv}}(o)|\,\text{p}_{\text{itv}}(s)) \times P(\text{r}_{\text{ior}}(o)|\,\text{r}_{\text{ior}}(s))$$

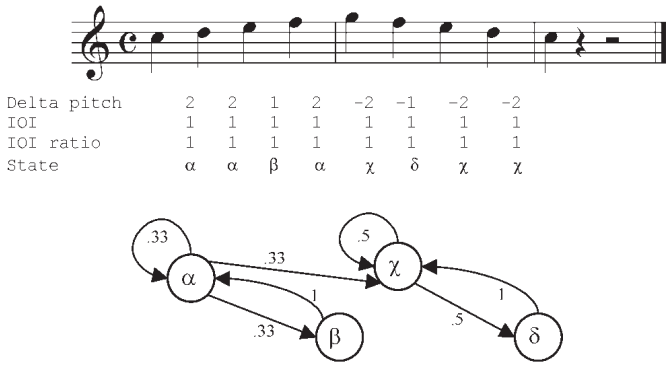


Figure 2.20: Markov model for a scalar passage (from (Pardo et al., 2004)).

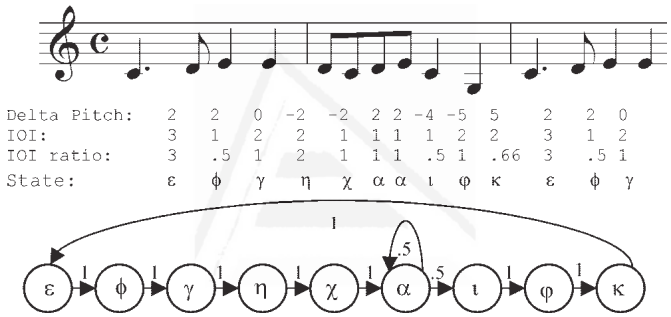


Figure 2.21: Markov model for a *Alouette* fragment (from (Pardo et al., 2004)).

In their system (Pardo et al., 2004), they do not perform the usual Baum-Welch Training, because their model requires only one exemplar, paired with a known error model defined in their paper obtained from a set of sung melodies. As mentioned above, the similarity value is the likelihood of the observed sequence given the HMM: let $\alpha_t(i)$ be the probability of the partial observation sequence $O_t = \{o(1), o(2), ..., o(t)\}$ to be produced by all possible state sequences that end at the i-th state:

$$\alpha_t(i) = P(o(1), o(2), ..., o(t)|q(t) = q_i)$$

Then, the unconditional probability of the partial observation sequence is the sum of $\alpha_t(i)$ over all N states. For computing that probability the *forward algorithm* is used, computed recursively as detailed in (Rabiner, 1989).

2.1.10 Neural networks

To the best of our knowledge, few works have been used to compare music in symbolic format. Two works (Harford, 2003; Yahzong et al., 2001) give a shallow description of the use of two different neural network architectures to indexing and retrieving of music.

One (Harford, 2003) is based on a fusion of SONNET (Self-Organizing Neural Network) architecture and an associative map derived from ARTMAP (Carpenter et al., 1991). By feeding the neural network input cells with pitch classes and IOI, the similarity in the way cells get activated for each of the songs to be compared is measured.

The second work (Yahzong et al., 2001) is based on the ability of the recurrent neural networks to predict time series (Elman, 1990). The input and output of the network is a series of pitch contours. Each of the songs to be compared is fed to the network and the output is collected. The output string is then compared using a distance named by the authors as *correlation degree*.

2.2 Polyphonic music

In order to choose a suitable representation for polyphonic music the kind of polyphony to be compared has to be established. In the previous definition of polyphonic music (page 4) we have differentiated two kinds of polyphony: one that considers a source as polyphonic if there are simultaneous notes played at a time, and other when there are simultaneous voices, even if they are arranged in a monody.

Working with the first model is the most straight-forward option and it is always available independently of the music source, notated or raw played. The second one is more difficult because some separation of voices is required previous to measure similarity. The separation of the individual voices from a polyphonic input can be done easily if the input is a file with already differentiated voices in the form of staves or instruments (like in MusicXML), tracks or channels (as in MIDI files). In turn, each voice may be polyphonic. If the input is a raw polyphonic stream, some techniques have been proposed recently to obtain that separation (Rafailidis et al., 2008).

Having done this consideration, polyphonic music similarity measurement systems can be classified in:

- Those systems that compare directly two polyphonic streams without any voice separation (Clausen et al., 2000; Lemström et al., 2008; Rizo et al., 2009b). We name them *pure polyphonic*. This is the approach followed in this dissertation.

- Algorithms or techniques that are able only to compare two monophonic lines. They may be denoted just as *monophonic* methods. They need some kind of reduction of the polyphony into monophony. The *skyline* (Uitdenbogerd and Zobel, 1998) seems to be best and most widely applied method to perform that reduction (see Uitdenbogerd and Zobel (2002, 2004) for a study on these techniques). Besides those reduction techniques, and under the assumption that the musical work is composed of a melody line with an accompaniment, that melody line can be extracted by using algorithms that classify voices as melody or accompaniment and are able to obtain it. We have introduced a method that seems to be robust enough for that task (Rizo et al., 2006a). As far as we know, the use of the melody track to compare polyphonic files has not been approached in the literature, although

there is a similar approach (Meek and Birmingham, 2001) that looks for the song main themes ranked by importance that may be used later to compare the song.

- Other group of methods try to locate a monophonic query into polyphonic documents, optionally with gaps, and respecting transposition invariance, either with some kind of voice separation looking for the part or voice that returns the best matching (Lemström and Mäkinen, 2005; Pardo and Sanghi, 2005) or just in the raw polyphonic sequence (Hanna et al., 2008). We denote them as *mono to poly* following the algorithm named *monopoly* in (Lemström and Tarhio, 2003).

- Finally, we consider an additional group that, to the best of our knowledge, has not been reported in the literature. It consists in obtaining the best alignment of the separated voices of the two songs to be compared. In order to align voices, a pre-filtering process to eliminate almost repeated voices within the songs should be applied. Both for that filtering, and for the alignment, the similarity of the voices must be measured. This can be done by using any of the other polyphonic methods. We have used this approach to refine one of the corpus used in the experimentation in order to find repeated MIDI files that differ only in the track layout.

Under the perspective of symbolic music encoding, the separation of voices only leads to a high level grouping of notes into several possibly polyphonic sequences, but it does not change anything in the simultaneous note encoding scheme. There are two main ways in which a polyphonic source can be encoded: either by representing individual notes, or by encoding simultaneous pitches, sometimes called unnamed chords [6] or *sonorities* (Taube, 1999), with more or less information of rhythm, and optionally with some kind of processing to map these pitches to named chords.

The main existing methods in the literature to encode and compare music roughly lie inside the same groups we have made for monophonic music: strings, n-grams, and geometric modeling. Besides, some methods based on the harmonic modeling have been proposed that will be reviewed below in the following sections.

2.2.1 String modeling

When only a note is sounding at a time in a sequential order the string representation of music is straight-forward. However, when the input is a polyphonic source this is not the case. An interesting analysis of different string representations for polyphonic music is developed in (Lemström and Pienimäki, 2006). In that review, three ways of building a string of symbols from the polyphonic source are proposed, namely *interleaved* and *non-interleaved* representations, and *onset-based* representation (see Fig. 2.22).

The *non-interleaved* representation corresponds to a sequential arrangement of all monophonic lines of the polyphonic piece, separated by any symbol not belonging to the alphabet. The main problem of this representation is that the meaningful extraction

[6] versus named chords as C♯$Maj7$

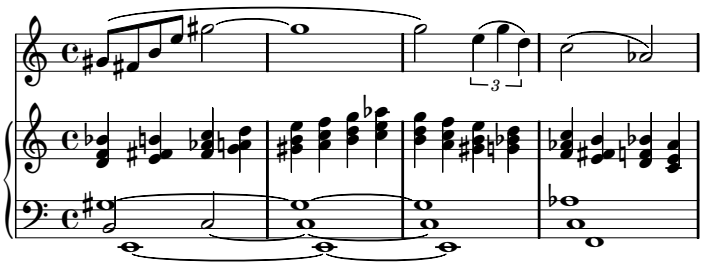
of monophonic lines from the polyphonic material is not solved so far (Rafailidis et al., 2008). The other possibility, the representation of all possible monophonic lines lead to a combinatorial explosion of possibilities, besides, the pattern matching process would generate too many false positives. The advantage is that any interval-like representation may be used to deal with transposition invariance.

A more adequate encoding is that of the *interleaved* representation (Pienimäki, 2002). All notes in the polyphony are sorted lexicographically, i.e., first the onset is considered and for equal onset times the pitch is used as sorting criteria. When using edit distances to compute similarity, big gaps may occur between contiguously perceived notes. Increasing the maximum allowed gap the same false positives problem is committed. Furthermore, the transposition invariance can be only managed by computing the brute-force solution of all possible transpositions.

The two previous representations model individual notes. If *sonorities* are modeled instead, the so-called *onset-based* representation may be used. After quantizing the input, notes with the same onset are grouped in sets of pitches. Given this representation, the search of a monophonic line in that structure is more or less straightforward, being the efficiency and, again, the transposition invariance the problem to be handled (Lemström and Tarhio, 2003).

As noted in (Lemström and Pienimäki, 2006), the duration information of all but the *non-interleaved* representation is lost, or at least, transformed by the fragmentation of long notes into several notes occurring in contiguous onsets or groups. This is the same case that was aforementioned in page 24 regarding the tied notes. Some authors have tackled this problem by fragmenting long notes into fixed duration tied notes as in (Hanna et al., 2008) (see Fig. 2.23), and the *notebits* in (Pardo and Sanghi, 2005).

A similar representation to the *onset-based* is the *quotiented sequence* in (Hanna and Ferraro, 2007) as can be noticed in Fig. 2.24. Citing (Hanna and Ferraro, 2007), "basically, quotiented sequences refer to sequences whose nodes are also sequences". In that work, *quotiented sequence* is just a string of sets of notes with their individual durations mapped into unnamed chords. The similarity is computed with adapted string matching techniques, and the comparison between sets of notes is achieved by comparing a note of the mono query with all the notes from the set, and considering only the most similar for the global score of similarity. If the named chords of the polyphonic source are extracted previously with any of the available methods (Pardo and Birmingham, 2002a; Temperley, 2001), this mapping from notes to chords is skipped and the polyphonic piece may be regarded as a linear string of chord symbols (e.g. CM FM G7 CM ...) whose similarity can be matched with any string matching algorithm. They key issue here again is the comparison of individual chords. In (Pienimäki and Lemström, 2004) that comparison is performed based on the chord tonal functions described in Def. 2.2.1.



(a) An excerpt of polyphonic music from Einojuhani Rautavaara's opera Thomas (1985).

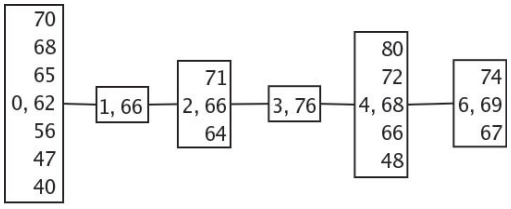
(0,68) (1,66) (2,71) (3,76) (4,80) %
(0,70) (2,71) (4,72) (6,74)%
(0,65) (2,66) (4,68) (6,69)%
(0,62) (2,64) (4,66) (6,67)%
(0,56) % (0,47) (4,48) % (0,40) %

(0,68) (1,66) (2,71) (3,76) (4,80)%
(0,70) (2,71) (4,72) (6,74)%
(0,65) (2,66) (4,68) (6,69)%
(0,62) (2,64) (4,66) (6,67) %
(0,56) % (0,47) (4,48) % (0,40) %

(b) Non-interleaved representation of the first bar of the excerpt in Fig. (a). Notes are represented by ordered pairs (o,p) where o represents the onset time and p the pitch information. We have emphasized (left) a solid occurrence, and (right) a distributed occurrence of an imaginary query pattern.

(0,40) (0,47) (0,56) (0.62) (0,65)
(0,68) (0,70) (1,66) (2,64) (2,66)
(2,71) (3,76) (4,48) (4,66) (4,68)
(4,72) (4,80) (6,67) (6,69) (6,74)

(c) The first bar of the excerpt in Fig. (a) in an onset-based representation.



(d) The first bar of the excerpt in Fig. (a) in an onset-based representation.

Figure 2.22: String polyphony representations (from (Lemström and Pienimäki, 2006)).

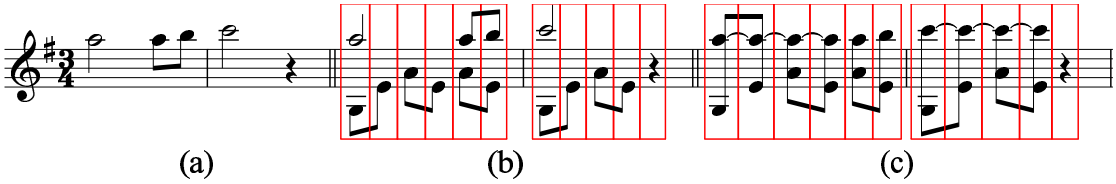


Figure 2.23: Illustration of the representation of polyphonic music with tied notes: a monophonic query corresponding to one voice (a), a first polyphonic representation based on onsets (b), and the representation chosen that considers tied notes (c) (from (Hanna et al., 2008)).

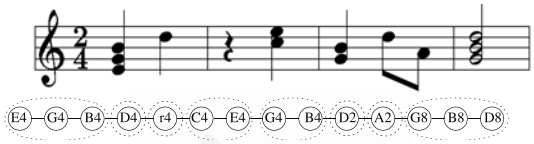


Figure 2.24: Example of polyphonic musical score and its related quotiented sequence (from (Hanna and Ferraro, 2007)).

Definition 2.2.1 *Let A and B be two chords, the distance $\mathcal{D}$ between them is defined as:*

$$\mathcal{D}(A,B) = \begin{cases} 0, & \text{if } A \text{ and } B \text{ are exactly the same chord} \\ a_1, & \text{if } A \text{ and } B \text{ are inversions of same chord} \\ a_2, & \text{if } A \text{ and } B \text{ are different chords of same function} \\ a_3, & \text{if } A \text{ and } B \text{ are major and minor forms of same chord} \\ 1, & \text{if } A \text{ and } B \text{ otherwise} \end{cases}$$

where coefficients a_1, a_2, and a_3 must be defined such that they weight the functional differences of chords: $a_1 \leq a_2 \leq a_3$. Tonal functions are defined traditionally: tonic *includes degrees I and VI, degrees V and VII are* dominant*, degrees IV and II are considered* subdominant*, and finally the III degree is not in any of those groups.*

The so-called *Tonal Pitch Step Distance* (Haas et al., 2008) that measures the distance between two chords, can be used also instead of that defined in Def. 2.2.1 to compute the substitution cost in any string matching algorithm on chord sequences. This distance between chords is based on the relative distance from each of the two compared chords to that defined on the tonic of the song key. As depicted in Fig. 2.25, now the problem is converted in a geometric one that is solved with the techniques in (Aloupis et al., 2006) described above in Fig. 2.17 of Section 2.1.6. Other similar distance that could be used is that defined in (Lerdahl, 2001).

Another possibility is that introduced in (Allali et al., 2007) to look for a monophonic query into a polyphonic song coded as a sequence of unnamed chords: following a

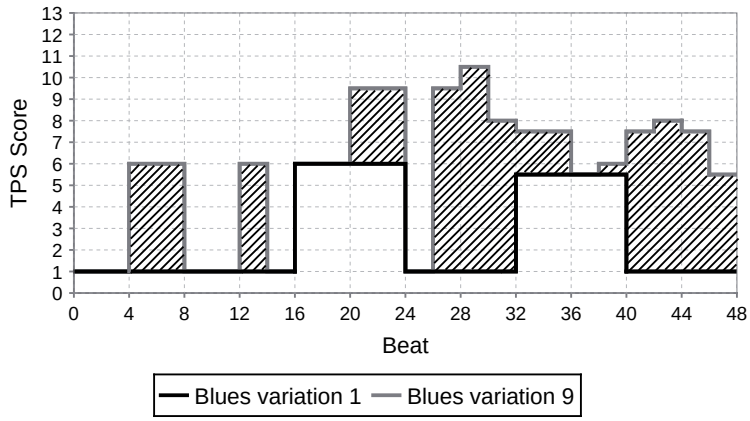


Figure 2.25: A plot of the step functions of two blues variations. The TPSD is defined as the hatched area between Blues variation 1 and 9 divided by the length of the shortest step function (which can be either one in this example) (from (Haas et al., 2008)).

standard string matching algorithm, the substitution cost of an individual note with a chord is that of the lowest cost of that note and each note of the chord.

Instead of using chords, in one of the several schemes presented in (Müllensiefen and Frieler, 2004a) named *Harmonic Edit-Distance*, the most probable tonality name is extracted for each bar in a so-called *harmonic string*, that will be compared using classical string matching algorithms.

In (Robine et al., 2008) different levels of descriptors are proposed: in the first level the key sequences, the second level contain the chord sequences, and finally the actual notes.

The fourth considered option is to obtain a sequence of descriptors from the polyphonic content. The difference between this option and the previous ones is that now notes are neither considered individually nor grouped, instead of that features are extracted with some windowing process. This is usually the approach used in audio cover identification (Adams et al., 2004; Bello, 2007; Downie et al., 2008; Serrà and Gomez, 2008). Usually a sequence is built with *chromagrams* (see Fig. 2.26) with 12 octave independent bins or *Harmonic Pitch Class Profiles (HPCP)* (Gómez, 2006) with 36 octave independent bins histogram representing the relative intensity of each semitone, and 1/3 of the 12 semitone equal tempered scale respectively. After that, in the case of (Serrà and Gomez, 2008) a global HPCP for each compared song is built as the average of all the HPCP, and it is used to deal with transposition invariance by looking for the rotation of the global HPCP that maximizes the dot product of both HPCP. After applying the best transposition, songs are compared by using sequence comparison methods like DTW or Smith-Waterman edit distance (see Section 2.1.4).

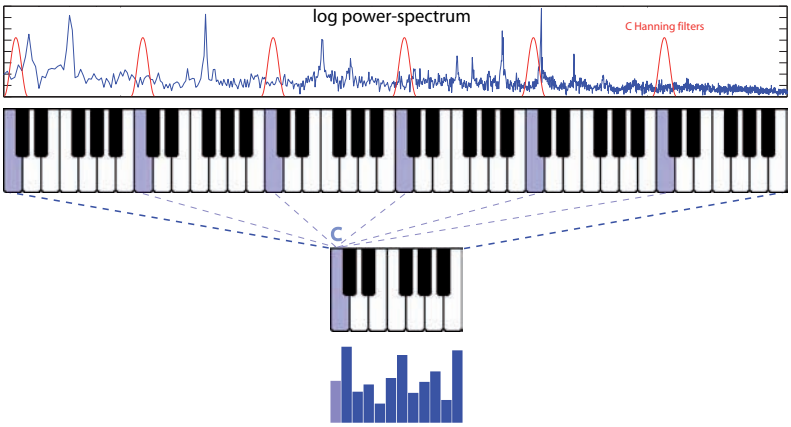


Figure 2.26: Computing schematic for building a chromagram. The power spectrum energy is accumulated into 12 pitch classes through a bank of filters tuned to the equal temperament chromatic scale (from (Jehan, 2005)).

Multi-feature polyphonic comparison.

Arguing that text-matching or pure-mathematical algorithms are not suitable for comparing music, and that more cognitive and musicological information should be used for the task, in (Meudic, 2003) it is proposed to extract three different sets of features to compute similarity between songs: pitches, contours, and rhythm.

Following the *onset-based* representation, he models the polyphonic sequence of all merged voices using equally spaced *beat sequences*, representing pitches, durations, and onsets extracted using their own beat tracking algorithm (Meudic, 2002a) (Fig. 2.27). The similarity value is a linear combination of the individual similarities of pitches, contours, and rhythm. Songs are previously normalized so that they contain the same number of *beat sequences*. For computing the pitch similarity only downbeats are considered, leaving apart the continuations (equivalent to the aforementioned tied notes), reducing thus the running times of algorithms under the rationale that "two sequences whose pitches coincide on the downbeat but differ elsewhere are often recognized as very similar" [7]. As each *beat sequence* is a pitch set, he introduces a distance between pitch sets based on the chords they form and the intersection between them: looking for exact occurrences of pitches or transposed versions of chords. The rhythmical similarity is obtained just by aligning the two compared sequences and looking for the coincidences in *beat sequences*. The upper and lower contours are compared using the intervals that produce them, heuristically taking a threshold of 5 half-tones to consider each pair of intervals similar.

[7] he states this fact has been confirmed by his experiments

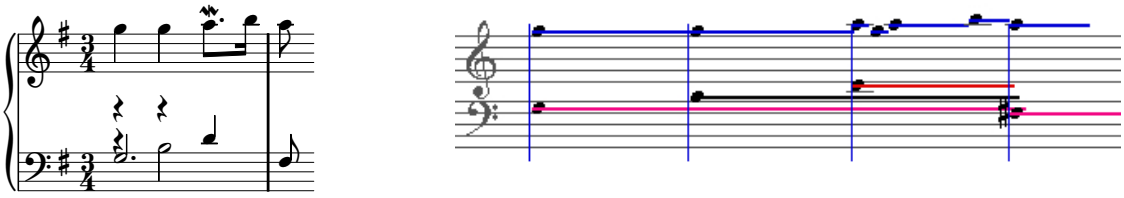
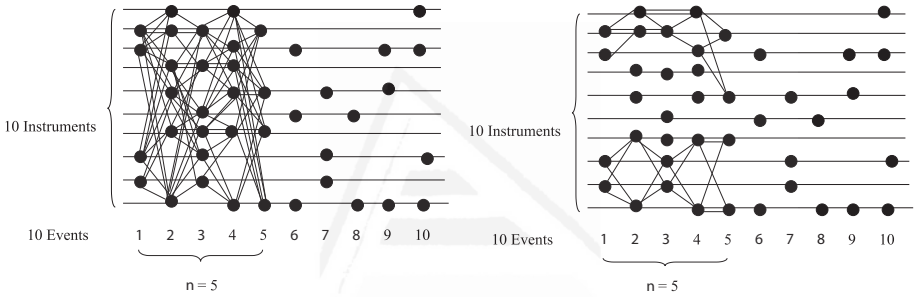


Figure 2.27: Beginning of the *Aria* of the Bach's *Goldberg variations*, BWV 988. The vertical lines delimit the beat segments. Horizontal lines are the durations of each event (from (Meudic, 2003)).



(a) All possible paths (b) Path selection

Figure 2.28: *n*-gram extraction from polyphonic sequences (from (Doraisamy, 2004)).

2.2.2 *n*-grams

In her thesis, Doraisamy (2004) uses non-probabilistic *n*-gram models, or *bag of words* models to index music collections and evaluate similarity as introduced in the monophonic methods review (see page 36). In her model a mapping from intervals and IOR to ASCII printable characters is presented, requiring a linear sequence to be computed. To this end, after a windowing process of the input polyphonic content, all possible pairs are obtained and converted into characters (see Fig. 2.28a). However, the combinatorial explosion of possibilities of all pairs when dealing with actual polyphonic pieces requires the pruning of some of those possibilities keeping only important paths (see Fig.2.28b). One straight-forward possibility is to use those paths that define the top and bottom envelope of the sequence. Other one is to keep the highest pitch at each onset, what is equivalent to perform a *skyline* reduction. For the results and conclusions presented in that thesis, the last possibility may be considered the best and most simple. The model also includes an extension of the classical *bag of words* model to include positional information in a polyphonic context, in such a way that multiple words may be located at the same position.

2.2.3 Harmonic methods

Pickens (2004) uses a harmonic characterization of the musical pieces to obtain their similarity or *evocativeness*, on the hypothesis that the chord sequence of a song cover is enough to evoke the original version. From the raw polyphonic content (either obtained from an audio to symbolic process or directly in that format) the so-called *simultaneities* are obtained and processed so that finally only pitch classes without durations are kept (see Fig. 2.29), being this sequence of *simultaneities* (encoded each simultaneity as a bit-vector of size 12) the input to the system, what leads to a system that is not note invariant to transposition [8]. That harmonic characterization is modeled using a language modeling approach, assuming that a piece of music d is generated by a model $p(d|M_d)$, being the unknown parameter the model M_d. Two ways of constructing M_d were presented.

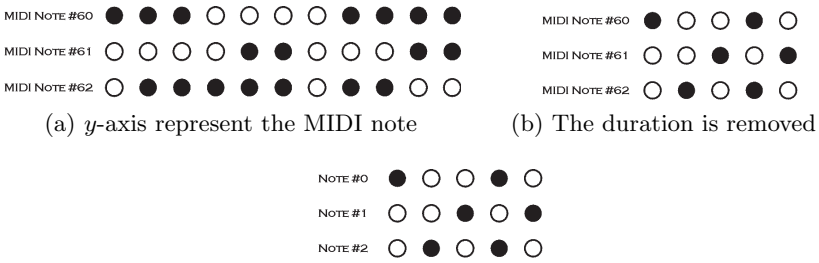
First, by means of a HMM for each piece, where the states are the 24 different major and minor triads, the observations are the sequence of *simultaneities*, and the transitions of the model are initialized according to the classical circle of fifths where tonalities are replaced with chords, upon the assumption that closer chords in that circle should lead to higher transition probabilities. The observation probabilities for each state are initialized using two different schemes, both use a measure of coincidence between the triad notes and the piece actual *simultaneities* notes. Finally the model is estimated using the standard Baum-Welch parameter re-estimation algorithm (Rabiner, 1989). Given a model for each piece, the similarity value of a new song or query is the likelihood of the model of having generated the query observation sequence, obtained by running the classical *forward algorithm* on the HMM.

The second model tries to avoid the lack of generalization due to the small training dataset used because each model is estimated for each song. This is accomplished, at least for the results and conclusions given, by the use of 1^{st}, 2^{nd}, and 0^{th} order Markov models. Now, the *simultaneities* are mapped heuristically into chords, and with the sequences of chords, a Markov model is constructed for each piece. One interesting point is that, instead of mapping each *simultaneities* into a single chord, a probability is assigned to each chord in what the author denotes as *partial observation vector*. Now, given a query song, it is modeled with a Markov model the same way as the collection pieces, and the similarity value between them is obtained by measuring the dissimilarity of the constructed model for the query and each collection piece model, being this dissimilarity measure the conditional relative entropy.

The harmonic characterization of pieces in order to compare songs has been used also in (Mardirossian and Chew, 2006). In that work, the 55-bin harmonic profile from $C\flat\flat$ to $C\,\mathrm{x}$ of each song is obtained with their *Center of Effect Generator* algorithm based on the *spiral array* (Chew, 2000), and then compared using either the L^1 or L^2 norms (see Fig. 2.30).

Following the early proposal of Shmulevich et al. (2001) to use Krumhansl's tone profiles (Krumhansl, 1990) for symbolic music retrieval, Müllensiefen and Frieler (2004a)

[8] however, all possible transpositions could be obtained, or use an interval from tonic pitch encoding p_{ift} instead.



(a) y-axis represent the MIDI note (b) The duration is removed

(c) y-axis represent the pitch class

Figure 2.29: Pickens' *simultaneities*. x-axis represents time, black circles represent notes sounding at that moment (from (Pickens, 2004)).

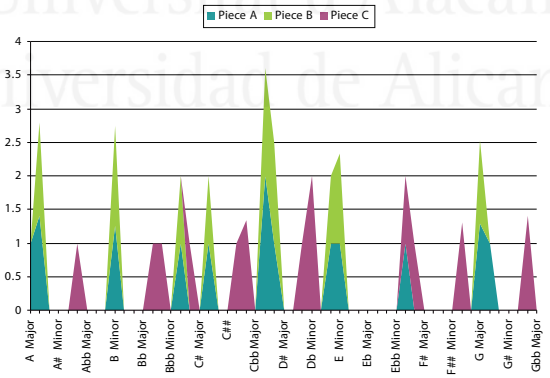


Figure 2.30: Plot of key histogram for three pieces (from (Mardirossian and Chew, 2006)). Note that not all 55 possible major and minor keys from C𝄫 to C x are present in the x-axis labels.

apply these pitch profiles in what they name *harmonic vector correlation* to compare music. A *harmonic vector* for a bar based on the Krumhansl pitch profiles (Krumhansl, 1990) is computed as follows: given the local tonality of the bar, a vector with a cell for each pitch class is constructed. The value for each pitch class is computed multiplying the sum of the IOIs of the notes of that pitch in the bar by the weight of that pitch class in the Krumhansl pitch profiles [9].

With those harmonic vectors, melodic similarity is measured by computing the correlation of those vectors for the two melodies compared. As both songs may have different lengths in number of bars, the minimum of the lengths is used and the exceeding bars are discarded. Although only used for comparing monophonic music, it may be easily extended to polyphony. The same system structure is used in (Miura and Shioya, 2003) where a vector representing the *pitch spectrum* for each bar is built for each song, and then compared using an ad-hoc measure based on the cosine similarity to compare vectors, and giving more importance to the first bars of the song.

2.2.4 Geometric modeling

Following the same scheme described in the monophonic representation systems (see Sect. 2.1.6), another way to encode the notes of a polyphonic melody is to depict it as a cartesian plot, where time is represented in the x axis, and pitch is mapped in the y axis, producing a sort of *piano-roll* picture (see Fig. 2.31). On that basis, notes may be represented just as points, with a radius that represents duration or any other quality of the note, and finally line segments like in a piano-roll score. The model can be extended to more dimensions to represent other note features.

The natural way of encoding simultaneous notes is the strength of these group of methods.

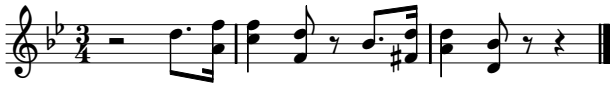
Quantized point-pattern similarity.

There is a group of systems that only use the note onsets of the piano-roll, i.e., the starting point of each segment.

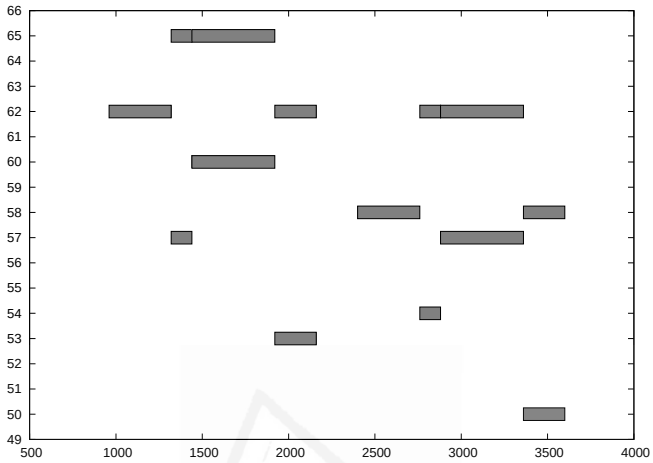
In order to compare raw polyphonic MIDI files, in (Mitzenmacher and Owen, 2001) the documents are first converted in a list of onset times or points for each pitch. Adapting the early approach in (Leppig, 1987) for monophonic music, where similarity is measured as the count of the matched notes versus the non-matched notes, for each MIDI pitch the intersection of the coincident times divided by the union of both sets is computed. Then, the similarity of two documents is obtained by calculating the weighted average of that coincidence measure for all pitches.

Clausen et al. (2000) used inverted file indices with the geometric representation in their *PROMS* system. The onset times of the notes are quantized to a pre-selected resolution so that both the pitch and time dimensions are discrete. Now, onset times are represented relatively to their metrical position within the measure (see Fig. 2.32). The

[9] that are just a vector with a real value for each pitch class in each tonality



(a) Excerpt of Beethoven's Hammerklaviersonate, Scherzo



(b)

Figure 2.31: Line-segment representation on a pitch-against-time Euclidean plane.

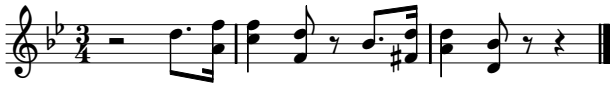
information within a measure constitutes one unit for the query. The retrieval method finds occurrences (total and partial) that have a similar metrical positions as the query; local time and pitch fluctuations cannot be dealt with. The tempo invariance can be obtained by conducting a metrical structure analysis phase and transposition invariance by using a mathematical trick that outperforms the brute-force solution.

Instead of all possible transpositions, the interval from tonic pitch encoding p_{ift} could be used.

In the original algorithm, the approximate search is accomplished by allowing a maximum of k dissimilarities between the query and the database document. To compare whole pieces of music, the original algorithm may be modified to return the normalized number of coincidences for the best alignment. In the sequel, we will refer to this technique as *PROMS similarity*, named after Clausen et al.'s original system.

Multidimensional point sets.

If instead of using different radius for representing other properties of notes (apart from time and pitch) like in (Typke, 2007) (see Sect. 2.1.6), more dimensions are added to the plot, the problem of comparing two pieces is converted into the one of mapping or translating two n-dimensional point sets. The *SIA(M)ESE* system (Wiggins et al., 2002) works on the translation vectors between the points in the piano-rolls. This way, finding a transposition invariant exact occurrence of a query pattern becomes straightforward:



(a) Excerpt of Beethoven's Hammerklaviersonate, Scherzo

{
{[8,74], [11,77], [11,69]},
{[12,77], [12,72], [16,74], [16,65], [20,70], [23,74], [23,66]},
{[24,74], [24,69], [28,70], [28,62]}
}

(b)

Figure 2.32: Representation in PROMS of fragment (from (Clausen et al., 2000))

find a translation vector that translates each point in the pattern to some point within the studied musical work. They look for either the longest time-contiguous subset of pattern notes into the score, or, under the assumption that usually the beginning and ending of a piece is more rememberable than the middle part, exact matches at the beginning and ending of the song (see Fig. 2.33). For large queries, in (Clifford et al., 2002) the *MSM* system is presented, that finds the largest subset of a pattern appearing in a point set at an arbitrary offset in any of the dimensions represented, thus allowing tempo and transposition invariance, and according to that work's authors, outperforming the *SIA(M)ESE* system.

Line segments.

In order to find exact matches of the starting points of the line segments representing the compared songs, Ukkonen et al. (2003a) show how to process the translation vectors in a suitable order to obtain an algorithm of a linear expected and a quadratic worst case time complexity (algorithm *P1*). Their method was also modified for the case of finding all partial occurrences of the pattern. This is doable in $O(nm \log m)$ time, where n and m refer to the length of the musical work and the pattern, respectively (algorithm *P2*). Moreover, they also suggested to replace the points in the piano-rolls by horizontal line segments (thus considering the length of the notes instead of the bare note-on information) and modified their algorithm to solve the problem of *the longest common total length*. For this problem, their *P3* algorithm requires some extra data structures but runs also in $O(nm \log m)$ time (with discrete input). This setting takes into account tempo fluctuations but not pitch fluctuations. One attempt in this direction is presented by Lubiw and Tanur (2004). More recently, Lemström et al. (2008) have developed more efficient versions of the *P2* algorithm using indexing: namely *P2v5* and *P2v6*. These algorithms are framed into the *C-Brahms* project (Lemström et al., 2003), generic name that we will use from here on to refer them.

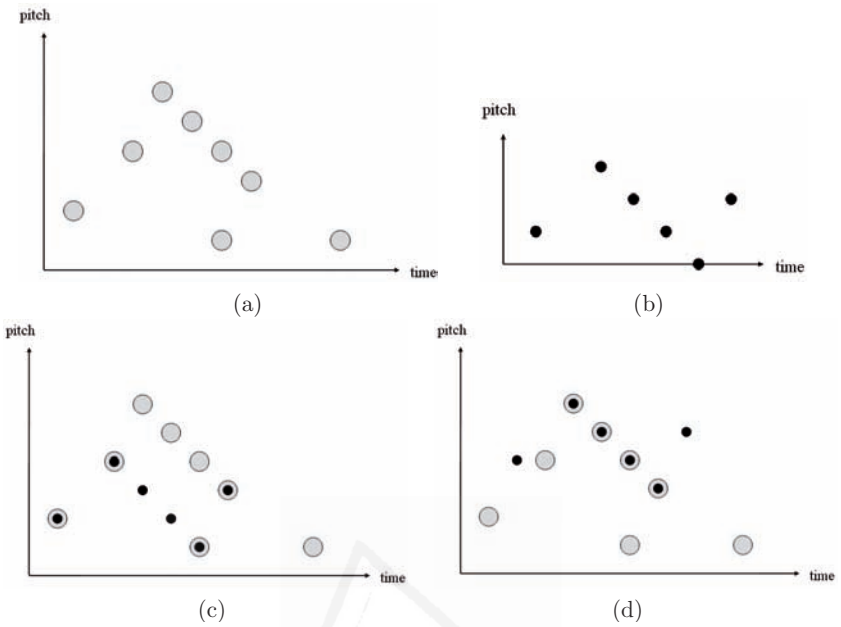


Figure 2.33: (a) Two-dimensional point set representation of a score; (b) two-dimensional point set representation of a pattern; (c) an approximate match of the pattern into the score that exactly matches the start and end of the pattern; (d) an approximate match of the pattern into the score that matches the longest time-contiguous subset of pattern notes into the score. (from (Typke et al., 2005b)).

2.2.5 Network flow distance for chords

By describing the Earth Mover Distance (Typke, 2007) (see Sect. 2.1.6) as a special case of a network flow distance (Ramon and Bruynooghe, 2001), "where a certain total amount of weight has to be moved across the network such that the flow is maximized and the costs are minimized", the problem of finding an optimal transportation between point sets is equivalent to that of finding a maximum flow at minimum cost through such a network.

In order to compare polyphonic music unnamed chords or *sonorities* are identified, and weights and arc costs are adjusted to reflect which notes belong to the same chord so that the distance is able to handle them.

For the interested reader, there is another work that based on the Typke's EMD distance model to compare melodies and allowing note-to-chord comparisons (Garbers et al., 2007).

2.2.6 Similarity by Kolmogorov complexity

There have been isolated attempts to use information theory measures to compare music. Ling and Sharp (2004) used a metrics based on the Kolmogorov complexity (Li and

Vitanyi, 1997). This metrics is based on the assumption that the compression of two similar pieces concatenated should be higher that two non-similar pieces because the more similar the pieces are, the more information they share.

In order to construct the binary string that will be fed into the LZW compressor (Welch, 1984), a skyline reduction from a whole polyphonic piece is performed and absolute pitches and pitch intervals are used to generate that string. In their work, they reported better performance than other models based on language modeling like bigrams and trigrams.

3

Music comparison with tree representation

"One of the most obvious facts about the experience of listening to practically any piece of music is that one perceives not merely an arbitrary sequence of note durations, but some sort of temporal structure in which the notes are grouped into various kinds of units." (Lee, 1985)

"Reductions offer a tempting solution to fuzzy matching because they suppress the differences of surface detail. But do they suppress the most appropriate surface details?" (Selfridge-Field, 1998)

3.1 Previous uses of trees in computer music

These two sentences support the hypothesis our approach is based on. The abstract data type tree seems to be the most suitable for capturing the temporal and hierarchical structure, and it is ready for representing the reduction of a work. Therefore, trees seem to be an adequate data structure to encode and process music in symbolic format for similarity computation. Despite this might look like an obvious assumption, as far as we know, trees had not been used as a method to represent music for comparison until our first proposal in (Rizo and Iñesta, 2002). However, some other uses of trees in the domain of computer music can be found in the literature.

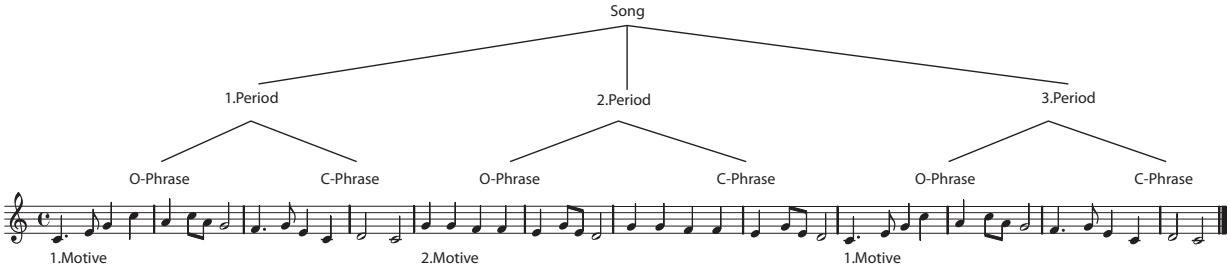


Figure 3.1: Form structure of "Alle Voegel Sind Schon Da." (from (Linster, 1992)).

A straight forward use of trees can be found in the representation of the formal architecture of a whole musical work (see Fig. 3.1). Another tree that instantaneously comes to mind is that of the hierarchy of subdivisions of the figure durations (see

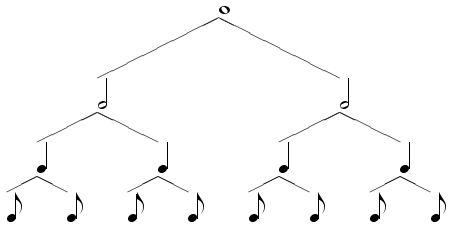


Figure 3.2: Duration hierarchy for note figures. From top to bottom: whole (4 beats), half (2 beats), quarter (1 beat), and eighth (1/2 beat) notes.

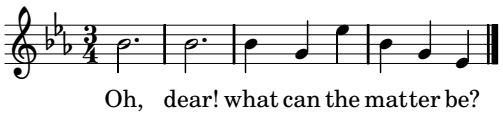
Fig. 3.2). Those two kinds of tree structures have been widely used in the domain of the rhythm perception. As noted in (Linster, 1992), the simplest rhythm patterns have a regular pulse, and multiples of this pulse are stressed in a hierarchical way, leading to the tree in Fig. 3.2. In order to describe formally the perception of more complex rhythms, a number of grammars and parsing strategies have been proposed. In (Lee, 1985) a review of the research of (Longuet-Higgins, 1978; Martin, 1972; Sundberg and Lindblom, 1992) shows different grammars to model the perceived rhythm from notational figures, under the assumption that a listener performs an unconscious grammar parsing (see Fig. 3.3).

$\frac{3}{4} \rightarrow \mathbf{\dot{d}}\cdot \mid \mathbf{-}\cdot \mid \frac{1}{4} + \frac{1}{4} + \frac{1}{4}$

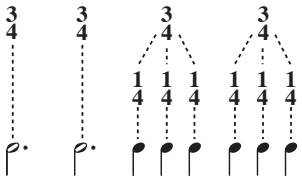
$\frac{1}{4} \rightarrow \mathbf{\text{♩}} \mid \mathbf{\text{𝄽}} \mid \frac{1}{8} + \frac{1}{8}$

$\frac{1}{8} \rightarrow \mathbf{\text{♪}} \mid \mathbf{\text{𝄾}} \mid \cdots$

(a) Grammar for meter 3/4



(b) First four bars of "Oh dear, what can the matter be"



(c) Parse tree of score with 3/4 grammar

Figure 3.3: Longuet-Higgins grammars (from (Lee, 1985) pages 54-55).

As pointed out in the work of Lee (1985), there are many rhythms that can be unambiguously analyzed using those grammars. However, there are also many situations in which the parsing is not so straight forward. See in Fig. 3.5c some possible parses for score in Fig. 3.5a using both the grammar for 3/4 in Fig. 3.3a and the grammar for 4/4 in Fig. 3.5b.

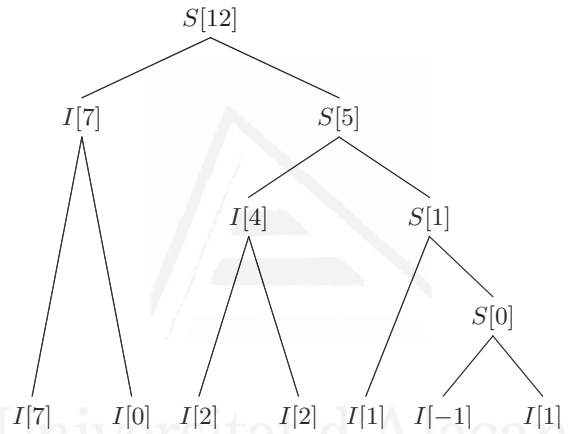
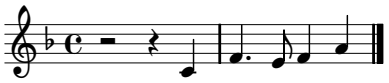
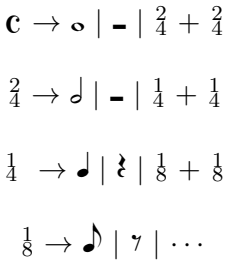


Figure 3.4: Parse tree from (Gilbert and Conklin, 2007).

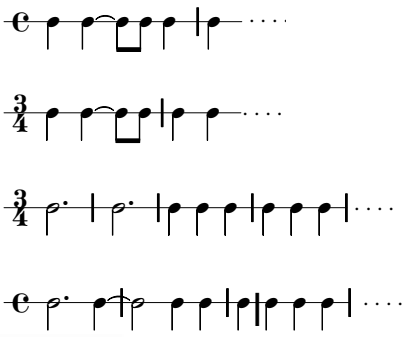
Being a very interesting approach, however, none of those works specifies an implementable computer algorithm or specification. Instead, two grammars that were conceived from a computational point of view were those found in (Bod, 2002) and (Gilbert and Conklin, 2007). The former tries to learn a grammar with the so-called Data-Oriented Parsing (DOP)using the *ESSEN* corpus (Schaffrath, 1995) (see Fig. 3.6). In that work, the task is to learn segmentations from a corpus of melodies with manually separated phrases. In (Gilbert and Conklin, 2007), monodies are parsed into tree structures using a probabilistic context-free grammar that is used to perform melodic reductions (see Fig. 3.4). This grammar is based in a melodic analysis like the one presented above in Section 2.1.3 (page 26) and was evaluated through a segmentation task.



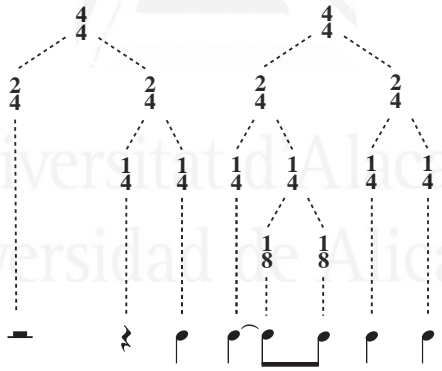
(a) First two bars of"Auld Lang Syne"



(b) Grammar for meter 4/4

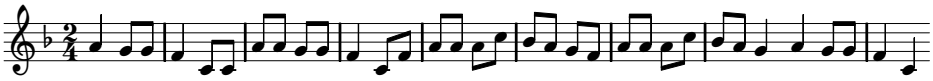


(c) Some possible parses of (a) using the grammar in (b) and that detailed in Fig. 3.3a.



(d) Some possible parses

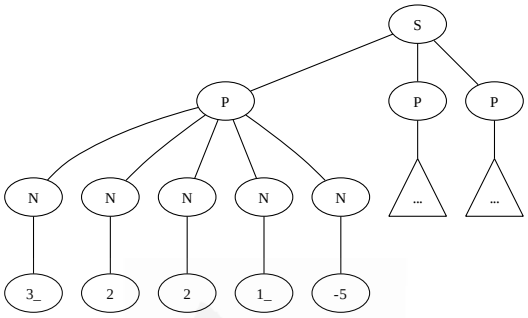
Figure 3.5: Longuet-Higgins grammar for meter 4/4 (from (Lee, 1985) pages 54-55) and some possible parses.



(a) Essen Folksong K0029, "Schalf Kindlein feste".

(3_221_-5)(-533221_-5)(13335432)(13335432_)(3_221_-5_)

(b) Bracket representation. The parenthesis delimit phrases.



(c) The labeled bracket representation in (b) for (first five notes of) the K0029 rendered as a tree.

Figure 3.6: Example of data used to learn grammar in (Bod, 2002).

As a way to conceptually represent music for a composition task, trees have been used in (Smaill et al., 1993) and (Balaban, 1996). In the latter, music is represented with a sort of object oriented structure where nodes represent musical objects and children inherit possibly transformed properties from their parents (Fig. 3.7).

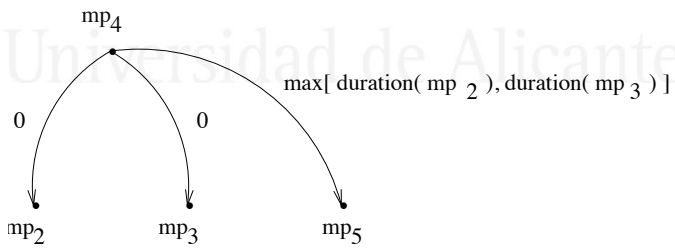


Figure 3.7: This tree describes hierarchically a piece, mp_4, that consists of simultaneous occurrences of mp_2 and mp_3, followed sequentially, by a piece mp_5 (from (Balaban, 1996)).

Some tree structures have been used to describe the hierarchical nature of pitch encodings (Höthker et al., 2001) (Fig. 3.8), however they describe meta-information and are not used to represent melodies.

In automatic composition, the *Wind in the Willows* system (Högberg, 2005) uses tree transducers to generate music. From a tree representation of a melody (Fig. 3.9a), the system uses tree transducers for structure and chord progressions (Fig. 3.9b) in order to create new music.

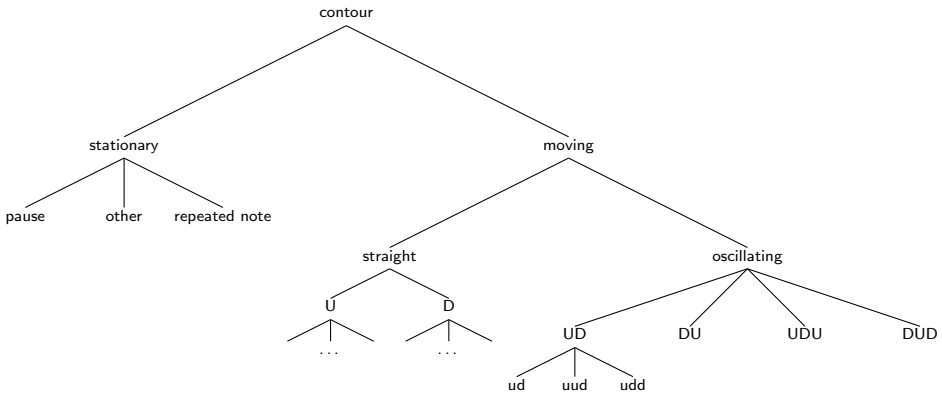
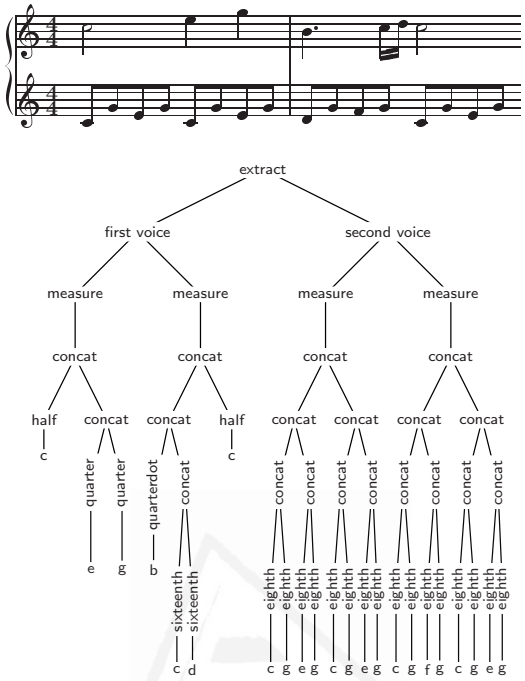


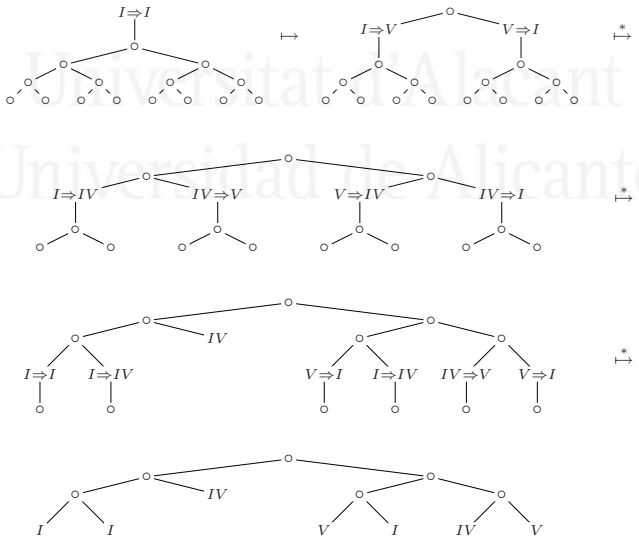
Figure 3.8: The tree-structure representation C2 in (Höthker et al., 2001).

Schenkerian analysis. During the late 19th and the early 20th centuries Heinrich Schenker (1868-1935) developed a method for analyzing tonal compositions based on the hierarchical structuration of music (see (Beach, 1977) for a compendium of his publications). He posed that a musical work can be eventually understood as a series of elaborations that leads from a basic harmonic and melodic structure to the actual sequence of notes that compose a melody. In order to develop the root idea into the actual notes, he used a hierarchical structure, i.e., a tree. However, as his theories were not developed from a formal point of view, there is not a "computable" methodology that drives an analysis from a piece. Instead, several non-deterministic principles have to be taken to derive the several levels of reduction of the schenkerian analysis. There have been a number of attempts to create an automatized system to perform a schenkerian analysis: Smoliar (1979), Kassler (1988), Kirlin and Utgoff (2008), and the excellent works of Marsden (2001, 2004, 2005, 2007). In his article "Generative structural representation of tonal music" (Marsden, 2005), he reviews the previous works in-depth. Nonetheless, as the theory is not deterministic, there are many situations where a decision has to be taken to disambiguate two possible valid analyses. None of those computer approaches to the schenkerian analysis has as its main objective to compare music, so as far as we know, there is no groundtruth that assesses that the computer made analyses are valid for the comparison goal.

One important concept for this thesis in those theories is the notion of *elaboration*. It can be seen as the contrary of the concept of *division* or *diminutions* (Forte and Gilbert, 1982), that refers to the process by which an interval formed by notes of longer value is expressed in notes of smaller value, comprising *passing notes* (**P**), the *neighbor notes* (**N**), the *consonant skips* (**CS**), the *repetitions* (**R**), the *suspensions* (**S**), and the *arpeggiations* (**Arp**). See Fig. 3.10 for an example of each.



(a) Above, the first two bars of the sonata in C-major KV545 by Mozart, and below, the same piece represented by a tree



(b) A transducer assigns a chord progression common to pop music.

Figure 3.9: Tree manipulation in *Wind in the Willows* (from (Högberg, 2005)).

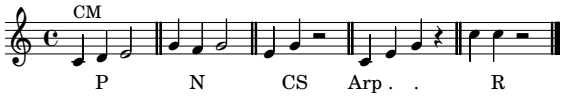
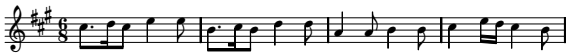
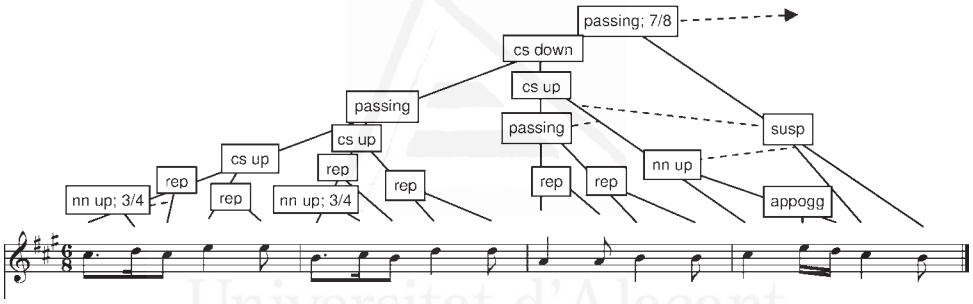


Figure 3.10: Basic elaborations

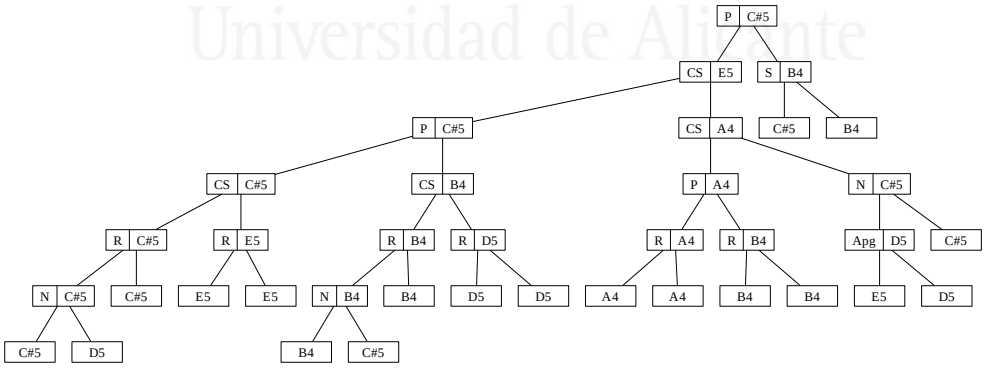


(a) Beginning of Mozart's piano sonata in A major, K.331.

(b) Schenker's analysis (from (Schenker, 1935), Fig. 157).



(c) Simple elaboration-tree representation in the original Marsden (2005) notation.



(d) Our standardized tree representation. The left box of each node contains the elaboration used, the right box has the note name.

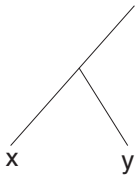
Figure 3.11: Marsden Schenkerian analysis of a musical excerpt. The tree notation in (d) introduced in the present work to homogenize reduction notation along the manuscript.

According to Marsden (2005), the analysis of the first two bars of the Mozart's piano sonata in A major, K.331 (Fig. 3.11a) leads to the tree in Fig. 3.11c. In that figure, our

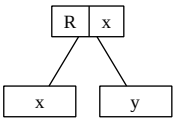
own notation of the reduction tree has been used (Fig. 3.11d) to homogenize this kind of information along this manuscript.

Being these theories relevant to our hypothesis of using reduced melodies, we have not been able to use them directly because, besides the non straight forward computability of the theories, there are relations that span across the strict parent-leaves relation of trees, what suggests that in Schenker theories are closer to graphs than to trees as noted in (Marsden, 2005).

Generative theory of tonal music. A somewhat similar approach to that of Schenker to explain hierarchically the inner relations in a musical piece is the Generative Theory of Tonal Music (GTTM) (Lerdahl and Jackendoff, 1983). Inspired in the linguistics grammars, the authors of GTTM wrote: "we take the goal of a theory of music to be a formal description of the musical intuitions of a listener who is experienced in a musical idiom". Although no formal grammar is defined in that work, an extensive series of rules that try to explain what the listener perceives is provided. Those rules cover from the base *tactum* and beat perception to how the whole piece is structured, passing through the individual dependencies between notes. Related to the concept of reduction and elaboration, Lerdahl and Jackendoff define the concept of *subordination* to denote which of two notes is more important in terms of structure, either the left note in the so-called *right elaboration* (see Fig. 3.12) or the right note in the *left elaboration* (see Fig. 3.13). In Fig. 3.14 we have plotted using our proposed notation an example of GTTM reduction of a musical excerpt.



(a) Original GTTM notation.



(b) Our own tree notation. The left box of each node contains the branching used (R stands for *right*), the right box has the note name.

Figure 3.12: GTTM: y is a *right-branching* elaboration of x.

(a) Original GTTM notation.

(b) Our own tree notation. The left box of each node contains the branching used (L stands for *left*), the right box has the note name.

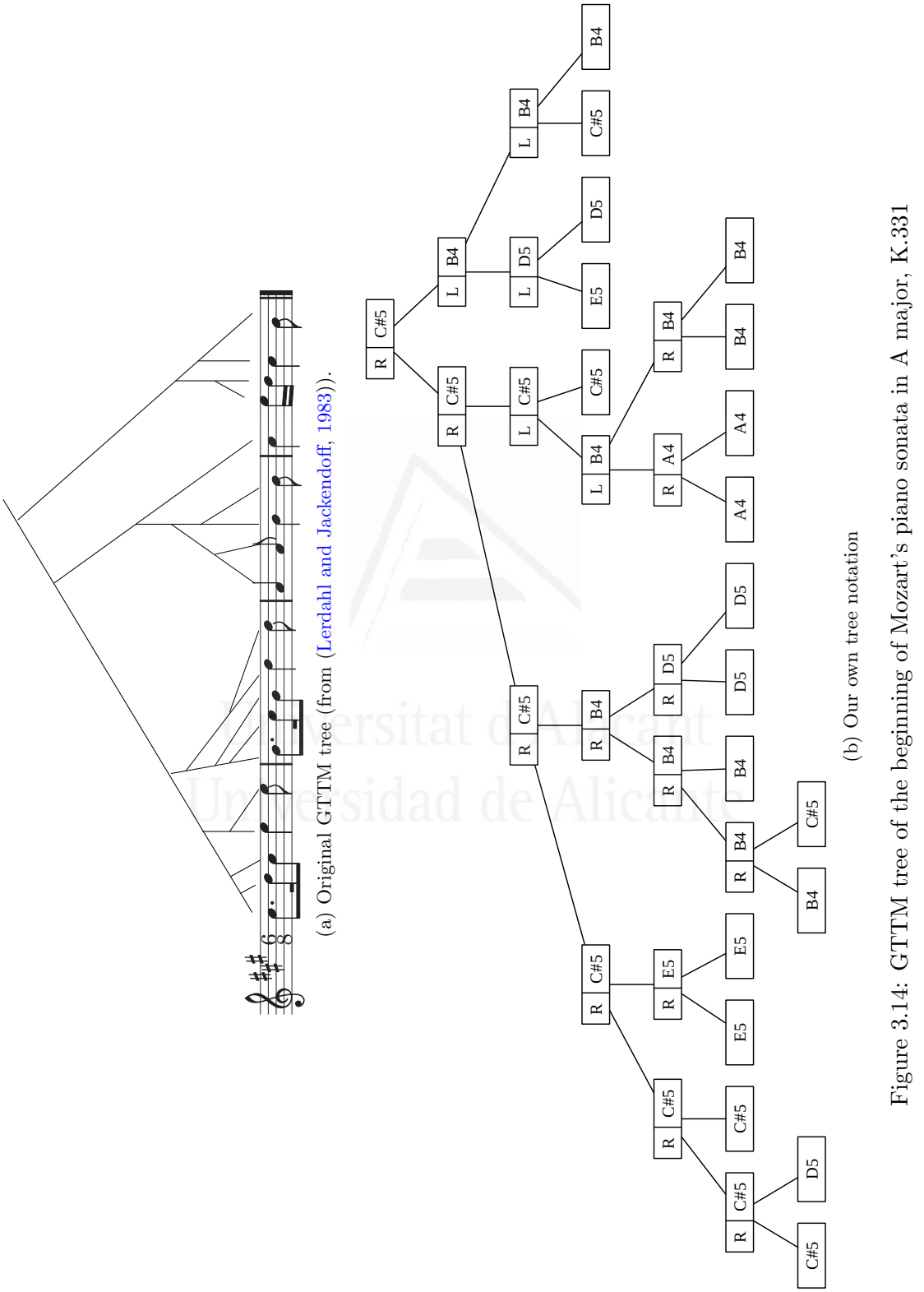
Figure 3.13: GTTM: x is a *left-branching* elaboration of y.

As it happens with Schenker theories, the GTTM is not directly computable. Most of the rules are ambiguous, in fact they are grouped in *well-formedness rules* and *preference rules* which lead to conflicts in many cases. Some attempts to implement the GTTM can be found in the literature (Hamanaka et al., 2007; Jones et al., 1993), being the work of Hamanaka (currently named *ATTA*) the most active and promising so far. However, the proposed systems, that have a multitude of parameters to be tuned, are still far to be able to work with whole pieces [1].

OpenMusic rhythm trees. In the context of assisted musical composition in the symbolic domain, a tool stands out from the rest: OpenMusic (Assayag et al., 1999). This tool uses trees as a natural way of representing the hierarchical nature of duration subdivision of musical figures and groupings like tuplets. It is implemented in LISP, where everything can be represented either with a list or a tree (list of lists). In order to be able to make the computer processing of complex rhythmic formulae easier, it uses the so-called *rhythm trees* (Agon et al., 2002), that represent the totality of the hierarchical relationship of traditional rhythmic notation: bars divided into groups divided into beats divided into notes. *Rhythm trees* only contain the rhythmic structure, they do not have any pitch information. This data is contained in synchronized lists that have the pitches for the leaves of the trees. The OpenMusic manual describes those trees as:

> Trees represented in list form $(D, \mathbf{S})$ where D is a number (integer or fractional) expressing a time extent, $\mathbf{S}$ is a list of items defining a set of proportions to take place in D, each item of $\mathbf{S}$ being either :
>
> - a number, possibly preceded by a dot to indicate that it is tied to the previous note
> - a sublist, having itself the same structure $(D, \mathbf{S})$.
>
> The actual duration indicated by D depends on the group of which its list is a part. At the highest level, the basic unit is assumed to be the

[1] Personal communication



(a) Original GTTM tree (from (Lerdahl and Jackendoff, 1983)).

(b) Our own tree notation

Figure 3.14: GTTM tree of the beginning of Mozart's piano sonata in A major, K.331

whole note. So, `(1 (1 1 1 1))` (Fig. 3.15) is a structure whose duration is a whole-note `(1)` which is subdivided into 4 equal parts, which would be 4 quarter notes in this case. Similarly, `(2 ( (1 (1 1 1 1)) (1 (1 1 1 1))))` (Fig. 3.16) is a structure with a duration of two whole-notes, containing two equal substructures, each lasting a whole-note. These substructures in turn contain 4 equal values, i.e., quarter notes. So it can be interpreted as a voice containing 2 measures in $\frac{4}{4}$.

By convention, when the value of D is on the measure level it will be expressed in a whole note unit. Following this rule, a three quarter notes $\frac{3}{4}$ bar is represented by `(3/4 (1 1 1))`.

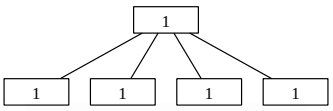


Figure 3.15: OpenMusic rhythm tree representation of list of lists `(1 (1 1 1 1))`

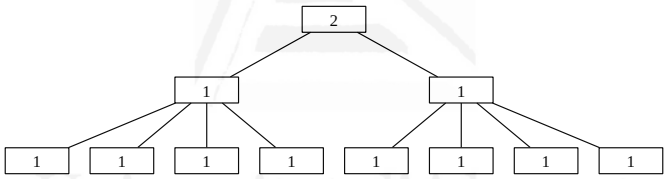


Figure 3.16: OpenMusic rhythm tree representation of list of lists `(2 ( (1 (1 1 1 1)) (1 (1 1 1 1)))`

An example of this representation for a particular melody can be found in Fig. 3.17a. In order to facilitate the interpretation of the tree, we have added some labels containing between parenthesis either a meter or the actual pitch that has been obtained from the list of pitches that accompanies the rhythm tree in OpenMusic to represent a melody.

Fig. 3.18a shows the already used example on the firsts bars of Mozart's K331 with the representation of OpenMusic.

Indexing. Finally, trees have been also used not as a means to represent music, but as an intermediate data structure for other goals like document structuration in searching (Blackburn, 2000; Drewes and Högberg, 2007; Skalak et al., 2008).

Our approach: metrical trees. Many of the just reviewed tree applications are very interesting and useful for the particular task they were designed for. However, all of them have some drawbacks for the similarity computation task: basically they are not ready for performing a straight forward reading of an input MIDI file, represent it, perform a

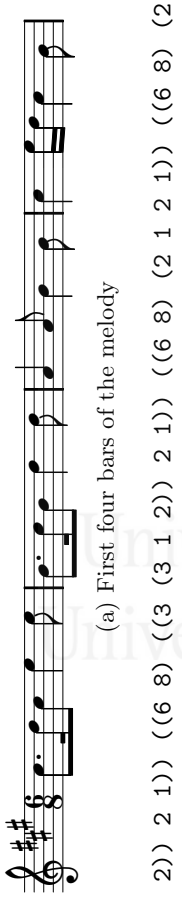
(a) First 5 bars of the melody Der Elsberger Bie vrie ischt auf Molankizle, Europa, Jugoslavija (Schaffrath, 1995)

```
(19/2 (((4 2) (-7 1)) ((4 2) (2 2 2 2)) ((3 2) (1 1 2 2)) ((4 2) (2 1 1 2 2)) ((4 2) (-3 1 1 3))))
```

(b) OM Rhythm Tree List for score above in (a)

(c) Our notation of OpenMusic rhythm tree merged with the pitches list obtained from the score.

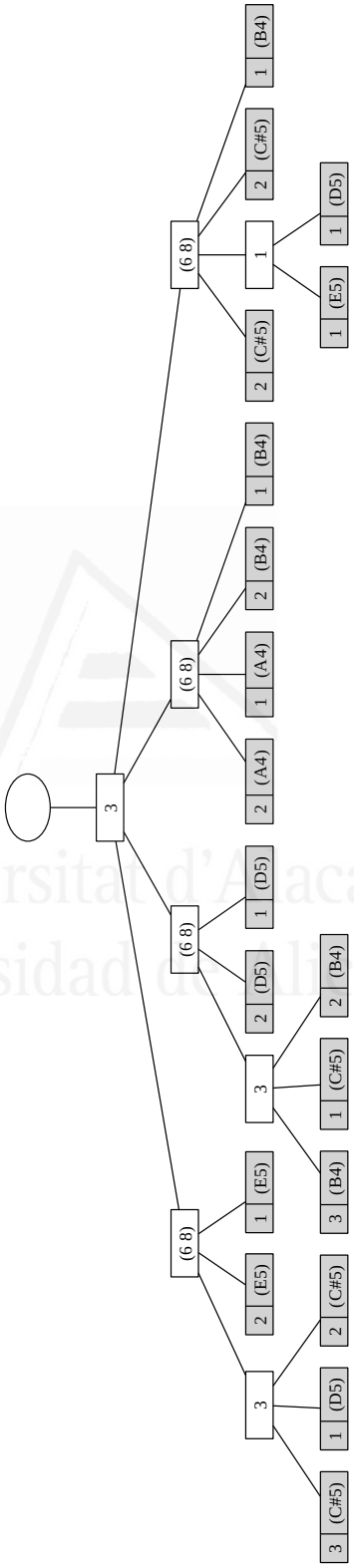
Figure 3.17: OpenMusic rhythm tree for score above in (a).



(a) First four bars of the melody

```
(3 (((6 8) ((3 (3 1 2)) 2 1)) ((6 8) ((3 (3 1 2)) 2 1)) ((6 8) (2 1 2 1)) ((6 8) (2 (1 (1 1)) 2 1))))
```

(b) OpenMusic Lists



(c) OpenMusic rhythm tree merged with the pitches list obtained from the score.

Figure 3.18: OpenMusic rhythm tree merged with pitches from the beginning of Mozart's piano sonata in A major, K.331. Note that the root label '3' is a simplification of 24/8 performed by OpenMusic.

reduction, and finally use the final tree to compare with other musical works encoded in the same kind of trees. Some approaches, like the OpenMusic representation seem to be adequate for the comparison task, but were discovered by the author of this dissertation after having proposed our first proposal of tree representation (Rizo and Iñesta, 2002). However, being somewhat similar, our representation encodes the kind of rhythm pulse proposed by several rhythm analysis as explored in (Lee, 1985).

Our objective is to use the tree data structure to better represent musical symbolic information in order to compare different pieces to judge their similarity. Our goal is not to depict faithfully all musical details, but to keep only those elements that are important for the comparison computation. Thus, our method is designed to be unambiguous and as much simple as possible.

We pose the importance of metrical structure against the linearity of other representations. The proposed trees, named *metrical trees*, avoid the necessity of encoding explicitly onsets and duration as it happens with string representations, leading to less degrees of freedom and less different possible encodings to be explored because only pitches need to be encoded. Besides, as stated in (Rolland, 1999), "melodic comparison requires taking into account not only the individual notes but also the structural information based on music theory and music recognition".

Next in this chapter, the tree construction method will be introduced. First some definitions and notations are stated. Then, the tree algorithms for the metrical tree construction are detailed. After that, some approaches to melodic reduction are shown. The comparison of melodies by the use of tree edit distances is treated in the next section. The method is finally extended for the representation of polyphonic music. Finally, the experiments section shows the performance of the proposed approach compared to existing music comparison paradigms.

3.2 Tree representation of monophonic metered music

A melody has two main dimensions: rhythm and pitch. In linear representations, those dimensions are coded by explicit symbols, but ordered trees are able to implicitly represent time in their structure, making use of the fact that the whole piece is divided hierarchically into bars, and note durations are multiples of basic time units, mainly in a binary (sometimes ternary) subdivision. The left to right ordering of tree nodes depicts the time flow. This way, trees are less sensitive to the codes used to represent melodies, since only pitch codes are needed to be established and thus there are less degrees of freedom for coding. Furthermore, this hierarchical organization allows to append additional information such as harmonic descriptors to groups of notes in a natural way. We name this kind of representation *metrical trees*.

In the western music from the common practice period, the lapse of time is usually divided first in bars regularly, then in beats and finally in subdivisions of those beats. The duration of the bars depends on the meter and tempo. The duration of the actual figures is designed according to a logarithmic scale: a *whole* note lasts twice than a *half*

note, that is two times longer than a *quarter* note, etc. (recall figure durations hierarchy in Fig. 3.2 at page 60).

In our tree model, each melody bar is represented by a tree, τ. Each note or rest is found in a leaf node. The left to right ordering of the leaves preserves the time order of the notes in the melody. The level of a leaf in the tree determines the duration of the note it represents, as displayed in Fig. 3.2: the root (level 1) represents the duration of the whole bar (e.g., a *whole* note for a $\frac{4}{4}$ bar), the two nodes in level 2, the duration of a *half*. In general, nodes in level l represent duration of a $1/2^{l-1}$ of a bar.

If only a whole note is found in the measure, the tree will consist of just the root, but if there were two half notes, this node would split into two children nodes. Thus, recursively, each node of the tree will split into two until representing the notes actually found in a measure.

During the tree construction, internal nodes are created when needed to reach the appropriate leaf level, filling the tree from left to right. Initially, only the leaf nodes contain a label value. The tree labels represent the corresponding pitch information in any representation of those introduced in section 2.1.1 (MIDI note, interval, etc.), and even harmonic data can be associated to the note. Once the tree is built, a bottom-up propagation of these labels can be performed to fully label all the nodes. The rules for this propagation will be introduced below.

An example of this scheme is presented in Fig. 3.19. Note that in this case the meter is a $\frac{2}{4}$ and the root duration is equal to a half. In the tree, the left child of the root has been split into two subtrees to reach level 3 that corresponds to the first note duration (as eighth note lasts $1/2^{3-1}$ of the bar, pitch B coded as **11** according to the pitch class representation). In order to represent the duration of the rest (coded as an empty label) and note G, coded as **7** (both last 1/8 of the bar), a new subtree is needed for the right child in level 3, providing two new leaves for representing them. The quarter note C (**0**) onsets at the second beat of the bar, so it is represented in level 2 according to its duration.

Fig. 3.19 depicts how the time order of the notes in the score is preserved by traversing the tree from left to right. Note also how onset times and durations are implicitly represented in the tree, in opposition to the explicit encoding of time when using strings. This representation is invariant under time scalings, as for instance, different meter representations of the same melody (e.g., 2/2, 4/4, or 8/8).

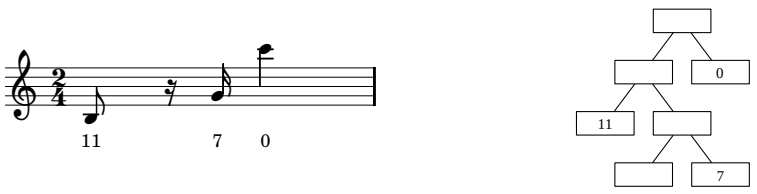
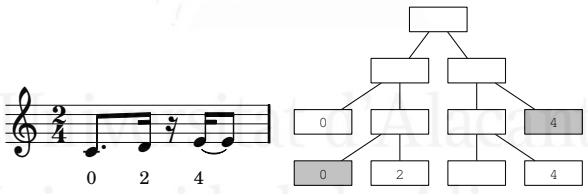


Figure 3.19: Simple example of tree construction using *pitch class* representation

.

Processing non binary durations.

In some occasions the situation can be more complicated. There are note durations that do not match a binary division of the whole bar or the parent node. This happens, for example, for dotted notes (duration is extended in an additional 50%) or tied notes whose durations are summed (see Fig. 3.20a). In such a case, a note cannot be represented just by one leaf in the proposed scheme. However, it is well-known (Mongeau and Sankoff, 1990) that our auditory system perceives a note of a given duration the same way as two notes of the same pitch, played one after the other, whose durations sum up to that of the single one. Therefore, when a note exceeds the considered duration, in terms of binary divisions of time, it is subdivided into notes of binary durations, and the resulting notes are coded in their proper tree levels. Nodes containing the continuation of another one can be encoded by appending any special symbol or indication to denote the continuation. Fig. 3.20b shows such an example and how it is handled by the scheme. This kind of duration splitting has been also used in other works (Hanna et al., 2008; Pardo and Sanghi, 2005).



(a) (b)

Figure 3.20: Tree representation of notes exceeding their notation duration: dotted and tied notes. Both **0** leaves correspond to the same dotted quarter note. The two **4** leaves represent the two tied notes. Shaded nodes denote continuation notes.

Other frequently used non binary divisions are ternary rhythms. In that case, the length of one bar is usually 3 beats and it is split into 3 quarter notes, etc. This is not a problem, since neither the tree construction method nor the metrics used to compare trees need to be binary; there can be any arbitrary number of children for a node. So, in ternary meters or ternary divisions, the number of children for a node is three. Now, durations at each level l is $1/3^{l-1}$ of the whole bar. This can be generalized to other more complicated cases that can appear in musical notations, like tuplets or compound meters. Fig. 3.21 gives an example of compound meter based on ternary divisions and the corresponding tree.

Figure 3.21: The compound meter 9/8 is based on ternary divisions. The tree construction method can also represent this melody.

There are other subtle situations that may appear in a score, like for example grace notes[2], that are not included in the cases described above. Nevertheless, in digital scores (e.g., MIDI files) these special notes are represented by short notes that are subsequently coded in the level corresponding to their written duration by our scheme.

Representation of complete melodies.

The method described above is able to represent a single bar as a tree, τ. A *bar* (or a measure) is a basic unit of rhythm structure in music. A melody is composed of a sequence of B bars. Let us now describe how to combine the set of trees $\{\tau_i\}_{i=1}^{B}$ representing the bars.

To build a tree, T, for a complete melody, the computed bar trees are joined in a particular order. For instance, the sub-trees can be grouped two by two, using always adjacent pairs. This operation is repeated hierarchically, bottom-up, with the new nodes until a single root is obtained. Let us denote the depth (or height) of a melody tree T by $h(T)$. With this grouping method, the trees grow in depth quickly:

$$h(T) = \log_2 B + 1 + \max_i h(\tau_i),$$

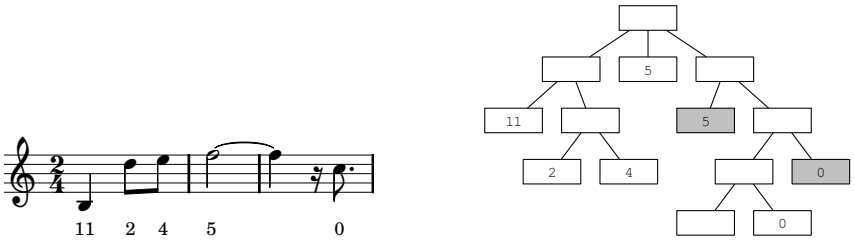
making the tree edit distance computation very time consuming, since it heavily depends on this tree feature as will be discussed below in Section 3.5.

The use of the hierarchy for capturing the musical architecture or structure of a whole work could be an option, but the automatic recognition of structure (Chai, 2005) is outside the scope of this dissertation. Then, our best choice is to build a tree with a root for the whole melody, whose children are the bar sub-trees. This way, the depth of a tree corresponding to a whole melody becomes

$$h(T) = 1 + \max_i h(\tau_i).$$

The smaller depth of the tree of the latter approach makes it the choice taken. Besides, this scheme is closer to the notion of regular time subdivision levels as studied in the common music theory. Fig. 3.22 displays an example of a simple melody, composed of three bars, and how it is represented by a tree composed of three sub-trees.

[2] A grace note is a very short note or a series or notes to achieve musical effects that occupies no time in the duration notation in a score. They are also known as *acciaccatura*.



(a) (b)

Figure 3.22: Melody and the corresponding tree. The root of the tree connects all the bar sub-trees.

In order to briefly compare our construction method with the other tree representation methods exposed in the beginning of this chapter, we have collected in figure 3.23 the already used fragment of Mozart's K331 piano sonata represented with the four previously depicted tree music representations using our standardized notation, along with our *metrical trees*. Our metrical trees do not focus on the analysis process through the bottom-up grouping of notes as happens with the Schenkerian analysis (see Fig. 3.23c) and the GTTM (see Fig. 3.23d), however, as we will describe below in Section 3.4, it is ready to support that kind of process. Instead, the figure shows how our approach focus more in a metrical based hierarchical subdivision of time, similar to that proposed by Lee (Lee, 1985) reviewed at the beginning of this chapter. The only approach able to represent rhythm is that of OpenMusic (see Fig. 3.23e), that it has a node representing each bar and contains the duration of the notes explicitly represented in the leaves. All those tree representations contain the actual notes in leaves, being our representation the only one that needs to split notes because the duration is implicit in the depth of nodes. Finally, all approaches represent time flow through the left to right ordering of nodes.

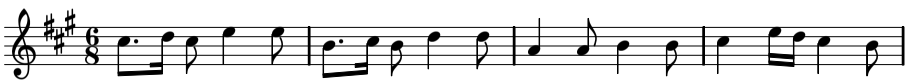
3.2.1 Examples of tree construction for different meters

In order to illustrate the metric tree construction from monophonic melodies, an example for each representative meter is shown in the appendix D.1 (page 173). The incipits in these examples have been obtained from the *kern scores* [3] library of virtual musical scores.

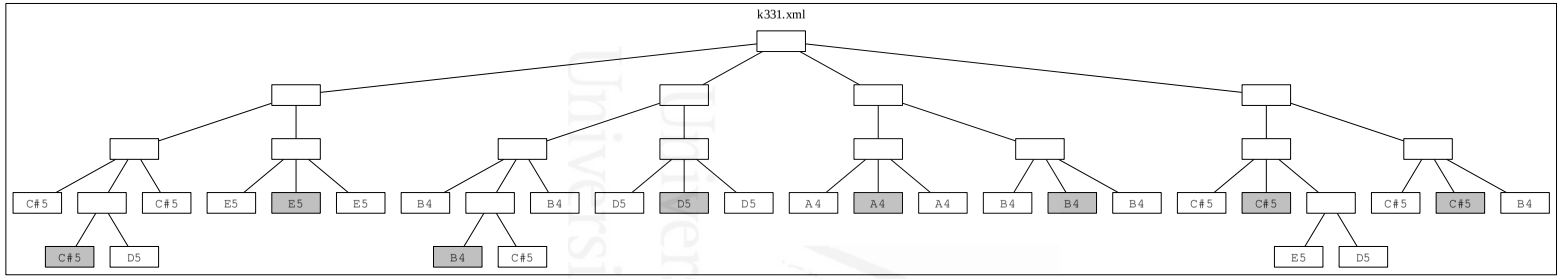
3.3 Metrical tree construction algorithms

Now that our tree representation of monophonic music has been depicted, in this section the construction of the tree in a more formal way will be detailed. First some required definitions and notations will be introduced, and finally the algorithms for constructing trees and some examples will be provided.

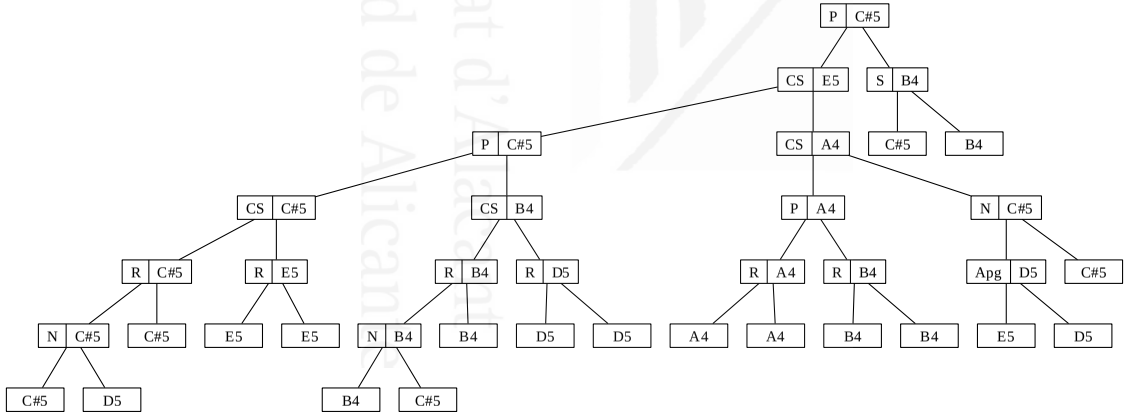
[3] http://kern.ccarh.org



(a) Beginning of Mozart's K331 Piano Sonata

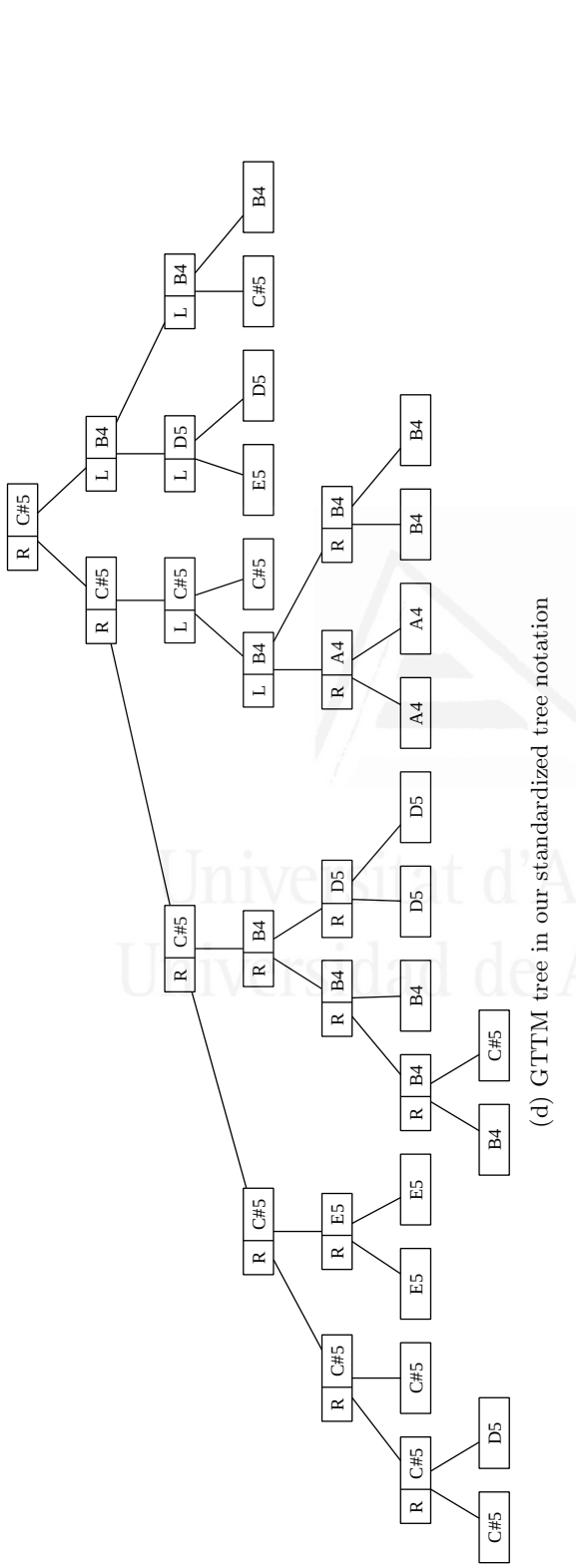


(b) Metrical tree

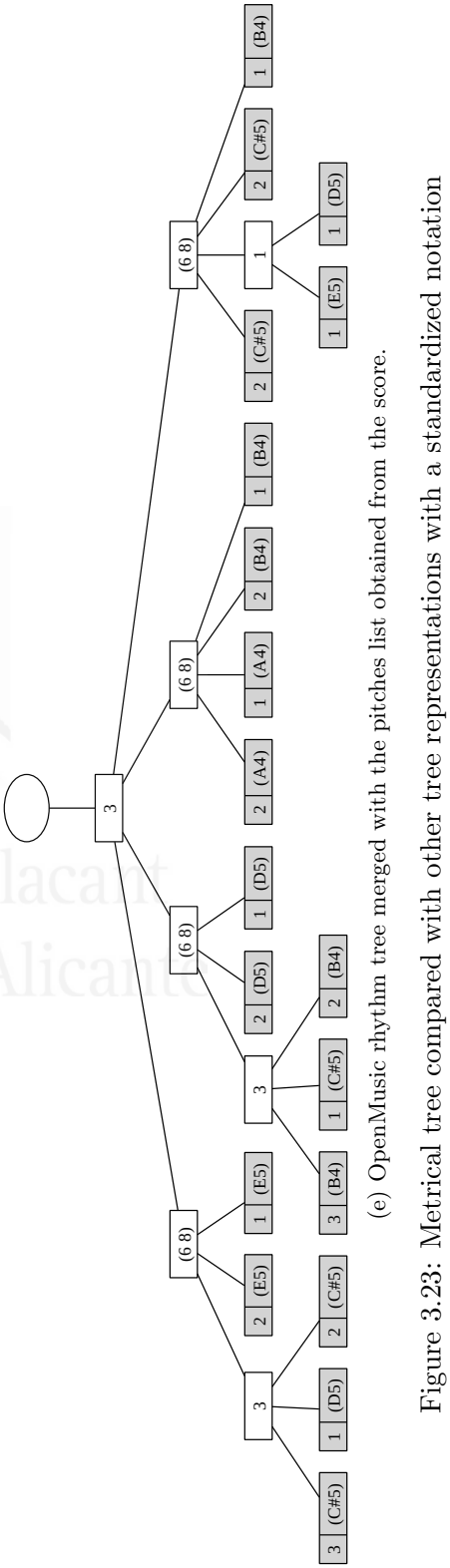


(c) Marsden Schenkerian analysis in our standardized tree notation. The left box of each node contains the elaboration used, the right box has the note name.

Figure 3.23: Metrical tree compared with other tree representations with a standardized notation (continued on next page)



(d) GTTM tree in our standardized tree notation



(e) OpenMusic rhythm tree merged with the pitches list obtained from the score.

Figure 3.23: Metrical tree compared with other tree representations with a standardized notation

3.3.1 Definition and notations

In this section, the terms and notations that will be used along this and the following chapters will be defined. These notations are based on the work of Valiente (2001).

Definition 3.3.1 *A labelled* graph $G = (V, E, L)$ *consists of a finite non-empty set* V *of vertices, a finite set* $E \subseteq V \times V$ *of arcs, and a set* L *of labels for nodes.*

Definition 3.3.2 *Let* $G = (V, E, L)$ *be a labelled graph. Each node contains a label, possibly empty. The labeling function will be defined by* $\text{label}: V \to L$*. The empty label will be denoted as* ϵ*.*

Definition 3.3.3 *A connected, directed, and acyclic graph* $T = (V, E, L)$ *is said to be a* labelled rooted tree *if there is a distinguished node* $r \in V$*, called the* root *of the tree and denoted by* $\text{root}: T \to V$ *such that for all nodes* $v \in V$*, there is only a path from root* r *to node* v*. Thus,* $\text{root}(T) = r$*.*

Definition 3.3.4 *Let us define also the function* $\text{rlabel}: G \to L$ *that returns the label of the root of the tree.*

All the trees used in this manuscript are labelled rooted trees, so both terms will be used interchangeably.

Definition 3.3.5 *Let* $T = (V, E, L)$ *be a rooted tree, the* level *or* depth *of a node* $v \in V$*, denoted by* $\text{depth}: V \to \mathbb{N} \cup \{0\}$*, is the length of the unique path from the root node* $\text{root}(T)$ *to node* v*.*

Definition 3.3.6 *The* height *of tree* $T = (V, E, L)$ *denoted by* $\text{h}: T \to \mathbb{N} \cup \{0\}$ *is defined as* $\text{h}(T) = \max_{v \in V}\{\text{depth}(v)\}$*.*

Definition 3.3.7 *Let* $T = (V, E, L)$*, and two nodes* $v, w \in V$*,* v *is said to be the parent of* w*, if* $(v, w) \in E$ *and* $\text{depth}(w) = \text{depth}(v) + 1$*. The parent of a node will be obtained by the function* $\text{par}: V \to V$*. The node* w *is said to be a child of* v*. Let us define the function* $\text{children}: V \to 2^V$ *as the set of all children of node* v*.* $\text{children}(v) = \{w | \text{par}(w) = v\}$*.*

Definition 3.3.8 *Let* $s, t \in V - \{\text{root}(T)\}$*, they are said to be* sibling *if* $\text{par}(s) = \text{par}(t)$*.*

Definition 3.3.9 *Let* $T = (V, E, L)$*, the arity, rank, or out-degree of a node* $v \in V$*, denoted by* $\text{rank}: V \to \mathbb{N} \cup \{0\}$*, is the number of children of a node.* $\text{rank}(v) = |\text{children}(v)|$*. When applied to a tree* T_m*,* $\text{rank}(T_m) = \max_{v \in V}\{\text{rank}(v)\}$*. When there is no possible confusion, the form* R_m *will be used as abbreviation for* $\text{rank}(T_m)$*.*

Definition 3.3.10 *Let* $T = (V, E, L)$ *be a tree, a node* $v \in V$ *is said to be a* leaf *if* $\text{rank}(v) = 0$*. A boolean function* $\text{leaf} : V \to \textbf{Bool}$ *is defined to denote it. Similarly,* $\text{leaves}(T)$ *is the set of nodes of that tree that have no children:* $\{v \in V | \text{leaf}(v) = \text{true}\}$*.*

Definition 3.3.11 *An* ordered tree $T = (V, E, L)$ *is a tree where the relative order of its children is fixed for each node. It allows us to define the enumeration function* child: $\mathbb{N} \times V \to V$, *such that* $\text{child}_i(v)$ *is just before* $\text{child}_j(v)$ *iff* $i = j - 1, 1 \leq i < j \leq \text{rank}(v)$. *The same function can be formulated using a tree as parameter:* child: $\mathbb{N} \times T \to V$, *such that for* $T_A \in T$, $\text{child}_i(T_A) = \text{child}_i(\text{root}(T_A))$.

Definition 3.3.12 *The postorder numbering of a tree consists of giving the visit order of each node of the tree following a postorder traversal of the tree. To uniquely identify nodes in a tree* T, *lets define* $T[i] \in \mathbb{N}$ *as the* i*th node in a postorder numbering, beginning from 1.*

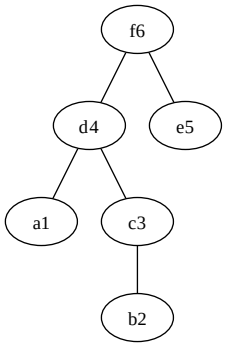
Definition 3.3.13 $\text{desc}(T[i])$ *is the set of all descendants of* $T[i]$ *including* $T[i]$ *itself. Note that this function not only returns the children of a node, but the children of the children and so on.*

Definition 3.3.14 *A* subtree $T' = (V', E', L)$ *of* $T = (V, E, L)$ *is a tree such that* $V' \subseteq V$, *and* $E' \subseteq E$.

Definition 3.3.15 *A* forest *is a disjoint union of trees, and an ordered forest has the property that its components follow an order being this way a sequence of trees that will be denoted as* $\mathcal{T}^+$. *The operation* $T[\text{child}_i(T)..\text{child}_j(T)]$ *is the set composed by the children of* T *from positions* i *to* j, *both included, and it forms a subforest. For abbreviating the notation,* $T[i..j]$ *will be used in the sequel to denote this operation.*

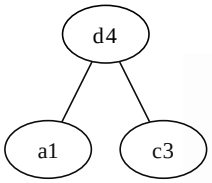
Examples of the definitions In order to illustrate the previous definitions, some examples are provided that have been obtained from the tree in Fig. 3.24a.

- Tree in Fig. 3.24a
  $T = (\{v_1, v_2, v_3, v_4, v_5, v_6\}, \{(v_6, v_4), (v_6, v_5), (v_4, v_1), (v_4, v_3), (v_3, v_2)\}, \{f6, d4, c5, a1, c3, b2\})$
- Subtree in Fig. 3.24b
  $T'_1 = (\{v_3\}, \{\}, \{c3\})$
- Subtree in Fig. 3.24c
  $T'_2 = (\{v_4, v_1, v_3\}, \{(v_4, v_1), (v_4, v_3)\}, \{d4, a1, c3\})$
- Ordered forest in Fig. 3.24d
  $F = ((\{v_2\}, \{\}, \{b2\}),$
  $(\{v_4, v_1, v_3\}, \{(v_4, v_1), (v_4, v_3)\}, \{d4, a1, c3\}), (\{v_5\}, \{\}, \{e5\})$
- Ordered subforest in Fig. 3.24e
  $T[1..3] = \{(\{v_1\}, \{\}, \{a1\}), (\{v_3, v_2\}, \{(v_3, v_2)\}, \{c3, b2\})\}$
- Descendants in Fig. 3.24f
  $\text{desc}(T[4]) = \{T[1], T[2], T[3], T[4]\}$

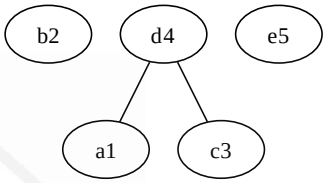


(a) Tree T (The accompanying numbers represent the postorder index)

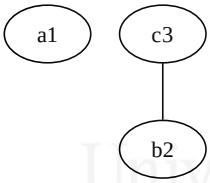
(b) Subtree T_1'



(c) Subtree T_2'



(d) Ordered forest



(e) Ordered subforest

(f) Descendants

Figure 3.24: Sample tree and some operations on it.

3.3.2 Construction of monophonic metrical trees

A melody is structured in bars, which contain the notes to be represented. Each bar is written according to a meter $M = (n, d)$ where $n \subset \mathbb{N}$ stands for the numerator, and $d \subset \mathbb{N}$ denotes the denominator of the meter, which determine the duration of that bar and its structure. The sequence of bars with their meters define a segmentation of the sequence, for our purposes it will be useful to know the meter of the bar where the note is located. Formally:

Definition 3.3.16 *Let us define a metered musical sequence* $S = (P, O, E, D, M)$*, being* $P = \{p_1, p_2, \ldots, p_{|S|}\}$ *a sequence of pitches,* $O = \{o_1, o_2, \ldots, o_{|S|}\}$ *the corresponding onset times,* $E = \{e_1, e_2, \ldots, e_{|S|}\}$ *indicates for each note if it is tied to the previous one,* $D = \{d_1, d_2, \ldots, d_{|S|}\}$ *denotes the durations, and* $M = \{m_1, m_2, \ldots, m_{|S|}\}$ *is the meter of the bar where each note is located. The elements in the sequence are ordered in time, such that* $\forall\ 1 \leq i \leq |S| - 1(o_i < o_{i+1})$*.*

Note in the previous definition that a tied note is represented by two successive elements in sequence with the same pitch but two onsets.

Definition 3.3.17 *A sequence* $S = (P, O, E, D, M)$ *is said to be monophonic if* $\forall\ 1 \leq i \leq |S| - 1(o_i + d_i) \leq o_{i+1}$.

Definition 3.3.18 *Let* $M = (n, d)$ *be a meter, being* n *is numerator and* d *its denominator, the duration of that meter* $\text{mdur}: M \to \mathbb{R}$ *is computed as* $\text{mdur}(n, d) = \frac{n \times 4}{d}$.

Definition 3.3.19 *Let* $M = (n, d)$ *be a meter. The function* $\text{compound}: M \to \textbf{Bool}$ *is defined as:* $\text{compound}(n, d) = (d = 2 \wedge n = 6) \vee (d > 2 \wedge (n \mod 3) = 0 \vee (d \mod 2) = 0 \wedge n/3 > 1)$.

This definition includes meters like $\frac{6}{8}$, $\frac{6}{2}$, and $\frac{12}{16}$.

Only western music from the common practice is inside the scope of this dissertation, so only meters from this period will be used. Nevertheless, the extension to include other meters is trivial.

As stated before in section 3.2 in page 76, each bar b_i is represented by a tree τ_i. For any bar b_i, its meter M_i is known. All nodes $v \in \text{desc}(\tau_i)$ are said to be in meter M_i.

Definition 3.3.20 *Let* $T = (V, E, L)$ *be a tree in meter* M, $\text{nodedur}(v), v \in V$ *is the time span of the node, and it is defined by its depth in the tree:* $\text{nodedur}(v) = \frac{1}{2^{\text{h}(v)}} \times \text{mdur}(M)$.

The recursive subdivision of a duration can lead to too deep trees representing notes shorter than a 256th. To avoid it a minimum duration for the notes is established and denoted by a constant kMINDURATION.

The algorithm to construct a tree from a monophonic sequence of notes is given in Algorithm 1. That algorithm is just a wrapper for the procedure that actually creates nodes from the musical sequence: (`fillTree`). For readability reasons, this algorithm has been simplified by removing the tuplet detection. The complete algorithm can be found in the Appendix B (page 164).

Algorithm 1: Construction of a metric tree from a metered monophonic melody

Input: Sequence S of notes
Output: Tree T
Data: i is the index of the current note of the sequence S of notes.
Data: *pendingDuration* is the duration still left of the current note to be put in the tree. Duration is encoded using r_{dabs} representation.
Data: *continuation* is a boolean value that is true when the next node to be created is representing a continuation of previous one
$i = 0$;
$pendingDuration = r_{dabs}(i)$;
$continuation = false$;
T = new empty tree **while** $(i \leq |S|) \vee (pendingDuration \geq$ kMINDURATION$)$ **do**
    τ = new tree with `nodedur`$(\tau) = \text{mdur}(m_i)$;
    `//see Alg. 2 in page 85`
    $continuation =$ `fillTree` $(\tau, i, pendingDuration, continuation)$;
    add τ as next child of T
end

Algorithm 2: fillTree

Input: Subtree tree τ
Data: *pendingDuration* is the duration still left of the current note to be put in the tree.
Data: i is the index of the current note of the sequence S of notes. p_i stands for the pitch of i-th note, and d_i for its duration
Data: *continuation* is a boolean value indicating the next node will be a continuation of previous one
Output: *continuation* the same parameter is also an output one

```
if i ≤ |S| then
    if pendingDuration ≥ nodedur(τ) then
        label(τ) = (p_i, continuation) ;
        pendingDuration = pendingDuration − nodedur(τ);
        if pendingDuration < kMINDURATION then
            i = i + 1;
            if i ≤ |S| then
                pendingDuration = d_i + pendingDuration;
            else
                pendingDuration = 0;
            end
            continuation = false;
        else
            continuation = true;
        end
    else
        if nodedur(τ) ≥ kMINDURATION then
            //see Alg. 3 in page 85
            a = computeArity(τ);
            for j = 1 to a do
                //createSubTree creates a new child for the given tree
                τ_j = createSubTree(τ);
                fillTree (τ_j, i, pendingDuration, continuation);
            end
        end
    end
end
```

Algorithm 3: computeArity

Input: Tree τ
Input: Current meter with b beats at its numerator
Output: Number of children of the tree given tree

```
if compound (n, d) then
    switch depth (τ) do
        case 1: return n/3;
        case 2: return 3;
        otherwise return 2;
    end
else
    switch depth (τ) do
        case 1: return b;
        otherwise return 2;
    end
end
```

3.4 Propagation

Two causes motivate the procedure described in this section. Firstly, most tree similarity algorithms need all the nodes (both internal and leaves) to have a label, but in the metric tree construction process, after the structure of the tree is completed, the pitch labels are just in the leaves. A set of rules are used for propagating the labels from the leaves upwards, labeling the internal nodes. The propagation of a label is decided on the basis that the note in that node is more important than that of the sibling node. The importance of a note is related to its capability to contribute to the melody identity. See in Figs. 3.25 [4], 3.26, and 3.27 how after propagating the first bars of the original theme and the first variation of Mozart's K331 become the same melody.

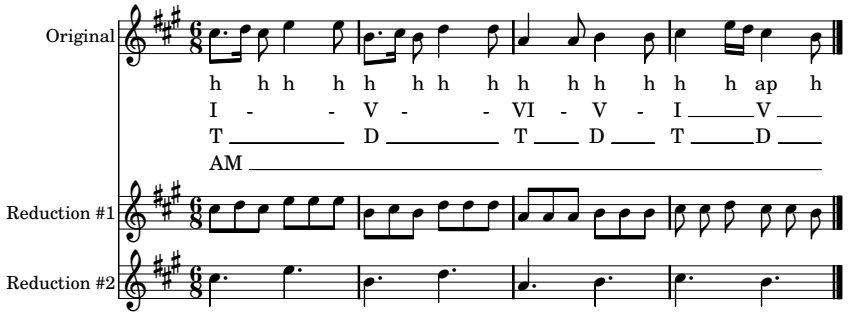
Secondly, the high complexity of the tree edit distance computing (detailed below in section 3.5), requires the trees to be as small as possible. When very short notes appear or they do not match exactly the binary or ternary subdivisions of the beat, the resulting trees are very deep. Thus, the label propagation rules are accompanied of a pruning action to delete little significant branches when applying the rules below a given *pruning level*. This process also contributes to remove irrelevant information that would make the classification more difficult, obtaining reduced trees able to keep the main musical features of the melody.

The decision what is relevant and what is not is a very difficult issue, note how both Shenkerian analysis and GTTM have problems on establishing in a deterministic way that decision. Our target is not to analyze works, but to decide which notes are more important in order to compare the songs in a computable way. Following these principles, two propagation schemes have been proposed, namely *heuristic*, *melodic-analysis* with two different variants. The former is based on empirical rules found after visual analysis of musical works against variations or renditions of them. The later uses the melodic analysis as a basis for determining the important notes. We have included also two very simple schemes that only rely on the metrical relative stress of notes: *left-propagation* and *right-propagation*.

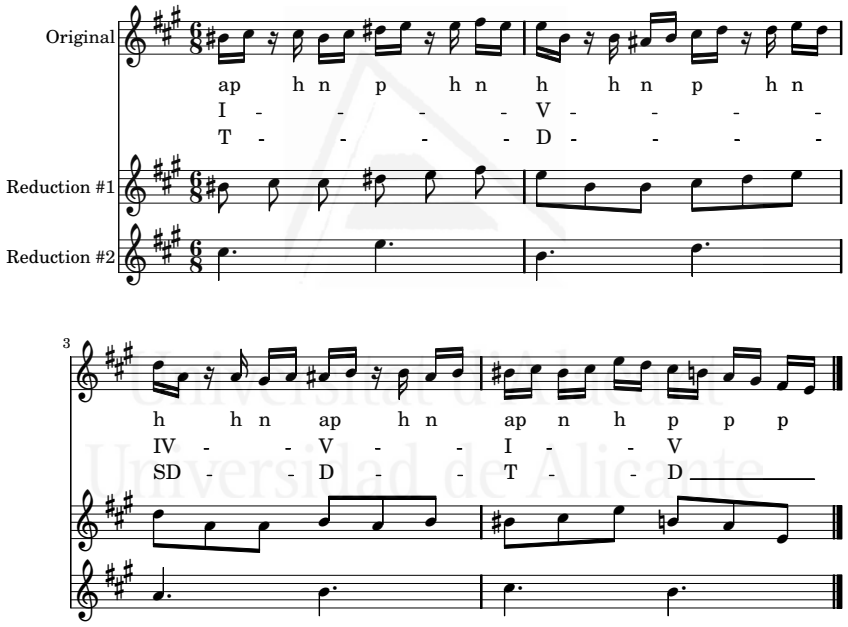
3.4.1 Heuristic propagation scheme

Given a *pruning level*, l, the rules for propagating the labels to internal nodes and pruning the tree are defined below. In each case, a rule is applied to a sub-tree, and if the level d of the sub-tree is below the pruning level l, the labels are propagated and the tree is pruned; otherwise, the rule only propagates labels, keeping the structure of the tree. This pruning level is equivalent to a quantization. This way, the notes represented will be always longer or equal to a $1/2^{N-1}$ fraction of the bar length, where N is the numerator of the bar meter. A value $l = \infty$ means that pruning is never applied.

[4] Tonal analysis by Plácido R. Illescas



(a) Mozart's K331 **theme**. The original score is notated with harmonic (h) and non-harmonic tones ('ap' for appoggiatura, etc.) as defined in Sec. 2.1.3, degrees (I, V, VI), tonal functions (T, D, S), and in the last line the tonality (AM). The staves named "Reduction #n" are empirically performed by the author by progressively removing shortest notes



(b) Mozart's K331, **first variation**, with a possible reduction performed by the author.

Figure 3.25: Possible propagation for Mozart's K331 first four bars of original theme and first variation. See tree representing that propagation in Figs. 3.26 (theme) and 3.27 (variation).

The set of propagation (and pruning when applicable) rules are described below and illustrated in figure 3.29. For the definitions, a parenthesis notation is used for the trees that simplifies the rules description, in such a way that a subtree, t, located in level d, having a father node with label a, and two children: left with label b and right with label c, is denoted as $t = a(bc)$. If a node has not a label, we will consider it as labelled with the empty label, ϵ. All the labels, except ϵ, are symbols in one of the Σ_p alphabets.

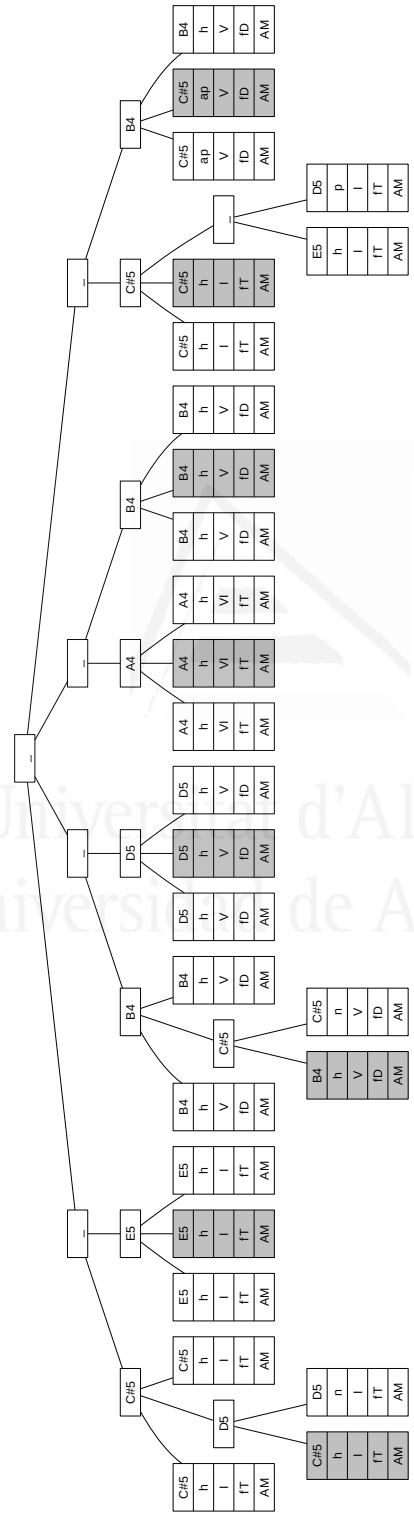


Figure 3.26: Tree containing reductions in the form of bottom-up propagations of K331 theme. The labels contain the note name and its attributes (melodic analysis, chord degree, tonal function, tonality) after that melodic tonal analysis

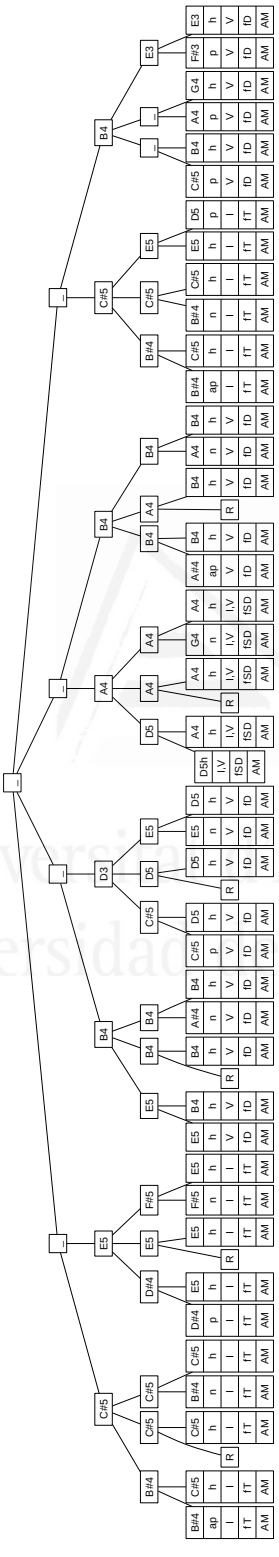


Figure 3.27: Tree containing reductions in the form of bottom-up propagations of K331, first variation. The labels contain the note name and its attributes (melodic analysis, chord degree, tonal function, tonality) after that melodic tonal analysis.

All the definitions have been stated for binary sub-trees but they can be extended for ternary trees, keeping the meaning of each situation. All these rules are illustrated in figure 3.29.

Definition 3.4.1 *Propagation rules:*

r1 This rule simply propagates/prunes a unary tree:

$$r1\big[\,\epsilon(a)\big] = \begin{cases} a & \text{if } d \geq l \\ a(a) & \text{otherwise} \end{cases}$$

If there is only one child it is automatically upgraded. This situation seldom appears but it can be found in the rightmost note of an incomplete measure or building the tree from a single measure.

r2 This rule makes the pitch propagate over a rest:

$$r2\big[\,\epsilon(\epsilon a)\big] = \begin{cases} a & \text{if } d \geq l \\ a(\epsilon a) & \text{otherwise} \end{cases}$$

$$r2\big[\,\epsilon(a\epsilon)\big] = \begin{cases} a & \text{if } d \geq l \\ a(a\epsilon) & \text{otherwise} \end{cases}$$

r3 The *r3* rule is also very simple, and joins equal pitches:

$$r3\big[\,\epsilon(aa)\big] = \begin{cases} a & \text{if } d \geq l \\ a(aa) & \text{otherwise} \end{cases}$$

If all the children of a node have the same label, they are deleted and its label is placed in the father node. Thus, two equal notes are equivalent to just one with double duration.

r4 If one of the children nodes has the same label as that of the father's sibling node, then the other label is propagated. This rule tries to avoid the propagation of a pitch that would be lost by the application of *r3* in the next step. This is formalized here for all possible cases:

$$r4\big[\,\epsilon(\epsilon(ba)b)\big] = \begin{cases} \epsilon(ab) & \text{if } d \geq l \\ \epsilon(a(ba)b) & \text{otherwise} \end{cases}$$

$$r4\big[\,\epsilon(\epsilon(ab)b)\big] = \begin{cases} \epsilon(ab) & \text{if } d \geq l \\ \epsilon(a(ab)b) & \text{otherwise} \end{cases}$$

$$r4\big[\,\epsilon(b\,\epsilon(ba))\big] = \begin{cases} \epsilon(ba) & \text{if } d \geq l \\ \epsilon(ba(ba)) & \text{otherwise} \end{cases}$$

$$r4\big[\,\epsilon(b\,\epsilon(ab))\big] = \begin{cases} \epsilon(ba) & \text{if } d \geq l \\ \epsilon(ba(ab)) & \text{otherwise} \end{cases}$$

r5 The *r5* rule limits the applicability of the former rules, that otherwise could propagate a very short pitch to a much longer note, eliminating other longer pitches. In order to avoid that, any rule (denoted as r in the rule below) can be applied only three times for the same label (this implies to stretch its length in a factor of 2^3 for binary meters).

$$r5\big[\,\epsilon(ab)\big] = \begin{cases} b & \text{if } d \geq l \\ b(ab) & \text{otherwise} \end{cases}$$

$$r5\big[\,\epsilon(ba)\big] = \begin{cases} b & \text{if } d \geq l \\ b(ba) & \text{otherwise} \end{cases}$$

iff

a is the root of $t = r\big[r\big[r\big[\ldots\big]\big]\big]$

and

a comes from a node 3 levels below.

r6 The *r6* rule is a "default" case, whenever no other rule may be applied:

$$r6\big[\,\epsilon(ab)\big] = a(ab)$$

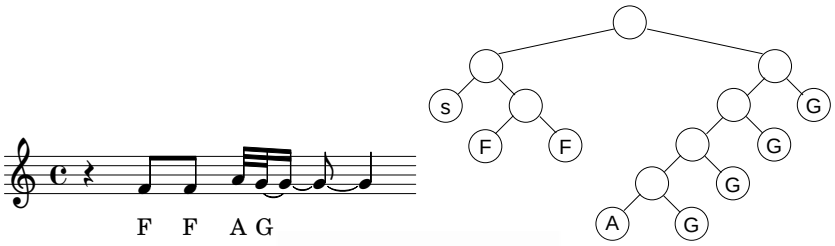
This rule upgrades the label of the left child, because in binary meters, the notes placed in odd beats are usually stressed. These notes are represented by left children in the tree.

All these rules are applied under certain conditions and precedence order that are described in the algorithm 4:

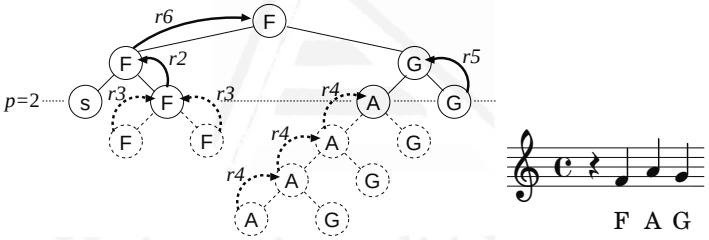
Algorithm 4: Application of rules for heuristic propagation

if arity(t) = 1 **then**
  r1
else if left-child(t) = ϵ or right-child(t) = ϵ **then**
  r2
else if left-child(t) = right-child(t) **then**
  r3
else if root(t) comes from a leaf 3 levels below **then**
  r5
else if $t = \epsilon(\epsilon(ba)b)$ or $t = \epsilon(\epsilon(ab)b)$ or $t = \epsilon(b\,\epsilon(ba))$ or $t = \epsilon(b\,\epsilon(ab))$ **then**
  r4
else
  r6
end if

An example of the application of these rules is displayed in Figure 3.28 with a pruning level $l = 2$. One measure with some notes with different durations is considered. In the left side of that figure, the score and the tree as it results from the construction procedure is presented. The labels in that tree are note names without octave.



(a) One measure-melody and its tree representation with pitch labels (only in the leaves now) before pruning and label propagation.



(b) Final tree with propagated and pruned nodes (in dashed lines after applying dashed rules). The equivalent melody to the pruned tree is also displayed

Figure 3.28: Effect of the heuristic propagation rules on a melody.

In Fig. 3.28a it can be observed how the propagation and pruning rules apply to the tree. A value of the pruning level $l = 2$ has been considered. This way, the rules applied below that level in the tree ($d \geq l$) are pruning rules, otherwise are just propagation rules. In the first half of the melody, the deepest levels have equal labels (F), so they are upgraded by the rule *r3* and then by *r2* because the sibling node is labelled with a rest. The second part shows how a very short note (A) is propagated by applying *r4* three times. Thus, *r5* is applied instead of *r6* that otherwise would have been applied, propagating 'A' again.

Note that in the score equivalent to that tree (Fig. 3.28b) only quarter notes (in this context, their duration can be stated as a $1/2^{l-1}$ of the measure) have survived to the propagation and pruning rules, keeping the main features of the original melody.

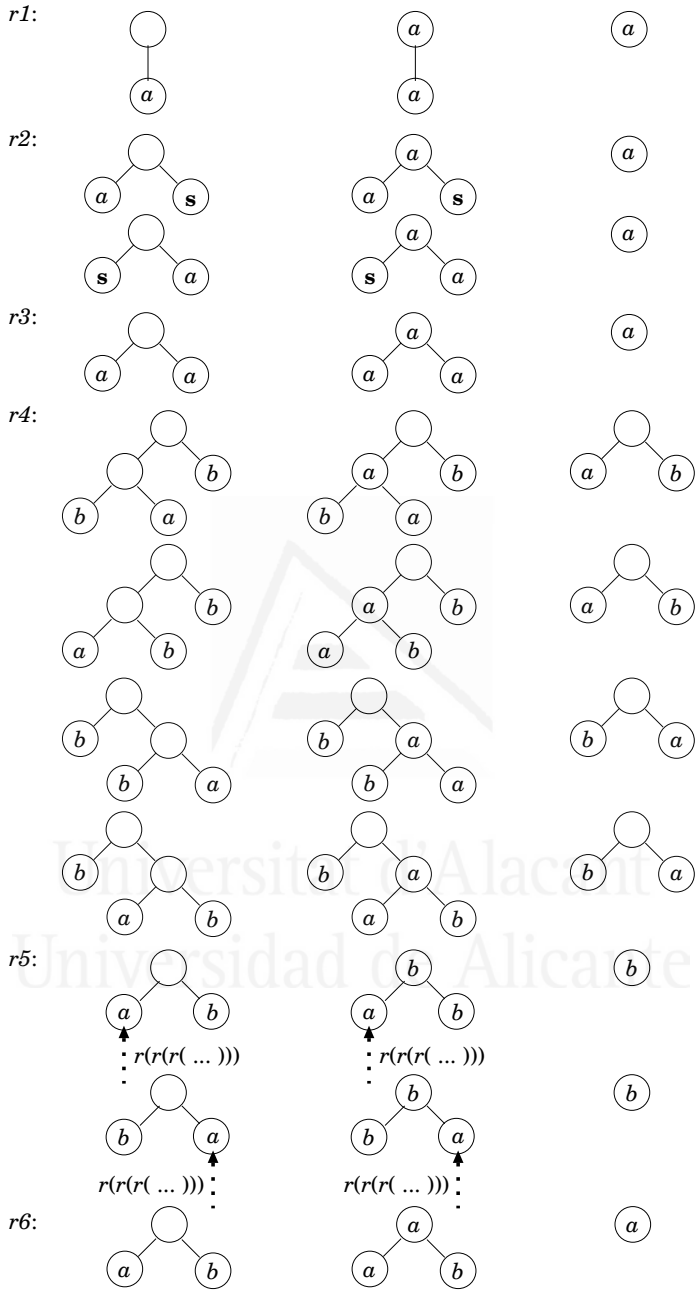


Figure 3.29: Propagation and pruning rules. (left column): original sub-tree; (center column) propagation rules; (right column): pruning rules. '**s**' representing rests stands for ϵ in the rule definitions.

3.4.2 Melodic-analysis propagation scheme

This analysis is based on the assumption that usually, the ornamentation of melodies is performed by introducing non-harmonic tones. Thus, in the propagation process, the decision on what note is more important is taken as a function of their melodic tagging (see Section 2.1.3). Whenever both notes are harmonic, the more stressed note, i.e., the left note, is propagated. The rules that define these situations are defined in Def. 3.4.2, and Algorithm 5 is given (note that the rules of the *heuristic* propagation are reused).

Definition 3.4.2 *Propagation rules:*

r7 This rule gives precedence to harmonic tones over non-harmonic tones.

$$r7\big[\,\epsilon(ab)\big] = \begin{cases} a(ab) & \textit{if } a \textit{ is a harmonic tone and } b \textit{ is a non-harmonic tone} \\ b(ab) & \textit{if } a \textit{ is a non-harmonic tone and } b \textit{ is a harmonic tone} \\ a(ab) & \textit{otherwise} \end{cases}$$

Algorithm 5: Application of rules for melodic analysis propagation

if arity(t) = 1 **then**
  r1
else if left-child(t) = ϵ or right-child(t) = ϵ **then**
  r2
else if left-child(t) = right-child(t) **then**
  r3
else
  r7
end if

Algorithm 6: Application of rules for left and right propagation

if arity(t) = 1 **then**
  r1
else if left-child(t) = ϵ or right-child(t) = ϵ **then**
  r2
else if left-child(t) = right-child(t) **then**
  r3
else
  r8 for left propagation, *r9* for right propagation
end if

A special scheme named *partial* propagation has been devised in order to check the suitability of the last case of rule $r7$. In this propagation, when both notes are harmonic tones, none is propagated, leaving the tree partially labelled.

3.4.3 Left and right propagations

The *left* (rule $r8$ in Def. 3.4.3) and *right* (rule $r9$ in Def. 3.4.3) propagation schemes just propagate notes on the basis of their position in the tree, and are devised to be a base case to compare the other more sophisticated schemes. Algorithm 6 is used to perform the propagation, using rule $r8$ for *left-propagation*, and rule $r9$ for *right-propagation*.

Definition 3.4.3 *Propagation rules:*

r8 Propagate always left node

$$r8\big[\,\epsilon(ab)\big] = a(ab)$$

r9 Propagate always right node

$$r9\big[\,\epsilon(ab)\big] = b(ab)$$

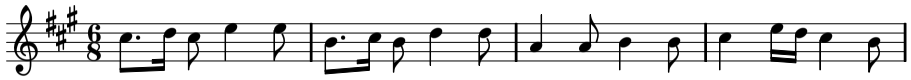
Recall Figure 3.23 in which we compared some existing tree representation methods with ours. We noted that both Schenker analysis and GTTM are based on a bottom-up grouping and reduction of the melody until reaching a common root. We have collected again, now in Fig. 3.30, both the GTTM and Marsden's Schenker analyses along with two propagations of the same work by using our methods. Our *melodic-analysis* propagation resembles the approach by Marsden, while keeping rhythm information. The example shown is simple and using both *left* propagation, and the *heuristic* method, they lead to the same tree. However, this is not always the case. The important point here is that following a pretty simple propagation scheme we have obtained, at least for that example, the same third level for our *melodic-analysis* propagation than the third level in the Marsden's Schenkerian analysis, and almost the same than the third level of the GTTM tree. This seems to be coherent basis for the tree similarity computation methods to compare fully labelled trees in order to measure the resemblance of music works.

3.5 Comparison of trees

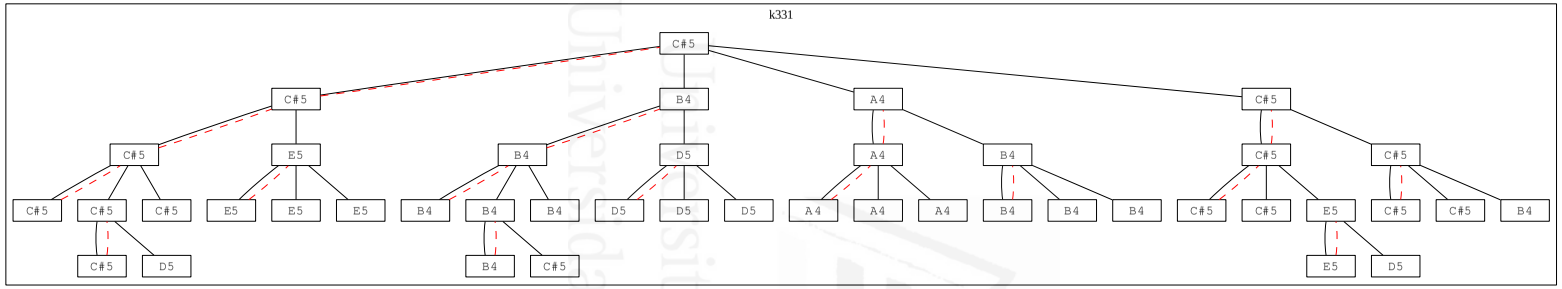
The goal of this work is to measure the similarity between two pieces of music represented by trees. Thus, in order to compare music, we need a measure to compare metric trees.

In the metric trees depicted so far, the tree structure describes meter and beat information. Leaves represent the actual pitches and inner nodes contain the result of the bottom-up propagation just described. The devised similarity measure should assign higher rates to the comparison of two variations of the same melody than to two different works. These variations are supposed to be represented by similar trees in structure and pitch labels.

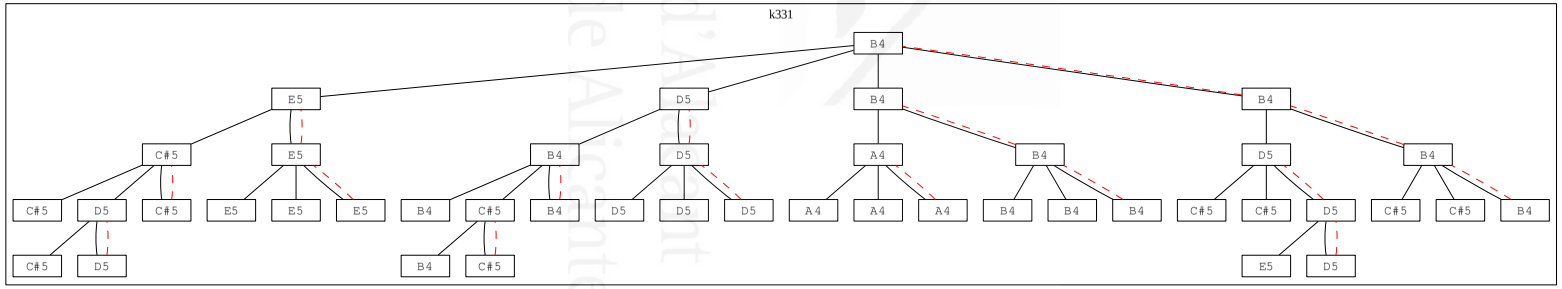
In general, trees can be divided in *ordered* and *not ordered trees*, and in *evolutionary* and *not evolutionary* trees. As the time dimension of music is represented implicitly in the left-to-right ordering of trees, we deal only with ordered trees. Regarding



(a) Beginning of Mozart's k331 Piano Sonata

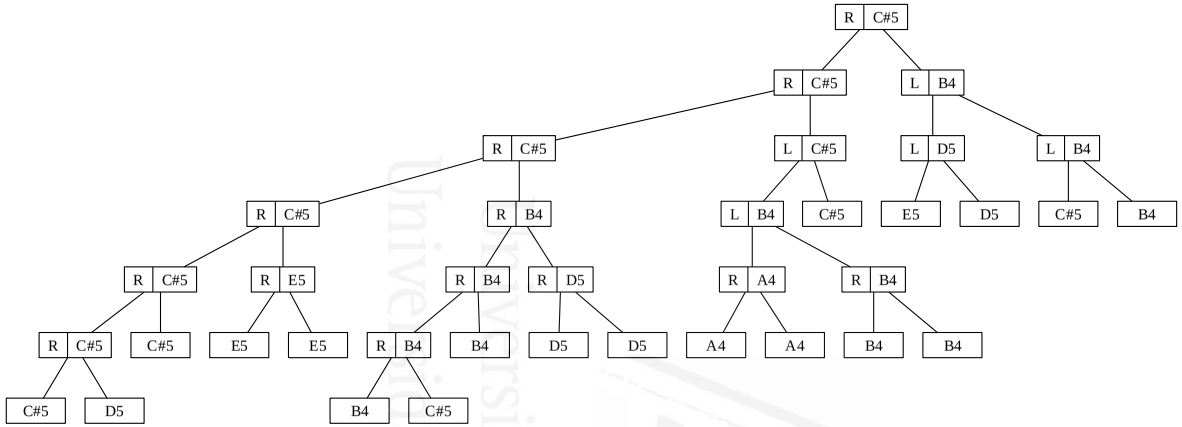


(b) Metric tree with propagation. In this example the *heuristic*, *melodic analysis*, and *left* propagations coincide

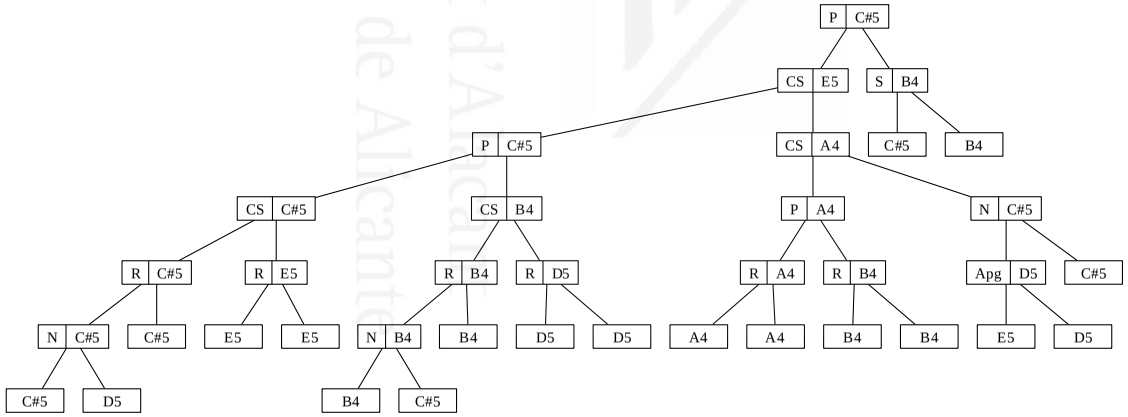


(c) Metric tree with *right* propagation

Figure 3.30: Metrical tree propagations compared with other tree representations with a standardized notation (continued on next page)



(d) GTTM tree in our standardized tree notation



(e) Schenkerian analysis (after Marsden) in our standardized tree notation. The left box of each node contains the elaboration used, the right box has the note pitch name.

Figure 3.30: Metrical tree propagations compared to other tree representations with a standardized notation.

the *evolutionary* trees, they are often used to conceptually represent the evolutionary relationship of species or organisms in biology, evolution of works in linguistics, statistical classifications, or even tracking computer viruses. The *metrical trees* we propose are not *evolutionary* trees because they do not have distinct labels, so those algorithms are not applicable for our problem.

There have been defined in the literature a number of similarity measures for the ordered non-evolutionary trees equivalent in most cases to those of strings reviewed in section 2.1.4. They have been used to compare ADN trees, XML structures, etc. Some of them measure the sequence of operations needed to transform one tree in another one (Tai, 1979), others look for the longest common path from the root to a tree node (Bille, 2007), and there are methods that allow wildcards in the matching process in the so-called *Variable-Length Doesn't Care (VLDC)* distance (Shasha and Zhang, 1997). Several taxonomies of these measures have been proposed. The interested reader can look up a hierarchy of tree edit distance measures in (Kuboyama et al., 2005) and (Valiente, 2001), and a survey in (Bille, 2005).

In this dissertation, four main techniques on tree distances have been considered: tree edit distances (Zhang and Shasha, 1989), constrained edit distances (Selkow, 1977), alignment distances (Jiang et al., 1995), and bottom-up distances (Valiente, 2001). Besides, a new distance between trees only labelled at leaves is proposed.

3.5.1 Tree mapping

When assessing the similarity between two trees T_A and T_B, we measure the cost of mapping the nodes of tree T_A to the nodes of the tree T_B (see Fig. 3.31). Non-matched nodes in T_B are said to be inserted, non-matched nodes in T_A are said to be deleted, and matched nodes in both trees having different labels are said to be relabelled.

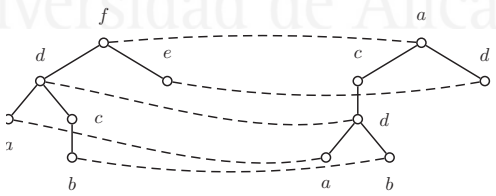


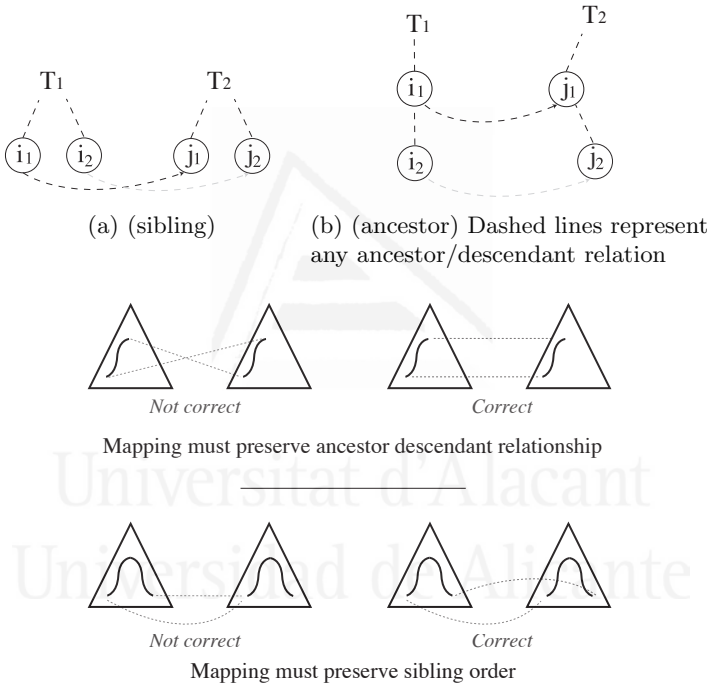
Figure 3.31: A possible mapping between trees T_A (left) and T_B (right) (from (Bille, 2005)).

Lemma 3.5.1 *Formally, a mapping from T_A to T_B, is a triplet (M, T_A, T_B), where M is any set of pair of integers $(i, j), 1 \leq i \leq |T_A|, 1 \leq j \leq |T_B|$ satisfying the following conditions (Shasha and Zhang, 1989) (see Fig. 3.32). For any pair of (i_1, j_1) and (i_2, j_2) in M:*

$$\begin{aligned} &\textit{(one-to-one)} && i_1 = i_2 \iff j_1 = j_2 \\ &\textit{(ancestor)} && T_A[i_1] \textit{ is an ancestor of } T_A[i_2] \iff T_B[j_1] \textit{ is an ancestor of } T_B[j_2] \\ &\textit{(sibling)} && T_A[i_1] \textit{ is to the left of } T_A[i_2] \iff T_B[j_1] \textit{ is to the left of } T_B[j_2] \end{aligned}$$

In the following, M is used instead of (M, T_A, T_B) if there is no possibility of confusion.

Definition 3.5.1 *The cost of M, denoted by $\gamma(M)$, is the sum of the cost of nodes inserted plus the cost of nodes deleted plus the cost of nodes relabeled in M, being usual to use the unit-cost setup, i.e., cost 1 for each node deleted or inserted, 1 for replacing two different labels, and 0 for replacing equal labels. The optimal mapping can be defined as the mapping with minimum cost:* $\delta(T_A, T_B) = \min_{(M,T_A,T_B)}\{\gamma(M)\}$.



(a) (sibling)

(b) (ancestor) Dashed lines represent any ancestor/descendant relation

(c) Mapping rules (from (Shasha and Zhang, 1989))

Figure 3.32: Mapping conditions.

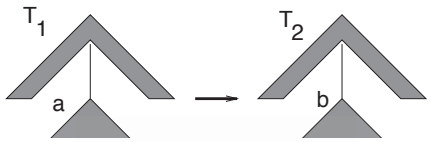
3.5.2 Tree Edit Distance

Analogous to the definition for strings, the classical edit distance between two trees, is the minimal cost to transform one input tree into an output one by edit operations. The main difference in the case of trees is the meaning of the edit operations. An *edit operation* on two trees $T_A = (V, E, L)$ and $T_B = (V', E', L')$ is any of the following:

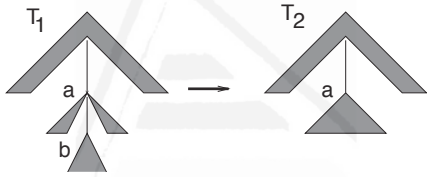
- *relabeling* the label l of a node $v \in V$ with the label l' of another node $w \in V'$, denoted by (v, w). It is equivalent to the substitution definition in strings (see Fig. 3.33a).

- *deletion* of a non-root node $v \in V$, denoted by (v, λ), consists of deleting it, making the children of v become the children of par(v), just in the position that was occupied by v, preserving this way the left to right ordering of leaves (see Fig. 3.33b).

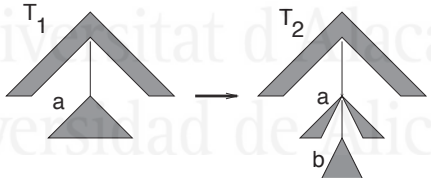
- *insertion* of a non-root node $w \in V'$, denoted by denoted by (λ, w), complements the *delete* operation. Given a sequence $w_i \cdots w_j$ of subtrees of a common parent w, the insertion of node w' makes those $w_i \cdots w_j$ subtrees children of w', and w' child of w (see Fig. 3.33c).



(a) Substitute operation



(b) Delete operation



(c) Insert operation

Figure 3.33: Tree edit operations (from (Isert, 1999)).

To each operation, a so-called *edit cost* $c_t \colon V \times V' \to \mathbb{R}$ is assigned, and it is based on the edit costs of the symbols of alphabet Σ at labels $c \colon \Sigma \times \Sigma \to \mathbb{R}$ (see Sec. 2.1.4 in page 29 for the definition of cost function c for strings). Therefore, $c_t(v, w)$ denotes the cost of applying the edit operation (v, w) where v is an input node and w is an output node. If $v = \lambda$, the operation denotes an insertion, if $w = \lambda$ the operation is a deletion. Note that the operation (λ, λ) is not allowed. For the substitution cost $c_t(v, w) = c(\text{label}(v), \text{label}(w))$. In the case of insertion and deletion, it is achieved by iteratively inserting and removing a series of nodes as defined below in Eq. 3.1 and 3.2.

The cost of deleting a complete tree rooted at $v \in V$ is defined recursively as:

$$c_t(v, \lambda) = c(\text{label}(v), \lambda) + \sum_{v_i \in \text{children}(v)} c_t(v_i, \lambda) \tag{3.1}$$

Similarly, the cost of inserting a tree rooted at $w \in V'$ is defined as:

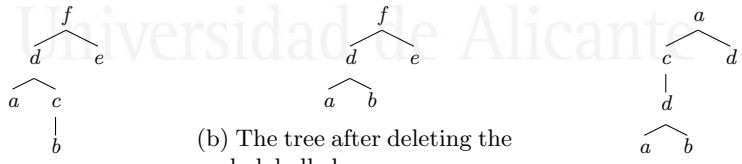
$$c_t(\lambda, w) = c(\lambda, \text{label}(w)) + \sum_{w_i \in \text{children}(w)} c_t(\lambda, w_i) \tag{3.2}$$

The difference between the mapping and the edit operations is mainly that the former only affects the nodes themselves that are inserted, deleted, or relabeled, without any change in the structure of the tree, while in edit operations, both the insertion and deletion can restructure the tree, affecting the children of the inserted or deleted nodes. This fact makes it necessary to define a new cost function for the edit distance different from that of mapping.

Definition 3.5.2 *An edit script is a triplet* (T_A, T_B, E_t) *where* $E_t = e_t = e_{t_1} \cdots e_{t_n}$ *is a sequence of edit operations* $e_{t_i} = (a_i, b_i) \in (V \cup \{\lambda\}) \times (V \cup \{\lambda\})$ *allowing the transformation of a tree* T_A *in a tree* T_B*. The cost of an edit script* $\pi_t(e_t)$ *is the sum of the costs of the edit operations involved in the script:* $\pi_t(e_t) = \sum_{i=1}^{n} c_t(e_{t_i})$.

Definition 3.5.3 *Let* $S_t(T_A, T_B)$ *be the set of all the scripts that enable the emission of* T_B *given* T_A*, the edit distance between* T_A *and* T_B *is defined by:* $d_t(T_A, T_B) = min_{e_t \in S_t(T_A, T_B)} \pi_t(e_t)$.

The sequence S of operations that leads from T_A to T_B is named S*-derivation*. There exists a mapping M from T_A to T_B such that the cost of the mapping is less or equal than the cost of the editing script: $\gamma(M) \leq \pi_t(S)$ (Zhang and Shasha, 1989). The mapping of Fig. 3.31 is the one corresponding to the edit script in Fig. 3.34.



(a) The initial tree

(b) The tree after deleting the node labelled c

(c) The tree after inserting the node labelled c and relabelling f to a and e to d

Figure 3.34: Transforming (a) into (c) by means of editing operations (from (Bille, 2005)).

The first author to give a solution to the general tree edit problem was Tai (1979), proposing an algorithm with time complexity $O(|T_A| \times |T_B| \times \text{depth}(T_A)^2 \times \text{depth}(T_B)^2)$, where $|T_i|$ denotes the number of nodes in tree T_i. This algorithm was improved by Zhang and Shasha (1989) giving a dynamic programming algorithm with time complexity $O(|T_A| \times |T_B| \times \min(\text{depth}(T_A), |\text{leaves}(T_A)|) \times \min(\text{depth}(T_B), |\text{leaves}(T_B)|))$, and a faster alternative to the algorithm using unit costs can be found in (Shasha and Zhang, 1990).

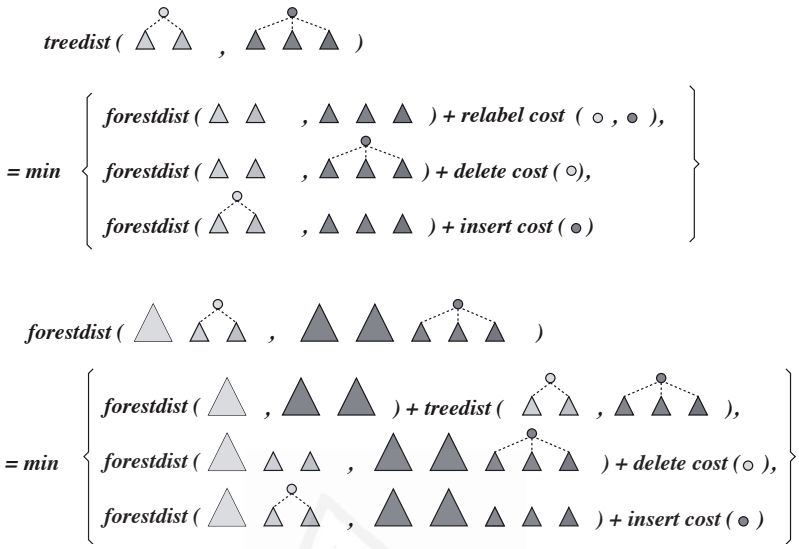


Figure 3.35: Zhasha and Zhang tree edit distance algorithm (from (Dulucq and Tichit, 2003)). *treedist* is the distance between two trees, *forestdist* stands for the distance between two ordered forests.

(a) $l(i_1)..i$ (b) $l(i_1)..i-1$

Figure 3.36: Scheme of recursion formula

The algorithm of Zhang and Shasha, that is based on the computation of the distance between ordered forests, is depicted graphically in Fig. 3.35.

Some comments about notation can help to easily understand the tree edit distance formula by Zhang and Shasha (1989) that is given below. The function $l(i)$ is used to denote the left-most descendant of node i. In order to define recursions, the relation of forest $l(i_1)..i$ with forest $l(i_1)..i-1$, where $i \in \text{desc}(i_1)$, is usually used. This means just removing the node i (see Fig. 3.36) . In the example of Fig. 3.37, $i_1 = 18, i = 17$, shaded nodes belong to the first set $(l(i_1)..i)$, and double line border nodes to the second one $(l(i_1)..i-1)$. Now, it easy to understand the first part of the Shasha and Zhang's general formula (Lemma 3.5.2). For example, in the insertion case, it means removing recursively the node i, what leads to the same calculation as the definition of $c_t(\lambda, T)$ in Equation 3.2.

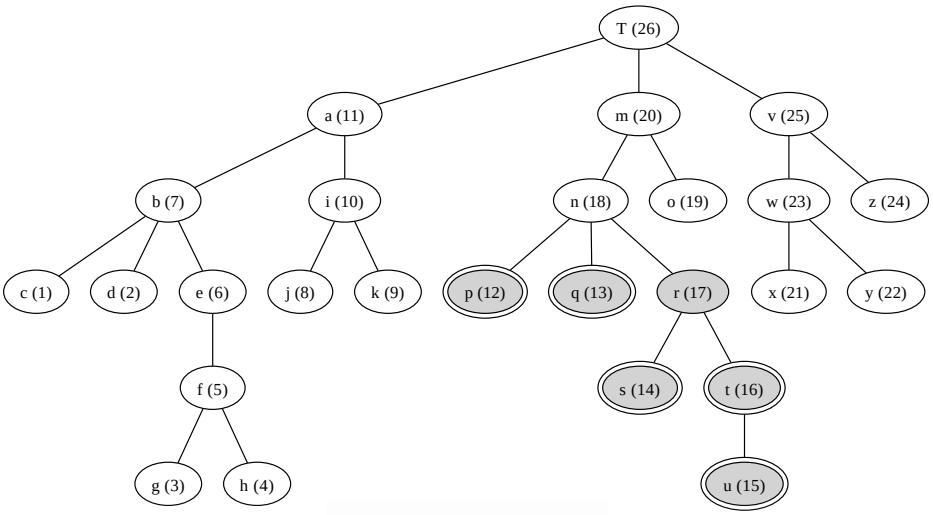


Figure 3.37: Tree example. The numbers between parenthesis denote the postorder index. The shaded nodes represent the forest defined by $l(18)..17$. The nodes with double line border are those in forest $l(18)..16$

Lemma 3.5.2 *Let* $i \in \text{desc}(i_1)$ *and* $j \in \text{desc}(j_1)$

$$
\begin{aligned}
&(i) \quad forestdist(\lambda, \lambda) = 0 \\
&(ii) \quad forestdist(l(i_1)..i, \lambda) = forestdist(l(i_1)..i-1, \lambda) + c(\text{rlabel}(T[i])) \\
&(iii) \quad forestdist(\lambda, l(j_1)..j) = forestdist(\lambda, l(j_1)..j-1) + c(\text{rlabel}(T[j]))
\end{aligned}
$$

Following the scheme depicted in Fig. 3.36, the second part of Shasha and Zhang's general formula (Lemma 3.5.3) is shown in Fig. 3.35.

Lemma 3.5.3 *Let* $i \in \text{desc}(i_1)$ *and* $j \in \text{desc}(j_1)$. *Then*

$$
\begin{aligned}
&(i) \quad \text{if } l(i) = l(i_1) \wedge l(j) = l(j_1), treedist(i,j) = forestdist(l(i_1)..i, l(j_1)..j) = \\
&\qquad \min \begin{cases} forestdist(l(i_1)..i-1, l(j_1)..j) + c_t(\text{rlabel}(T_A[i]), \lambda) \\ forestdist(l(i_1)..i, l(j_1)..j-1) + c_t(\text{rlabel}(\lambda), T_B[j]) \\ forestdist(l(i_1)..i-1, l(j_1)..j-1) + c_t(\text{rlabel}(T_A[i]), T_B[j]) \end{cases} \\
&(ii) \quad \text{if } l(i) \neq l(i_1) \vee l(j) \neq l(j_1) \text{ (i.e. otherwise)}, forestdist(l(i_1)..i, l(j_1)..j) = \\
&\qquad \min \begin{cases} forestdist(l(i_1)..i-1, l(j_1)..j) + c_t(\text{rlabel}(T_A[i]), \lambda) \\ forestdist(l(i_1)..i, l(j_1)..j-1) + c_t(\text{rlabel}(\lambda), T_B[j]) \\ forestdist(l(i_1)..i-1, l(j_1)..j-1) + treedist(i,j) \end{cases}
\end{aligned}
$$

Finally, the algorithm follows a dynamic programming scheme, where all possible edit sequences are computed bottom-up, keeping only that of minimum cost. The process is sped-up by selecting only those roots of subtrees that need separate computations, i.e., those combinations of i_1 and i in the expression $l(i_1)..i$ that represent different trees.

3.5.3 Tree Alignment Distance

Two trees are said to be *isomorphic* if they are either both empty, or having identical root labels, the subtrees rooted at corresponding children of their roots are also isomorphic (Valiente, 2001). An alignment between two trees is obtained by making both trees isomorphic if labels are ignored through the insertion of nodes with special labels so-called *spaces* (Bille, 2007), that is, inserting and removing the required nodes to both trees so that they have the same shape. By superimposing both trees after the insertion of those *spaces*, the alignment cost is the sum of the substitution costs of all pairs of matched nodes in both aligned trees (see Fig. 3.38). The *optimal alignment* is the alignment of minimum cost.

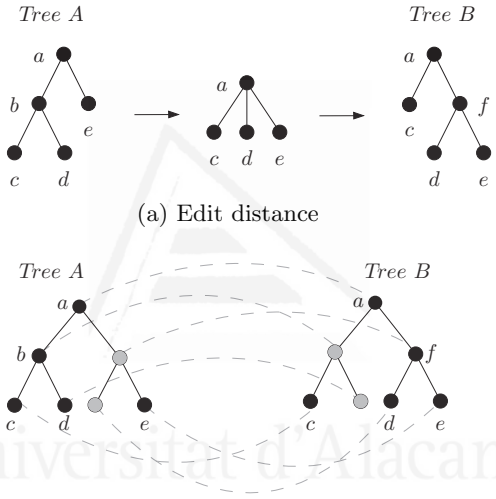


Figure 3.38: Tree edit distance vs. alignment distance. (a) shows the optimal mapping for the two trees A and B with the edit distance: the mapping contains a deletion and an insertion. The *spaces* are plotted as not labeled grayed nodes. With the alignment in (b), the optimal mapping involves four not matched labels. (from (Jiang et al., 1995; Touzet, 2006)).

In strings, the edit distance is equivalent to the optimal alignment distance. However, for trees it may be different (Bille, 2007). The tree alignment distance corresponds to a restricted edit distance where all insertions must be performed before any deletions. Hence, the edit distance is always lower or equal than the optimal alignment (Bille, 2005). It seems that alignment charges more for the structural dissimilarity at the top levels of the trees than at the lower levels, whereas edit treats all the levels the same (Jiang et al., 1995). This assumption is important for us, because at higher levels the represented notes are longer than those at lower levels. In that work, an algorithm to solve the problem in $O(|T_A| \times |T_B| \times (\text{rank}(T_A) + \text{rank}(T_B))^2)$ time and $O(|T_A| \times |T_B| \times (\text{rank}(T_A) + \text{rank}(T_B)))$ space is given.

The alignment distance is defined as (Jiang et al., 1995):

Lemma 3.5.4 *Let T_A and T_B be to trees, for any s,t such that $1 \leq s \leq \text{rank}(T_A)$, and $1 \leq t \leq \text{rank}(T_B)$,*

$$D(T_A[i_1..i_s], T_B[j_1..j_t]) =$$

$$\min \begin{cases} D(T_A[i_1..i_{s-1}], T_B[j_1..j_t]) + D(T_A[i_s], \lambda) \\ D(T_A[i_1..i_s], T_B[j_1..j_{t-1}]) + D(\lambda, T_B[i_t]) \\ D(T_A[i_1..i_{s-1}], T_B[j_1..j_{t-1}]) + D(T_A[i_s], T_B[i_t]) \\ c(\lambda, \text{rlabel}(j_t)) + \min_{1 \leq k < s}\{D(T_A[i_1..i_{k-1}], T_B[j_1..j_{t-1}]) + \\ \quad D(T_A[i_k..i_s], T_B[j_t])\} \\ c(\text{rlabel}(i_s), \lambda) + \min_{1 \leq k < t}\{D(T_A[i_1..i_{s-1}], T_B[j_1..j_{k-1}]) + \\ \quad D(T_A[i_s], T_B[j_k..j_t])\} \end{cases}$$

Implementation improvement.

The problem of this distance is that it heavily depends on the rank of the trees. In our case, the maximum rank is determined by the number of bars in the song. This factor makes this distance extremely slow when comparing long songs.

As stated above, the tree alignment distance corresponds to a restricted edit distance where all insertions must be performed before any deletions (Bille, 2005). Thus, we can use any non-constrained edit distance, as the one of Zhang and Shasha (1989) described above in Section 3.5.2, and keep only those edit scripts that hold this restriction.

The implementation of the edit distance by Zhang and Shasha (1989) uses a dynamic programming table **D**. Each cell in **D** represents the best cost of each edit decision. In order to retrieve the edit script path, another table **S** is required that stores for each cell in **D**, which other cell precedes it.

The invalid edit sequences (i.e. deletions before insertions) must be assigned an ∞ cost. To be able to detect those invalid edit sequences, the best path in **S** must be traversed for each edit decision. This traversing operation at each step would make this alternative implementation of the algorithm even more time consuming than the original.

In order to avoid traversing at each step the partial edit script path, we can use a finite deterministic automata, and keep at each step the current state of the automata (see Fig. 3.39). Any edit operation that leads to the error state must be given an infinity cost so that this path is never taken. This way, the alignment distance will have the same temporal complexity than the edit distance by Zhang and Shasha (1989), that behaves better than the original alignment distance by Jiang et al. (1995) for complete songs.

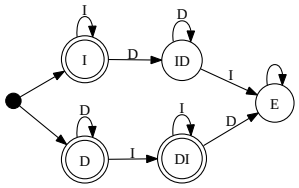


Figure 3.39: Automaton reflecting valid edit operations for the tree edit distance be an alignment distance. 'I' stands for the insert operation, 'D' for delete. The substitution operation is always valid, so it is not reflected in this automaton. The state 'E' stands for "error". Note that the automaton must be read reverse, because in the Zhang and Shasha (1989) the operations are computed also from the end to the beginning.

3.5.4 1-degree tree edit distance

An interesting variant of the edit distance is sometimes called *degree-1 edit distance* (Selkow, 1977) or *top-down distance*, where deletions and insertions are constrained to leaves. Thus, in order to delete an inner node, all its descendants must be deleted before. "It is easy to see that if $T_A[i]$ maps to $T_B[j]$, then the parent of $T_A[i]$ must map to the parent of $T_B[j]$. The reason is that if $T_A[i]$ is not deleted, its parent cannot be deleted or inserted, meaning that if two nodes are matched, then their parents must also be matched" (Shasha and Zhang, 1997). This fact imposes a hard restriction on how the mappings between nodes are made.

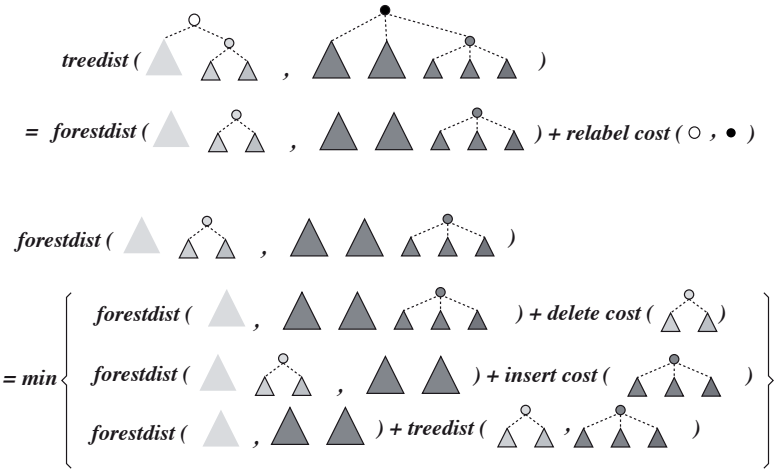


Figure 3.40: Selkow tree edit distance algorithm.

The Selkow's distance between two trees is depicted graphically in Fig. 3.40, and Lemma 3.5.5 defines the distance. The Selkow algorithm has its strength in its low temporal cost $O(|T_A| \times |T_B|)$.

Lemma 3.5.5
Let us use R_A *for* $\text{rank}(T_A)$ *and* R_B *for* $\text{rank}(T_B)$*, and let* $treedist(i..i', j..j')$ *stand for* $treedist(T_A[\text{child}_i(T_A)..\text{child}_{i'}(T_A)], T_B[\text{child}_j(T_B)..\text{child}_{j'}(T_B)])$

$$
\begin{aligned}
&(i) && treedist(\lambda, \lambda) = 0\\
&(ii) && treedist(T_A, \lambda) = c_t(T_A, \lambda)\\
&(iii) && treedist(\lambda, T_B) = c_t(\lambda, T_B)\\
&(iv) && treedist(T_A, T_B) = c(\text{rlabel}(T_A), \text{rlabel}(T_B)) + forestdist(1..R_A, 1..R_B)\\
&(v) && forestdist(\lambda, \lambda) = 0\\
&(vi) && forestdist(i..i', \lambda) = forestdist(i..i'-1, \lambda) + c_t(T_A[i'], \lambda)\\
&(vii) && forestdist(\lambda, j..j') = forestdist(\lambda, j..j'-1) + c_t(\lambda, T_B[j'])\\
&(viii) && forestdist(i..i', j..j') =\\
& && \min \begin{cases} forestdist(i..i'-1, j..j') + c_t(T_A[i'], \lambda),\\ forestdist(i..i', j..j'-1) + c_t(\lambda, T_B[j']),\\ forestdist(i..i'-1, j..j'-1) + treedist(T_A[i'], T_B[j']) \end{cases}
\end{aligned}
$$

3.5.5 Bottom-up distance

The bottom-up distance between two non-empty rooted trees T_A and T_B is equal to $1 - f/\max(|T_A|, |T_B|)$, where f is the size of a largest common forest of T_A and T_B (see Fig. 3.41), and it can be computed in $O(|T_A| + |T_B|)$ (Valiente, 2001).

However, this complexity is actually $O(|T_A| \times |T_B| \times \log(T_A + T_B))$, because in the original paper the computing of the bottom-up mapping is not included in the complexity calculation [5].

In order to compute that distance the following special mappings must be defined.

Definition 3.5.4 *A mapping* M *from a tree* T_A *to a tree* T_B *is* isolated-subtree *if it satisfies the following condition: for all* $(i_1, j_1), (i_2, j_2) \in M$*, the rightmost node of* $T_A[i_1]$ *is to the left of* $T_A[i_2]$ *iff the rightmost node of* $T_B[j_1]$ *is to the left of* $T_B[j_2]$*.*

Definition 3.5.5 *Let* T_A *and* T_B *be two trees, a mapping* M *is said to be* top-down *if* $\forall (i, j) \in M \rightarrow (\text{par}(i), \text{par}(j)) \in M$ *(see Fig. 3.42a).*

Definition 3.5.6 *An isolated-subtree mapping from a tree* T_A *to a tree* T_B *is said to be* bottom-up *if* $\forall (i, j) \in M \rightarrow (i_1, j_1), \ldots, (i_k, j_k) \in M$*, where* $i_1, \ldots, i_k \in \text{children}(T_A)$ *and* $j_1, \ldots, j_k \in \text{children}(T_B)$ *(see Fig. 3.42b).*

[5] The nested loop in the mapping function (lines from 3 to 12 of the algorithm included in (Valiente, 2001)) that traverses in level-order all the nodes of both trees leads to $O(|T_A| \times |T_B| \times \log(T_A + T_B))$, where the logarithm corresponds to the map operations on a $(|T_A| + |T_B|)$ size map inside the double loop.

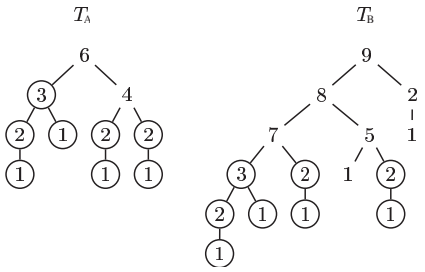
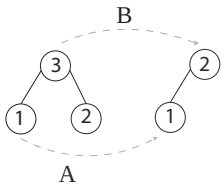
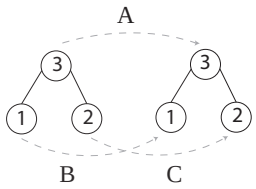


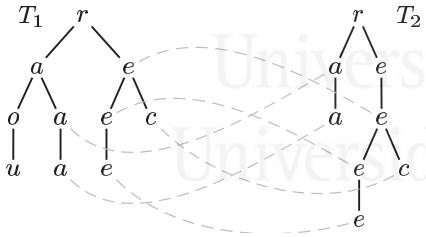
Figure 3.41: A largest common forest between two rooted trees.



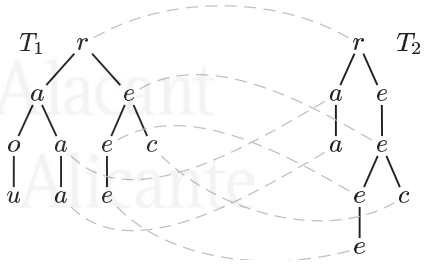
(a) Top-down mapping: if A is a mapping, then B must be also a mapping



(b) Bottom-up mapping: if A is a mapping, then B and C must be also mappings



(c) Sample bottom-up mapping (from (Valiente, 2001)).



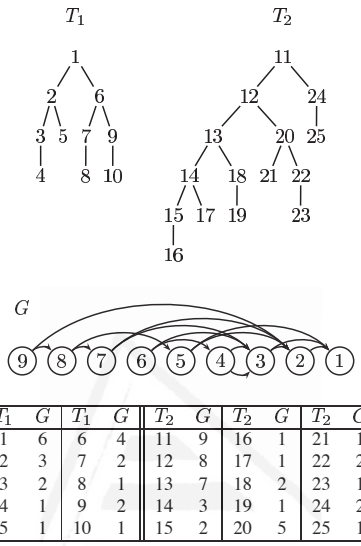
(d) Sample isolated subtree mapping which is neither top-down nor bottom-up (from (Valiente, 2001)).

Figure 3.42: Mapping examples

The bottom-up distance from tree to tree is the cost of a least-cost bottom-up mapping between both trees, i.e. the bottom-up distance is defined in such a way that two nodes v_i and w_i match only when all the children nodes of v_i and w_i match. In order to do so, first a compacted directed acyclic graph representation G of the forest F consisting of the disjoint union of the sets of nodes in T_A and T_B, together with a corresponding map between the nodes of T_A and T_B and the nodes of G (see Fig. 3.43). Then, according to graph G, a mapping M from T_A to T_B is extracted. Finally, the distance between both trees is obtained as the mapping cost defined in Def. 3.5.1.

The algorithm is based on the exact match of node labels. Therefore, the main drawback of this method to compare music is that similarity between the represented concepts on the labels is limited to a boolean decision.

Another disadvantage of this algorithm to compare metrical trees is the hard restriction imposed by the bottom-up mapping to the structure of the compared trees.



T_1	G	T_1	G	T_2	G	T_2	G	T_2	G
1	6	6	4	11	9	16	1	21	1
2	3	7	2	12	8	17	1	22	2
3	2	8	1	13	7	18	2	23	1
4	1	9	2	14	3	19	1	24	2
5	1	10	1	15	2	20	5	25	1

Figure 3.43: Compacted directed acyclic graph representation of two rooted trees (from (Valiente, 2001)).

3.5.6 Partially labelled tree comparison

The presented similarity measures between trees are designed to work with fully labelled trees. In order to apply those algorithms to trees labelled only at the leaves, the non-labelled inner nodes can be assigned a special label "empty". However it is expected that they don't work as well as they do with fully labelled trees.

In order to overcome this situation two approaches are possible. The first one consists of labeling all nodes using any bottom-up propagation scheme based on the application domain specific knowledge . The main drawback to that option is that any intermediate process might add noise to the resulting trees. The second approach is the definition of a similarity function designed just to compare those partially labelled trees.

The *partially labelled tree comparison algorithm* (Rizo and Iñesta, 2010) s_p is based on the assumption that the similarity value between a labelled leaf and a non-labelled inner node should be the average of chances of finding that leaf in the descendants of that inner node. Fig. 3.44a shows the simplest case of having two leaf trees: $s_p(T_A, T_B) = \delta(x, y)$, where $\delta(x, y) = 1 \iff x = y$, and 0 elsewhere. For comparing the trees shown in Fig. 3.44b, the chances of finding the label x in T_B are computed as $s_p(T_A, T_B) = (\delta(x, y) + \delta(x, z))/2$. If instead of being a label, y were another tree, the function should be computed recursively. Finally, when none of the trees is composed by a single leaf

(Fig. 3.44c), the similarity of the ordered forests wx and yz can be computed like an edit distance between sequences wx and yz where each symbol is a tree.

This similarity method omits the accounting of the insertion or deletion of nodes and just measures the chance of finding coincident labels, paying more attention to the information hierarchically contained in the tree than to the tree structure.

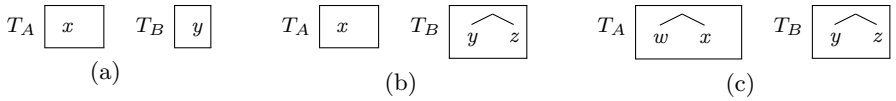


Figure 3.44: Similarity function s_p representative cases

Being designed for working with partially labelled trees, however, we can slightly adapt the original idea to work also with fully labelled trees. The case of comparing a leaf to a non-leaf tree (Fig. 3.45a), is computed as $s_p(T_A, T_B) = (\delta(x, b) + \delta(x, y) + \delta(x, z))/3$. Likewise, the similarity $s_p(T_A, T_B)$ between two fully labelled trees (Fig. 3.45b) is computed as the edit distance between sequences wx and yz, where each symbol is a tree, plus the similarity between labels a and b.

Figure 3.45: Similarity function s_p working with fully labelled trees. Note that the inner nodes (in this case root nodes) labels correspond to one of the children in the metrical trees.

Let $s_p \colon T \times T \to \mathbb{R}$ be a similarity function between trees and $sf_p \colon \mathcal{T}^+ \times \mathcal{T}^+$ be a similarity function between forests. Let us also use rlabel$\colon T \to L$ that returns the label of the root of the tree, and R_m as an abbreviation for rank(T_m). The similarity between two trees, fully or partially labelled, is defined as:

Definition 3.5.7

$$(i)\quad s_p(T_A,T_B)=$$

$$\begin{cases}\delta(\text{rlabel}(T_A),\text{rlabel}(T_B)) & : \textit{if } \text{leaf}(T_A)\wedge\text{leaf}(T_B) \quad (1)\\ \frac{\delta(\text{rlabel}(T_A),\text{rlabel}(T_B))+\sum_{j=1}^{R_B} s_p(T_A,\text{child}_j(T_B))}{1+R_B} & : \textit{if } \text{leaf}(T_A)\wedge\neg\,\text{leaf}(T_B) \quad (2)\\ \frac{\delta(\text{rlabel}(T_A),\text{rlabel}(T_B))+\sum_{i=1}^{R_A} s_p(\text{child}_i(T_A),T_B)}{1+R_A} & : \textit{if } \neg\,\text{leaf}(T_A)\wedge\text{leaf}(T_B) \quad (3)\\ \frac{\delta(\text{rlabel}(T_A),\text{rlabel}(T_B))+sf_p(T_A,T_B)}{\max(R_A,R_B)+1} & : \textit{otherwise} \quad (4)\end{cases}$$

$$(ii)\quad sf_p(\lambda,\lambda)=0$$

$$(iii)\quad sf_p(i..i',\lambda)=sf_p(i..i'-1,\lambda)$$

$$(iv)\quad sf_p(\lambda,j..j')=sf_p(\lambda,j..j'-1)$$

$$(v)\quad sf_p(i..i',j..j')=$$

$$\max\begin{cases}sf_p(i..i'-1,j..j')\\ sf_p(i..i',j..j'-1)\\ sf_p(i..i'-1,j..j'-1)+s_p(T_A[i'],T_B[j'])\end{cases}$$

The simplest situation in Fig. 3.44a is solved by case (i)-(1). Cases (i)-(2) and (i)-(3) solve the problems depicted in Fig. 3.44b and 3.45a. Finally, (i)-(4) computes the similarity for Fig. 3.45b. After comparing the roots, the ordered forests composed by the tree children (Fig. 3.44c) are compared with the similarity function between forests sf_p in the indirect recurrence (ii) to (v).

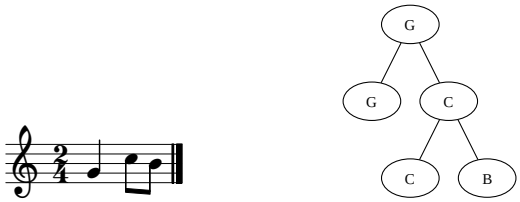
The pseudocode that computes this similarity measure is detailed in appendix B.3.

Complexity of the partial edit distance.

In order to calculate the time complexity of s_p and sf_p, the functions $\mathcal{T}_s$ and $\mathcal{T}_{sf}$ will be used respectively. In both cases the size of the problem is the number of nodes of the compared trees: $(|T_A|,|T_B|)$.

$$\mathcal{T}_s(|T_A|,|T_B|)=\begin{cases}1 & : \quad if\ |T_A|=1\wedge|T_B|=1\\ R_B\times\mathcal{T}_s(|T_A|,|T_B|/\,\text{rank}(T_B)) & : \quad if\ |T_A|=1\wedge|T_B|>1\\ R_A\times\mathcal{T}_s(|T_A|/\,\text{rank}(T_A),|T_B|) & : \quad if\ |T_A|>1\wedge|T_B|=1\\ \mathcal{T}_{sf}(|T_A|,|T_B|) & : \quad if\ |T_A|>1\wedge|T_B|>1\end{cases}$$

The function sf_p can be solved using a dynamic programming scheme as the used for any edit distance. On a classical edit distance, where the substitution cost has constant complexity, given the problem size $(|T_A|,|T_B|)$, the complexity is $O(|T_A|\times|T_B|)$ because its implementation is a simple double loop traversing a $|T_A|\times|T_B|$ matrix. However, in our case, in each step of that iteration, the s_p is called. Under these assumptions, and following the development in appendix B.2, the temporal complexity of the algorithm is $O(|T_A|\times|T_B|)$.



(a) Source incipit (b) Tree representing score in (a)

Figure 3.46: Sample incipit with its tree representation used to illustrate edit operations in next figures.

3.6 Tree edit operations in metric trees

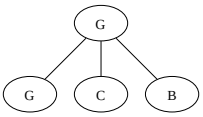
In this section, the meaning that each edit operation has on the score is described. The simplest one is node relabeling (see Fig. 3.47). That operation does not lead to any ambiguity to the resulting tree. However, a relabeling of the root 'G' to 'F' would produce a tree not translatable into a score, because the new 'F' label is not found in any leave and thus cannot represent a valid propagation or reduction of the tree. For these kind of operations, it is important to note that the edit operations are part of a script of editions, so this operation can be just an intermediate operation.

The deletion of an inner node changes the rhythm pattern coded by the source tree. The example of Fig. 3.48 shows how a binary rhythm is translated into a ternary rhythm. The removal of a leaf node can cause the same problem regarding the musical correctness of the resulting tree. Fig. 3.49 shows that the deletion of a leaf node yields an incomplete tree, that may be interpreted as the score in Fig. 3.49b. Again, this operation must be seen as a part of an edit script, as shown in the example of Fig. 3.49c, where that single child is also deleted leaving now a valid tree.

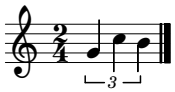
The insertion of nodes makes changes on the structure of the tree, and thus, on the rhythmical pattern it describes. Examples in Fig. 3.50 illustrate this fact.

(a) Tree after relabelling node labelled 'B' as 'A' in tree of Fig. 3.46b (b) Score equivalent to tree in (a)

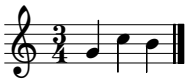
Figure 3.47: Relabel node operation



(a) Tree after deleting inner node labelled 'C' in three of Fig. 3.46b

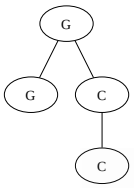


(b) Score equivalent to tree in (a)

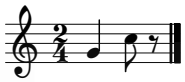


(c) Another score equivalent to tree in (a)

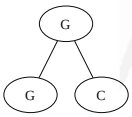
Figure 3.48: Deletion of inner node



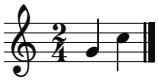
(a) Tree after deleting leaf node labelled 'B' in tree of Fig. 3.46b



(b) Score equivalent to tree in (a). The rest could also be placed before the 8th note.



(c) Tree after deleting leaf node labelled 'C' in tree in (a)



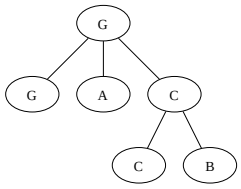
(d) Score equivalent to tree in (c)

Figure 3.49: Delete of leaf node operation

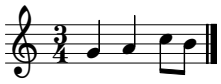
3.7 Tree representation of polyphonic music

The introduced methodology is only designed to deal with monophonic music. If the input works are polyphonic in any of the classifications in Section 2.2 (page 43), two strategies are possible: one is to reduce the input work to monophonic, and the other is to deal with the polyphonic content directly. In this section, the way to represent and compare polyphonic content with trees is described. Our approach is to consider a source as polyphonic if there are simultaneous notes played at a time.

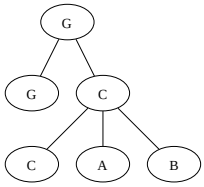
To represent polyphonic music all voices are inserted in the same tree following the rules of the monophonic music representation. Node labels now represent sets of pitch classes. Under this approach, each node will contain all the notes played at a given time (whose depth is conditioned by the shortest one). A node representing only rests has an empty set as the label. Fig. 3.51 contains a polyphonic example and its tree representation.



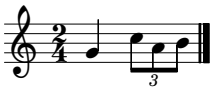
(a) Tree after inserting node labelled 'A' as child of the root in tree of Fig. 3.46b



(b) Score equivalent to tree in (a)



(c) Tree after inserting node labelled 'A' as child of the inner node 'C' Fig. 3.46b



(d) Score equivalent to tree in (c)

Figure 3.50: Insert node operation

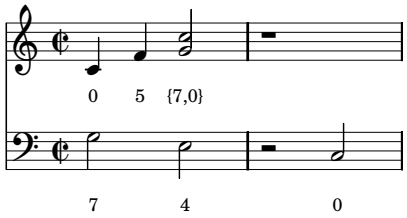
3.7.1 Bottom-up propagation of labels

Once the tree is constructed, a label propagation step is performed. The propagation process is performed recursively in a post-order traversal of the tree. Labels are propagated bottom-up using set algebra. Let $\text{rlabel}(\tau)$ be the label of the root node of the subtree τ expressed as a set of pitches in any of the encodings presented in Section 2.1.1. When the label of the node is a rest, the label is the empty set: $\text{rlabel}(\tau) = \varnothing$. Then, given a subtree τ with $c_i \in \text{children}(\tau)$, the upwards propagation of labels is performed as $\text{rlabel}(\tau) = \bigcup_i \text{rlabel}(c_i)$. The upwards propagation goes until level two, that is, the root representing the whole piece of music always remains empty. The root could also be filled with propagated pitches, however, as the root represents the whole song, in most cases it would contain all pitch classes.

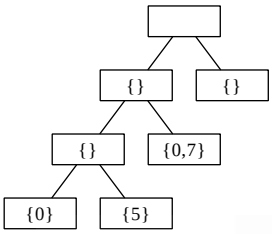
Fig. 3.52 shows the tree in Fig. 3.51 after propagating its labels. In that figure, the half note C (pitch class 0) in the second bar (right-most leaf), is promoted ($\varnothing \cup \{0\} = \{0\}$). In the first bar, the node containing the label $\{0, 5, 7\}$ contained only the quarter note G (pitch class 7) before propagation. The propagation operation merges all pitches in that branch ($\{0\} \cup \{5\} \cup \{7\} = \{0, 5, 7\}$).

3.7.2 Multiset labels

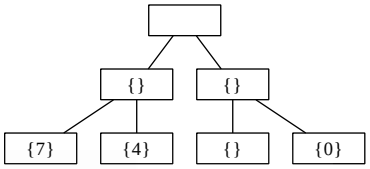
The current polyphonic representation may have a drawback after the propagation step: if the lower levels of the tree contain scales covering a whole octave, the sets of propagated inner nodes would contain all the pitch classes. This way, the inner nodes representing two different musical works would be the same and the comparison methods would always consider both to be similar.



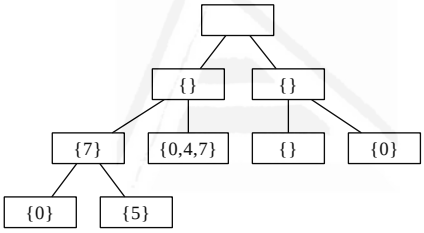
(a) Score to be represented



(b) Tree representing top voice



(c) Tree representing bottom voice



(d) Polyphonic tree representing both voices

Figure 3.51: An example of a polyphonic music and the corresponding tree. Labels belong to pitch alphabet p_{12}. Note that in polyphonic trees empty labels are explicitly represented by the empty set.

To overcome this problem the set label is replaced by a *multiset*, where longer notes have higher cardinality than short ones, i.e., giving lower importance to those pitch classes propagated from deeper levels of the tree.

A *multiset* (aka. a *bag*) is a pair (X, f), where X is an ordered set, and f is a function mapping $f : X \to \mathbb{N}$. For any $x \in X$, $f(x)$ is called the *multiplicity* of x. Using this definition and expressing the values of $f(x)$ as an ordered set, we see that the multiset $\{1, 1, 3\} = (\{1, 3\}, \{2, 1\})$ meaning that $f(1) = 2$ and $f(3) = 1$.

Now, all the node labels in a tree are represented by a multiset (X, f). Once again, we start from the leaves by setting:

$$X = \Sigma_p, \tag{3.3}$$

$$f(p) = \begin{cases} 2^{h-d}/F, \text{ if p} \in \text{rlabel}(\tau) \\ 0, \text{ otherwise} \end{cases} \tag{3.4}$$

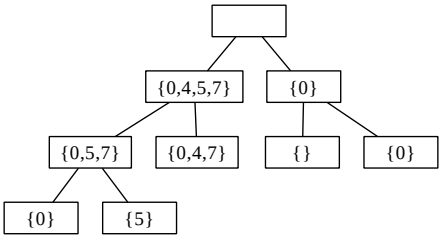


Figure 3.52: Tree of Fig. 3.51 after bottom-up label propagation.

where $d = \text{depth}(v)$ returns the level of the node v in the tree τ, the height $h = \text{h}(\tau)$, and $F = \max_{q \in \Sigma_p} f(q)$.

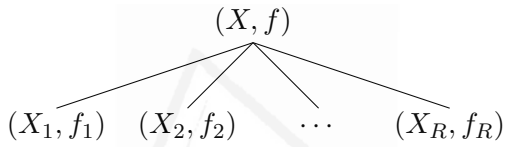


Figure 3.53: Sample subtree

The propagation is performed analogously to that of above, using the multiset union operation instead of set union. Let (X, f) be the multiset contained in the label of the root of the subtree to be propagated, (X_i, f_i) the multiset in the label of the i-th child, and let us also denote R as the rank of the subtree as illustrated in Fig. 3.53. The union of multisets for its propagation that will replace (X, f) by (X', f') as described below in Def. 3.7.1.

Definition 3.7.1

$$X' = \bigcup_{i=1}^{R} X_i \cup X$$

$$f'(x) = f(x) + \sum_{i=1}^{R} (f_i(x)/2)$$

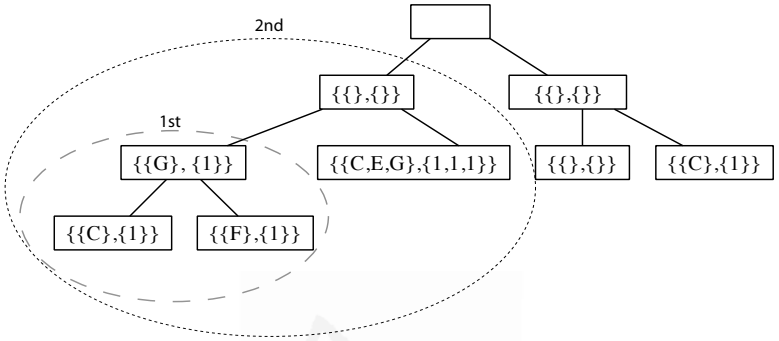
Note that the division by 2 is performed because the level of the children is one less than the root of the subtree.

Fig. 3.54 illustrates this representation. It can be noticed that the note 'F' has a lower weight at the root level of the first bar compared to the other longer notes.

Pruning

Due to the presence of very short notes that eventually will have a low weight in the final node labels, the pruning of the trees from a given pruning level l equivalent to that

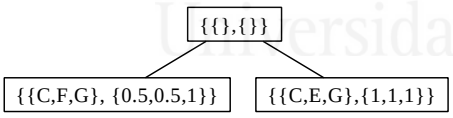
used for monophonic trees has been considered, thus making trees smaller and tree edit algorithms faster.



(a) Multiset label version of the tree in Fig. F.3d using p_{cmn} where the octave has been omitted to simplify the figure. The labels contain multisets (X, f).

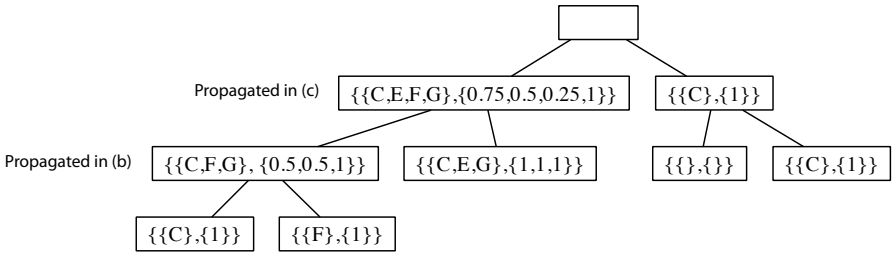
$$f'(G) = f(G) = 1$$
$$f'(C) = f_1(C)/2 = 0.5$$
$$f'(F) = f_2(F)/2 = 0.5$$

(b) Propagation of subtree '1st' in (a)



$$f'(C) = f_1(C)/2 + f_2(C)/2 = 0.25 + 0.5 = 0.75$$
$$f'(E) = f_2(E)/2 = 0.5$$
$$f'(F) = f_1(F)/2 = 0.25$$
$$f'(G) = f_1(G)/2 + f_2(G)/2 = 0.5 + 0.5 = 1$$

(c) Propagation of subtree '2nd' in (a) after having propagated '1st' in (b)



(d) Propagated tree after having propagated all subtrees

Figure 3.54: The first two steps of the propagation of the polyphonic tree in (a) that leads to fully labelled tree in (d) is depicted.

3.7.3 Polyphonic trees comparison

The comparison of polyphonic trees is analogous to that of monophonic trees. However, the symbols contained in the leaves are no longer individual pitches but multisets. This makes it necessary to define the substitution cost for pitch multisets.

Let $\mathcal{M} = (X, f)$ be a multiset that corresponds to a node label. We represent it by using a vector $\mathbf{v}_{\mathcal{M}} \in \mathbb{R}^{|\Sigma_p|}$, such that $\mathbf{v}_{\mathcal{M}}[p] = f(p), \forall p \in X$ according to definition in Equation 3.4. Then, the substitution cost c_{sbn} between two multisets $\mathcal{M}_a = (X_a, f_a)$ and $\mathcal{M}_b = (X_b, f_b)$ is defined as a distance between the corresponding vectors:

$$c_{sbn}(\mathcal{M}_a, \mathcal{M}_b) \triangleq d_{eq}(\mathbf{v}_{\mathcal{M}_a}, \mathbf{v}_{\mathcal{M}_b}) \tag{3.5}$$

For computing the distance d_{eq}, a number of distances between vectors have been tested (Deza and Deza, 2009; Han and Kamber, 2000; Ryu et al., 1998): Manhattan (L1), Cosine similarity, Euclidean distance (L2), Jaccard coefficient^{-1-1}, Log distance, Matching coefficient, Multisets distance, Overlap coefficient, Probabilities, Variational distance, Hellinger distance, Harmonic Mean. They are all described in the appendix A.

Pitch encoding in multisets

The comparison of multisets of size N makes tree distances at least N times slower than the analogous monophonic version. This makes it important to choose a pitch encoding rich enough to allow to represent the merged content after propagation, and on the contrary, of the smallest possible size to avoid slowing too much the comparison process. The p_{ift} pitch encoding seems to be the best choice because it seems to be the one with the best trade-off between richness and size, and it also allows for transposition invariant comparison.

4 Experiments

For this section the following notation will be used. The terms *representation* and *method* will be employed interchangeably for denoting each paradigm, namely strings, trees, n-grams, graphs, etc. All methods but graphs have a number of possible combinations of parameters and algorithms. Some examples of these parameters are the different n-gram sizes, the pitches and duration combinations on string methods, and the tree edit distance with the pitch representation, propagation method, and pruning level for our tree proposal. Each one of this combinations will be denoted interchangeably *setup* or *configuration*.

The experiments have been devised in order to check the fitness of the tree representation proposed for the task of measuring similarity between musical works. As introduced in the previous sections, there are many different setups that can be adopted in order to construct and compare trees. In the experimentation, all possible settings have been explored trying to discover the impact of each component of the system for the task.

As there is not any representative and big enough ground-truth in the research community for comparing the different algorithms available, several corpora, both monophonic and polyphonic, that focus on different aspects of similarity have been collected.

Our proposal has been compared with the systems that may be considered the most representative in state-of-the-art song comparison. For those systems a big number of possible parameter combinations and configurations have been tested, specially, those relative to parameters shared with the tree representations, as it happens with pitch encodings.

Finally, the different parameters and algorithms for constructing and comparing trees have been analyzed.

4.1 Music classification

In order to evaluate the suitability of the similarity measuring methods explored, the following music retrieval system has been designed. Let $\mathcal{G}^N$ be a corpus with a collection of N music files in some symbolic format, such as SMF and MusicXML (Good and Actor, 2003). Following a classical pattern recognition naming convention, these files

are referred as *prototypes*, and each one belongs to a so-called *class* of a set $\mathcal{C}$ of classes. The classification process can be regarded as a mapping $c : \mathcal{G}^N \rightarrow \mathcal{C}$. In our setup, each *class* represents a song, and each *prototype* $p_i \in \mathcal{G}^N$ is a different rendition or variation of the original song. The *class* of a prototype is denoted by $c(p_i)$. The objective of the classification is to know which *class* (song) each *prototype* corresponds to. To that end, a nearest-neighbor scheme has been followed: let $s_\alpha(p_x, p_y)$ be the similarity value between two musical works $p_x, p_y \in \mathcal{G}^N$ $(1 \leq x, y \leq N)$ using the configuration α, the file p_i is classified as $c(p_j)$, where p_j is the work such that $j = \text{argmax}_{k=0}^{N-1}\{s_\alpha(p_i, p_k)\}$.

Instead of a classifier, the system can be also devised as a music retriever one. To that end, for a given song file q or *query*, the similarity measure is computed with respect to all the files in the corpus, and they are sorted in descending order using that measure in a list $L(q)$, being $L_0(q)$ the song with the maximum similarity value, $L_1(q)$ the second one, and so on. Note that the similarity values might be repeated for successive positions in the list. From that point of view, the nearest-neighbor scheme consists in retrieving $c(L_0(q))$.

4.2 Evaluation metrics

Several measures for evaluating the algorithms have been used in order to assess different performance aspects. They are formulated upon the basis of a music retrieval system, and are described just below:

Success rate. Given a set $\mathcal{Q} \subset \mathcal{G}^N$ of query prototypes, and for $q \in \mathcal{Q}$, let $L(q)$ be the resulting list of prototypes retrieved ordered by descending similarity value, the *success rate* is defined as the number of queries for which the most similar prototype belongs the same class as the query. Besides, in order to be considered a successful classification, the resulting similarity value of the first retrieved item must be strictly greater than that value for the second. This is done to avoid giving good classification rates to algorithms that are not able to correctly order the results computing the same similarity value to several prototypes.

Precision. The precision is defined as the rate $\frac{\mid \textit{relevant documents retrieved} \mid}{\mid \textit{documents retrieved} \mid}$. The definition of *relevance* and the *documents retrieved* for our problem is not straightforward, because for each trial, all the documents in the corpus are retrieved, sorted by the similarity value. In order to complement the *success rate* for those cases in which several prototypes have the same similarity value and are located in the first positions of $L(q)$, we consider as the set $\mathcal{D}$ of *documents retrieved* all documents that are returned in the first positions with all the same similarity value: $\mathcal{D} = \{L_k(q) \mid s_\alpha(L_0(q), q) = s_\alpha(L_k(q), q)\}$. The set $\mathcal{R}$ of *relevant documents* for the query q, are those items in the corpus belonging to the same class as the query, $\mathcal{R} = \{q' \in \mathcal{Q} \mid c(q') = c(q)\}$. The set $\mathcal{R}_r$ of *relevant documents retrieved* is composed of those *relevant documents* that are in

the set of *documents retrieved*, $\mathcal{R}_r = \{q' \in \mathcal{D} \mid c(q') = c(q)\}$. Finally, the *Precision (PR)* for a query q is defined as:

$$PR = \frac{\mid \mathcal{R}_r \mid}{\mid \mathcal{D} \mid} \tag{4.1}$$

Precision-at-n. This measure is defined as the number of relevant documents within the first n returned items. In our case, the *relevant documents retrieved* definition is the one expressed for the *precision* measure, i.e., they are those documents belonging to the same class of the query.

Precision-at-$|class|$ or *(PaC)* When the number of prototypes per class is balanced in the corpus, the n in the *precision-at-n* measure can be fixed. However, this is not the case for our corpora, so instead of fixing that n, it is variable and equal to the number of prototypes in the corpus that belong to the class of the query, i.e., $n = |\mathcal{R}|$.

Given the composition of our corpora, there are many cases where for a prototype there are others very similar and their retrieval seems to be easy. The *Precision-at-class (PaC)* measure is the most restrictive and measures not only the correct retrieval of the first slot, but all expected slots.

Recall. The Recall (RC) is utilized in its usual definition: $\frac{|relevant\ documents\ retrieved|}{|relevant\ documents|}$, where the meaning of those terms are those expressed in the *precision* measure:

$$RC = \frac{\mid \mathcal{R}_r \mid}{\mid \mathcal{R} \mid} \tag{4.2}$$

In an ideal case, all the prototypes belonging to the same class of the query would be retrieved with the same highest similarity value. However, it is very difficult that the similarity values coincide, and thus, this value is expected to be very low. High values mean that the prototypes used are actually equal after being encoded with the chosen representation.

f-measure. It is defined as $\frac{2 \times PR \times RC}{PR + RC}$. Here, like in the previous definitions, when the denominator is 0, a 0 is returned as the value of the measure.

Reciprocal rank. The success rate only takes into account the first item returned, and it does not give value to the following positions in the retrieved list. The *Reciprocal rank (RR)* scores all the retrieved documents using the following formula:

$$RR = \frac{1}{\min\{k \mid c(L_k(q)) = c(q)\}} \tag{4.3}$$

When only the mean is reported, this measure is usually referred as *Mean Reciprocal Rank (MRR)*.

Intra-class vs. inter-class measure. This measure is useful for evaluating the ability of the similarity algorithm to give high similarity values to all prototypes in the same class and to give low similarity values to different class prototypes. The higher this value, the better the algorithm is. It was introduced by Bunke and Csirik (1992) to serve as evaluation measure for learning string distance edit costs. Given a corpus, it is defined as the ratio A/B, where A is the average *intra-class* distance, i.e., the average distance among all prototypes of the same class, and B is the average *inter-class* distance, i.e, the average distance among all pair of prototypes belonging to different classes. In order to integrate this measure in our system, it is only necessary to note that we are dealing with similarity values instead of distances. Thus, this measure may be expressed as:

$$II = \frac{A + \epsilon}{B + \epsilon} \tag{4.4}$$

where an ϵ is introduced to avoid divisions by 0. In our implementation, we have used the minimum value of the double Java data type $\epsilon = 2^{-1074}$

Time. Running times are measured in milliseconds taking into account only the test phase, leaving aside the construction of the representations that may be done off-line. All experiments have been performed using a Sun machine with 8 Gb RAM and 8 Intel(R) Xeon(R) CPU X5355 running at 2.66GHz, with a SUSE Linux version with kernel version 2.6.

4.3 Corpora

As stated above, there is not a single way to define music similarity. Thus, two songs can be considered similar if they fall in one of the following three cases:

- Renderings or interpretations of the same original work. It includes both the interpretation of a song with possible mistakes and embellishments, and in the case of polyphonic music, the different distribution of voices and tracks.
- Variations in the classical music form.
- Songs processed from an input source and altered in some way. The polyphonic to monophonic *skyline* process is an example.

In order to take all three cases into account several monophonic and polyphonic corpora has been collected, being each corpus representative of one kind of similarity. Below, these corpora are described. The detail of their contents can be found in appendix C.

These corpora contain two main file encodings: SMF and MusicXML. The only important consideration about the encoding is that in those corpora encoded with MusicXML, the pitch enharmonic spelling is available, not being the case for SMF files.

Monophonic corpora

PASCAL. The *PASCAL* [1] corpus contains different renditions of the same theme of a number of songs. It consists of a set of 420 monophonic 8-12 bar incipits of 20 worldwide well known tunes of different musical genres. For each song, a canonic version were created by writing the score in a musical notation application and exported to MIDI and MP3 formats. The MP3 files has been given to three amateur and two professional musicians who listened to the songs (mainly to identify the part of the tune to be played) and played them with MIDI controllers several times with different embellishments and performance errors. This way, for each of the 20 original scores, 20 different variations were built. This corpus tries to represent different possibilities of how the same song can be played.

VARM. Being our starting point that the hierarchical structure of trees should capture the essence of the themes, even if they are enriched with music variation techniques, this corpus has been built from monophonic themes and variations as written by the composers. It is formed by 9 different themes plus a total of 36 variations of these themes. They are short monophonic sequences of an average of eight measures, all them encoded with MusicXML.

104. This corpus, first collected for the work on metrical trees representations (Rizo and Iñesta, 2002), includes 8 bar incipits of 102 melodies MIDI files downloaded from Internet and processed using a *skyline* process. This melodies correspond to different renderings of 10 well-known tunes of different genres.

MIREX. The *MIREX* corpus corresponds to a published subset of that collected by Typke et al. (2004) and used in the similarity computation contest (Downie, 2006) for training purposes. This ground-truth consists of a set of 11 queries and a 581 document collection, all of them containing around 10 to 40 notes in a range of 3 to 6 bars. For each query, this document collection is ordered according to the similarity of each song to the query. The similarity was obtained by averaging the 35 human experts judgements on the resemblance of each document to the query. Several documents may share the same position in the list. In order to evaluate the quality of a similarity algorithm the authors of this corpus proposed the so-called Average Dynamic Recall (ADR) measure (Typke et al., 2006) that accounts for this ground-truth organization.

Polyphonic corpora

Each of the polyphonic corpus is presented in two formats:

- As it is, to be processed by pure polyphonic comparison methods
- In monophonic format after an *skyline* reduction process

[1] This name is given after its origin, a project developed under an European Union Network of Excellence named PASCAL (Pattern Analysis, Statistical Modeling and Computational Learning).

ICPS. This corpus has 68 MIDI files corresponding to different interpretations of the incipits of seven musical works. They have been played by the author using a MIDI keyboard and, for some songs, an accompaniment with the *Band in a Box* commercial application was added.

VARP. This corpus consists of 85 sequences representing variations of 16 different themes of classical works as written by the composer. In this case, the variations are founded mostly on the harmonic structure of a main theme. The average length of the themes is 8 bars.

INET. It is made of 101 whole MIDI files downloaded from the Internet, corresponding to cover versions of 31 different popular songs. They contain full-length real-time sequences with mistakes and different arrangements of the original versions.

COVERS. Finally, the corpus that has been considered as the target of a realistic application scenario is made up of 2279 MIDI files collected from several personal collections or downloaded from the Internet. It has been named *COVERS* and it is composed of MIDI files with different renderings of 790 full-length pop-rock songs, that have been manually tagged. This corpus has been split into ten balanced folds for performing a cross-validation evaluation.

As noted above in page 119, some of the songs of the corpus are used as queries that are compared to the rest of the songs of that corpus in order to evaluate each similarity method. Two different ways of selecting prototypes to be the queries has been used. In the case of the academic music (the monophonic corpus *VARM* and the polyphonic corpus *VARP*), each class or song has one prototype that the composer named as *theme*, and a set of prototypes that were given the name of *variations*. We have used those *theme* prototypes as queries. For the *PASCAL* corpus, the melody extracted directly from the score is taken as the query, and the different renderings as versions. For the rest of corpora one file is chosen randomly as the query and the other renderings as versions.

4.4 Method parameters and configurations

In the background chapter several methods have been described with a number of possible parameters and configurations. In this section, the different ways of represent and compare monophonic music that have been tested are described. First our trees proposal is detailed, then the other approaches from the literature.

Metric tree configurations

Trees can be built and compared in 1680 different ways. This is the product of the combination of all factors described in Section 3.3:

pitch: The eight different pitch encodings introduced in Section 2.1.1, namely: p_{abs}, p_{21}, p_{40}, p_c, p_{12}, p_{hdc}, p_{int}, and p_{ift}.

pruning level: The tree pruning explained above in Section 3.4 has been explored from level $l = 1$ that leads to trees containing only one node for each bar, to $l = 6$ containing the maximum resolution used in this dissertation that is $1/2^6$ of a bar, leading to a 64th note for a $\frac{4}{4}$ meter.

propagation method: Introduced also in Section 3.4, the explored propagation methods are *None*, *Partial*, *Heuristic*, *Melodic*, *Left*, and *Right*.

distance: The different tree similarity measurement algorithms described in Section 3.5 are: *Align*, *Partial*, *Selkow*, *Shasha*, and *Valiente*. In all cases the result of the distance has been normalized using the maximum of the number of nodes of both compared trees.

Although some of these factors are parameters for the construction of the trees, and others are edit distance algorithms, in the sequel they all will be considered as parameters for simplicity of the exposition.

Polyphonic tree configurations

pruning level: This is equivalent to that found in monophonic metrical trees.

multiset distance: The different multiset distances introduced in Section 3.7.3: *Manhattan* (*L1*), *Cosine similarity*, *Euclidean distance* (*L2*), *Jaccard coefficient*$^{-1}$, *Log distance*, *Matching coefficient*, *Multisets distance*, *Overlap coefficient*, *Probabilities*, *Variational distance*, *Hellinger distance*, and *Harmonic Mean*.

edit distance: For the polyphonic trees only the Selkow edit distance has been used because it is the only one with a practicable performing time, and our proposed partial distance cannot be applied because it compares the equality of the labels of the nodes, operation that does not make sense for multisets.

Rest of parameters

A number of the similarity measurement methods detailed in the introduction (Section 2.1) have been implemented in our framework, in some cases from an available implementation [2], and in other cases from the description found in the respective publications.

As detailed in the introduction, all those methods have parameters or configurations that must be set to perform the actual comparisons. With the objective to find the best setup for each method, all possible configurations have been tested, that are listed just below.

[2]Thanks to Iman Suyoto for the *fanimae* implementation of the n-grams methods, Kjell Lemström for the *C-Brahms* geometrical algorithms, and Alberto Pinto for his graph approach implementation

String edit distances (S). Among all the explored methods, maybe the closest one to our proposal is the string edit distance approach. Thus, this has been the method that most exhaustively has been explored. All parameters about pitch and rhythm have been tested that result in 2240 different setups:

pitch : The eight different pitch encodings introduced in Section 2.1.1: p_{abs}, p_{21}, p_{40}, p_c, p_{12}, p_{hdc}, p_{int}, and p_{ift}. Besides, an special pitch is used when only the rhythm is compared.

rhythm : The seven duration and time encodings detailed in Section 2.1.2: r_{dabs}, r_{tabs}, r_{acc}, r_c, r_{hdc}, r_{ioi}, and r_{ior}. The special rhythm symbol is used when only the pitch information is considered.

k : It is the linear combination constant detailed in Definition 2.1.22, and has been tested with values $k \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$.

coupling : The coupling of rhythm and pitch (Section 2.1.4) has been tested in the two possible setups: *coupled* and *decoupled*. Note that parameter k, when applied to *decoupled* configuration, instead of combine the partial results of pitch and duration, it serves as weighting constant of pitch or rhythm symbols.

rests : The inclusion (*withRests*) or ellision (*withoutRests*) introduced in Section 2.1.4 has been also tested.

NHT : One of the contributions of the trees approach is the use of tonal information in order to discriminate notes. This information has been also used in string representation (Section 2.1.4), leading to two possible configurations: *withNHT* leaves all notes, and *withoutNHT* removes all notes that are not harmonic tones, simulating this way the promotion of notes in the tree representation.

Mongeau-Sankoff (MS). This an interesting variant of the string edit distance that adds fragmentation and consolidations to the edit operations as detailed in Section 2.1.4.

The possible 97 setups of this distance are similar to those used for classical string edit distances:

pitch: Only p_{abs}, p_c, and p_{itv} have been used.

duration : r_{dabs} and r_{ioi}.

k : $\{0.1, 0.5, 0.9\}$.

coupling : Only *coupled* representations.

rests, NHT : exactly as the classical string edit distances.

pitch comparison : the pitch comparison has been performed in two possible ways: *matrixPitch* that compares the equality of the compared pitches in p_{itv} encoding, and *Sankoff*, that uses the weights proposed in the original paper (Mongeau and Sankoff, 1990) for comparing pitches.

C-Brahms. For the geometric methods (Section 2.1.6) the C-Brahms system (Lemström et al., 2003) has been used. All the different comparison and indexing techniques *P1*, *P2*, *P2v5*, *P2v6* (Lemström et al., 2008), and *P3* have been tested.

PROMS. Another geometric-like system that only uses onsets and text-retrieval techniques is PROMS (Section 2.2.4).

In the original algorithm, the approximate search is accomplished by allowing k dissimilarities as a maximum between the query and the database document. To our needs, where whole pieces of music are to be compared, the original algorithm has been modified to return the normalized number of coincidences in the best alignment. The different setups of the PROMS establish the quantization r per bar. Values from 4 (meaning a resolution of a quarter note for a 4/4 meter) to 28 have been tested in our experiments. The PROMS representation is very sensitive to quantization. This quantization produces chords with closer notes. The quantization of the polyphonic scores leads to too dense chords, and as r raises, it gets less dense.

n-grams and edit distances by Uitdenboderg. The n-grams approach used (Section 2.1.5) follows the approach by (Suyoto and Uitdenbogerd, 2005). We have implemented the algorithms proposed in that paper and present in the *fanimae* sofware: the *n-gram counting algorithm*, using sizes $n \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$, and some string edit distances proposed in that paper: the *simple local alignment* as described in page 31, the *start match alignment*, and finally the string edit distance using the *directed-modulo-12* pitch representation (see p_{dm} in Section 2.1.1) with cost 1 for insertion and deletions, and 2 for substitutions.

Graphs (G). The graph approach (Section 2.1.8) does not have any parameter to be set.

All methods have been implemented in Java to be integrated in our system, except for C-Brahms, for which an adaptation layer has been created to be plugged into our system. We have followed a strict object oriented approach that enables the system to be easily extended with new representation approaches, comparison algorithms, and retrieval schemes.

4.5 Best result for each corpus

Recall we denote as *setup* a possible combination of factors or parameters of each method, i.e., a combination of prunning level, pitch, tree distance, propagation strategy.

All possible setups for each method in each corpus have been tested generating a huge amount of results. In order to have a first insight on the behavior of each method, its best setup in each corpus has been selected using as sorting criteria the precision-at-$|class|$ that is the most restrictive measure. These results are detailed in Table 4.1.

This table is valid for having a first overall view of the performance of each method on each kind of similarity represented by each corpus. However, there are some important drawbacks that arise if we want to give an averaged global result from these figures:

- The three corpora used are not class balanced
- They represent different aspects of similarity
- The differences shown may not be significant
- The setup that has achieved the best result for each method in each corpus is different
- Time, although not included in the table for space reasons, varies from setup to setup in such a way that the best setup may be very slow while the second best setup may be very fast

The next section describes a more complex experiment design that enables us to extract global conclusions while overcoming those drawbacks.

4.6 Experiment setup and results

The experiments in both the monophonic and polyphonic case have been designed in the same way.

The experiments are two folded. On one hand, the objective is to study the tree representation approach for the music similarity computation task. On the other hand, to put in context our proposal by comparing the results of the tree approach to those of other representations. For those two purposes four steps were followed:

1. The first step was to select those setups that achieved a good trade-off between time and accuracy that perform robust for all considered kinds of musical similarity represented in the different corpora.
2. Having selected a list of adequate configurations of each method in the first step, they were used both in a realistic scenario that is represented by the *COVERS* and the *MIREX* corpora.
3. From the previous results, one of the selected setups is chosen to be the finest configuration of the method.
4. Finally, after giving the results of our proposal compared with other state-of-the-art methods, a more in-detail analysis of each factor of tree construction and comparison is done.

Each one of these steps is explained in more details next.

Table 4.1: Best results for each method and corpus in terms of precision-at-$|class|$. The heading of the table has been abbreviated to fit in the page: 'S' stands for Success rate, 'PR' for precision, 'PaC' is precision-at-$|class|$, 'RC' abbreviates recall, 'F' is f-measure, 'M' for mean reciprocal rank (MRR), finally 'I' stands for intra-class vs. inter-class ratio.

Method	**104**							**PASCAL**							**VARM**						
Monophonic	S	M	PaC	PR	RC	F	I	S	M	PaC	PR	RC	F	I	S	M	PaC	PR	RC	F	I
Metric trees	0.8	**0.9**	0.5	**0.9**	**0.2**	0.3	**0.1**	**1.0**	**1.0**	0.8	**1.0**	**0.1**	0.1	**0.0**	0.8	0.9	**0.8**	0.8	0.4	0.5	**0.1**
PROMS	0.6	0.8	0.4	0.7	**0.2**	0.2	0.4	0.9	0.9	0.4	0.9	0.0	0.1	0.1	**1.0**	**1.0**	0.7	**1.0**	**0.5**	**0.6**	0.3
Strings	0.7	**0.9**	**0.6**	**0.9**	**0.2**	0.3	**0.1**	0.8	**1.0**	0.8	**1.0**	**0.1**	0.1	**0.0**	**1.0**	**1.0**	**0.8**	**1.0**	**0.5**	**0.6**	**0.1**
M.Sankoff	**0.9**	**0.9**	**0.6**	**0.9**	0.1	0.2	**0.1**	**1.0**	**1.0**	0.5	**1.0**	**0.1**	0.1	**0.0**	**1.0**	**1.0**	**0.8**	**1.0**	**0.5**	**0.6**	**0.1**
***n*-grams**	0.4	**0.9**	**0.6**	**0.9**	**0.2**	**0.4**	1.0	0.7	**1.0**	**0.9**	**1.0**	**0.1**	0.1	**0.0**	0.6	0.7	0.5	0.6	0.3	0.3	**0.1**
C-BRAHMS	0.6	0.7	0.3	0.7	0.1	0.2	1.0	0.5	0.6	0.2	0.6	0.0	0.1	0.1	0.8	0.9	0.5	0.8	0.3	0.4	0.4
Graphs	0.3	0.6	0.3	0.6	**0.2**	0.2	**0.1**	0.5	**1.0**	0.2	0.9	**0.1**	**0.2**	**0.0**	0.0	0.2	0.1	0.0	0.0	0.0	**0.1**

	INET							**ICPS**							**VARP**						
Polyphonic	S	M	PaC	PR	RC	F	I	S	M	PaC	PR	RC	F	I	S	M	PaC	PR	RC	F	I
Trees	**0.5**	0.6	0.6	0.4	0.5	0.7	**0.0**	**0.9**	**0.9**	**0.7**	**0.1**	**0.2**	**0.9**	**0.1**	0.6	0.6	**0.6**	**0.4**	0.4	0.7	**0.0**
PROMS	**0.5**	**0.7**	**0.7**	**0.5**	**0.6**	**0.8**	0.1	0.6	0.6	0.3	**0.1**	0.1	0.7	0.2	**0.8**	**0.8**	**0.6**	**0.4**	**0.5**	**0.8**	0.3
C-BRAHMS	**0.5**	0.5	0.5	0.4	0.4	0.6	2.4	0.7	0.8	0.3	**0.1**	**0.2**	**0.9**	0.3	0.7	0.7	**0.6**	**0.4**	**0.5**	0.7	0.3

4.6.1 Method setups selection

The objective of this step is to decide which setups of each method perform the best in average for all kinds of similarity reflected in each corpus, while taking into account the class unbalance of some of the corpora used, the different types of corpora content, and processing times. The basis of the process for selecting good setups are:

1. The use of the precision-at-$|class|$ as the retrieval quality measure because it is the one that takes more advantage of the results obtained for each query, being as well the one that shows more differences among methods. The success rate and the reciprocal rank only consider the first correct returned item for each query. Instead, the precision-at-$|class|$ is measuring all correct items returned within the first positions. The number of considered retrieved items has a strong implication on how the results can be analyzed. Consider each query retrieval result as a sample of a statistical population. The success rate returns only values 0 or 1. The precision-at-$|class|$ returns a full range between 0 and 1 instead, that allows for a finer statistical analysis of the results.

2. The comparison of results among different setups of a method in a given corpus is performed by using statistical significance tests.

3. A voting algorithm is used to combine the results of each setup in all corpora to rank the setups while overcoming the unbalance of corpora and their different nature.

Let α_i be a setup or configuration of a method. The process that ranks all possible setups of a method is summarized in Fig. 4.1 and is composed of the following steps:

1. For each setup α_i and each corpus $\mathcal{G}_j$ compute the average and standard deviation of the precision-at-$|class|$ (abbreviated as $PaC(\alpha_i, \mathcal{G}_j)$) for all queries in that $\mathcal{G}_j$.

2. Given the population of results of all setups, it is ordered descending in terms of mean of $PaC(\alpha_i, \mathcal{G}_j)$, grouping the setups whose precisions-at-$|class|$ differences are not statistically significant. This step produces a rank of setups for each corpus, where the setups with non significant different performances are in the same slot of the ranking. For testing significance of groups, non-parametrical tests have been used: Kruskal-Wallis (Kruskal and Wallis, 1952) test for groups of size greater than two, and Mann-Whitney (Mann and Whitney, 1947) for pairs of samples, using a typical setup $p < 0.05$. The non-parametrical tests have been chosen instead of the ANOVA (Anscombe, 1948) test because the later makes the assumption on the normality and equality or homogeneity of variances (homoscedasticity) that we cannot assume.

3. Combine the different ranks obtained of each setup in all corpora by the use of a *Borda count* voting method of group decision theory (Ho et al., 1994). This combination outputs a score for each setup that indicates how well each setup has

performed in comparison to other setups in all corpora. It is important to note that this score gives a ranking of performances for all setups in a method, and its scope is restricted to that method. It has not sense to compare the *Borda count* among methods because a method can be consistently bad with all setups, and however, the best among all its setups will have a high *Borda count*. The *Borda count* score is computed as follows: for each corpus, give a score 1 to the setups in the first slot, 1/2 for those in the second slot, 1/3 for the third, $\cdots$, $1/s$ for the s-th ranking slot. Note that if two setups are in the first slot, for the third setup occupying the second slot the 1/3 score is given. Then, the *Borda count* score of each setup corresponds to the accumulated scores obtained for all corpora.

4. Having the precision-at-$|class|$, processing times, and *Borda count* scores, we have enough information to select as the most suitable setups those having a good trade-off among these three measures.

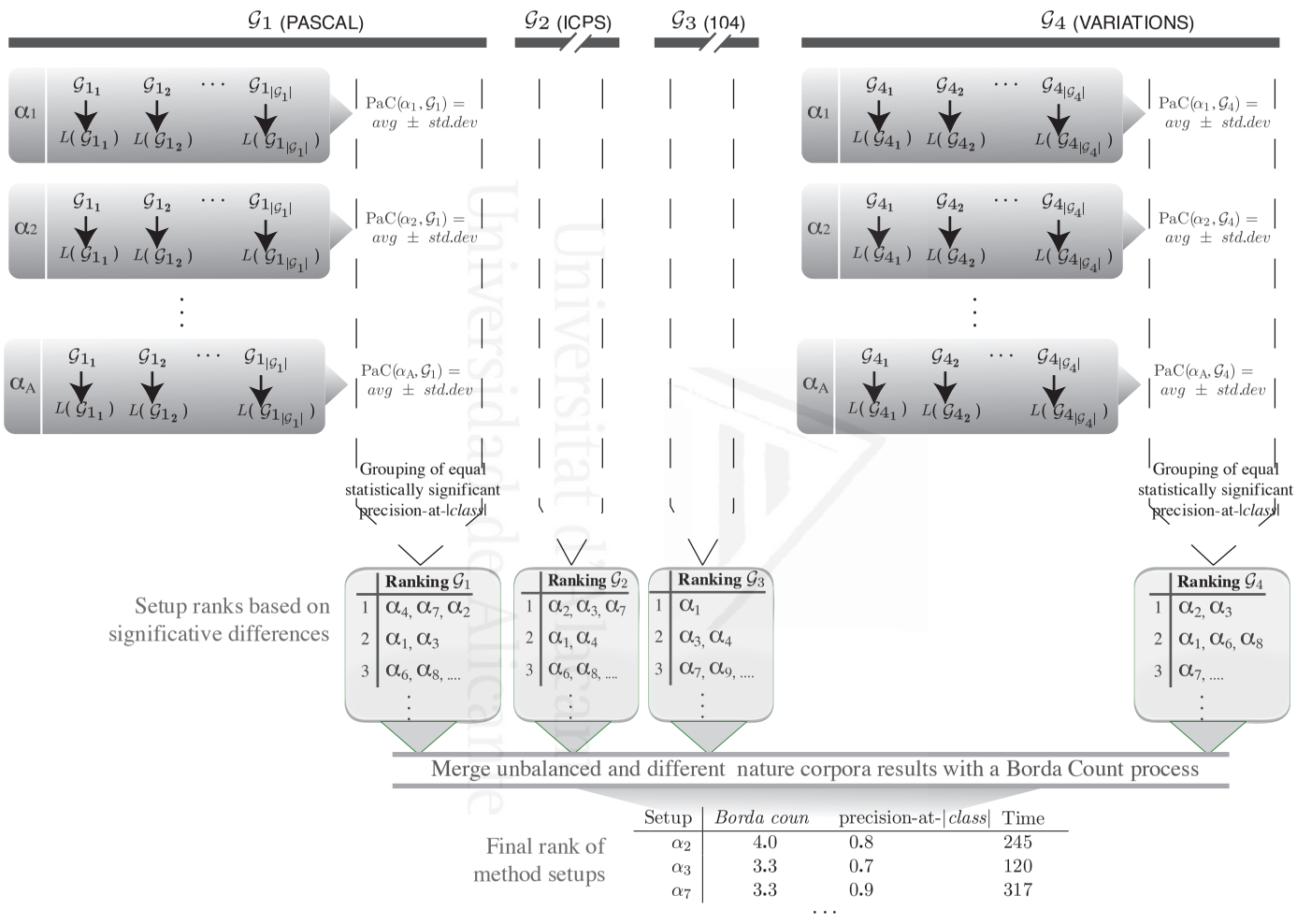


Figure 4.1: Experimental process to find the best setup α_i for each method.

Selection of setups of monophonic methods

The corpora *PASCAL*, *104*, *VARM* have been used to perform the experiments with monophonic methods. Besides, an skyline version of *ICPS* has been used in order to evaluate the behavior of the methods with a corpus that it is expected to contain a lot of noise. Recall that *VARM* contains theme and variations as written by the composer, *104* are melodies downloaded from internet, and *PASCAL* and *ICPS* are different renderings of the same song extract.

The plot in Fig. 4.2 shows graphically the behavior of the different setups of methods. Note that the Graph method has not been plot because it only has a possible setup and we don't have the necessity of selecting the best one. A *Borda count* threshold has been established empirically to remove setups that do not behave consistently among the different corpora and keep enough setups to follow with the next experiment step.

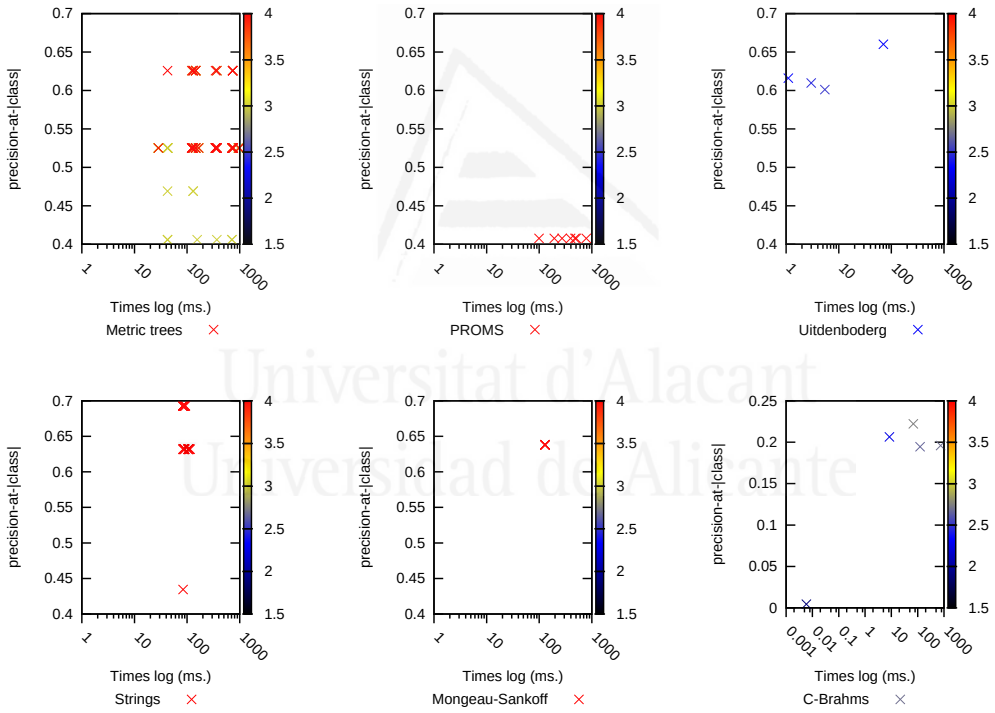


Figure 4.2: Precision-at-|*class*| and times trade-off for each method. The color shows the *Borda count* score. Recall the *Borda count* only indicates consistence among results in different corpora. Note the C-Brahms plot does not follow the same ordinate axis as the rest of methods.

The full list of results can be found in appendix E.1. Among all these setups, only those having a processing time faster than 200 ms and having the best *Borda count* have been selected for being applied to the realistic scenario experiment. The methods *Metric trees*, *String edit distances*, *PROMS*, and *Mongueau-Sankoff* reach in several setups the

Table 4.2: Method selected setups count

Method	Total setups	Selected setups
Trees	1680	57
Strings	2240	61
Mongeau-Sankoff	97	7
Uitdenboderg	14	4
PROMS	6	2
C-Brahms	5	4
Graphs	1	1

maximum *Borda count* score 4.0, that means that these setups are among the best in the four used corpora. Only 16 setups of trees reach the *Borda count* score 4.0, so we have included also the setups with at least a *Borda count* 3.0. In the case of *Uitdenboderg* and *C-Brahms*, as this maximum value has not been obtained by any setup, only the time and precision-at-$|class|$ criteria has been followed to select setups. This decision reduces considerably the number of setups of each method as shown in Table 4.2.

The plot in Fig. 4.2 shows that the string based methods are the ones reporting best precision-at-$|class|$, while following the *Borda count*, they are consistent along the corpora. The proposed metric tree methods yield the best trade-off of consistence with precision-at-$|class|$ and processing times. The most surprising result has been that of the 2-grams, that while not being very consistent along corpora, have been extremely fast with a good precision-at-$|class|$.

Selection of setups of polyphonic methods

The same methodology has been applied for the selection of the polyphonic methods. Plots in Fig. 4.3 show the trade-off between processing times and precision-at-$|class|$. The detail of all setups can be found in appendix E.2. In the polyphonic case, having less possible setups, all values with a non-zero *Borda count* have been included in the appendix.

The plot in Fig. 4.3 shows that the best *Borda count* is obtained by the C-Brahms method, however, at a low precision-at-$|class|$ rate. For the trees and the PROMS methods, the most robust (best *Borda count*) are not the fastest setups, but a good compromise among time, precision-at-$|class|$, and *Borda count* can be easily spotted in the plots. The reported *Borda count* values are lower than those found in the monophonic experiments. This is due to the fact that the precision-at-$|class|$ obtained by the different setups are now in most cases significant, and thus, the best ranked setup uses not to be shared among setups.

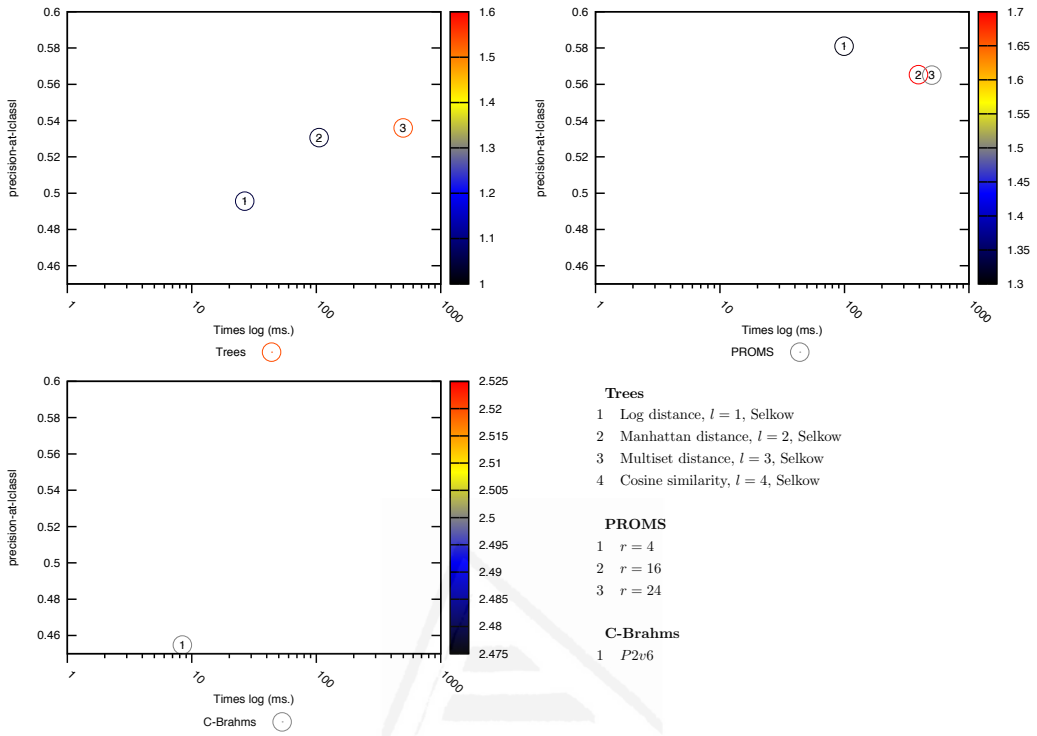


Figure 4.3: Precision-at-$|class|$ and times trade-off for each polyphonic method. The color shows the *Borda count* score.

4.6.2 Application to realistic scenarios

The datasets utilized for testing the feasibility of the approach were designed to test different aspects of music variability. The corpora *104*, *PASCAL*, *VARM*, *VARP*, *INET*, *ICPS* were small song sets, aiming to reach conclusions on the system architecture without spending large computation times in carrying out the experiments. Now, we try to apply the lessons learned from those experiments in more realistic scenarios.

In this set of experiments, the MRR has been used instead of the precision-at-$|class|$ because now we want to measure the performance of each method from the point of view of the the retrieval system user.

First using as a corpus the *COVERS* dataset, whose songs have been applied the skyline algorithm in order to apply the monophonic methods. However, it is important to note that this corpus does not contain all kinds of similarity that the small corpora represent. Ten balanced subsets have been constructed for performing a ten-fold cross-validation evaluation. The queries-dataset retrieval scheme used previously has been used for each fold. The results are obtained as the means and sample standard-deviations of the different quality measures described in Section 4.2 for the 10 sub-experiments. Only the setups of each method considered "suitable" after the previous experiment have been tested. Note that the previous *Borda count* methodology employed in that

experiment has not been applied now because, on the contrary as before, the different folds of *COVERS* are balanced and contain homogeneous content.

The *MIREX* corpus results has been reported because it is publicly available and these results could be useful for other researchers to compare their results with ours, keeping in mind that this is not the corpus used for the MIREX contest, but only the training dataset.

Monophonic results on *COVERS* corpus

The selected setups from the previous section have been applied to a *skyline* version of the *COVERS* corpus. The result of the best setup in average of each method is detailed in Table 4.3. All values are averaged from the ten folds of the corpus. For each method, the setup with best trade-off MRR vs. time has been plotted in Fig. 4.4. The full detail of results has been placed in appendix, Table E.10.

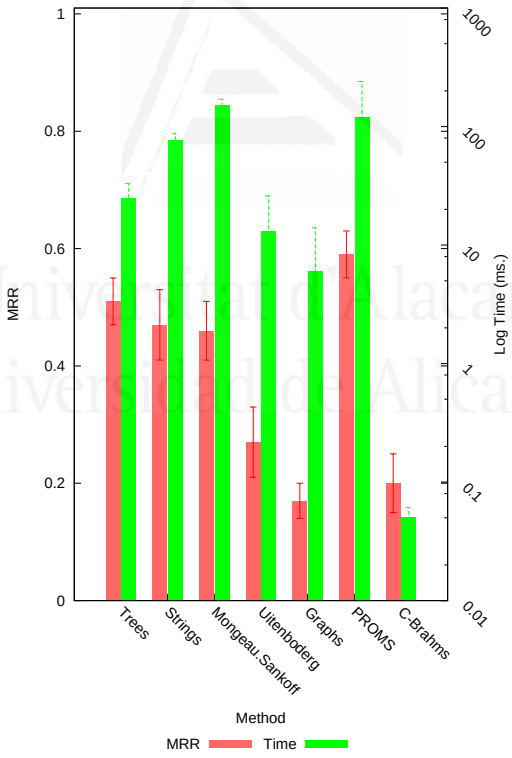


Figure 4.4: Skyline *COVERS* corpus results for selected setups.

The method that has performed the best in terms of retrieval quality has been PROMS, the fastest methods have been C-Brahms, Uitdenboderg, and Graphs, and the one with the best trade-off between retrieval quality and processing time is the proposed

Table 4.3: Results for skyline *COVERS* corpus with monophonic methods. All values are averaged from the ten folds of the corpus.

Method (and setups)	MRR	Precision-at-$\|class\|$	Time	Success	Precision	Recall	Intraclass vs. interclass
Metric trees: p_{hdc}, $l = 2$, Heuristic prop., Partial similarity							
	0.51 ± 0.04	0.37 ± 0.03	**25 ± 8**	0.43 ± 0.04	0.43 ± 0.03	0.30 ± 0.03	0.0145 ± 0.0010
CBRAHMS: P2v6							
	0.20 ± 0.05	0.11 ± 0.03	**0.051 ± 0.010**	0.14 ± 0.04	0.15 ± 0.05	0.11 ± 0.05	1.4 ± 1.1
Graphs							
	0.17 ± 0.03	0.08 ± 0.03	**6 ± 8**	0.09 ± 0.03	0.09 ± 0.03	0.06 ± 0.02	0.0097 ± 0.0011
PROMS: $r = 4$							
	0.59 ± 0.04	0.44 ± 0.05	120 ± 120	0.53 ± 0.05	0.53 ± 0.05	0.36 ± 0.05	0.031 ± 0.003
Strings: p_{hdc}, $k = 0.9$, r_{tabs}, *decoupled*, *noRests*, *withNHT*							
	0.47 ± 0.06	0.34 ± 0.06	77 ± 10	0.43 ± 0.07	0.43 ± 0.07	0.28 ± 0.05	0.0118 ± 0.0007
Mongueau & Sankoff: p_{itv}, $k = 0.1$, r_{dabs}, *matrixPitch*, *noRests*, *withNHT*							
	0.46 ± 0.05	0.33 ± 0.06	150 ± 20	0.42 ± 0.06	0.42 ± 0.06	0.27 ± 0.06	0.0104 ± 0.0008
Uitdenboderg: $4 - gram$							
	0.27 ± 0.06	0.16 ± 0.04	**13 ± 13**	0.19 ± 0.05	0.19 ± 0.05	0.13 ± 0.04	0.029 ± 0.004

metric tree approach, using our proposed tree partial similarity algorithm, with *heuristic* propagation, normalization, pruning at level $l = 2$ and *high-definition contour* pitch p_{hdc}.

From the full listing of results in the appendix, other setups of trees are similar in performance, but this one seems to be the fastest one with best performance among all tree configurations.

Polyphonic results on *COVERS* corpus

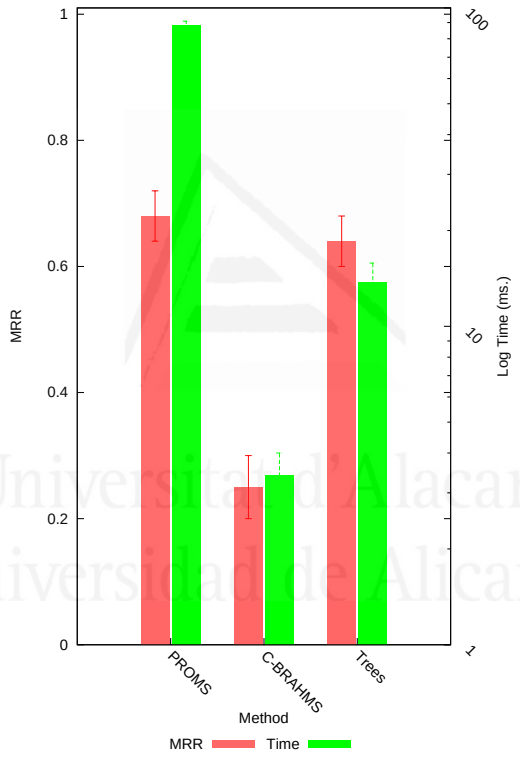


Figure 4.5: *COVERS* corpus results for selected polyphonic setups.

The polyphonic methods have been tested with the original polyphonic *COVERS* corpus, divided into ten folds, without any *skyline* process applied. Selected results are plotted in Fig. 4.5 and detailed in Table 4.4, and the full detail can be found in appendix E.4.

The best performance has been obtained again by PROMS in terms of MRR. However, the rest of quality measures report comparable results between PROMS and the trees approach, being the latter the one with best trade-off between performance and time. The C-Brahms has reported the best processing times.

It is very interesting to look at the intraclass vs. interclass value. The trees are the ones that most separate the distances of the songs belonging to different classes from those belonging to the same class.

Monophonic results on *MIREX* corpus

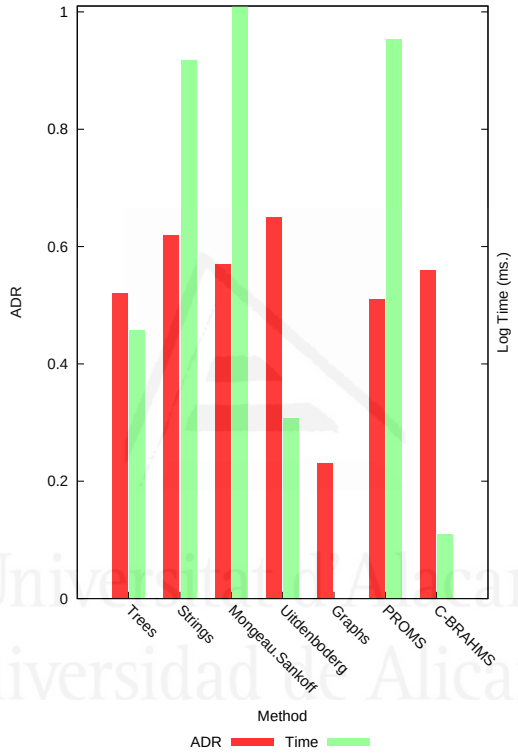


Figure 4.6: *MIREX* corpus results for selected setups

The *MIREX* corpus, used for training in that contest in its 2005 edition, has been tested with the selected setups of methods. Results are plotted in Fig. 4.6 and detailed in appendix E.5, page 195. For each method, the setups with best trade-off between ADR and time have been plotted (recall that the quality measure used for this corpus is ADR that accounts for this ground-truth organization).

Note that these results cannot be compared with the published results of that contest because we are working only with the training set part of it. However, we include it here because that training set is publicly available and can be useful for comparison with other methods not included in this manuscript. One thing to be considered about this corpus is that what is evaluated is the similarity of incipits as perceived by a group of people that judged their resemblance, and not the different versions or covers of songs

Table 4.4: Results for *COVERS* corpus with polyphonic methods.

Method (and setups)	MRR	Precision-at-$\lvert class \rvert$	Time	Success	Precision	Recall	Intraclass vs. interclass
CBRAHMS: *P2v6*							
	0.25 ± 0.05	0.15 ± 0.04	3.4 ± 0.6	0.19 ± 0.05	0.19 ± 0.05	0.13 ± 0.04	0.4 ± 0.4
PROMS: $r = 4$							
	0.68 ± 0.04	0.56 ± 0.04	87.9 ± 3.0	0.64 ± 0.05	0.65 ± 0.05	0.45 ± 0.05	0.029 ± 0.002
Polyphonic trees: *Log multiset distance,* $l = 1$							
	0.64 ± 0.04	0.54 ± 0.03	13.7 ± 2.1	0.62 ± 0.04	0.62 ± 0.03	0.43 ± 0.03	0.0089 ± 0.0009

Table 4.5: Example of different *Borda count* rankings

Sorted list of results	*Borda count*	*Nauru* variant
0.9	1	1
0.9	1	1
0.7	1/3	1/2
0.7	1/3	1/2
0.5	1/5	1/3
0.3	1/6	1/4
0.3	1/6	1/4
0.1	1/8	1/5

that are explicitly the same work with different renderings or variations, as it happens with the result of corpora used in this dissertation.

In this small dataset, the best method has been the Uitdenboderg approach with its 4-grams approach. The strings methods reported best ADR than our trees approach, however, the trees method obtains the second best trade-off between ADR and processing time.

4.6.3 Best setup selection

After having analyzed a number of possible setups for each method running on different kind corpora, to conclude, it is desirable to select only one configuration for each method as the best one. For doing this selection, all results on all corpora will be used to elaborate a ranking of setups. The MRR and time will be the measures employed for deciding the best setups, because they are the measures most perceived by an end-user.

In order to give a ranking of the setups of methods that have reported the best results in terms of MRR in average in all corpora, a variant *Borda count* known as *Nauru* has been used. In this variant, repeated items do not decrease the ranking score of the following ones (see Table 4.5). Note in that table, that the result ranked third, has given a 1/3 value in the classical *Borda count* configuration because it has been ranked in the third position, and it has been given a 1/2 in the *Nauru* variant because repeated results do not decrease the subsequent positions. This variant has been used for comparing the results of all methods because not all methods have the same number of setups, and thus, the comparison using the classical *Borda count* configuration would not be fair. Thus, the setup that has obtained the best *Borda count* in its *Nauru* variant can be considered the best in average for all corpora.

The criteria to select a setup as the best one will be its robustness measured with the *Narnu Borda count* variant, and the processing times.

Monophonic results

The plot in Fig. 4.7 describes the performance of the setups of each method in all monophonic corpora and the behavior in average based on the *Borda count*. Detailed results for the monophonic methods are listed in appendix, table E.17 in page 200.

The proposed tree approach is the one that has more setups in the top positions of that ranking. Besides, it is the most robust for all corpora and the one with the best trade-off between MRR and time. The results of Uitdenboderg are comparable to those of trees in terms of time and *Borda count*. Some methods, like strings or PROMS, that had given a good performance for a given setup in some corpus, have no setup performing as well for all corpora. Finally, the fastest methods have been the C-Brahms and Graphs having, however, the worst performances.

The selected setups of monophonic methods are:

Metric trees p_c, *heuristic* propagation, and Selkow tree edit distance with pruning level $l = 2$.

Strings p_{itv}, $k = 0.9$, r_{tabs}, decoupled, removing rests and leaving non-harmonic tones.

Mongeau-Sankoff p_{ift}, $k = 0.1$, r_{ioi}, *matrixPitch*, removing rests and leaving non-harmonic tones.

Uitdenboderg String edit distance using the *directed-modulo-12* pitch representation. The best n-gram configuration has been the 4-grams setup.

PROMS $r = 8$ or $r = 4$.

C-Brahms $P2v6$.

Graphs Pinto approach that is the only one used in this manuscript.

Figure 4.7: Best precision-at-$|class|$ and times for each method.

Polyphonic results

Plot in Fig. 4.8 shows the performance of best setups of all methods on the polyphonic corpora, and the details can be found in appendix, Table E.18. In the polyphonic domain, the trees approach has had no setup robust enough for giving the best results on all corpora, even having some configurations that had given the best results for individual corpora. In any case, the trade-off of time and MRR in average behaves consistently. The best polyphonic setup for method each is:

Trees Harmonic mean, with $l = 2$ because it has the best trade-off among all evaluated corpora and time.

PROMS $r = 8$ or $r = 4$.

C-Brahms $P2v6$, having a *Borda count* a bit worse than $P2v5$, it is much faster.

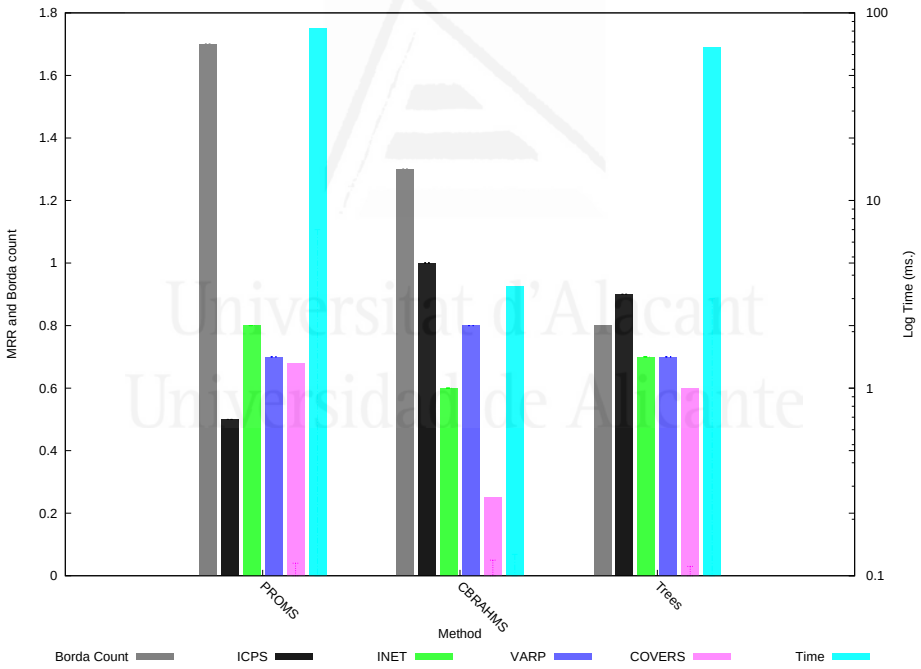


Figure 4.8: Best precision-at-$|class|$ and times for each method.

Table 4.6: Preprocessing times.

Method	Average preprocessing times per song	
	Monophonic	**Polyphonic**
PROMS	0.2 ± 0.1 ms	0.5 ± 0.4 ms
Graphs	0.6 ± 0.1 ms	
CBrahms	3.6 ± 0.2 ms	4.1 ± 0.4 ms
Uitdenboderg	7.0 ± 0.8 ms	
Strings	7.9 ± 6.8 ms	
Trees	293.9 ± 2.3 ms	37.2 ± 13.9 ms

4.6.4 Tree construction and comparison parameters analysis

The last phase of the experiments is to analyze in-depth the behavior of each way of construct and compare trees in order to extract conclusions about the suitability of each one by themselves alone or in combination in order to look for interesting synergies among parameters.

Besides studying the suitability for the music comparison task, the performance of the proposed partially labelled tree distance has been studied.

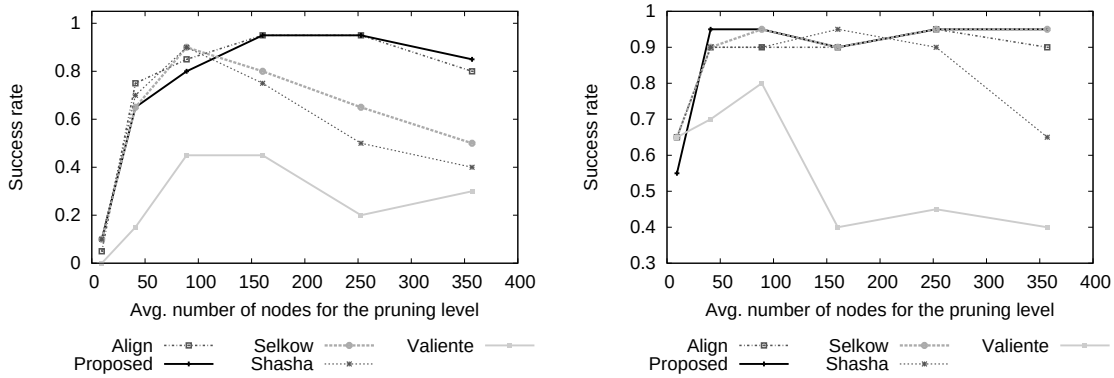
Preprocessing times

Being the skyline version of the *COVERS* corpus the one with more realistic input files, it has been chosen to study the preprocessing times not included in the previous steps of the experiment (see Table 4.6). These times must be considered only for the first reading of the corpus used, and as the internal encodings of any representation can be saved in text files (we have implemented this facility), and the fact that none of the construction times are bigger than a second, the corpus encoding times can be ignored for a running end-user application. Anyway, this can be considered the main drawback of our proposal that should be improved in the future.

Partial edit distance performance

One of the contributions of this work is the partially labelled tree distance. In order to evaluate its performance, the corpus *PASCAL* has been used, feeding all algorithms with several versions of the trees that have been pruned from level $l = 1$ to $l = 6$ (which is the maximum depth found in the corpus), and using as fixed parameters the pitch encoding p_{ift} and the melodic propagation [3]. For each pruning level, the average number of nodes has been extracted to be used as x-axis in the plots.

[3]We have selected this corpus and parameters for having been used in previous published works by the author (Bernabeu et al., 2009; Habrard et al., 2008)

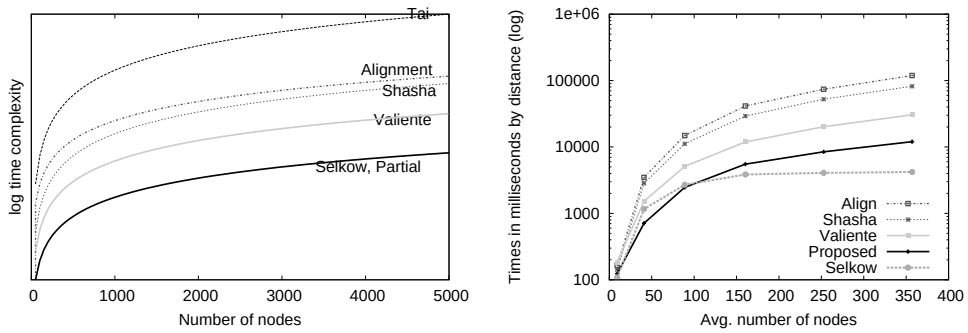


(a) Partially labelled trees (b) Fully labelled trees

Figure 4.9: Success rates of proposed method compared to classical tree edit distances.

The results plotted in Fig. 4.9a show that the proposed algorithm behaves the best for non-pruned trees or $l = 6$ (see results for $x > 350$). For pruned trees, it behaves also the best in average.

When working with fully labelled trees (see Fig. 4.9b), the success rates of the proposed method are comparable to the success rates of the Selkow and the alignment distance. This alignment distance corresponds to our implementation proposed in page 105. The original implementation of Jiang (Jiang et al., 1995) led to intractable times with songs longer than four bars. The plot in Fig. 4.10a shows the theoretical evolution of computing times of the main tree edit distance algorithms given their time complexity, and Fig. 4.10b describes the experimental processing times of those distances in our experiments. The actual times confirm the prediction from the theoretical complexities, being our proposed algorithm the second fastest one.



(a) Complexities theoretical processing times (b) Actual processing times

Figure 4.10: Time processing evolution of algorithm.

Thus, it seems that the proposed similarity is suitable for its purpose, being able to compare successfully both trees labelled only at leaves and fully labelled trees better than the other methods in terms of trade-off between time and success rate.

Monophonic trees

The previous sections have shown that the proposed approach reports good trade-off in terms of retrieval quality and processing times. In this section, an analysis of the impact of each parameter of the tree method is performed.

Table 4.7 shows the average behavior of the different parameters in terms of precision-at-$|class|$ for the different monophonic corpora and ALL representing the merged results. The values of mean and standard deviations have been obtained from the different precision-at-$|class|$ for each query of each corpus. The dispersion in the results show that no parameter can be considered by itself as the best without being combined with others. However, it seems clear that the tree distance by Valiente is not suitable for the task.

All the possible combinations of parameters in groups of two and three for each of the monophonic corpora have been visually analyzed [4]. In order to let the reader have a summary of all tables not included in the manuscript for its excessive length, Fig. 4.11 shows graphically the values contained in those tables. A slight improvement on the performance in terms of have precision-at-$|class|$ has been detected in pruning levels from $l = 2$ to $l = 4$, and the best trade-off between time and performance is located in $l = 2$. The propagation methods *None* and *Partial* lead in general worse results, but for the corpus *ICPS* in its *skyline* version, that has a lot of noise coming from the polyphonic to monophonic reduction, and it does not work well with any propagation method based in any musicological assumption based on the metrical position of notes.

Regarding the processing times, neither the pitch encoding or the propagation method alter the comparison algorithm times, as it was expected. The parameters that do have a deep impact on time are those of pruning level and the similarity or distance algorithms, as shown above in the review of the performance of the partial edit distance performance.

Apart from these conclusions, looking at the different combinations of parameters in the plot, only the inadequate combinations can be detected, i.e., all those painted with cool colors (blues and greens).

Polyphonic trees

The polyphonic trees have less parameters that can be combined, i.e., only the multiset distance used to compare the labels of the nodes, and the pruning level. Fig. 4.12 describes the MRR of all combinations between multiset distance and pruning level for all polyphonic corpora. The plot only shows that there are two multiset distances, *1/Jaccard* and the *Helliger* distance that consistently perform the worst for all cases. For

[4] The document with the details of each possible combination of parameters can be downloaded from http://www.dlsi.ua.es/gent/drizo/thesis/metrictreeanalysis.pdf

Table 4.7: Metric trees precision-at-$|class|$: all parameters, all corpora

	PASCAL	104	VARM	ICPS	ALL
Pitch					
p_{abs}	0.32 ± 0.08	0.34 ± 0.07	0.52 ± 0.14	0.29 ± 0.08	0.37 ± 0.13
p_{21}	0.40 ± 0.11	0.38 ± 0.09	0.52 ± 0.13	0.32 ± 0.08	0.40 ± 0.13
p_{40}	0.40 ± 0.11	0.38 ± 0.09	0.52 ± 0.13	0.32 ± 0.08	0.40 ± 0.13
p_c	0.4 ± 0.2	0.34 ± 0.13	0.4 ± 0.2	0.40 ± 0.11	0.4 ± 0.2
p_{12}	0.39 ± 0.12	0.38 ± 0.08	0.47 ± 0.14	0.32 ± 0.07	0.39 ± 0.12
p_{hdc}	0.4 ± 0.2	0.37 ± 0.14	0.4 ± 0.2	0.38 ± 0.10	0.4 ± 0.2
p_{int}	0.4 ± 0.2	0.36 ± 0.11	0.4 ± 0.2	0.39 ± 0.09	0.4 ± 0.2
p_{ift}	0.5 ± 0.2	0.40 ± 0.08	0.43 ± 0.13	0.42 ± 0.11	0.45 ± 0.14
Prunning level					
$l = 1$	0.3 ± 0.2	0.31 ± 0.09	0.4 ± 0.2	0.33 ± 0.09	0.34 ± 0.13
$l = 2$	0.5 ± 0.2	0.40 ± 0.08	0.50 ± 0.14	0.37 ± 0.12	0.43 ± 0.14
$l = 3$	0.5 ± 0.2	0.40 ± 0.11	0.5 ± 0.2	0.38 ± 0.10	0.44 ± 0.14
$l = 4$	0.4 ± 0.2	0.38 ± 0.10	0.46 ± 0.14	0.38 ± 0.10	0.42 ± 0.14
$l = 5$	0.4 ± 0.2	0.36 ± 0.10	0.47 ± 0.14	0.34 ± 0.09	0.39 ± 0.14
$l = 6$	0.4 ± 0.2	0.35 ± 0.09	0.5 ± 0.2	0.33 ± 0.09	0.38 ± 0.14
Propagation					
None	0.29 ± 0.13	0.31 ± 0.10	0.4 ± 0.2	0.39 ± 0.11	0.36 ± 0.14
Partial	0.3 ± 0.2	0.30 ± 0.09	0.4 ± 0.2	0.35 ± 0.10	0.33 ± 0.13
Heuristic	0.5 ± 0.2	0.41 ± 0.09	0.5 ± 0.2	0.34 ± 0.10	0.43 ± 0.14
Melodic	0.5 ± 0.2	0.39 ± 0.09	0.5 ± 0.2	0.35 ± 0.10	0.4 ± 0.2
Left	0.5 ± 0.2	0.39 ± 0.09	0.50 ± 0.14	0.34 ± 0.09	0.43 ± 0.14
Right	0.42 ± 0.14	0.39 ± 0.09	0.40 ± 0.14	0.37 ± 0.10	0.39 ± 0.12
Distance					
Align	0.44 ± 0.14	0.39 ± 0.07	0.46 ± 0.11	0.39 ± 0.09	0.42 ± 0.11
Partial	0.4 ± 0.2	0.37 ± 0.09	0.51 ± 0.13	0.39 ± 0.11	0.43 ± 0.13
Selkow	0.5 ± 0.2	0.42 ± 0.08	0.55 ± 0.12	0.40 ± 0.09	0.47 ± 0.13
Shasha	0.4 ± 0.2	0.41 ± 0.08	0.47 ± 0.13	0.33 ± 0.10	0.41 ± 0.14
Valiente	0.26 ± 0.12	0.25 ± 0.08	0.29 ± 0.13	0.26 ± 0.05	0.27 ± 0.10

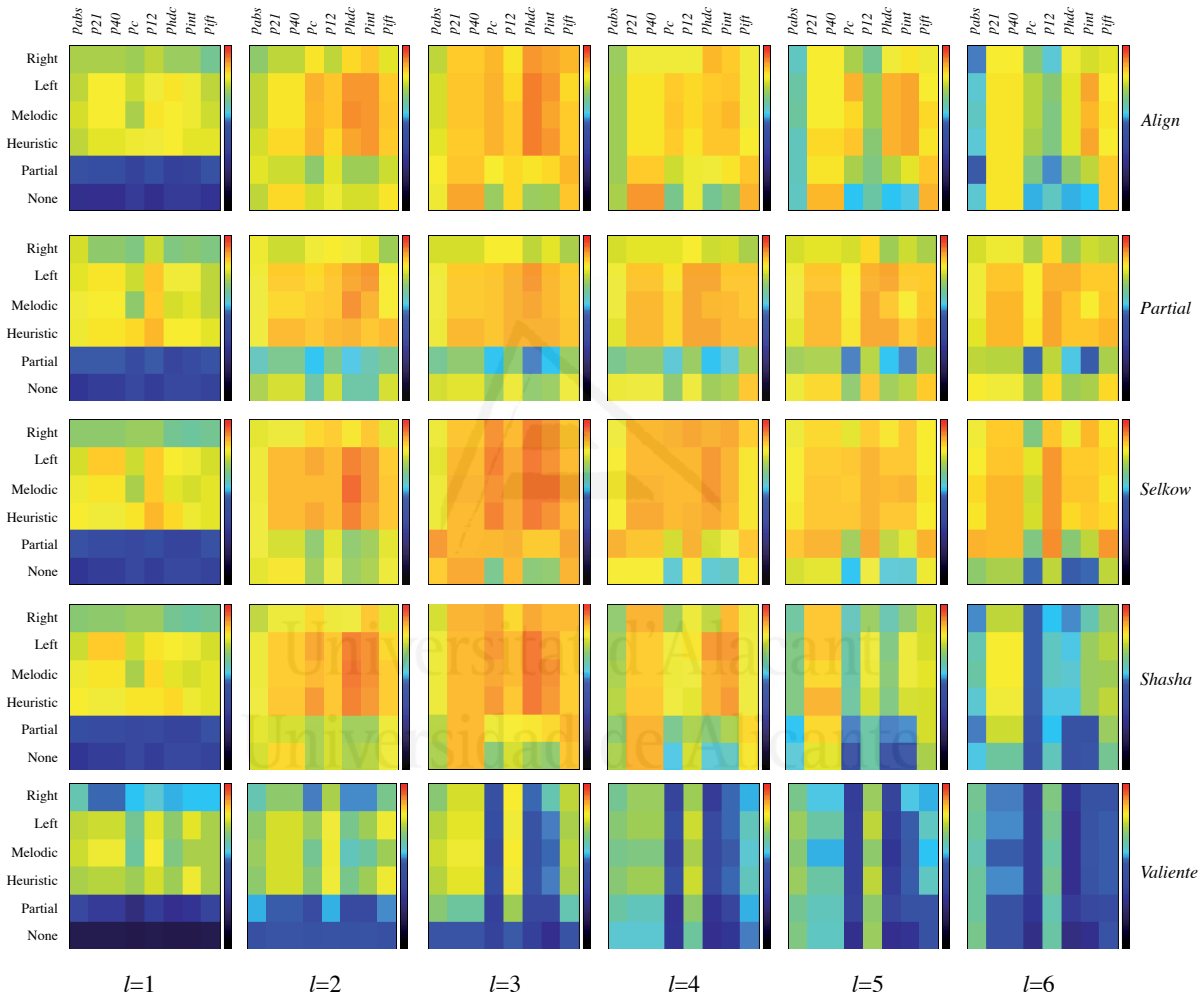


Figure 4.11: Average precision-at-$|class|$ plotted as heat maps for all monophonic corpora in all possible combination of parameter. The warmer the color (red, orange) the best precision-at-$|class|$.

the other cases no distance, pruning level, or any combination of both can be considered as the best for the three corpora.

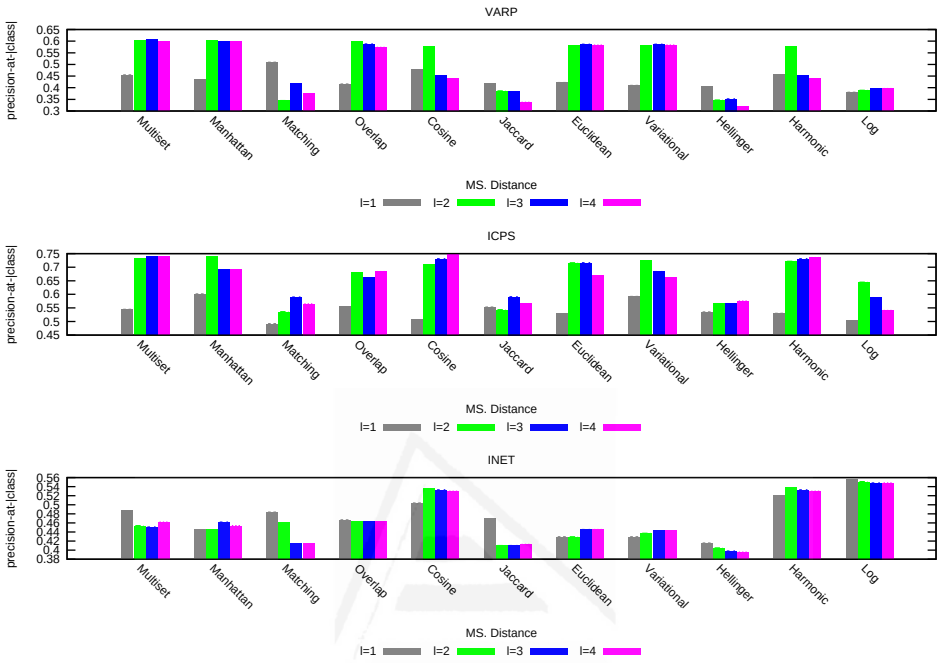


Figure 4.12: Multiset distance vs. pruning level for the polyphonic corpora

5

Conclusions, contributions, and future work

This dissertation has introduced a tree representation of music encoded in symbolic format for similarity computation, and it has been shown to report a good trade-off between comparison time and accuracy among the presented methods in the explored corpora.

In order to frame our proposal into the related literature, both the current state of the art on music similarity measurement, and the different successful uses of the abstract data type tree for symbolic music processing in other domains, have been explored.

Let us recall the hypothesis introduced in the beginning of the manuscript:

> The abstract data type tree seems to be the most suitable for capturing the temporal and hierarchical structure, and it is adequate for representing the reduction of a work. Therefore, trees seem to be an adequate data structure to encode and process music in symbolic format for similarity computation.

The experiments presented in the previous chapter seem to support that hypothesis. In the following paragraphs, each of its key concepts, and some other contributions of this thesis, will be discussed.

··· suitable for capturing the temporal and hierarchical structure

The proposed representation has shown to be able to represent the rhythmic information of a musical work in an implicit way based on the metrical structure of common music period works.

The main advantages over linear structures are the simplicity on representing both monophonic and polyphonic music in the same kind of structure, and the versatility of that structure to fit more elaborated information as the musical form or harmony. Although in a preliminary stage, it should be noted that some experiments that include harmonic information in the labels of the tree, point that this kind of information can improve the representation power of trees supported by better accuracy in similarity computation [1].

[1] the results raise in *VARM* corpus until a 100% success rate

From the experimental tuning point of view, the trees approach has shown to need less parameters to fit than its linear equivalent, the string representations.

The proposed tree structure has been used successfully for other tasks. For example, tonality guessing following a bottom-up traversal of polyphonic trees (Rizo et al., 2006b). This method has been used in this thesis to obtain the tonality for those MIDI files whose key signature was not present (required for pitch encodings as p_{ift} and to perform a simple pitch spelling for encodings as p_{21} and p_{40}), and from the results it can be deduced that this process has performed consistent enough for the task it was designed. For example, the best setups for the strings in the monophonic *COVERS* corpus used p_{21} and p_{40} encodings, or the best Mongeau and Sankoff setup used the p_{ift} pitch encoding. Another application was automatic composition (Espí et al., 2007), where the metrical trees are used as the way of representing music in symbolic format in a genetic system, which uses subtree interchanging as crossover operation.

Our approach has two main drawbacks: its tight dependency on the meter structure of the input source, and its difficulty to represent ties, dots, and syncopations. The first problem can be overcome through an a priori metrical analysis of the work (Eck and Casagrande, 2005; Meudic, 2002b). In any case, the experiments show that metrical information is important for similarity computation: the best setups in the experiments contained metrical information, PROMS uses bar information in the representation, and strings have reported its best performance using absolute time as rhythm encoding. The second drawback, from the representation point of view, has been solved by the addition of a special symbol that encodes the concept of note continuation. For the comparison task, it is not a problem as other authors point out (Hanna et al., 2008; Mongeau and Sankoff, 1990; Pardo and Sanghi, 2005).

One derived problem that comes from the fixed metrical structure is the excessive growth of trees when very little notes and performance imprecisions are found. This problem has been addressed by the proposed propagation processes. However, this is an open research direction, maybe by applying advanced quantization algorithms (Agon et al., 1994; Cemgil et al., 2000), or by converting the fixed metric tree structure into a more flexible one, able to respect someway the meter, beat and tactum structure, and capable of working both with monophonic and polyphonic music.

The last aspect to be improved from the representation point of view is the substitution of the current root grouping of the trees representing bars by a hierarchical structure representing the architecture of the musical work.

··· ready for representing the reduction of a work

Being a process originally designed to fully label the tree from the leaves, the bottom-up propagation of the labels in the leaves maybe has become the most powerful tool among all the proposals of this dissertation. The results show that any of the propagations guided by the metrical structure is valid for comparison task. In other case, after pruning the trees at the highest levels, the performance rates would decrease. The results also

show that for corpora with little noise [2] the music theory can help in deciding which notes are more important to describe the essence of a melody.

After having pruned trees until level two, for the implementation of a real work retrieval system, the root sequence could be kept and used as a thumbnail of the song. Thus, a fast indexing system can be built on top of the thumbnail of the songs in the database.

However, this tool still has to be refined by including more musical knowledge or by constructing systems able to learn the reduction technique either from examples or by the use of an optimization algorithm. Furthermore, the polyphonic propagation proposed is somewhat simplistic and should incorporate the learnings of the monophonic case. The main difficulty for achieving this objective is the musical analysis preprocess required to apply the propagation rules to the polyphonic content. The way the propagation in the tree is performed is a key issue, because it determines the way the song is summarized. Some experiments using different weights for the node label propagation in the tree, depending on a number of factors (harmonicity, position in the tree, etc.) are currently being explored. For example, besides using in the multisets the cardinality based on the accumulated duration of notes, some tests have been done on using the Krumhansl tonality vector (Krumhansl, 1990) as proposed in (Müllensiefen and Frieler, 2004b), that, so far, have given no improvements.

The use of genre information, as that obtained from methods in whose creation the author has collaborated (Pérez-Sancho et al., 2009), may be a source of data that can be used to improve the propagation quality.

Finally, one alternative to tackle the polyphonic case is to represent a tree for each voice and combine the partial results for comparison. However, voice separation is an open problem (Rafailidis et al., 2008).

Comparison algorithms

The comparison of trees has been performed using a sort of tree distances and a new similarity measure proposed in this thesis. Only one tree distance among all the explored methods has demonstrated to be inadequate for our task: the Valiente bottom-up distance. The reason of its poor results must be explained by the structure of the compared trees it imposes. Our new proposed implementation of the Jiang alignment distance has performed as expected, with times similar to the Shasha and Zhang distance and making it practicable. The other distances have shown to behave correctly for the similarity computation task, being the Selkow distance the best one. One common issue for all algorithms, is that the normalization of distances has improved significantly the results. A new similarity measure to compare partially labelled trees has been proposed yielding results comparable, and sometimes better, to those of the existing methods, reporting among the best computation times. Currently, we are beginning to apply this new algorithm can be applied to other tasks like the comparison of Quad Trees (Finkel and Bentley, 1974) with promising results.

[2] *VARM* and *VARP* corpora

Some other tree distances may be applied, and maybe the processing times can be improved, but from the results obtained it does not seem to have room for better results with other kind of distances. Maybe, where more research has to be done is on the label comparison. One thing to be done is to add the propagation information to weight the label substitution cost.

In this dissertation no fine tuning of edit costs has been presented. The author shows in (Habrard et al., 2008) that results can be substantially improved by the application of cost fitting methods like genetic algorithms, or by the use of stochastic distances in both strings and tree methods, able to learn from a training set the cost matrices. In that paper the *PASCAL* corpus was divided into three folds for cross-validation, and represented with p_{ift} pitch encoding with strings without rhythm, and trees using melodic propagation, no pruning, and Selkow edit distance. An important improvement of accuracy was obtained using both genetic algorithms and stochastic methods, showing besides, that the costs learned with the stochastic approach were able to explain the most common changes between the original theme and the different renderings of it.

The pre-filtering of the nodes in polyphonic trees before comparing, and the use of an adaptation of our proposed partial tree distance to deal with multi-sets is another research direction that should be explored. A similar approach that removes *non interesting* notes in the MIDI file can be found in (Madsen et al., 2008).

Another work area that is opened is the application of the tree comparison techniques that are applied now to our metrical and polyphonic trees, to the other tree representations explored in the state of the art on music tree representation section. In his work, Alan Marsden expresses the need of an objective application in order to check the quality of the Shenkerian analysis. The similarity computation is a good candidate for that.

In order to implement a large-size retrieval system, the simple 1-NN approach used to evaluate our system is not practicable. Rizo et al. (2006c) proposed a new approximate nearest neighbor search for non-vector representation of patterns (metrical trees) to speed up the classification.

Finally, as not taken into account for evaluating our approach, it is worth to note that the current implementation of our tree construction process, is too slow and should be improved to build a real retrieval system by debugging the current implementation.

Combination of methods

We have performed an exhaustive exploration of the parameter space for some of the principal methods of similarity measurement of symbolically encoded music, leading to a total of 4088 setups, summing up all configurations of the explored methods. The combination of them is a promising way to definitely reach the best possible results. The author has explored (Rizo et al., 2008, 2009a,b) the way of selecting the best ensemble of setups for the polyphonic similarity task based on choosing the most diverse methods with best retrieval quality rate.

For the processing of polyphonic music with monophonic methods that can be later integrated in those ensembles, the used skyline algorithm can be replaced with a more

sophisticated preprocess: to extract the melody track following the statistical methods proposed by the authors in (Rizo et al., 2006a).

Corpora and results analysis

The majority of works that evaluate music similarity only use a single corpus, that may have a biased representation of what music similarity is. Furthermore, in some cases as MIREX, contains a subjectively evaluated ranking of prototypes. We propose a different way of evaluating the different comparison paradigms. On one hand we have gathered 7 different corpora, both monophonic and polyphonic, representing different ways of conceiving what music similarity is. On the other, a methodology to extract conclusions on the performance of several algorithms on different nature and size corpora has been proposed.

Publications

The majority of the content of this manuscript has been previously published previously, in a more reduced form, in conference proceedings, journals and book chapters:

1. Rizo D. and Iñesta, J,M. "New partially labelled tree similarity measure: a case study. " Structural, Syntactic, and Statistical Pattern Recognition. Lecture Notes In Computer Science, vol. 6218, pp. 296–305 (2010)

2. Rizo, D. and Lemström, K. and Iñesta, J.M. "Tree representation in combined polyphonic music comparison" Computer Music Modeling and Retrieval. Genesis of Meaning in Sound and Music. Lecture Notes in Computer Science, vol. 5493, pp. 177–195 (2009)

3. Rizo, D. and Lemström, K. and Iñesta, J.M. "Ensemble of state-of-the-art methods for polyphonic music comparison" Proceedings of the Workshop on Exploring Musical Information Spaces, ECDL 2009, ISBN: 978-84-692-6082-1, pp. 46–51, Corfu, Greece (2009)

4. Habrard, A.; Iñesta, J.M.; Rizo, D.; Sebban, M. "Melody recognition with learned edit distances" Structural, Syntactic, and Statistical Pattern Recognition. Lecture Notes in Computer Science, vol. 5342, pp. 86-96 (2008)

5. Rizo, D., Lemström, K., Iñesta, J.M. "Tree structured and combined methods for comparing metered polyphonic music" Proc. Computer Music Modeling and Retrieval 2008 (CMMR'08), ISBN: 978-87-7606-027-5, pp. 263–278, Copenhagen, Denmark (2008)

6. Rizo, D.; Iñesta, J.M.; Ponce de León, P.J. "Tree Model of Symbolic Music for Tonality Guessing" Proc. of the IASTED Int. Conf. on Artificial Intelligence and Applications, AIA 2006, ISBN: 0-88986-404-7, pp. 299-304, Innsbruck, Austria (2006)

7. Rizo, D.; Moreno-Seco, F.; Iñesta, J.M.; Micó, L. "Efficient search with tree-edit distance for melody recognition" Pattern Recognition: Progress, Directions and Applications, pp. 218-244 (2006)

8. Rizo D., Iñesta J.M., Ponce de León, P.J., Pertusa A. "Reconocimiento automático de la tonalidad en partituras digitales" Proc. I Workshop sobre Reconocimiento de Formas y Análisis de Imágenes, CEDI 2005, pp. 53-60 (2005)

9. Rizo, D., Iñesta, J.M. "Tree Symbolic Music Representation for Key Finding" 1st Annual Music Information Retrieval Evaluation eXchange (MIREX 2005), held in conjunction with ISMIR 2005, London, UK (2005)

10. D. Rizo, F. Moreno-Seco, J.M. Iñesta "Tree-structured representation of musical information" Lecture Notes in Computer Science - Lecture Notes in Artificial Intelligence, vol. 2652, pp. 838-846 (2003)

11. Rizo, D.; Iñesta-Quereda, J.M. "Tree-structured representation of melodies for comparison and retrieval" Proceedings of International Workshop on Pattern Recognition in the Information Society, PRIS 2002, pp. 155 (2002)

Besides, during the elaboration of this thesis, the author has worked in other music information retrieval areas, collaborating in the following publications:

1. Pérez-Sancho, C.; Rizo, D.; Iñesta, J.M.; Ponce de León, P.J.; Kersten, S.; Ramirez, R. "Genre classification of music by tonal harmony" Intelligent Data Analysis, vol. in press (2010)

2. Bernabeu, J.F., Calera-Rubio, J., Iñesta, J.M., Rizo, D. "A probabilistic approach to melodic similarity" Proceedings of MML 2009, pp. 48-53 (2009)

3. Pérez-García, T.; Iñesta, J.M.; Rizo, D. "metamidi: a tool for automatic metadata extraction from MIDI files" Proceedings of the Workshop on Exploring Musical Information Spaces, ECDL 2009, ISBN: 978-84-692-6082-1, pp. 36–40, Corfu, Greece (2009)

4. Pérez-Sancho, C.; Rizo, D; Iñesta, J.M. "Genre classification using chords and stochastic language models" Connection Science, vol. 21, pp. 145-159 (2009)

5. Illescas, P.R., Rizo, D., Iñesta, J.M. "Learning to analyse tonal music" Proc. Int. Workshop on Machine Learning and Music, MML 2008, pp. 25-26, Helsinki, Finland (2008)

6. Pérez-Sancho, C.; Rizo, D.; Iñesta, J.M. "Stochastic text models for music categorization" Lecture Notes in Computer Science, vol. 5342, pp. 55-64 (2008)

7. Pérez-Sancho, C.; Rizo, D.; Kersten, S.; Ramírez, R. "Genre classification of music by tonal harmony" Proc. Int. Workshop on Machine Learning and Music, MML 2008, pp. 21-22, Helsinki, Finland (2008)

8. Ponce de León, P.J., Rizo, D., Ramírez, R. "Melody characterization by a fuzzy rule system" Proc. Int. Workshop on Machine Learning and Music, MML 2008, pp. 35-36, Helsinki, Finland (2008)

9. Ponce de León, P.J.; Iñesta, J.M.; Rizo, D. "Mining digital music score collections: melody extraction and genre recognition" Pattern Recognition, ISBN: 978-3-902613-24-4, pp. 559-590, Vienna, Austria (2008) "Pattern Recognition" book, published by IN-TECH

10. Ponce de León, P.J.; Rizo, D.; Ramirez, R.; Iñesta, J.M. "Melody Characterization by a Genetic Fuzzy System" Proceedings of the 5th Sound and Music Computing Conference , pp. 15-23 (2008)

11. Ramírez, R., Pérez, A.; Kersten, S., Rizo, D., Illescas, P.R., Iñesta, J.M. "Modeling celtic violin expressive performance" Proc. Int. Workshop on Machine Learning and Music, MML 2008, pp. 7-8, Helsinki, Finland (2008)

12. Espí D., Ponce de León P.J., Pérez-Sancho C., Rizo D., Iñesta J.M., Moreno-Seco F., Pertusa A. "A cooperative approach to style-oriented music composition" Proc. of the Int. Workshop on Artificial Intelligence and Music, MUSIC-AI, pp. 25-36, Hyderabad, India (2007)

13. Pedro J. Ponce de León, José M. Iñesta, David Rizo "Towards a human-friendly melody characterization by automatically induced rules" Proceedings of the 8th Int. Conf. on Music Information Retrieval, ISMIR 2007, pp. 437–440, Vienna (2007)

14. Plácido R. Illescas, David Rizo, José M. Iñesta "Harmonic, melodic, and functional automatic analysis" Proceedings of the 2007 International Computer Music Conferrence, pp. 165–168 (2007)

15. Rizo D., Ponce de León P. J., Pérez-Sancho C., Pertusa A., Iñesta J. M. "A Pattern Recognition Approach for Melody Track Selection in MIDI Files" Proc. of the 7th Int. Symp. on Music Information Retrieval ISMIR 2006, ISBN: 1-55058-349-2, pp. 61-66, Victoria, Canada (2006)

16. Rizo D., Ponce de León P.J., Pertusa A., Iñesta J.M. "Melodic track identification in MIDI files" Proc. of the 19th Int. FLAIRS Conference, ISBN: 978-1-57735-261-7 (2006)

A

Multiset distances

The similarity computation between vectors of features has been widely studied in the pattern matching and information retrieval literature (Deza and Deza, 2009; Han and Kamber, 2000; Ryu et al., 1998). In this dissertation we have selected some of the well known vector distances and in some cases they have been adapted to our problem of comparing multi-sets in the representation on polyphonic music with trees.

Manhattan, L1

$$\frac{\sum_{p=0}^{11} |\mathbf{v}_{\mathcal{M}_a}[p] - \mathbf{v}_{\mathcal{M}_b}[p]|}{12} \tag{A.1}$$

Cosine similarity

$$1 - \frac{\sum_{p=0}^{11} (\mathbf{v}_{\mathcal{M}_a}[p])(\mathbf{v}_{\mathcal{M}_b}[p])}{\sqrt{\sum_{i=1}^{12} \mathbf{v}_{\mathcal{M}_a}[p]^2 \sum_{i=1}^{12} \mathbf{v}_{\mathcal{M}_b}[p]^2}} \tag{A.2}$$

If the denominator takes a 0 value, the expression returns 1.

Euclidean distance, L2

$$\frac{\sqrt{\sum_{p=0}^{11} (\mathbf{v}_{\mathcal{M}_a}[p] - \mathbf{v}_{\mathcal{M}_b}[p])^2}}{12} \tag{A.3}$$

1/Jaccard coefficient

$$1 - \frac{count_{p=0}^{11}(\mathbf{v}_{\mathcal{M}_a}[p] > 0 \wedge \mathbf{v}_{\mathcal{M}_b}[p] > 0)}{count_{p=0}^{11}(\mathbf{v}_{\mathcal{M}_a}[p] > 0 \vee \mathbf{v}_{\mathcal{M}_b}[p] > 0)} \tag{A.4}$$

The possible singular case of 0/0 is solved to 0.

Log distance

Even though the absolute difference between two short notes can be the same as the difference between two long notes, the perceived difference is not the same. With this distance we try to reflect that situation.

$$\frac{\sqrt{\sum_{p=0}^{11}(ln(\mathbf{v}_{\mathcal{M}_a}[p]+1)-ln(\mathbf{v}_{\mathcal{M}_b}[p]+1))^2}}{12} \tag{A.5}$$

Matching coefficient

$$1-\frac{count_{p=0}^{11}(\mathbf{v}_{\mathcal{M}_a}[p]>0\wedge\mathbf{v}_{\mathcal{M}_b}[p]>0)}{12} \tag{A.6}$$

Multisets distance

$$\frac{\sum_{p=0}^{11}min(1,(\mathbf{v}_{\mathcal{M}_a}[p]-\mathbf{v}_{\mathcal{M}_b}[p])^2)}{12} \tag{A.7}$$

Note that the maximum difference between two components of the multiset has been limited to 1. Initially, unit cost has been assigned to insertion and deletion operations.

Overlap coefficient

$$\frac{count_{p=0}^{11}(\mathbf{v}_{\mathcal{M}_a}[p]=\mathbf{v}_{\mathcal{M}_b}[p])}{12} \tag{A.8}$$

Note the equality between floats is performed as $|\mathbf{v}_{\mathcal{M}_a}[p]-\mathbf{v}_{\mathcal{M}_b}[p]|<0.001$

Probabilities The next distances make use of probabilities. We have used the normalized form of the vectors as an approach to express these probabilities:

$$\forall_{p=0}^{11}p(\mathbf{v}_{\mathcal{M}_a}[p])=\frac{\mathbf{v}_{\mathcal{M}_a}[p]}{\sum_{q=0}^{11}\mathbf{v}_{\mathcal{M}_a}[q]} \tag{A.9}$$

Variational distance

$$\sum_{p=0}^{11}(p(\mathbf{v}_{\mathcal{M}_a}[p])-p(\mathbf{v}_{\mathcal{M}_b}[p])) \tag{A.10}$$

Hellinger distance

$$1-\frac{\sum_{p=0}^{11}(p(\mathbf{v}_{\mathcal{M}_a}[p])p(\mathbf{v}_{\mathcal{M}_b}[p]))}{12} \tag{A.11}$$

Harmonic Mean

$$\sum_{i=1}^{12} \frac{p(\mathbf{v}_{\mathcal{M}_a}[p])\ p(\mathbf{v}_{\mathcal{M}_b}[p])}{p(\mathbf{v}_{\mathcal{M}_a}[p]) + p(\mathbf{v}_{\mathcal{M}_b}[p])} \tag{A.12}$$

B
Algorithms

B.1 Complete `fillTree` algorithm

The code in Alg. 8 includes a simple tuple detection to that specified in Alg. 2 of page 85.

Algorithm 7: possibleTuplets

Input: Tree τ
Input: Current meter with b beats at its numerator
Output: Number of subdivisions for tuplet. A 0 value is returned when no possible tuplet will be detected

```
if compound (n, d) then
    switch depth (τ) do
        case 1: return 0;
        case 2: return 2;
        otherwise return 3;
    end
else
    switch depth (τ) do
        case 1: return 0;
        otherwise return 3;
    end
end
```

Algorithm 8: fillTree (complete version with tuplets)

Input: Subtree tree τ

Data: *pendingDuration* is the duration still left of the current note to be put in the tree, *isTuplet* boolean value that is true if the next notes are forming a tuple, *possibleTuplet* possible irregular subdivision to search in order to detect tuplets, *expectedTupleElementDuration* is the duration that the next note should have to belong to a tuple, i is the index of the current note of the sequence S of notes, p_i stands for the pitch of i-th note, and d_i for its duration, *continuation* is a boolean value indicating the next node will be a continuation of previous one.

Output: *continuation* the same parameter is also an output one

```
if i ≤ |S| then
    if pendingDuration ≥ nodedur(τ) then
        label(τ) = (p_i, continuation) ;
        pendingDuration = pendingDuration − nodedur(τ);
        if pendingDuration < kMINDURATION then
            i = i + 1;
            if i ≤ |S| then
                pendingDuration = d_i + pendingDuration;
            else
                pendingDuration = 0;
            end
            continuation = false;
        else
            continuation = true;
        end
    else
        if nodedur(τ) ≥ kMINDURATION then
            possibleTuple = possibleTuplets(τ);
            if pendingDuration ≠ 0 then
                //0 means no possible tuplet can be detected
                expectedTupleElementDuration = nodedur(τ)/possibleTuple;
                for j = 1 to possibleTuple do
                    if d_{i+j} ≠ expectedTupleElementDuration then
                        isTuplet = false;
                end
                if isTuplet then
                    for j = 1 to possibleTuple do
                        τ_j = createSubTree(τ);
                        label(τ_j) = (p_i, continuation) ;
                        i = i + 1;
                        continuation = false;
                    end
            if ¬isTuplet then
                //see Alg. 3 in page 85
                a = computeArity(τ);
                for j = 1 to a do
                    //createSubTree creates a new child for the given tree
                    τ_j = createSubTree(τ);
                    fillTree (τ_j, i, pendingDuration,continuation);
                end
        end
    end
end
```

B.2 Complexity of the partially labeled tree comparison algorithm

The complete development of the partially labeled tree similarity algorithm introduced in page 109 is developed below:

$$\begin{aligned}
\mathcal{T}(|T_A|,|T_B|) &= c + \mathcal{T}_f(|T_A|-1,|T_B|-1)\\
\mathcal{T}_f(|T_A|,|T_B|) &= \sum_{i=1}^{R_A}\sum_{j=1}^{R_B}\mathcal{T}(\frac{|T_A|}{R_A},\frac{|T_B|}{R_B})\\
&= R_A \times R_B \times \mathcal{T}(\frac{|T_A|}{R_A},\frac{|T_B|}{R_B})
\end{aligned}$$

$$\begin{aligned}
\mathcal{T}(|T_A|,|T_B|) &= c + \mathcal{T}_f(|T_A|-1,|T_B|-1)\\
&= c + R_A \times R_B \times \left(c + \mathcal{T}_f(\frac{|T_A|}{R_A}-1,\frac{|T_B|}{R_B}-1)\right)\\
&= c + R_A \times R_B \times \left(c + R_A \times R_B \times \mathcal{T}\left(\frac{\frac{|T_A|}{R_A}-1}{R_A},\frac{\frac{|T_B|}{R_B}-1}{R_B}\right)\right)\\
&= c + R_A \times R_B \times c + R_A \times R_B \times R_A \times R_B \times \mathcal{T}\left(\frac{\frac{|T_A|}{R_A}-1}{R_A},\frac{\frac{|T_B|}{R_B}-1}{R_B}\right)\\
&= c + R_A \times R_B \times c + R_A \times R_B \times R_A \times R_B\\
&\times \left(c + \mathcal{T}_f\left(\frac{\frac{|T_A|}{R_A}-1}{R_A}-1,\frac{\frac{|T_B|}{R_B}-1}{R_B}-1\right)\right)\\
&= c + R_A \times R_B \times c + R_A \times R_B \times R_A \times R_B\\
&\times \left(c + R_A \times R_B \times \mathcal{T}\left(\frac{\frac{\frac{|T_A|}{R_A}-1}{R_A}-1}{R_A},\frac{\frac{\frac{|T_B|}{R_B}-1}{R_B}-1}{R_B}\right)\right)\\
&= c + R_A \times R_B \times c + (R_A \times R_B)^2 + (R_A \times R_B)^3\\
&\quad \times \mathcal{T}\left(\frac{\frac{\frac{|T_A|}{R_A}-1}{R_A}-1}{R_A},\frac{\frac{\frac{|T_B|}{R_B}-1}{R_B}-1}{R_B}\right)\\
&\cdots\\
&= \sum_{k=0}^{n} c \times (R_A \times R_B)^k + (R_A \times R_B)^n \times \mathcal{T}\left(\frac{\frac{\frac{|T_A|}{R_A}-1}{R_A}-1}{R_A},\frac{\frac{\frac{|T_B|}{R_B}-1}{R_B}-1}{R_B}\right)
\end{aligned}$$

The following condition holds:

$$\mathcal{T}\left(\frac{\frac{|T_A|}{R_A}-1}{R_A}-1, \frac{\frac{|T_B|}{R_B}-1}{R_B}-1\right) < \mathcal{T}\left(\frac{|T_A|}{R_A^3}, \frac{|T_B|}{R_B^3}\right)$$

Thus, the previous development can be rewritten as:

$$\mathcal{T}(|T_A|,|T_B|) = \sum_{k=0}^{n} c \times (R_A \times R_B)^k + (R_A \times R_B)^n \times \mathcal{T}\left(\frac{|T_A|}{R_A^n}, \frac{|T_B|}{R_B^n}\right)$$

This recurrence follows until one of the base cases of s_p is hold, i.e., when at least one of the trees becomes a leaf. As each call removes one level, it can be stated that $n = \min(\mathrm{h}(T_A), \mathrm{h}(T_B))$. Due to the nature of the trees, we can say that $\mathrm{h}(T_A) \approx \log_{R_A} |T_A|$. Lets take for example $|T_A| < |T_B|$, then $n = \log_{R_A} |T_A|$, that transform the recurrence in:

$$\begin{aligned}
\mathcal{T}(|T_A|,|T_B|) &= \sum_{k=0}^{\log_{R_A}|T_A|} c \times (R_A \times R_B)^k + (R_A \times R_B)^{\log_{R_A}|T_A|} \times \mathcal{T}\left(\frac{|T_A|}{R_A^{\log_{R_A}|T_A|}}, \frac{|T_B|}{R_B^{\log_{R_A}|T_A|}}\right) \\
&= \sum_{k=0}^{\log_{R_A}|T_A|} c \times (R_A \times R_B)^k + (R_A \times R_B)^{\log_{R_A}|T_A|} \times \mathcal{T}\left(\frac{|T_A|}{|T_A|}, \frac{|T_B|}{R_B^{\log_{R_A}|T_A|}}\right) \\
&= \sum_{k=0}^{\log_{R_A}|T_A|} c \times (R_A \times R_B)^k + (R_A \times R_B)^{\log_{R_A}|T_A|} \times \mathcal{T}\left(1, \frac{|T_B|}{R_B^{\log_{R_A}|T_A|}}\right)
\end{aligned}$$

Following the logarithm base change rules $\log_{R_A} |T_A| = \frac{\log_{R_B}|T_A|}{\log_{R_B} R_A}$:

$$\begin{aligned}
R_B^{\log_{R_A}|T_A|} &= R_B^{\frac{\log_{R_B}|T_A|}{\log_{R_B} R_A}} \\
&= R_B^{(\log_{R_B}|T_A|)\times\frac{1}{\log_{R_B} R_A}} \\
&= |T_A|^{(\log_{R_B} R_A)^{-1}}
\end{aligned}$$

Using that equivalence:

$$\begin{aligned}\mathcal{T}(|T_A|,|T_B|) &= \sum_{k=0}^{\log_{R_A}|T_A|} c \times (R_A \times R_B)^k + (R_A \times R_B)^{\log_{R_A}|T_A|} \times \mathcal{T}\left(1, \frac{|T_B|}{|T_A|^{(\log_{R_B} R_A)^{-1}}}\right) \\ &= \sum_{k=0}^{\log_{R_A}|T_A|} c \times (R_A \times R_B)^k + R_A^{\log_{R_A}|T_A|} \times R_B^{\log_{R_A}|T_A|} \times \mathcal{T}\left(1, \frac{|T_B|}{|T_A|^{(\log_{R_B} R_A)^{-1}}}\right) \\ &= \sum_{k=0}^{\log_{R_A}|T_A|} c \times (R_A \times R_B)^k + |T_A| \times |T_A|^{(\log_{R_B} R_A)^{-1}} \times \mathcal{T}\left(1, \frac{|T_B|}{|T_A|^{(\log_{R_B} R_A)^{-1}}}\right)\end{aligned}$$

Lets solve the sum in the first part of the formula:

$$\begin{aligned}\sum_{k=0}^{\log_{R_A}|T_A|} c \times (R_A \times R_B)^k &= c \times \sum_{k=0}^{\log_{R_A}|T_A|} (R_A \times R_B)^k \\ &= c \times \frac{(R_A \times R_B)^{\log_{R_A}|T_A|+1} - 1}{R_A \times R_B - 1} \\ &= c \times \frac{R_A^{\log_{R_A}|T_A|} \times R_B^{\log_{R_A}|T_A|} \times R_A \times R_B - 1}{R_A \times R_B - 1} \\ &= c \times \frac{|T_A| \times |T_A|^{(\log_{R_B} R_A)^{-1}} \times R_A \times R_B - 1}{R_A \times R_B - 1}\end{aligned}$$

Using that partial result, the complete formula results in:

$$\begin{aligned}\mathcal{T}(|T_A|,|T_B|) = c \times & \frac{|T_A| \times |T_A|^{(\log_{R_B} R_A)^{-1}} \times R_A \times R_B - 1}{R_A \times R_B - 1} \\ & + |T_A| \times |T_A|^{(\log_{R_B} R_A)^{-1}} \times \mathcal{T}\left(1, \frac{|T_B|}{|T_A|^{(\log_{R_B} R_A)^{-1}}}\right)\end{aligned}$$

The base class of s_p, i.e., when one of the trees consists of only a leaf node, the algorithm just traverses in pre-order the non leaf tree. Thus, in that case, $\mathcal{T}(T) \in O(T)$. If the first node to become leaf is T_A as we have stated before, the complexity of s_p is the number of nodes that are left in the second tree:

$$\begin{aligned}\mathcal{T}(|T_A|,|T_B|) = c \times & \frac{|T_A| \times |T_A|^{(\log_{R_B} R_A)^{-1}} \times R_A \times R_B - 1}{R_A \times R_B - 1} \\ & + |T_A| \times |T_A|^{(\log_{R_B} R_A)^{-1}} \times \frac{|T_B|}{|T_A|^{(\log_{R_B} R_A)^{-1}}} \\ = c \times & \frac{|T_A| \times |T_A|^{(\log_{R_B} R_A)^{-1}} \times R_A \times R_B - 1}{R_A \times R_B - 1} + |T_A| \times |T_B| \\ & \in O(|T_A| \times |T_B|)\end{aligned}$$

B.3 Algorithm for the partially labeled tree comparison algorithm

Algorithm 9: s_p

Input: Trees T_A and T_B
Output: Similarity value
if $\text{leaf}(T_A) \wedge \text{leaf}(T_B)$ **then**
  return $c(\text{rlabel}(T_A), \text{rlabel}(T_B))$
else if $\text{leaf}(T_A) \wedge \neg\,\text{leaf}(T_B)$ **then**
  return $-c(\text{rlabel}(T_A), \text{rlabel}(T_B)) + \sum_{j=1}^{\text{rank}(T_A)} \frac{s_p(T_A, \text{child}_j(T_B))}{\text{rank}(T_B)}$
else if $\neg\,\text{leaf}(T_A) \wedge \text{leaf}(T_B)$ **then**
  return $-c(\text{rlabel}(T_A), \text{rlabel}(T_B)) + \sum_{i=1}^{\text{rank}(T_B)} \frac{s_p(\text{child}_i(T_A), T_B)}{\text{rank}(T_A)}$
else
  return $-c(\text{rlabel}(T_A), \text{rlabel}(T_B)) + \frac{s_{fp}(T_A, T_B)}{\max(\text{rank}(T_A), \text{rank}(T_B))}$
end

Algorithm 10: s_{fp}

Input: Trees T_A and T_B
Output: Similarity value
Data: $\mathbf{D}$ is a matrix of $(\text{rank}(T_A)+1) \times (\text{rank}(T_B)+1)$ dimension
Data: $cins, csus, cdel \in \mathbb{R}$
for $i \leftarrow 0$ **to** $\text{rank}(T1)$ **do**
  for $j \leftarrow 0$ **to** $\text{rank}(T2)$ **do**
    if $i \geq 1$ **then**
      $cdel \leftarrow \mathbf{D}[i-1][j]$
    end
    if $j \geq 1$ **then**
      $cins \leftarrow \mathbf{D}[i][j-1]$
    end
    if $i \geq 1 \wedge j \geq 1$ **then**
      $csus \leftarrow \mathbf{D}[i-1][j-1] + s_p(\text{child}_i(T_A), \text{child}_j(T_B))$
    end
    if $i \geq 1 \vee j \geq 1$ **then**
      $\mathbf{D}[i][j] \leftarrow \min(cdel, cins, csus)$
    end
  end
  return $\mathbf{D}[\text{rank}(T_A)][\text{rank}(T_B)]$
end

C Corpora

The names of the songs found in each of the corpora are detailed below. For each song there are several cover files or variations plus the main prototype, theme or query file. The complete dataset can be obtained upon request to the author.

C.1 104 (monophonic)

- "Africa" (Toto)
- "Canon in D major (J. Pachelbel)
- "Eine kleine Nachtmusik K. 525, 1st movement (W.A. Mozart)
- "Love theme from The Godfather film" (N. Rota)
- "Help" (The Beatles)
- "Imagine" (John Lennon)
- "Love Story film theme" (F. Lai)
- "Theme from New York, New York" (J. Kander)
- "Love Theme from Titanic" (J. Horner)
- "Unchained melody" (A. North)

C.2 VARM (monophonic)

The following RISM content can be downloaded from RISM at http://www.rism.org.uk/.

- "Aria with Variations", RISM ID. no 13931
- "17 Variations sérieuses / pour le Piano", RISM ID. no 32308 (F. Mendelssohn Bartholdy)

- "Minuet (Variations)", RISM ID. no 34326 (M. Benedetto M)
- "Lesson (Variations)", RISM ID. no 34344 (M. Coyle)
- "Felton's Minuet (Variations)", RISM ID. no 34373 (W. Felton)
- "The Highland Laddie (Variations)", RISM ID. no 34382 (J. Parry)
- "Theme and variations, K.265" (W.A. Mozart)
- "Sonata K.331, 1st movement theme and variations" (W.A. Mozart)
- "Goldberg variations, BWV 988 (bass voice)" (J.S. Bach)

C.3 Pascal (monophonic)

- "Alouette" (french children song)
- "Macarena" (Los del Rio)
- "Avemaria" (F. Schubert)
- "Ode to joy, from the 9th symphony" (L.V. Beethoven)
- "Boléro" (M.Ravel)
- "Oh! Susanna" (S. Foster)
- "La Cucaracha" (mexican song)
- "Pink Panther" (H. Mancini)
- "La Cumparsita" (G.M. Rodriguez)
- "Silent night" (christmas carol)
- "Frère Jacques" (french children song)
- "Tico-Tico no Fubá" (Zequinha de Abreu)
- "Guantanamera" (José Fernández Díaz)
- "Toccata and fugue in D minor" (J.S.Bach)
- "Happy birthday" (Patty Hill and Mildred J. Hill)
- "Twinkle twinkle little star" (children song)
- "Jingle bells" (J. Pierpoint)
- "When the Saints Go Marching In" (J.M. Black)
- "Lohengrin, wedding march" (R. Wagner)
- "Yesterday" (The Beatles)

C.4 ICPS (polyphonic)

- "Ave Maria" (F. Schubert)
- "Boléro" (M. Ravel)
- "Alouette" (trad.)
- "Happy Birthday" (trad.)
- "Frère Jacques" (trad.)
- "Jingle Bells" (Christmas carol)
- "When The Saints Go Marching In" (jazz standard)

C.5 VARP (polyphonic)

- "Variations on a rococo theme", Op.33 (P. Tchaikovsky)
- Variations on Bach's "English suites" BWV 806-808 (suite 1 courante II, suite 2, 3, and 6 sarabande) (anon.)
- "Goldberg variations", BWV 988 (J.S. Bach)
- "Variations on a theme of Beethoven", Op. 35 (C. Saint-Saëns)
- Variations on Beethoven's Quartet Op.18 No.5 (anon.)
- "Diabelli variations", Op. 120 (L.V. Beethoven)
- "Symphonic variations", Op.78 (A. Dvořák)
- Variations on Beethoven's Für Elise (anon.)
- "L' ecole moderne, Les arpèges", Op.10 No.9 (H. Wieniawski)
- "Theme and variations for horn and piano", Op.10 (J. Labor)
- "Theme and variations for guitar" (S. Ravitz)
- Rock variations on Pachelbel's cannon (anon.)
- "Symphonic Studies", Op.13 (R. Schumann)

C.6 INET (polyphonic)

This corpus contains songs by 10cc, Cher, 4 non blondes, Coldplay, AC/DC, Collective Soul, ABBA, Daft Punk, Darude, Aha, Depeche Mode, Dido, All4One, Dire Straits, Aphaville, Village People, Beatles, Bee Gees, Bob Marley, and Bruce Springsteen.

C.7 COVERS (polyphonic)

This corpus is the subset of a larger POP genre MIDI files database of all the songs that have at least two different renderings each one. The sub-genres found are 60's, Fiesta, New wave, Pop Latin, Reggae, Soul, 90s, New age, Pop-American, Pop-Standard, Singers.

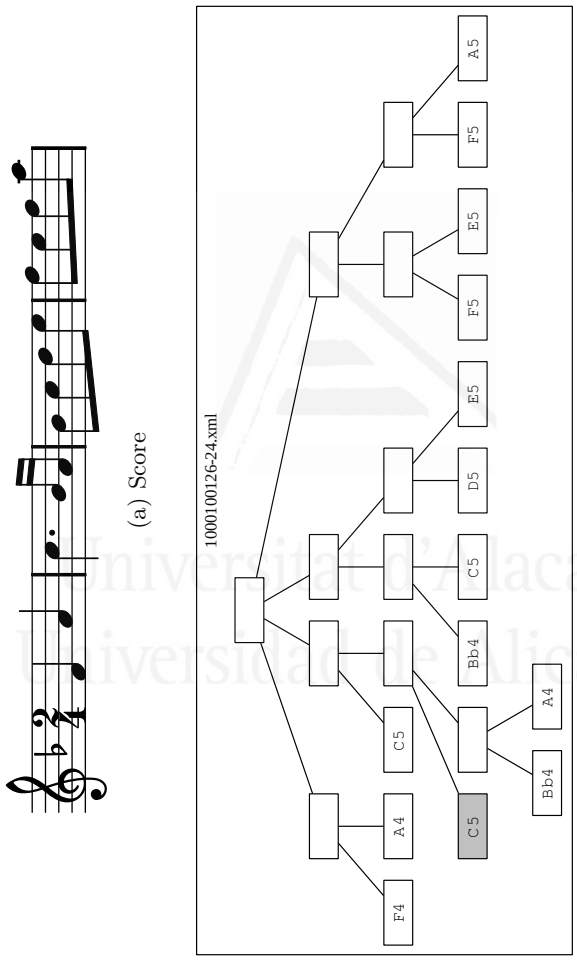
They are listed below: Association, Beatles, Bee Gees, Bobby Darin, Box Tops, Connie Francis, Frankie Valli, Hermans Hermits, Lovin Spoonful, Neil Sedaka, Paul Anka, Shadows, Spencer Davis Group, The Mammas and the Pappas, Three Dog Night, Ace of Base, All Saints, Anastacia, Aqua, B*Witched, Backstreet Boys, Boyz II Men, Boyzone, Britney Spears, Christina Aguilera, Destiny's Child, Fine Young Cannibals, Geri Halliwell, Hanson, 'N Sync, Pink, Right Said Fred, S Club 7, Savage Garden, Spice Girls, Sugababes, Take that, Usher, West Life, Venga Boys, Enigma, Enya, Frank Mills, Jean Michel Jarre, Kitaro, Aha, B-52's, Bangles, Berlin, Cars, Cyndi Lauper, Depeche Mode, Duran Duran, Eurythmics, Falco, Frankie Goes to Hollywood, Howard Jones, Human League, INXS, Katrina and the Waves, Level 42, Madness, Men At Work, Orchestral Manouvers Dark, Simple Minds, Spandau Ballet, Tears for Fears, The Knack, Wham, Yazoo, Alan Jackson, Anne Murray, B. J. Thomas, Garth Brooks, John Anderson, John M. Montgomery, Kenny Rogers, Little Texas, Peter Paul and Mary, Sammy Kershaw, Shania Twain, Tracy Byrd, Gloria Estefan, Morris Albert, Ricky Martin, Abba, Alphaville, Annie Lennox, Barbra Streisand, Barry Manilow, Bette Midler, Boonie Tyler, Bruce Hornsby, Captain and Tennille, Carpenters, Celine Dion, Cher, David Gates, Eric Carmen, George Michael, Glenn Medeiros, Irene Cara, Jamiroquai, Jennifer Warnes, Johnny Rivers, Kim Carnes, Kylie Minogue, Lisa Stansfield, Madonna, Michael Jackson, Michael Sembello, Pet Shop Boys, Renato Zero, Richard Marx, Rick Astley, Robert Palmer, Roxette, Seal, Simon and Garfunkel, Tom Jones, Wet Wet, Bob Marley, Eddie Grant, Inner Circle, Johnny Nash, Shaggy, UB40, Al Stewart, America, Arlo Guthrie, Art Garfunkel, Billy Joel, Boz Scaggs, Bread, Carly Simon, Carole King, Cat Stevens, Chris Rea, Christopher Cross, Dan Fogelberg, Dido, Don Mc Lean, Donovan, Elton John, George Harrison, Gerry Rafferty, Gilbert O'Sullivan, Harry Nilsson, James Taylor, Janis Ian, Jim Croce, Jimmy Buffett, John Denver, John Lennon, Joni Mitchell, Leo Sayer, Neil Diamond, Nelly Furtado, Paul Young, Sinead O'Connor, Van Morrison, Des'ree, Lionel Richie, Mariah Carey, Natalie Cole, Sade, Whitney Houston.

D

Examples of trees

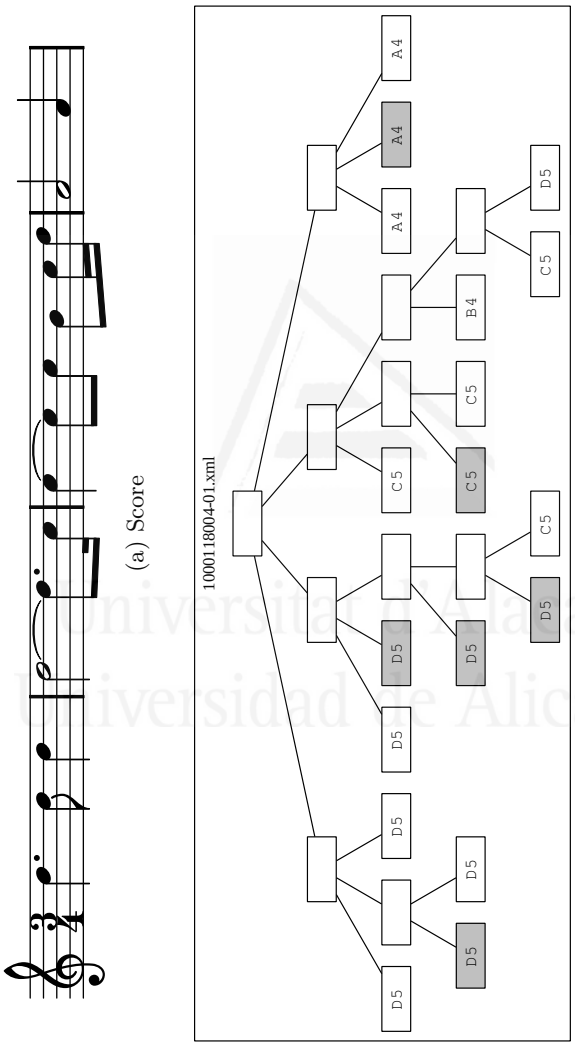
D.1 Examples of tree construction for different meters

The following figures show examples of construction of metrical trees for the main meters. In the figures, the shaded nodes are those containing a continuation note.



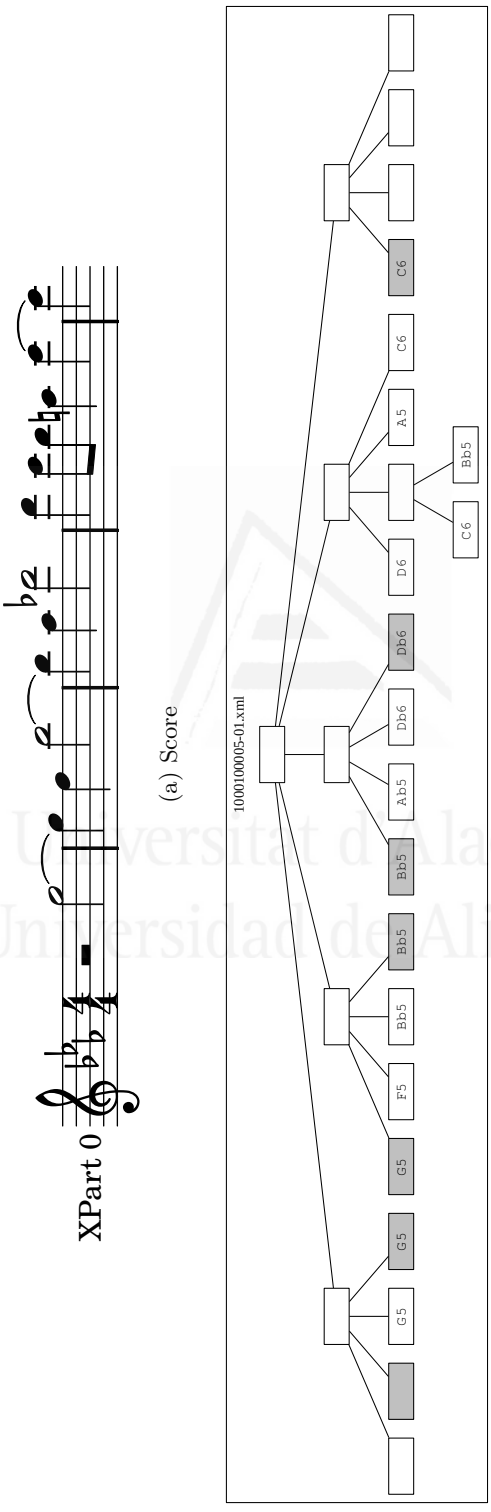
(a) Score (b) Tree

Figure D.1: 2/4 meter. Bach, Johann Sebastian; Clavierbung, 3. Theil, org. Duetto. 2.



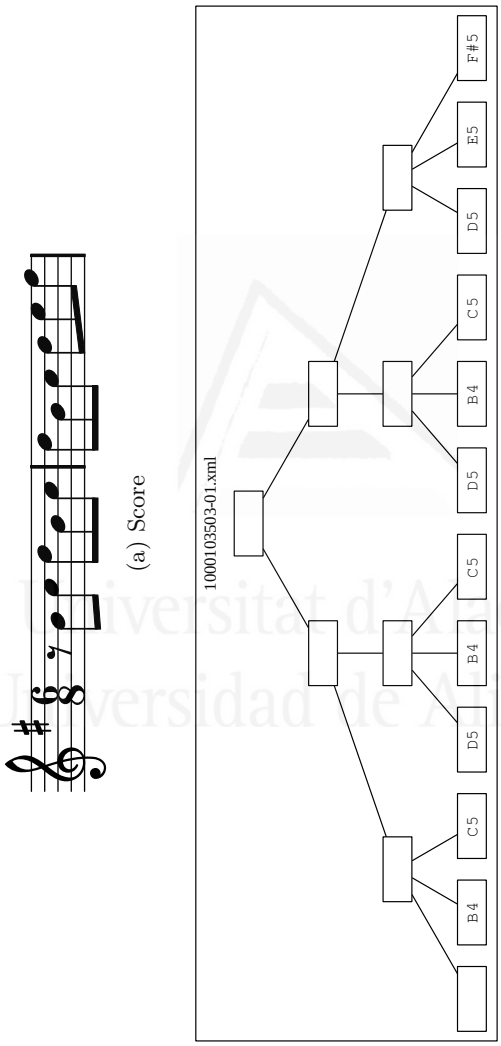
(a) Score (b) Tree

Figure D.2: 3/4 meter. Couperin, Louis; Sarabande



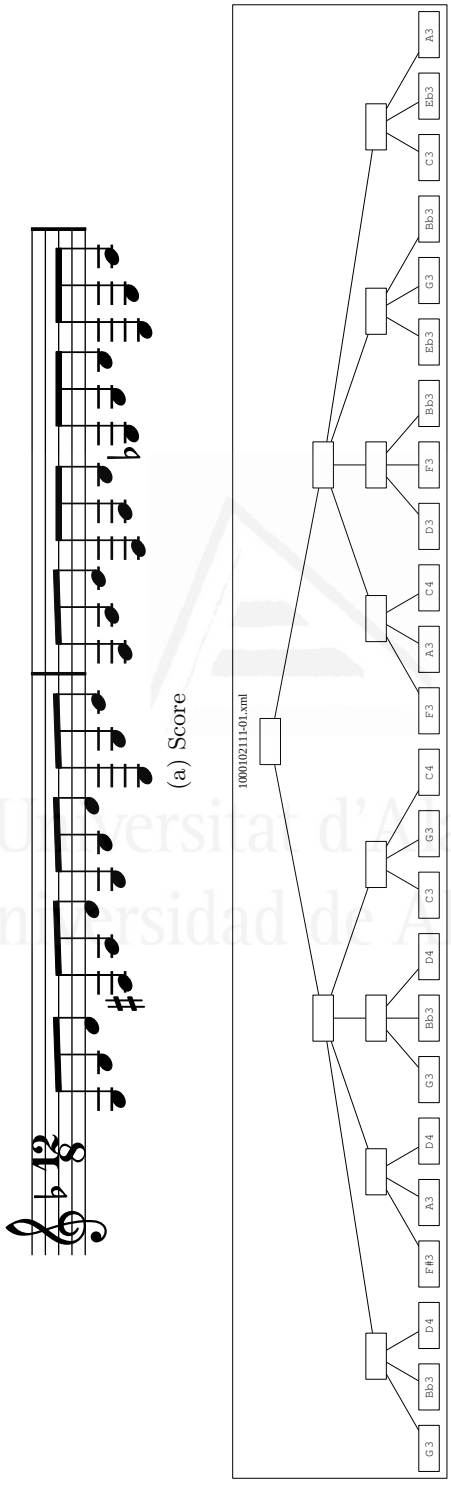
(a) Score (b) Tree

Figure D.3: 4/4 meter. Pergolesi, Giovanni Battista; Stabat mater, Grave



(a) Score (b) Tree

Figure D.4: 6/8 meter. Mozart, Wolfgang Amadeus; Don Giovanni, pf. Allegro



(a) Score (b) Tree

Figure D.5: 12/8 meter. Händel, Georg Friedrich; Caro autor di mia doglia, bc.

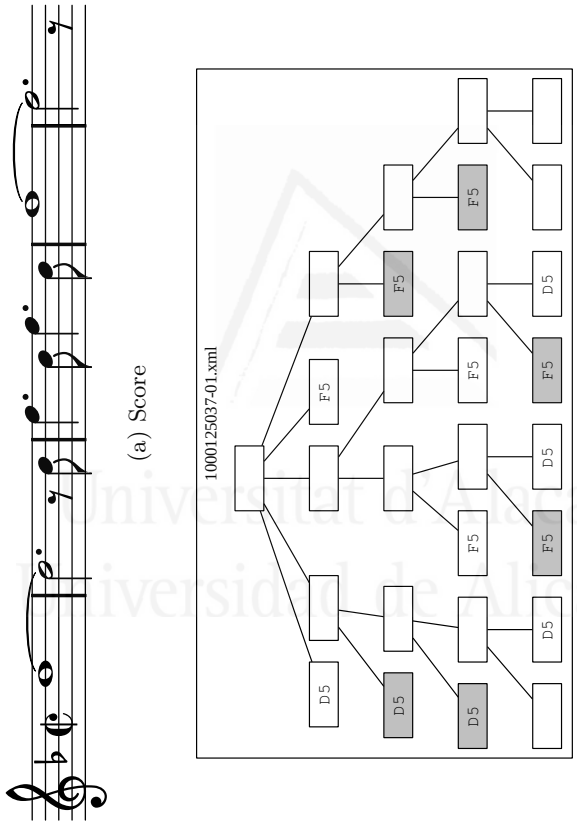
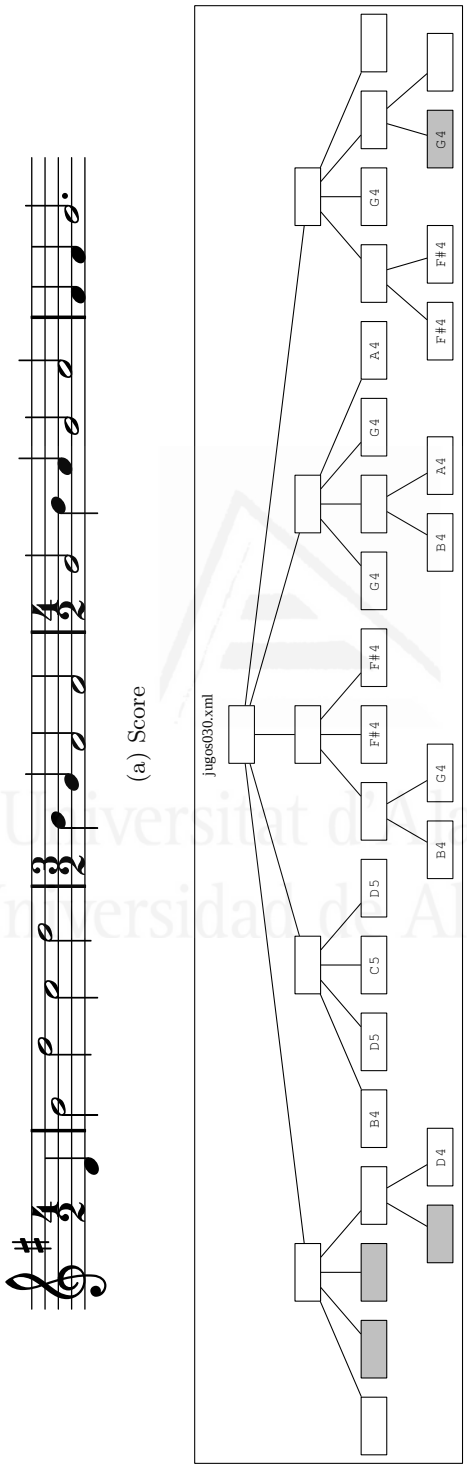
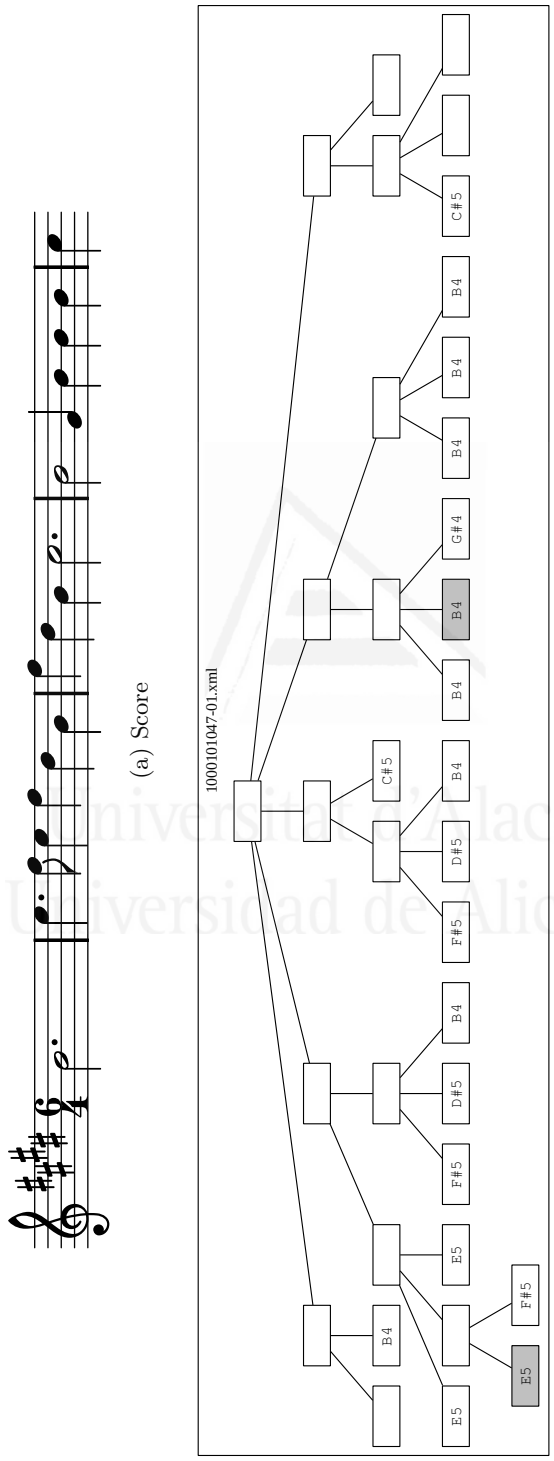


Figure D.6: 2/2 meter. Gluck, Christoph Willibald; Alceste, Intrada: Pi tosto adagio



(a) Score (b) Tree

Figure D.7: 3/2 and 4/2 meters. Der Elsberger Bie vrie ischt auf Molankizle, Essen, Europa, Jugoslavija



(a) Score

(b) Tree

Figure D.8: 6/4 meter. Swan, Timothy; The morning is charming.

E

Experiment results

E.1 Setup selection results for monophonic methods

Table E.1: Selected setups for method Metric trees

	Setup	Borda count	PaC	Time
1	p_c. $l = 2$. Heuristic prop.. Partial	3.0	0.5	28
2	p_{hdc}. $l = 2$. Heuristic prop.. Partial	3.0	0.5	28
3	p_{hdc}. $l = 2$. Left prop.. Partial	3.0	0.5	28
4	p_{itv}. $l = 2$. Left prop.. Partial	3.0	0.5	28
5	p_{itv}. $l = 2$. Melodic prop.. Partial	3.0	0.5	28
6	p_c. $l = 2$. Heuristic prop.. Selkow	3.0	0.5	42
7	p_c. $l = 2$. Left prop.. Selkow	3.0	0.5	42
8	p_{hdc}. $l = 2$. Heuristic prop.. Selkow	3.0	0.5	42
9	p_{hdc}. $l = 2$. Left prop.. Selkow	3.0	0.5	42
10	p_{hdc}. $l = 2$. No prop.. Selkow	3.0	0.4	42
11	p_{itv}. $l = 2$. Left prop.. Selkow	3.0	0.5	42
12	p_{itv}. $l = 2$. Melodic prop.. Selkow	3.0	0.5	42
13	p_{itv}. $l = 2$. No prop.. Selkow	3.0	0.4	42
14	p_{itv}. $l = 2$. Right prop.. Selkow	3.0	0.5	42
15	p_{ift}. $l = 2$. Heuristic prop.. Selkow	3.0	0.5	42
16	p_{hdc}. $l = 3$. Left prop.. Partial	3.0	0.6	122
17	p_c. $l = 3$. Heuristic prop.. Partial	3.0	0.5	123
18	p_{hdc}. $l = 3$. Melodic prop.. Partial	3.0	0.5	123
19	p_{itv}. $l = 3$. Melodic prop.. Partial	3.0	0.5	123
20	p_{hdc}. $l = 2$. Melodic prop.. Shasha	3.0	0.5	129
21	p_{ift}. $l = 2$. Melodic prop.. Shasha	3.0	0.5	129
22	p_c. $l = 2$. Heuristic prop.. Shasha	3.0	0.5	130
23	p_c. $l = 2$. Left prop.. Shasha	3.0	0.5	130
24	p_{hdc}. $l = 2$. Heuristic prop.. Shasha	3.0	0.6	130
25	p_{hdc}. $l = 2$. Left prop.. Shasha	3.0	0.6	130
26	p_{ift}. $l = 2$. Heuristic prop.. Shasha	3.0	0.5	130
27	p_{itv}. $l = 2$. Right prop.. Shasha	3.0	0.5	131
28	p_{hdc}. $l = 3$. Melodic prop.. Selkow	3.0	0.6	145
29	p_{itv}. $l = 3$. Melodic prop.. Selkow	3.0	0.6	145

continued on next page

continued from previous page

	Setup	Borda count	PaC	Time
30	p_{ift}. $l = 3$. Melodic prop.. Selkow	3.0	0.5	145
31	p_c. $l = 3$. Left prop.. Selkow	3.0	0.6	146
32	p_c. $l = 3$. Right prop.. Selkow	3.0	0.5	146
33	p_{hdc}. $l = 3$. Right prop.. Selkow	3.0	0.5	146
34	p_{itv}. $l = 3$. Heuristic prop.. Selkow	3.0	0.6	146
35	p_{itv}. $l = 3$. Left prop.. Selkow	3.0	0.6	146
36	p_{ift}. $l = 3$. Left prop.. Selkow	3.0	0.5	146
37	p_{hdc}. $l = 2$. No prop.. Align	3.0	0.4	155
38	p_{itv}. $l = 2$. Heuristic prop.. Align	3.0	0.5	165
39	p_{itv}. $l = 2$. Melodic prop.. Align	3.0	0.5	165
40	p_{itv}. $l = 2$. Right prop.. Align	3.0	0.5	165
41	p_{ift}. $l = 2$. Melodic prop.. Align	3.0	0.5	165
42	p_{itv}. $l = 4$. Left prop.. Partial	3.0	0.5	346
43	p_{itv}. $l = 4$. Melodic prop.. Partial	3.0	0.5	346
44	p_{ift}. $l = 4$. Melodic prop.. Partial	3.0	0.5	346
45	p_{abs}. $l = 4$. No prop.. Selkow	3.0	0.4	367
46	p_{hdc}. $l = 4$. Melodic prop.. Selkow	3.0	0.6	367
47	p_{itv}. $l = 4$. Heuristic prop.. Selkow	3.0	0.5	367
48	p_{itv}. $l = 4$. Melodic prop.. Selkow	3.0	0.5	367
49	p_{ift}. $l = 4$. Heuristic prop.. Selkow	3.0	0.6	367
50	p_{itv}. $l = 4$. Right prop.. Selkow	3.0	0.5	368
51	p_{ift}. $l = 3$. No prop.. Shasha	3.0	0.4	707
52	p_{ift}. $l = 5$. No prop.. Selkow	3.0	0.4	714
53	p_{hdc}. $l = 5$. Left prop.. Selkow	3.0	0.5	715
54	p_{ift}. $l = 5$. Left prop.. Selkow	3.0	0.5	715
55	p_c. $l = 5$. Heuristic prop.. Selkow	3.0	0.5	716
56	p_{hdc}. $l = 5$. Heuristic prop.. Selkow	3.0	0.5	716
57	p_{hdc}. $l = 5$. Melodic prop.. Selkow	3.0	0.5	718
58	p_{ift}. $l = 5$. Melodic prop.. Selkow	3.0	0.5	718
59	p_{itv}. $l = 5$. Right prop.. Selkow	3.0	0.5	720
60	p_c. $l = 3$. Heuristic prop.. Shasha	3.0	0.5	734
61	p_c. $l = 3$. Melodic prop.. Shasha	3.0	0.5	734
62	p_{hdc}. $l = 3$. Melodic prop.. Shasha	3.0	0.5	734
63	p_{ift}. $l = 3$. Melodic prop.. Shasha	3.0	0.6	734
64	p_c. $l = 3$. Left prop.. Shasha	3.0	0.5	736
65	p_c. $l = 3$. Right prop.. Shasha	3.0	0.5	736
66	p_{hdc}. $l = 3$. Left prop.. Shasha	3.0	0.6	736
67	p_{itv}. $l = 3$. Right prop.. Shasha	3.0	0.5	736
68	p_{ift}. $l = 3$. Right prop.. Shasha	3.0	0.5	736
69	p_{itv}. $l = 5$. Heuristic prop.. Partial	3.0	0.5	745
70	p_{itv}. $l = 5$. Left prop.. Partial	3.0	0.5	747
71	p_{ift}. $l = 5$. Melodic prop.. Partial	3.0	0.5	749
72	p_{ift}. $l = 3$. No prop.. Align	3.0	0.5	925
73	p_c. $l = 3$. Left prop.. Align	3.0	0.5	973
74	p_{hdc}. $l = 3$. Left prop.. Align	3.0	0.5	973
75	p_{hdc}. $l = 3$. Heuristic prop.. Align	3.0	0.5	1007
76	p_{itv}. $l = 6$. Heuristic prop.. Selkow	3.0	0.5	1222

continued on next page

continued from previous page

	Setup	Borda count	PaC	Time
77	p_{itv}. $l = 6$. Heuristic prop.. Partial	3.0	0.5	1417
78	p_{itv}. $l = 4$. Left prop.. Shasha	3.0	0.5	2862
79	p_{itv}. $l = 4$. Heuristic prop.. Shasha	3.0	0.5	2880
80	p_{hdc}. $l = 4$. Left prop.. Align	3.0	0.5	4061
81	p_c. $l = 5$. Melodic prop.. Align	3.0	0.4	12809
82	p_{itv}. $l = 5$. Melodic prop.. Align	3.0	0.5	12809
83	p_{ift}. $l = 5$. Melodic prop.. Align	3.0	0.5	12809
84	p_c. $l = 5$. Left prop.. Align	3.0	0.4	12821
85	p_{ift}. $l = 5$. Left prop.. Align	3.0	0.5	12821
86	p_{hdc}. $l = 2$. Melodic prop.. Partial	4.0	0.5	28
87	p_{hdc}. $l = 2$. Melodic prop.. Selkow	4.0	0.6	42
88	p_{itv}. $l = 3$. Left prop.. Partial	4.0	0.5	122
89	p_{ift}. $l = 3$. Left prop.. Partial	4.0	0.5	122
90	p_{hdc}. $l = 3$. Heuristic prop.. Partial	4.0	0.6	123
91	p_{itv}. $l = 3$. Heuristic prop.. Partial	4.0	0.5	123
92	p_{ift}. $l = 3$. Melodic prop.. Partial	4.0	0.5	123
93	p_{itv}. $l = 2$. Melodic prop.. Shasha	4.0	0.5	129
94	p_{itv}. $l = 2$. Heuristic prop.. Shasha	4.0	0.6	130
95	p_{itv}. $l = 2$. Left prop.. Shasha	4.0	0.6	130
96	p_{ift}. $l = 2$. Left prop.. Shasha	4.0	0.5	130
97	p_c. $l = 3$. Melodic prop.. Selkow	4.0	0.6	145
98	p_{hdc}. $l = 3$. Heuristic prop.. Selkow	4.0	0.5	146
99	p_{hdc}. $l = 3$. Left prop.. Selkow	4.0	0.6	146
100	p_{ift}. $l = 3$. Heuristic prop.. Selkow	4.0	0.5	146
101	p_{hdc}. $l = 2$. Left prop.. Align	4.0	0.5	165
102	p_c. $l = 4$. Heuristic prop.. Partial	4.0	0.6	346
103	p_c. $l = 4$. Left prop.. Partial	4.0	0.5	346
104	p_{hdc}. $l = 4$. Heuristic prop.. Partial	4.0	0.5	346
105	p_{hdc}. $l = 4$. Left prop.. Partial	4.0	0.5	346
106	p_{ift}. $l = 4$. Left prop.. Partial	4.0	0.5	346
107	p_c. $l = 4$. Melodic prop.. Selkow	4.0	0.5	367
108	p_{hdc}. $l = 4$. Heuristic prop.. Selkow	4.0	0.6	367
109	p_{ift}. $l = 4$. Melodic prop.. Selkow	4.0	0.5	367
110	p_c. $l = 4$. Left prop.. Selkow	4.0	0.5	368
111	p_{hdc}. $l = 4$. Left prop.. Selkow	4.0	0.5	368
112	p_{itv}. $l = 4$. Left prop.. Selkow	4.0	0.5	368
113	p_{ift}. $l = 4$. Right prop.. Selkow	4.0	0.5	368
114	p_c. $l = 5$. Left prop.. Selkow	4.0	0.5	715
115	p_{itv}. $l = 5$. Left prop.. Selkow	4.0	0.5	715
116	p_{itv}. $l = 5$. Heuristic prop.. Selkow	4.0	0.5	716
117	p_c. $l = 5$. Melodic prop.. Selkow	4.0	0.5	718
118	p_{itv}. $l = 5$. Melodic prop.. Selkow	4.0	0.5	718
119	p_{hdc}. $l = 3$. Heuristic prop.. Shasha	4.0	0.6	734
120	p_{itv}. $l = 3$. Heuristic prop.. Shasha	4.0	0.6	734
121	p_{itv}. $l = 3$. Melodic prop.. Shasha	4.0	0.6	734

continued on next page

continued from previous page

	Setup	Borda count	PaC	Time
122	p_{hdc}. $l = 3$. Right prop.. Shasha	4.0	0.5	736
123	p_{itv}. $l = 3$. Left prop.. Shasha	4.0	0.6	736
124	p_{hdc}. $l = 5$. Heuristic prop.. Partial	4.0	0.5	745
125	p_{hdc}. $l = 5$. Left prop.. Partial	4.0	0.5	747
126	p_{ift}. $l = 5$. Left prop.. Partial	4.0	0.5	747
127	p_c. $l = 3$. Melodic prop.. Align	4.0	0.5	975
128	p_{hdc}. $l = 3$. Melodic prop.. Align	4.0	0.5	975
129	p_{hdc}. $l = 6$. Heuristic prop.. Selkow	4.0	0.5	1222
130	p_{hdc}. $l = 6$. Heuristic prop.. Partial	4.0	0.5	1417
131	p_{itv}. $l = 5$. Heuristic prop.. Align	4.0	0.5	12802
132	p_{itv}. $l = 5$. Left prop.. Align	4.0	0.5	12821

Table E.2: Selected setups for method C-BRAHMS

	Setup	Borda count	PaC	Time
1	P1	1.9	0.0	0
2	P2v6	2.2	0.2	8
3	P2v5	2.8	0.2	68
4	P2	2.7	0.2	123
5	P3	2.7	0.2	742

Table E.3: Selected setups for method PROMS

	Setup	Borda count	PaC	Time
1	r =4	4.0	0.4	99
2	r =8	4.0	0.4	193
3	r =12	4.0	0.4	272
4	r =16	4.0	0.4	393
5	r =20	4.0	0.4	477
6	r =24	4.0	0.4	503
7	r =28	4.0	0.4	785

Table E.4: Selected setups for method Uitdenboderg

	Setup	Borda count	PaC	Time
1	2-grams	2.3	0.6	1
2	3-grams	2.4	0.6	2
3	4-grams	2.5	0.6	5
4	Edit distance	2.3	0.7	70

Table E.5: Selected setups for method Strings

	Setup	Borda count	PaC	Time
1	p_{hdc}. $k = 0.1$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	83
2	p_{hdc}. $k = 0.1$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.6	83
3	p_{hdc}. $k = 0.3$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.6	83
4	p_{itv}. $k = 0.1$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	83
5	p_{itv}. $k = 0.1$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.7	83
6	p_{itv}. $k = 0.9$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	83
7	p_{ift}. $k = 0.1$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	83
8	p_{ift}. $k = 0.1$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.7	83
9	p_{ift}. $k = 0.3$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	83
10	p_{ift}. $k = 0.5$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	83
11	p_{ift}. $k = 0.5$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.7	83
12	p_{ift}. $k = 0.7$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	83
13	p_{ift}. $k = 0.7$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.7	83
14	p_{ift}. $k = 0.9$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	83
15	p_{ift}. $k = 0.9$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.7	83
16	p_{hdc}. $k = 0.3$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	84
17	p_{hdc}. $k = 0.5$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	84
18	p_{hdc}. $k = 0.5$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.6	84
19	p_{hdc}. $k = 0.7$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	84
20	p_{hdc}. $k = 0.7$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.4	84
21	p_{hdc}. $k = 0.9$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.7	84
22	p_{itv}. $k = 0.3$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	84
23	p_{itv}. $k = 0.3$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.7	84
24	p_{itv}. $k = 0.5$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	84
25	p_{itv}. $k = 0.5$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.7	84
26	p_{itv}. $k = 0.7$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	84
27	p_{itv}. $k = 0.7$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.7	84
28	p_{itv}. $k = 0.9$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.7	84
29	p_{ift}. $k = 0.3$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.7	84
30	p_{c}. $k = 0.3$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	85
31	p_{c}. $k = 0.9$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	85
32	p_{hdc}. $k = 0.9$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	85
33	p_{c}. $k = 0.7$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.6	86
34	p_{c}. $k = 0.5$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.6	87
35	p_{c}. $k = 0.7$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	87
36	p_{12}. $k = 0.7$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.6	87
37	p_{c}. $k = 0.3$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.6	88
38	p_{c}. $k = 0.9$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.6	88
39	p_{12}. $k = 0.1$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.6	88
40	p_{12}. $k = 0.5$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.6	88
41	p_{12}. $k = 0.9$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.6	88
42	p_{12}. $k = 0.3$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	89
43	p_{12}. $k = 0.7$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	89
44	p_{12}. $k = 0.1$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	90

continued on next page

continued from previous page

	Setup	Borda count	PaC	Time
45	p_{12}. $k = 0.3$. r_{tabs}. decoupled. withRests. withNHT	4.0	0.6	90
46	p_{12}. $k = 0.5$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	90
47	p_{12}. $k = 0.9$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	90
48	p_c. $k = 0.5$. r_{tabs}. decoupled. withoutRests. withNHT	4.0	0.7	92
49	p_{40}. $k = 0.9$. r_{tabs}. coupled. withoutRests. withNHT	4.0	0.6	105
50	p_{hdc}. $k = 0.9$. r_{tabs}. coupled. withRests. withNHT	4.0	0.6	105
51	p_{ift}. $k = 0.9$. r_{tabs}. coupled. withoutRests. withNHT	4.0	0.6	105
52	p_{ift}. $k = 0.9$. r_{tabs}. coupled. withRests. withNHT	4.0	0.6	105
53	p_{21}. $k = 0.9$. r_{tabs}. coupled. withRests. withNHT	4.0	0.6	106
54	p_{40}. $k = 0.9$. r_{tabs}. coupled. withRests. withNHT	4.0	0.6	106
55	p_{itv}. $k = 0.9$. r_{tabs}. coupled. withoutRests. withNHT	4.0	0.6	106
56	p_{itv}. $k = 0.9$. r_{tabs}. coupled. withRests. withNHT	4.0	0.6	106
57	p_{hdc}. $k = 0.9$. r_{tabs}. coupled. withoutRests. withNHT	4.0	0.6	107
58	p_c. $k = 0.9$. r_{tabs}. coupled. withoutRests. withNHT	4.0	0.6	108
59	p_c. $k = 0.9$. r_{tabs}. coupled. withRests. withNHT	4.0	0.6	110
60	p_{12}. $k = 0.9$. r_{tabs}. coupled. withRests. withNHT	4.0	0.6	111
61	p_{12}. $k = 0.9$. r_{tabs}. coupled. withoutRests. withNHT	4.0	0.6	113

Table E.6: Selected setups for method Mongeau and Sankoff

	Setup	Borda count	PaC	Time
1	p_{ift}. $k = 0.1$. r_{dabs}. matrixPitch. withoutRests. withNHT	4.0	0.6	128
2	p_{ift}. $k = 0.1$. r_{dabs}. matrixPitch. withRests. withNHT	4.0	0.6	128
3	p_{ift}. $k = 0.1$. r_{ioi}. matrixPitch. withoutRests. withNHT	4.0	0.6	128
4	p_{ift}. $k = 0.5$. r_{dabs}. matrixPitch. withoutRests. withNHT	4.0	0.6	128
5	p_{ift}. $k = 0.5$. r_{dabs}. matrixPitch. withRests. withNHT	4.0	0.6	128
6	p_{ift}. $k = 0.5$. r_{ioi}. matrixPitch. withRests. withNHT	4.0	0.6	128
7	p_{ift}. $k = 0.1$. r_{ioi}. matrixPitch. withRests. withNHT	4.0	0.6	129

E.2 Setup selection results for polyphonic methods

Table E.7: Selected setups for method Trees

	Setup	Borda count	PaC	Time
5	Hellinger distance. $l = 2$.	0.4	0.6	153
2	Cosine similarity. $l = 2$.	0.3	0.6	101
15	Multisets distance. $l = 4$.	0.7	0.5	2083
20	Multisets distance. $l = 3$.	1.5	0.5	499
3	Multisets distance. $l = 2$.	0.7	0.5	103
12	Hellinger distance. $l = 3$.	0.3	0.5	694
10	Cosine similarity. $l = 3$.	0.3	0.5	528
9	Manhattan (L1). $l = 3$.	0.4	0.5	509

continued on next page

continued from previous page

	Setup	Borda count	PaC	Time
1	Overlap coefficient. $l = 2$.	0.3	0.5	97
21	Cosine similarity. $l = 4$.	1.2	0.5	2159
19	Manhattan (L1). $l = 2$.	1.0	0.5	105
16	Hellinger distance. $l = 4$.	0.4	0.5	2731
14	Manhattan (L1). $l = 4$.	0.3	0.5	1970
7	Overlap coefficient. $l = 3$.	0.2	0.5	505
8	Euclidean distance (L2). $l = 3$.	0.2	0.5	507
11	Probabilities. $l = 3$.	0.2	0.5	693
4	Probabilities. $l = 2$.	0.2	0.5	133
6	Log distance. $l = 2$.	0.6	0.5	348
13	Log distance. $l = 3$.	0.4	0.5	1607
17	Log distance. $l = 4$.	0.4	0.5	6137
18	Log distance. $l = 1$.	1.0	0.5	26

Table E.8: Selected setups for method PROMS

	Setup	Borda count	PaC	Time
2	r =12	0.8	0.6	272
5	r =4	1.3	0.6	99
1	r =8	0.8	0.6	193
6	r =16	1.7	0.6	393
7	r =24	1.5	0.6	503
3	r =20	0.8	0.6	477
4	r =28	0.8	0.5	785

Table E.9: Selected setups for method C-BRAHMS

	Setup	Borda count	PaC	Time
3	P2v6	2.5	0.5	8
4	P2v5	2.3	0.5	68
2	P3	1.5	0.2	742
5	P2	2.3	0.5	123
1	P1	1.4	0.0	0

E.3 Monophonic COVERS results

Table E.10: COVER Skyline results of selected methods

Method (and setups)	**MRR**	**PaC**	**Time**
C-BRAHMS:			
P2	0.20 ± 0.05	0.11 ± 0.04	19 ± 7
P2v5	0.21 ± 0.05	0.12 ± 0.04	1.02 ± 0.11
P2v6	0.20 ± 0.05	0.11 ± 0.03	0.051 ± 0.010
Graphs			
Pinto	0.17 ± 0.03	0.08 ± 0.03	6 ± 8
PROMS:			
PR, r =4	0.59 ± 0.04	0.44 ± 0.05	120 ± 120
PR, r =8	0.59 ± 0.04	0.45 ± 0.06	200 ± 200
String:			
p_{21}, k0.9, r_{tabs}, coupled, withRests, withNHT	0.47 ± 0.06	0.34 ± 0.06	90 ± 40
p_{40}, k0.9, r_{tabs}, coupled, withoutRests, withNHT	0.47 ± 0.06	0.34 ± 0.06	75 ± 7
p_{40}, k0.9, r_{tabs}, coupled, withRests, withNHT	0.47 ± 0.06	0.34 ± 0.06	100 ± 60
p_c, k0.3, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	100 ± 200
p_c, k0.3, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	59 ± 6
p_c, k0.5, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	61 ± 7
p_c, k0.5, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	70 ± 20
p_c, k0.7, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	61 ± 8
p_c, k0.7, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	59 ± 6
p_c, k0.9, r_{tabs}, coupled, withoutRests, withNHT	0.47 ± 0.06	0.34 ± 0.06	100 ± 60
p_c, k0.9, r_{tabs}, coupled, withRests, withNHT	0.47 ± 0.06	0.34 ± 0.06	75 ± 8
p_c, k0.9, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	59 ± 5
p_c, k0.9, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	60 ± 4
p_{12}, k0.1, r_{tabs}, decoupled, withoutRests, withNHT	0.45 ± 0.06	0.32 ± 0.07	80 ± 20
p_{12}, k0.1, r_{tabs}, decoupled, withRests, withNHT	0.45 ± 0.06	0.32 ± 0.07	47 ± 7
p_{12}, k0.3, r_{tabs}, decoupled, withoutRests, withNHT	0.45 ± 0.06	0.32 ± 0.07	60 ± 20
p_{12}, k0.3, r_{tabs}, decoupled, withRests, withNHT	0.45 ± 0.06	0.32 ± 0.07	50 ± 20
p_{12}, k0.5, r_{tabs}, decoupled, withoutRests, withNHT	0.45 ± 0.06	0.32 ± 0.07	51 ± 9
p_{12}, k0.5, r_{tabs}, decoupled, withRests, withNHT	0.45 ± 0.06	0.32 ± 0.07	50 ± 12
p_{12}, k0.7, r_{tabs}, decoupled, withoutRests, withNHT	0.45 ± 0.06	0.32 ± 0.07	48 ± 7
p_{12}, k0.7, r_{tabs}, decoupled, withRests, withNHT	0.45 ± 0.06	0.32 ± 0.07	90 ± 90

continued on next page

continued from previous page

Method (and setups)	**MRR**	**PaC**	**Time**
p_{12}, k0.9, r_{tabs}, coupled, withoutRests, withNHT	0.47 ± 0.06	0.34 ± 0.06	90 ± 30
p_{12}, k0.9, r_{tabs}, coupled, withRests, withNHT	0.47 ± 0.06	0.34 ± 0.06	77 ± 12
p_{12}, k0.9, r_{tabs}, decoupled, withoutRests, withNHT	0.45 ± 0.06	0.32 ± 0.07	60 ± 20
p_{12}, k0.9, r_{tabs}, decoupled, withRests, withNHT	0.45 ± 0.06	0.32 ± 0.07	50 ± 10
p_{hdc}, k0.1, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	80 ± 50
p_{hdc}, k0.1, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	61 ± 7
p_{hdc}, k0.3, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	60 ± 5
p_{hdc}, k0.3, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	70 ± 30
p_{hdc}, k0.5, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	60 ± 7
p_{hdc}, k0.5, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	61 ± 7
p_{hdc}, k0.7, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	100 ± 200
p_{hdc}, k0.7, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	61 ± 8
p_{hdc}, k0.9, r_{tabs}, coupled, withoutRests, withNHT	0.47 ± 0.06	0.34 ± 0.06	77 ± 10
p_{hdc}, k0.9, r_{tabs}, coupled, withRests, withNHT	0.47 ± 0.06	0.34 ± 0.06	90 ± 50
p_{hdc}, k0.9, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	80 ± 60
p_{hdc}, k0.9, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	80 ± 50
p_{itv}, k0.1, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	62 ± 9
p_{itv}, k0.1, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	60 ± 5
p_{itv}, k0.3, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	70 ± 40
p_{itv}, k0.3, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	59 ± 5
p_{itv}, k0.5, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	59 ± 6
p_{itv}, k0.5, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	58 ± 5
p_{itv}, k0.7, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	100 ± 200
p_{itv}, k0.7, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	70 ± 40
p_{itv}, k0.9, r_{tabs}, coupled, withoutRests, withNHT	0.47 ± 0.06	0.34 ± 0.06	75 ± 5
p_{itv}, k0.9, r_{tabs}, coupled, withRests, withNHT	0.47 ± 0.06	0.34 ± 0.06	75 ± 7
p_{itv}, k0.9, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	59 ± 5
p_{itv}, k0.9, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	59 ± 5
p_{ift}, k0.1, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	100 ± 110
p_{ift}, k0.1, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	60 ± 5
p_{ift}, k0.3, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	100 ± 70
p_{ift}, k0.3, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	61 ± 8
p_{ift}, k0.5, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	60 ± 6
p_{ift}, k0.5, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	59 ± 5
p_{ift}, k0.7, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	61 ± 7
p_{ift}, k0.7, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	80 ± 60
p_{ift}, k0.9, r_{tabs}, coupled, withoutRests, withNHT	0.47 ± 0.06	0.34 ± 0.06	77 ± 7
p_{ift}, k0.9, r_{tabs}, coupled, withRests, withNHT	0.47 ± 0.06	0.34 ± 0.06	120 ± 140
p_{ift}, k0.9, r_{tabs}, decoupled, withoutRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	60 ± 6
p_{ift}, k0.9, r_{tabs}, decoupled, withRests, withNHT	0.46 ± 0.06	0.33 ± 0.07	61 ± 9
M.Sankoff:			
p_{ift}, k0.1, r_{dabs}, matrixPitch, withoutRests, withNHT	0.46 ± 0.05	0.33 ± 0.06	98 ± 9
p_{ift}, k0.1, r_{dabs}, matrixPitch, withRests, withNHT	0.46 ± 0.05	0.33 ± 0.06	120 ± 40
p_{ift}, k0.1, r_{ioi}, matrixPitch, withoutRests, withNHT	0.46 ± 0.05	0.33 ± 0.06	140 ± 90

continued on next page

continued from previous page

Method (and setups)	MRR	PaC	Time
p_{ift}, k0.1, r_{ioi}, matrixPitch, withRests, withNHT	0.46 ± 0.05	0.33 ± 0.06	120 ± 30
p_{ift}, k0.5, r_{dabs}, matrixPitch, withoutRests, withNHT	0.45 ± 0.05	0.32 ± 0.05	140 ± 90
p_{ift}, k0.5, r_{dabs}, matrixPitch, withRests, withNHT	0.45 ± 0.05	0.32 ± 0.05	120 ± 30
p_{ift}, k0.5, r_{ioi}, matrixPitch, withRests, withNHT	0.45 ± 0.05	0.32 ± 0.05	110 ± 30
Trees:			
p_c, $l = 2$, Heuristic prop., Partial	0.49 ± 0.04	0.35 ± 0.04	26 ± 9
p_c, $l = 2$, Heuristic prop., Selkow	0.51 ± 0.02	0.38 ± 0.02	34 ± 3
p_c, $l = 2$, Heuristic prop., Shasha	0.51 ± 0.02	0.38 ± 0.02	120 ± 40
p_c, $l = 2$, Left prop., Selkow	0.50 ± 0.02	0.37 ± 0.02	34 ± 4
p_c, $l = 2$, Left prop., Shasha	0.51 ± 0.02	0.38 ± 0.02	120 ± 40
p_c, $l = 3$, Heuristic prop., Partial	0.48 ± 0.04	0.35 ± 0.04	110 ± 20
p_c, $l = 3$, Left prop., Selkow	0.49 ± 0.03	0.36 ± 0.04	119 ± 8
p_c, $l = 3$, Melodic prop., Selkow	0.49 ± 0.03	0.35 ± 0.04	150 ± 40
p_c, $l = 3$, Right prop., Selkow	0.43 ± 0.04	0.29 ± 0.05	119 ± 8
p_{hdc}, $l = 2$, Heuristic prop., Partial	0.51 ± 0.04	0.37 ± 0.03	25 ± 8
p_{hdc}, $l = 2$, Heuristic prop., Selkow	0.50 ± 0.03	0.37 ± 0.04	36 ± 6
p_{hdc}, $l = 2$, Heuristic prop., Shasha	0.50 ± 0.04	0.37 ± 0.04	120 ± 50
p_{hdc}, $l = 2$, Left prop., Align	0.32 ± 0.04	0.19 ± 0.04	200 ± 300
p_{hdc}, $l = 2$, Left prop., Partial	0.51 ± 0.04	0.37 ± 0.03	25 ± 9
p_{hdc}, $l = 2$, Left prop., Selkow	0.50 ± 0.03	0.37 ± 0.04	34 ± 3
p_{hdc}, $l = 2$, Left prop., Shasha	0.50 ± 0.03	0.37 ± 0.04	120 ± 40
p_{hdc}, $l = 2$, Melodic prop., Partial	0.50 ± 0.03	0.36 ± 0.03	25 ± 9
p_{hdc}, $l = 2$, Melodic prop., Selkow	0.50 ± 0.03	0.37 ± 0.03	36 ± 7
p_{hdc}, $l = 2$, Melodic prop., Shasha	0.49 ± 0.03	0.36 ± 0.04	120 ± 40
p_{hdc}, $l = 2$, No prop., Align	0.26 ± 0.05	0.15 ± 0.04	150 ± 70
p_{hdc}, $l = 2$, No prop., Selkow	0.31 ± 0.06	0.19 ± 0.05	34 ± 3
p_{hdc}, $l = 3$, Heuristic prop., Partial	0.50 ± 0.04	0.36 ± 0.03	130 ± 50
p_{hdc}, $l = 3$, Heuristic prop., Selkow	0.48 ± 0.04	0.34 ± 0.05	126 ± 10
p_{hdc}, $l = 3$, Left prop., Partial	0.50 ± 0.04	0.36 ± 0.03	110 ± 20
p_{hdc}, $l = 3$, Left prop., Selkow	0.48 ± 0.04	0.34 ± 0.05	128 ± 14
p_{hdc}, $l = 3$, Melodic prop., Partial	0.49 ± 0.03	0.36 ± 0.03	110 ± 20
p_{hdc}, $l = 3$, Melodic prop., Selkow	0.47 ± 0.04	0.33 ± 0.05	120 ± 7
p_{hdc}, $l = 3$, Right prop., Selkow	0.41 ± 0.04	0.27 ± 0.04	119 ± 7
p_{itv}, $l = 2$, Heuristic prop., Align	0.27 ± 0.04	0.16 ± 0.04	160 ± 70
p_{itv}, $l = 2$, Heuristic prop., Shasha	0.42 ± 0.04	0.30 ± 0.04	120 ± 50
p_{itv}, $l = 2$, Left prop., Partial	0.46 ± 0.04	0.32 ± 0.05	25 ± 7
p_{itv}, $l = 2$, Left prop., Selkow	0.43 ± 0.04	0.31 ± 0.04	36 ± 8
p_{itv}, $l = 2$, Left prop., Shasha	0.42 ± 0.04	0.30 ± 0.04	300 ± 400
p_{itv}, $l = 2$, Melodic prop., Align	0.26 ± 0.04	0.15 ± 0.04	160 ± 70
p_{itv}, $l = 2$, Melodic prop., Partial	0.46 ± 0.04	0.33 ± 0.05	27 ± 8
p_{itv}, $l = 2$, Melodic prop., Selkow	0.42 ± 0.03	0.29 ± 0.02	36 ± 6
p_{itv}, $l = 2$, Melodic prop., Shasha	0.42 ± 0.03	0.29 ± 0.03	130 ± 60
p_{itv}, $l = 2$, No prop., Selkow	0.31 ± 0.05	0.19 ± 0.05	34 ± 4
p_{itv}, $l = 2$, Right prop., Align	0.24 ± 0.03	0.13 ± 0.03	160 ± 70

continued on next page

continued from previous page

Method (and setups)	MRR	PaC	Time
p_{itv}, $l = 2$, Right prop., Selkow	0.37 ± 0.03	0.23 ± 0.02	35 ± 3
p_{itv}, $l = 2$, Right prop., Shasha	0.36 ± 0.04	0.23 ± 0.04	120 ± 50
p_{itv}, $l = 3$, Heuristic prop., Partial	0.44 ± 0.04	0.30 ± 0.04	110 ± 20
p_{itv}, $l = 3$, Heuristic prop., Selkow	0.40 ± 0.05	0.27 ± 0.04	120 ± 7
p_{itv}, $l = 3$, Left prop., Partial	0.44 ± 0.04	0.30 ± 0.04	100 ± 20
p_{itv}, $l = 3$, Left prop., Selkow	0.40 ± 0.05	0.26 ± 0.04	120 ± 7
p_{itv}, $l = 3$, Melodic prop., Partial	0.43 ± 0.04	0.29 ± 0.04	110 ± 20
p_{itv}, $l = 3$, Melodic prop., Selkow	0.39 ± 0.04	0.26 ± 0.04	120 ± 8
p_{ift}, $l = 2$, Heuristic prop., Selkow	0.47 ± 0.05	0.34 ± 0.05	35 ± 6
p_{ift}, $l = 2$, Heuristic prop., Shasha	0.47 ± 0.05	0.34 ± 0.05	110 ± 40
p_{ift}, $l = 2$, Left prop., Shasha	0.47 ± 0.05	0.34 ± 0.06	130 ± 50
p_{ift}, $l = 2$, Melodic prop., Align	0.32 ± 0.05	0.20 ± 0.04	160 ± 70
p_{ift}, $l = 2$, Melodic prop., Shasha	0.47 ± 0.05	0.35 ± 0.05	110 ± 40
p_{ift}, $l = 3$, Heuristic prop., Selkow	0.47 ± 0.05	0.34 ± 0.06	130 ± 20
p_{ift}, $l = 3$, Left prop., Partial	0.35 ± 0.05	0.23 ± 0.05	100 ± 20
p_{ift}, $l = 3$, Left prop., Selkow	0.47 ± 0.05	0.34 ± 0.06	119 ± 7
p_{ift}, $l = 3$, Melodic prop., Partial	0.35 ± 0.05	0.23 ± 0.04	100 ± 20
p_{ift}, $l = 3$, Melodic prop., Selkow	0.46 ± 0.05	0.33 ± 0.06	119 ± 7
UM:			
Uitdenboderg, alg, n, gramn, 2, grams	0.11 ± 0.03	0.06 ± 0.02	3 ± 2
Uitdenboderg, 3-grams	0.19 ± 0.04	0.11 ± 0.03	5 ± 3
Uitdenboderg, 4-grams	0.27 ± 0.06	0.16 ± 0.04	13 ± 13
Uitdenboderg, Edit distance	0.17 ± 0.03	0.10 ± 0.03	70 ± 50

E.4 Polyphonic COVERS results

Table E.11: COVER results of selected methods

Method (and setups)	MRR	PaC	Time
C-BRAHMS:			
P2	0.23 ± 0.04	0.12 ± 0.04	190 ± 20
P2v5	0.26 ± 0.04	0.15 ± 0.03	51 ± 7
P2v6	0.25 ± 0.05	0.15 ± 0.04	3.9 ± 0.6
PROMS:			
PR, r =12	0.68 ± 0.04	0.54 ± 0.04	180 ± 12

continued on next page

continued from previous page

Method (and setups)	MRR	PaC	Time
PR, r =16	0.67 ± 0.04	0.54 ± 0.05	230 ± 20
PR, r =20	0.49 ± 0.04	0.34 ± 0.04	270 ± 20
PR, r =24	0.67 ± 0.04	0.54 ± 0.05	320 ± 20
PR, r =28	0.38 ± 0.05	0.24 ± 0.04	390 ± 20
PR, r =4	0.68 ± 0.04	0.56 ± 0.04	83 ± 7
PR, r =8	0.69 ± 0.04	0.55 ± 0.03	132 ± 8
Trees:			
1, Cosine similarity, $l = 1$	0.61 ± 0.00	0.52 ± 0.00	2.8 ± 0.2
1, Cosine similarity, $l = 2$	0.62 ± 0.00	0.53 ± 0.00	47 ± 2
1, Cosine similarity, $l = 3$	0.62 ± 0.00	0.52 ± 0.00	246 ± 14
1, Cosine similarity, $l = 4$	0.62 ± 0.01	0.53 ± 0.01	1070 ± 50
Euclidean distance, $l = 1$	0.43 ± 0.01	0.31 ± 0.02	2.5 ± 0.2
Euclidean distance, $l = 2$	0.44 ± 0.01	0.33 ± 0.02	44 ± 2
Euclidean distance, $l = 3$	0.43 ± 0.01	0.33 ± 0.02	229 ± 14
Euclidean distance, $l = 4$	0.44 ± 0.01	0.32 ± 0.03	1010 ± 70
Harmonic mean, $l = 1$	0.60 ± 0.01	0.51 ± 0.00	4.0 ± 0.2
Harmonic mean, $l = 2$	0.61 ± 0.00	0.53 ± 0.00	65 ± 3
Harmonic mean, $l = 3$	0.61 ± 0.01	0.52 ± 0.01	330 ± 20
Harmonic mean, $l = 4$	0.62 ± 0.01	0.52 ± 0.00	1380 ± 70
Hellinger distance, $l = 1$	0.49 ± 0.05	0.36 ± 0.04	3.8 ± 0.3
Hellinger distance, $l = 2$	0.47 ± 0.04	0.36 ± 0.04	62 ± 5
Hellinger distance, $l = 3$	0.46 ± 0.03	0.35 ± 0.04	320 ± 30
Hellinger distance, $l = 4$	0.47 ± 0.03	0.36 ± 0.03	1340 ± 100
Jackard coefficient, $l = 1$	0.48 ± 0.01	0.37 ± 0.01	2.6 ± 0.2
Jackard coefficient, $l = 2$	0.51 ± 0.03	0.41 ± 0.02	44 ± 2
Jackard coefficient, $l = 3$	0.52 ± 0.03	0.43 ± 0.02	226 ± 13
Jackard coefficient, $l = 4$	0.52 ± 0.03	0.43 ± 0.03	970 ± 50
Log distance, $l = 1$	0.63 ± 0.02	0.55 ± 0.01	13.3 ± 0.7
Log distance, $l = 2$	0.63 ± 0.03	0.56 ± 0.02	182 ± 8
Log distance, $l = 3$	0.62 ± 0.03	0.54 ± 0.01	820 ± 50
Log distance, $l = 4$	0.62 ± 0.03	0.53 ± 0.01	3330 ± 130
Manhattan, $l = 1$	0.57 ± 0.01	0.46 ± 0.01	2.35 ± 0.14
Manhattan, $l = 2$	0.56 ± 0.00	0.45 ± 0.04	41 ± 2
Manhattan, $l = 3$	0.56 ± 0.00	0.45 ± 0.04	214 ± 13
Manhattan, $l = 4$	0.57 ± 0.01	0.45 ± 0.01	950 ± 50
Matching coefficient, $l = 1$	0.45 ± 0.01	0.32 ± 0.01	2.49 ± 0.13
Matching coefficient, $l = 2$	0.50 ± 0.03	0.40 ± 0.04	40.2 ± 1.1
Matching coefficient, $l = 3$	0.50 ± 0.06	0.39 ± 0.07	208 ± 10
Matching coefficient, $l = 4$	0.50 ± 0.06	0.40 ± 0.05	910 ± 40
Multiset distance, $l = 1$	0.54 ± 0.01	0.41 ± 0.01	2.4 ± 0.2

continued on next page

continued from previous page

Method (and setups)	MRR	PaC	Time
Multiset distance, $l = 2$	0.55 ± 0.01	0.45 ± 0.04	41 ± 2
Multiset distance, $l = 3$	0.55 ± 0.00	0.43 ± 0.02	215 ± 14
Multiset distance, $l = 4$	0.56 ± 0.01	0.45 ± 0.01	950 ± 50
Overlap coefficient, $l = 1$	0.51 ± 0.01	0.39 ± 0.02	2.5 ± 0.2
Overlap coefficient, $l = 2$	0.54 ± 0.03	0.44 ± 0.01	43 ± 2
Overlap coefficient, $l = 3$	0.56 ± 0.02	0.44 ± 0.01	226 ± 14
Overlap coefficient, $l = 4$	0.56 ± 0.04	0.44 ± 0.03	980 ± 50
Variational distance, $l = 1$	0.40 ± 0.03	0.28 ± 0.02	3.8 ± 0.2
Variational distance, $l = 2$	0.40 ± 0.04	0.29 ± 0.03	63 ± 3
Variational distance, $l = 3$	0.40 ± 0.04	0.28 ± 0.04	330 ± 20
Variational distance, $l = 4$	0.39 ± 0.04	0.27 ± 0.04	1360 ± 70

E.5 MIREX results

Table E.12: MIREX ADR results for method Metric trees

Setup	ADR	Time
p_{hdc}, $l = 3$, Left prop., Selkow	0.5	0.54
p_{hdc}, $l = 3$, Heuristic prop., Selkow	0.5	0.54
p_{itv}, $l = 2$, Left prop., Shasha	0.5	0.53
p_{itv}, $l = 3$, Left prop., Partial	0.5	0.53
p_{ift}, $l = 3$, Heuristic prop., Selkow	0.5	0.52
p_{ift}, $l = 2$, Left prop., Shasha	0.5	0.52
p_{itv}, $l = 2$, Melodic prop., Shasha	0.5	0.51
p_{itv}, $l = 2$, Heuristic prop., Shasha	0.5	0.51
p_{ift}, $l = 3$, Left prop., Partial	0.5	0.50
p_{itv}, $l = 3$, Heuristic prop., Partial	0.5	0.49
p_{ift}, $l = 3$, Melodic prop., Partial	0.5	0.49
p_c, $l = 3$, Melodic prop., Selkow	0.5	0.46
p_{hdc}, $l = 2$, Melodic prop., Selkow	0.4	0.43
p_{hdc}, $l = 3$, Heuristic prop., Partial	0.4	0.41
p_{hdc}, $l = 2$, Melodic prop., Partial	0.4	0.41
p_{hdc}, $l = 2$, Left prop., Align	0.4	0.36

Table E.13: MIREX ADR results for method PROMS

Setup	ADR	Time
PR, r =8	0.5	0.53
PR, r =4	0.5	0.52

Table E.14: MIREX ADR results for method Uitdenboderg

Setup	ADR	Time
Uitdenboderg, alg, string, Edit distance, distance, 112	0.7	0.68
Uitdenboderg, alg, n, gramn, 4, grams	0.7	0.65
Uitdenboderg, alg, n, gramn, 3, grams	0.6	0.63
Uitdenboderg, alg, n, gramn, 2, grams	0.5	0.51

Table E.15: MIREX ADR results for method Strings

Setup	ADR	Time
p_{ift}, k0. 9, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{ift}, k0. 7, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{ift}, k0. 5, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{ift}, k0. 3, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{ift}, k0. 1, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{itv}, k0. 9, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{itv}, k0. 7, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{itv}, k0. 5, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{itv}, k0. 3, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{itv}, k0. 1, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{hdc}, k0. 9, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{hdc}, k0. 7, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{hdc}, k0. 5, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{hdc}, k0. 3, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{hdc}, k0. 1, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_c, k0. 9, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_c, k0. 7, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_c, k0. 5, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_c, k0. 3, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{12}, k0. 9, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{12}, k0. 7, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{12}, k0. 5, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{12}, k0. 3, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{12}, k0. 1, r_{tabs}, decoupled, withoutRests, withNHT	0.6	0.62
p_{40}, k0. 9, r_{tabs}, coupled, withoutRests, withNHT	0.6	0.62
p_{ift}, k0. 9, r_{tabs}, coupled, withoutRests, withNHT	0.6	0.62
p_{itv}, k0. 9, r_{tabs}, coupled, withoutRests, withNHT	0.6	0.62
p_{hdc}, k0. 9, r_{tabs}, coupled, withoutRests, withNHT	0.6	0.62
p_c, k0. 9, r_{tabs}, coupled, withoutRests, withNHT	0.6	0.62
p_{12}, k0. 9, r_{tabs}, coupled, withoutRests, withNHT	0.6	0.62
p_{40}, k0. 9, r_{tabs}, coupled, withRests, withNHT	0.6	0.58
p_{21}, k0. 9, r_{tabs}, coupled, withRests, withNHT	0.6	0.58
p_{ift}, k0. 9, r_{tabs}, coupled, withRests, withNHT	0.6	0.58
p_{itv}, k0. 9, r_{tabs}, coupled, withRests, withNHT	0.6	0.58

continued on next page

continued from previous page

Setup	ADR	Time
p_{hdc}, k0. 9, r_{tabs}, coupled, withRests, withNHT	0.6	0.58
p_c, k0. 9, r_{tabs}, coupled, withRests, withNHT	0.6	0.58
p_{12}, k0. 9, r_{tabs}, coupled, withRests, withNHT	0.6	0.58
p_{ift}, k0. 9, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{ift}, k0. 7, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{ift}, k0. 5, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{ift}, k0. 3, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{ift}, k0. 1, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{itv}, k0. 9, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{itv}, k0. 7, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{itv}, k0. 5, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{itv}, k0. 3, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{itv}, k0. 1, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{hdc}, k0. 9, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{hdc}, k0. 7, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{hdc}, k0. 5, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{hdc}, k0. 3, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{hdc}, k0. 1, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_c, k0. 9, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_c, k0. 7, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_c, k0. 5, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_c, k0. 3, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{12}, k0. 9, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{12}, k0. 7, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{12}, k0. 5, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{12}, k0. 3, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58
p_{12}, k0. 1, r_{tabs}, decoupled, withRests, withNHT	0.6	0.58

Table E.16: MIREX ADR results for method Mongeau and Sankoff

Setup	ADR	Time
p_{ift}, k0. 5, r_{dabs}, matrixPitch, withoutRests, withNHT	0.6	0.58
p_{ift}, k0. 1, r_{dabs}, matrixPitch, withoutRests, withNHT	0.5	0.55
p_{ift}, k0. 1, r_{ioi}, matrixPitch, withoutRests, withNHT	0.5	0.54
p_{ift}, k0. 5, r_{dabs}, matrixPitch, withRests, withNHT	0.5	0.54
p_{ift}, k0. 5, r_{ioi}, matrixPitch, withRests, withNHT	0.5	0.53
p_{ift}, k0. 1, r_{dabs}, matrixPitch, withRests, withNHT	0.5	0.53
p_{ift}, k0. 1, r_{ioi}, matrixPitch, withRests, withNHT	0.5	0.53

E.6 Global monophonic results

Table E.17: Global monophonic results. 'TA' stands for trees, 'UM' for Uitdenboderg, 'PR' for PROMS, 'S' for strings, 'MS' for Monguea-Sankoff, and 'MRX' stands for MIREX and 'B.C' for *Borda count*

Setup	PSCL	104	VARM	ICPS	MRX	COVERS	**Avg.**	**Time**	**B.C.**
TA:p_c. $l = 2$. Left prop.. Shasha	1.0	0.7	0.7	1.0	0.4	0.51 ± 0.02	0.7 ± 0.3	120 ± 40	2.6
TA:p_c. $l = 2$. Heuristic prop.. Shasha	1.0	0.8	0.7	1.0	0.4	0.51 ± 0.02	0.7 ± 0.3	120 ± 40	2.6
TA:p_c. $l = 2$. Heuristic prop.. Selkow	1.0	0.7	0.7	1.0	0.4	0.51 ± 0.02	0.7 ± 0.3	34 ± 3	2.6
TA:p_{hdc}. $l = 2$. Melodic prop.. Selkow	1.0	0.8	0.8	1.0	0.4	0.50 ± 0.03	0.8 ± 0.3	36 ± 7	2.5
UM:Uitdenboderg. Edit distance	1.0	0.9	0.7	0.9	0.7	0.17 ± 0.03	0.7 ± 0.3	69 ± 8	2.4
TA:p_{hdc}. $l = 3$. Left prop.. Selkow	1.0	0.8	0.8	1.0	0.5	0.48 ± 0.04	0.8 ± 0.2	128 ± 14	2.4
TA:p_{hdc}. $l = 3$. Heuristic prop.. Selkow	1.0	0.8	0.8	1.0	0.5	0.48 ± 0.04	0.8 ± 0.2	126 ± 10	2.4
TA:p_{hdc}. $l = 2$. Melodic prop.. Shasha	1.0	0.8	0.8	1.0	0.4	0.49 ± 0.03	0.8 ± 0.3	120 ± 40	2.4
TA:p_{hdc}. $l = 2$. Left prop.. Shasha	1.0	0.8	0.8	1.0	0.5	0.50 ± 0.03	0.8 ± 0.2	120 ± 40	2.4
TA:p_{hdc}. $l = 2$. Left prop.. Selkow	1.0	0.8	0.8	1.0	0.4	0.50 ± 0.03	0.8 ± 0.2	34 ± 3	2.4
TA:p_{hdc}. $l = 2$. Heuristic prop.. Shasha	1.0	0.8	0.8	1.0	0.4	0.50 ± 0.04	0.8 ± 0.2	120 ± 50	2.4
TA:p_{hdc}. $l = 2$. Heuristic prop.. Selkow	1.0	0.8	0.8	1.0	0.4	0.50 ± 0.03	0.8 ± 0.2	36 ± 6	2.4
TA:p_c. $l = 2$. Left prop.. Selkow	1.0	0.7	0.7	1.0	0.4	0.50 ± 0.02	0.7 ± 0.3	34 ± 4	2.4
TA:p_{hdc}. $l = 3$. Right prop.. Selkow	0.9	0.9	0.8	1.0	0.5	0.41 ± 0.04	0.8 ± 0.2	119 ± 7	2.3
TA:p_{hdc}. $l = 3$. Melodic prop.. Selkow	1.0	0.8	0.8	1.0	0.5	0.47 ± 0.04	0.8 ± 0.2	120 ± 7	2.3
TA:p_c. $l = 3$. Melodic prop.. Selkow	1.0	0.8	0.7	1.0	0.5	0.49 ± 0.03	0.8 ± 0.2	150 ± 40	2.3
TA:p_c. $l = 3$. Left prop.. Selkow	1.0	0.8	0.8	1.0	0.5	0.49 ± 0.03	0.8 ± 0.2	119 ± 8	2.3
TA:p_{hdc}. $l = 2$. Left prop.. Align	1.0	0.9	0.6	1.0	0.4	0.32 ± 0.04	0.7 ± 0.3	200 ± 300	2.2
TA:p_{hdc}. $l = 2$. Left prop.. Partial	1.0	0.8	0.8	0.9	0.4	0.51 ± 0.04	0.7 ± 0.2	25 ± 9	2.1
TA:p_{hdc}. $l = 2$. Melodic prop.. Partial	1.0	0.9	0.8	0.9	0.4	0.50 ± 0.03	0.7 ± 0.2	25 ± 9	2.0
UM:Uitdenboderg. 4-grams	1.0	0.9	0.4	0.7	0.7	0.27 ± 0.06	0.7 ± 0.3	13 ± 13	1.9
UM:Uitdenboderg. 3-grams	1.0	0.9	0.5	0.7	0.6	0.19 ± 0.04	0.7 ± 0.3	5 ± 3	1.9
TA:p_{itv}. $l = 3$. Melodic prop.. Selkow	1.0	0.9	0.9	1.0	0.6	0.39 ± 0.04	0.8 ± 0.3	120 ± 8	1.9
TA:p_{hdc}. $l = 3$. Left prop.. Partial	1.0	0.8	0.8	0.9	0.4	0.50 ± 0.04	0.7 ± 0.2	110 ± 20	1.9
TA:p_{hdc}. $l = 3$. Melodic prop.. Partial	1.0	0.8	0.7	0.9	0.4	0.49 ± 0.03	0.7 ± 0.2	110 ± 20	1.8
TA:p_c. $l = 3$. Heuristic prop.. Partial	1.0	0.7	0.6	0.9	0.3	0.48 ± 0.04	0.7 ± 0.3	110 ± 20	1.8
TA:p_c. $l = 2$. Heuristic prop.. Partial	1.0	0.7	0.6	0.9	0.3	0.49 ± 0.04	0.7 ± 0.3	26 ± 9	1.8
TA:p_{hdc}. $l = 2$. Heuristic prop.. Partial	1.0	0.8	0.8	0.9	0.4	0.51 ± 0.04	0.7 ± 0.2	25 ± 8	1.7
TA:p_c. $l = 3$. Right prop.. Selkow	0.9	0.9	0.8	1.0	0.5	0.43 ± 0.04	0.8 ± 0.3	119 ± 8	1.7
TA:p_{itv}. $l = 3$. Left prop.. Selkow	1.0	0.8	0.8	1.0	0.5	0.40 ± 0.05	0.8 ± 0.2	120 ± 7	1.5
TA:p_{itv}. $l = 2$. Heuristic prop.. Align	1.0	0.8	0.8	0.9	0.4	0.27 ± 0.04	0.7 ± 0.3	160 ± 70	1.5
TA:p_{hdc}. $l = 3$. Heuristic prop.. Partial	1.0	0.7	0.8	0.9	0.4	0.50 ± 0.04	0.7 ± 0.2	130 ± 50	1.5
TA:p_{itv}. $l = 3$. Heuristic prop.. Selkow	1.0	0.8	0.8	1.0	0.5	0.40 ± 0.05	0.8 ± 0.2	120 ± 7	1.4
TA:p_{itv}. $l = 2$. Left prop.. Partial	1.0	0.8	0.8	0.8	0.5	0.46 ± 0.04	0.7 ± 0.2	25 ± 7	1.4
TA:p_{itv}. $l = 2$. Melodic prop.. Align	1.0	0.8	0.8	0.9	0.4	0.26 ± 0.04	0.7 ± 0.3	160 ± 70	1.3
PR:PR. r =8	0.9	0.7	0.9	0.6	0.5	0.59 ± 0.04	0.7 ± 0.2	200 ± 200	1.3
MS:p_{ift}. k0.1. r_{ioi}. matrixPitch. withoutRests. withNHT	0.9	0.8	1.0	0.7	0.5	0.46 ± 0.05	0.7 ± 0.2	140 ± 90	1.3
UM:Uitdenboderg. 2-grams	1.0	0.8	0.6	0.7	0.5	0.11 ± 0.03	0.6 ± 0.3	1.14 ± 0.03	1.2
TA:p_{itv}. $l = 2$. Right prop.. Align	0.9	0.8	0.8	1.0	0.4	0.24 ± 0.03	0.7 ± 0.3	160 ± 70	1.2
PR:PR. r =4	0.9	0.7	0.8	0.5	0.5	0.59 ± 0.04	0.7 ± 0.2	110 ± 130	1.2
TA:p_{itv}. $l = 2$. No prop.. Selkow	0.7	0.6	0.7	1.0	0.4	0.31 ± 0.05	0.6 ± 0.3	34 ± 4	1.1
TA:p_{hdc}. $l = 2$. No prop.. Selkow	0.7	0.6	0.7	1.0	0.3	0.31 ± 0.06	0.6 ± 0.3	34 ± 3	1.1
TA:p_{hdc}. $l = 2$. No prop.. Align	0.8	0.7	0.7	1.0	0.3	0.26 ± 0.05	0.6 ± 0.3	150 ± 70	1.1
CBRAHMS:cbrahmsqp26	0.3	0.6	0.8	1.0	0.6	0.20 ± 0.05	0.6 ± 0.3	0.051 ± 0.010	1.1

continued on next page

continued from previous page

Setup	PSCL	104	VARM	ICPS	MRX	COVERS	**Avg.**	**Time**	**B.C.**
TA:p_{itv}. $l = 2$. Left prop.. Shasha	1.0	0.8	0.9	0.9	0.5	0.42 ± 0.04	0.8 ± 0.2	300 ± 400	1.0
TA:p_{itv}. $l = 2$. Left prop.. Selkow	1.0	0.8	0.9	0.9	0.5	0.43 ± 0.04	0.7 ± 0.2	36 ± 8	1.0
TA:p_{itv}. $l = 2$. Melodic prop.. Shasha	1.0	0.9	0.9	0.8	0.5	0.42 ± 0.03	0.8 ± 0.2	130 ± 60	0.9
TA:p_{itv}. $l = 2$. Melodic prop.. Selkow	1.0	0.8	0.9	0.8	0.5	0.42 ± 0.03	0.7 ± 0.2	36 ± 6	0.9
TA:p_{itv}. $l = 2$. Heuristic prop.. Shasha	1.0	0.8	0.9	0.9	0.5	0.42 ± 0.04	0.7 ± 0.2	120 ± 50	0.9
TA:p_{itv}. $l = 3$. Heuristic prop.. Partial	1.0	0.8	0.8	0.8	0.5	0.44 ± 0.04	0.7 ± 0.2	110 ± 20	0.8
MS:p_{ift}. k0.1. r_{ioi}. matrixPitch. withRests. withNHT	0.9	0.8	0.9	0.7	0.5	0.46 ± 0.05	0.7 ± 0.2	120 ± 30	0.8
MS:p_{ift}. k0.1. r_{dabs}. matrixPitch. withRests. withNHT	1.0	0.8	0.9	0.7	0.5	0.46 ± 0.05	0.7 ± 0.2	120 ± 40	0.8
TA:p_{itv}. $l = 3$. Left prop.. Partial	1.0	0.7	0.8	0.8	0.5	0.44 ± 0.04	0.7 ± 0.2	100 ± 20	0.7
TA:p_{itv}. $l = 2$. Right prop.. Shasha	0.9	0.9	0.8	0.9	0.5	0.36 ± 0.04	0.7 ± 0.2	120 ± 50	0.7
TA:p_{itv}. $l = 2$. Right prop.. Selkow	0.9	0.9	0.8	0.9	0.4	0.37 ± 0.03	0.7 ± 0.2	35 ± 3	0.7
TA:p_{itv}. $l = 2$. Melodic prop.. Partial	1.0	0.8	0.9	0.7	0.5	0.46 ± 0.04	0.7 ± 0.2	27 ± 8	0.7
TA:p_{ift}. $l = 3$. Heuristic prop.. Selkow	1.0	0.9	0.7	0.9	0.5	0.47 ± 0.05	0.7 ± 0.2	130 ± 20	0.6
S:p_{itv}. k0.9. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	59 ± 5	0.6
S:p_{itv}. k0.5. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	59 ± 6	0.6
S:p_{itv}. k0.3. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	70 ± 40	0.6
S:p_{itv}. k0.1. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	62 ± 9	0.6
S:p_{ift}. k0.7. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	61 ± 7	0.6
S:p_{ift}. k0.5. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	60 ± 6	0.6
S:p_{ift}. k0.3. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	100 ± 70	0.6
S:p_{ift}. k0.1. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	100 ± 110	0.6
S:p_{hdc}. k0.9. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	80 ± 60	0.6
S:p_{hdc}. k0.7. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	100 ± 200	0.6
S:p_{hdc}. k0.5. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	60 ± 7	0.6
S:p_{hdc}. k0.3. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	60 ± 5	0.6
S:p_c. k0.9. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	59 ± 5	0.6
S:p_c. k0.7. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	61 ± 8	0.6
S:p_c. k0.5. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	61 ± 7	0.6
MS:p_{ift}. k0.5. r_{ioi}. matrixPitch. withRests. withNHT	1.0	0.9	0.9	0.7	0.5	0.45 ± 0.05	0.7 ± 0.2	110 ± 30	0.6
TA:p_{itv}. $l = 3$. Melodic prop.. Partial	1.0	0.8	0.8	0.8	0.5	0.43 ± 0.04	0.7 ± 0.2	110 ± 20	0.5
TA:p_{ift}. $l = 3$. Melodic prop.. Selkow	1.0	0.9	0.8	0.9	0.5	0.46 ± 0.05	0.7 ± 0.2	119 ± 7	0.5
TA:p_{ift}. $l = 3$. Left prop.. Selkow	1.0	0.9	0.7	0.9	0.5	0.47 ± 0.05	0.7 ± 0.2	119 ± 7	0.5
TA:p_{ift}. $l = 2$. Left prop.. Shasha	1.0	0.7	0.7	0.9	0.5	0.47 ± 0.05	0.7 ± 0.2	130 ± 50	0.5
TA:p_{ift}. $l = 2$. Heuristic prop.. Shasha	1.0	0.7	0.7	0.9	0.5	0.47 ± 0.05	0.7 ± 0.2	110 ± 40	0.5
TA:p_{ift}. $l = 2$. Heuristic prop.. Selkow	1.0	0.8	0.7	0.9	0.5	0.47 ± 0.05	0.7 ± 0.2	35 ± 6	0.5
S:p_{itv}. k0.9. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	59 ± 5	0.5
S:p_{itv}. k0.9. r_{tabs}. coupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.47 ± 0.06	0.7 ± 0.2	75 ± 5	0.5
S:p_{itv}. k0.9. r_{tabs}. coupled. withRests. withNHT	1.0	0.8	0.7	0.8	0.6	0.47 ± 0.06	0.7 ± 0.2	75 ± 7	0.5
S:p_{itv}. k0.7. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.45 ± 0.06	0.7 ± 0.2	100 ± 200	0.5
S:p_{itv}. k0.7. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	70 ± 40	0.5
S:p_{itv}. k0.5. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	58 ± 5	0.5
S:p_{itv}. k0.3. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	59 ± 5	0.5
S:p_{itv}. k0.1. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	60 ± 5	0.5
S:p_{ift}. k0.9. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.45 ± 0.06	0.7 ± 0.2	57 ± 9	0.5
S:p_{ift}. k0.9. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	61 ± 9	0.5
S:p_{ift}. k0.9. r_{tabs}. coupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.47 ± 0.06	0.7 ± 0.2	77 ± 7	0.5
S:p_{ift}. k0.9. r_{tabs}. coupled. withRests. withNHT	1.0	0.8	0.7	0.8	0.6	0.47 ± 0.06	0.7 ± 0.2	120 ± 140	0.5
S:p_{ift}. k0.7. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	80 ± 60	0.5
S:p_{ift}. k0.5. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	59 ± 5	0.5
S:p_{ift}. k0.3. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	61 ± 8	0.5

continued on next page

continued from previous page

Setup	PSCL	104	VARM	ICPS	MRX	COVERS	**Avg.**	**Time**	**B.C.**
S:p_{ift}. k0.1. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	60 ± 5	0.5
S:p_{hdc}. k0.9. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	80 ± 50	0.5
S:p_{hdc}. k0.9. r_{tabs}. coupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.47 ± 0.06	0.7 ± 0.2	77 ± 10	0.5
S:p_{hdc}. k0.9. r_{tabs}. coupled. withRests. withNHT	1.0	0.8	0.7	0.8	0.6	0.47 ± 0.06	0.7 ± 0.2	90 ± 50	0.5
S:p_{hdc}. k0.7. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	61 ± 8	0.5
S:p_{hdc}. k0.5. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	61 ± 7	0.5
S:p_{hdc}. k0.3. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	70 ± 30	0.5
S:p_{hdc}. k0.1. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.45 ± 0.06	0.7 ± 0.2	85 ± 9	0.5
S:p_{hdc}. k0.1. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	61 ± 7	0.5
S:p_c. k0.9. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	60 ± 4	0.5
S:p_c. k0.9. r_{tabs}. coupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.47 ± 0.06	0.7 ± 0.2	100 ± 60	0.5
S:p_c. k0.9. r_{tabs}. coupled. withRests. withNHT	1.0	0.8	0.7	0.8	0.6	0.47 ± 0.06	0.7 ± 0.2	75 ± 8	0.5
S:p_c. k0.7. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	59 ± 6	0.5
S:p_c. k0.5. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	70 ± 20	0.5
S:p_c. k0.3. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.45 ± 0.06	0.7 ± 0.2	100 ± 140	0.5
S:p_c. k0.3. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.46 ± 0.06	0.7 ± 0.2	59 ± 6	0.5
S:p_{40}. k0.9. r_{tabs}. coupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.47 ± 0.06	0.7 ± 0.2	75 ± 7	0.5
S:p_{40}. k0.9. r_{tabs}. coupled. withRests. withNHT	1.0	0.8	0.7	0.8	0.6	0.47 ± 0.06	0.7 ± 0.2	100 ± 60	0.5
S:p_{21}. k0.9. r_{tabs}. coupled. withRests. withNHT	1.0	0.8	0.7	0.8	0.6	0.47 ± 0.06	0.7 ± 0.2	90 ± 40	0.5
S:p_{12}. k0.9. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.45 ± 0.06	0.7 ± 0.2	60 ± 20	0.5
S:p_{12}. k0.9. r_{tabs}. coupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.47 ± 0.06	0.7 ± 0.2	90 ± 30	0.5
S:p_{12}. k0.9. r_{tabs}. coupled. withRests. withNHT	1.0	0.8	0.7	0.8	0.6	0.47 ± 0.06	0.7 ± 0.2	77 ± 12	0.5
S:p_{12}. k0.7. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.45 ± 0.06	0.7 ± 0.2	48 ± 7	0.5
S:p_{12}. k0.5. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.45 ± 0.06	0.7 ± 0.2	51 ± 8	0.5
S:p_{12}. k0.3. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.45 ± 0.06	0.7 ± 0.2	60 ± 20	0.5
S:p_{12}. k0.1. r_{tabs}. decoupled. withoutRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.45 ± 0.06	0.7 ± 0.2	80 ± 20	0.5
MS:p_{ift}. k0.5. r_{dabs}. matrixPitch. withoutRests. withNHT	1.0	0.8	0.8	0.7	0.6	0.45 ± 0.05	0.7 ± 0.2	140 ± 90	0.5
MS:p_{ift}. k0.5. r_{dabs}. matrixPitch. withRests. withNHT	1.0	0.8	0.9	0.7	0.5	0.45 ± 0.05	0.7 ± 0.2	120 ± 30	0.5
G:Pinto	1.0	0.6	0.2	0.7	0.2	0.17 ± 0.03	0.5 ± 0.3	1.92 ± 0.12	0.5
TA:p_{ift}. $l = 3$. Left prop.. Partial	1.0	0.8	0.7	0.9	0.5	0.35 ± 0.05	0.7 ± 0.2	100 ± 20	0.4
TA:p_{ift}. $l = 2$. Melodic prop.. Shasha	1.0	0.7	0.8	0.9	0.5	0.47 ± 0.05	0.7 ± 0.2	110 ± 40	0.4
S:p_{12}. k0.9. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.45 ± 0.06	0.7 ± 0.2	50 ± 10	0.4
S:p_{12}. k0.7. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.45 ± 0.06	0.7 ± 0.2	90 ± 90	0.4
S:p_{12}. k0.5. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.45 ± 0.06	0.7 ± 0.2	50 ± 12	0.4
S:p_{12}. k0.3. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.45 ± 0.06	0.7 ± 0.2	50 ± 20	0.4
S:p_{12}. k0.1. r_{tabs}. decoupled. withRests. withNHT	1.0	0.8	0.8	0.8	0.6	0.45 ± 0.06	0.7 ± 0.2	47 ± 7	0.4
MS:p_{ift}. k0.1. r_{dabs}. matrixPitch. withoutRests. withNHT	1.0	0.8	0.8	0.7	0.5	0.46 ± 0.05	0.7 ± 0.2	150 ± 20	0.4
TA:p_{ift}. $l = 3$. Melodic prop.. Partial	1.0	0.8	0.7	0.9	0.5	0.35 ± 0.05	0.7 ± 0.2	100 ± 20	0.3
TA:p_{ift}. $l = 2$. Melodic prop.. Align	1.0	0.7	0.8	0.9	0.4	0.32 ± 0.05	0.7 ± 0.3	160 ± 70	0.3

E.7 Global polyphonic results

Table E.18: Global polyphonic results. The time corresponds to that obtained with the *COVERS* corpus.

Setup	ICPS	INET	VARP	COVERS	**Avg.**	**Time**	***Borda count***
PROMS:r =8	0.6	0.8	0.7	0.69 ± 0.04	0.71 ± 0.05	132 ± 8	1.7
PROMS:r =4	0.5	0.8	0.7	0.68 ± 0.04	0.67 ± 0.10	83 ± 7	1.7
PROMS:r =16	0.7	0.7	0.9	0.67 ± 0.04	0.74 ± 0.09	230 ± 20	1.7
PROMS:r =28	1.0	0.7	0.8	0.38 ± 0.05	0.7 ± 0.3	390 ± 20	1.4
CBRAHMS:P2v5	1.0	0.6	0.8	0.26 ± 0.04	0.7 ± 0.3	45 ± 3	1.4
CBRAHMS:P2v6	1.0	0.6	0.8	0.25 ± 0.05	0.7 ± 0.3	3.48 ± 0.13	1.3
PR:r =12	0.7	0.7	0.8	0.68 ± 0.04	0.72 ± 0.03	180 ± 12	1.1
PR:r =24	0.8	0.7	0.8	0.67 ± 0.04	0.74 ± 0.07	320 ± 20	1.0
PR:r =20	0.7	0.7	0.8	0.49 ± 0.04	0.69 ± 0.14	270 ± 20	1.0
Trees::Overlap coefficient. $l = 3$.	0.9	0.6	0.7	0.56 ± 0.02	0.7 ± 0.2	226 ± 14	0.8
Trees::Multiset distance. $l = 4$.	0.9	0.6	0.7	0.57 ± 0.03	0.7 ± 0.2	950 ± 50	0.8
Trees::Manhattan. $l = 3$.	0.9	0.6	0.7	0.57 ± 0.03	0.7 ± 0.2	214 ± 13	0.8
Trees::Manhattan. $l = 2$.	0.9	0.6	0.7	0.57 ± 0.03	0.7 ± 0.2	41 ± 2	0.8
Trees::Log distance. $l = 4$.	0.9	0.6	0.5	0.63 ± 0.03	0.7 ± 0.2	3330 ± 130	0.8
Trees::Log distance. $l = 3$.	0.9	0.6	0.5	0.63 ± 0.03	0.7 ± 0.2	820 ± 50	0.8
Trees::Log distance. $l = 2$.	0.9	0.6	0.5	0.63 ± 0.03	0.7 ± 0.2	182 ± 8	0.8
Trees::Harmonic mean. $l = 4$.	0.9	0.6	0.6	0.60 ± 0.02	0.7 ± 0.2	1380 ± 70	0.8
Trees::Harmonic mean. $l = 3$.	0.9	0.7	0.7	0.60 ± 0.03	0.7 ± 0.2	330 ± 20	0.8
Trees::Harmonic mean. $l = 2$.	0.9	0.7	0.7	0.60 ± 0.03	0.72 ± 0.14	65 ± 3	0.8
Trees::Cosine similarity. $l = 4$.	0.9	0.6	0.6	0.60 ± 0.03	0.7 ± 0.2	1070 ± 50	0.8
Trees::Cosine similarity. $l = 3$.	0.9	0.7	0.6	0.60 ± 0.03	0.7 ± 0.2	246 ± 14	0.8
Trees::Cosine similarity. $l = 2$.	0.9	0.7	0.7	0.60 ± 0.03	0.72 ± 0.14	47 ± 2	0.8
Trees::Variational distance. $l = 3$.	0.9	0.5	0.7	0.44 ± 0.04	0.7 ± 0.2	330 ± 20	0.7
Trees::Variational distance. $l = 2$.	0.9	0.5	0.7	0.44 ± 0.04	0.7 ± 0.2	63 ± 3	0.7
Trees::Overlap coefficient. $l = 2$.	0.9	0.6	0.7	0.56 ± 0.02	0.7 ± 0.2	43 ± 2	0.7
Trees::Multiset distance. $l = 3$.	0.9	0.6	0.7	0.57 ± 0.03	0.7 ± 0.2	215 ± 14	0.7
Trees::Multiset distance. $l = 2$.	0.9	0.6	0.7	0.56 ± 0.03	0.7 ± 0.2	41 ± 2	0.7
Trees::Manhattan. $l = 4$.	0.9	0.6	0.7	0.57 ± 0.03	0.7 ± 0.2	950 ± 50	0.7
Trees::Euclidean distance. $l = 3$.	0.9	0.6	0.7	0.47 ± 0.04	0.7 ± 0.2	229 ± 14	0.7
Trees::Log distance. $l = 1$.	0.9	0.7	0.6	0.64 ± 0.04	0.68 ± 0.13	13.3 ± 0.7	0.6
CBRAHMS:P2	0.9	0.5	0.7	0.23 ± 0.04	0.6 ± 0.3	178.9 ± 1.4	0.6
CBRAHMS:P3	0.4	0.3	0.3	0.12 ± 0.02	0.26 ± 0.10	1140 ± 110	0.2
CBRAHMS:P1	0.0	0.1	0.0	0.0 ± 0.0	0.03 ± 0.05	0.0025 ± 0.0005	0.2

F

Resumen en castellano

Introducción

Actualmente, la disponibilidad de grandes cantidades de música almacenadas en formatos digitales en colecciones públicas y privadas crea problemas para su organización y para localizar información en ellas. Se hace necesaria la existencia de herramientas capaces de clasificar y permitir al usuario la búsqueda de música por contenido. Éstas pertenecen al dominio del *reconocimiento de formas* en lo que se conoce como Recuperación de Información Musical o MIR.

Nuestro trabajo se enmarca en este dominio, en concreto en la representación de información musical y su comparación. En esta tesis se propone una nueva representación de música codificada en formato simbólico mediante el uso de árboles y un conjunto de métodos para la evaluación de la similitud entre dos piezas musicales usando dicha representación.

El uso de estos métodos de comparación resuelve el problema de la clasificación musical basada en el contenido, en particular la localización de duplicados o trabajos similares y la búsqueda de un contenido musical. Las aplicaciones de éstos abarcan desde tareas de estudio y análisis en musicología hasta la detección de plagios.

La comunidad MIR suele categorizar los datos musicales en *audio digital*, *metadatos*, *partituras impresas* y *representaciones simbólicas*. El primero incluye todos los formatos de ficheros que almacenan grabaciones reales de información sonora tras su digitalización en formatos bien conocidos como los *wav*, *aiff*, *off* o *mp3*. Estos ficheros pueden contener cualquier tipo de sonido, desde el ruido de un coche hasta música, de forma que si se pretende analizar musicalmente los ficheros, previamente es necesario inferir esa información musical desde los datos en bruto. Los *metadatos* contienen información referente a la pieza musical como son el intérprete, compositor y título. Por *partitura impresa* entendemos cualquier representación gráfica que una persona puede leer pero que para un ordenador es simplemente una imagen. Finalmente, las *representaciones simbólicas* incluyen partituras en formatos digitales que contienen información acerca de notas, claves, pentagramas, que deben ser convertidas en una partitura impresa por un programa de notación musical para que puedan ser leídas por una persona. La música se puede representar también por la grabación de una interpretación en un *formato simbólico*. Como ejemplo de estos formatos son *MusicXML* (Good and Actor, 2003), *SCORE* (Selfridge-Field, 1997), *kern* (Huron, 2002) o *GUIDO* (Hoos et al., 1998). En

este tipo de formatos se almacenan las operaciones que generan la música, como son el inicio y fin de notas, su volumen, o la intensidad con la que se ha pulsado la tecla de un piano. El formato más extendido es el *SMF* (MID, 1996).

En esta tesis estamos interesados en las dos *representaciones simbólicas* porque queremos investigar en información de similitud musical empleando términos musicales como notas, grados, etc..., y los datos simbólicos son los más cercanos a este tipo de información.

Hoy en día, los métodos de similitud musical se dividen básicamente en aquellos que trabajan con música monofónica y los que son capaces de procesar contenido musical polifónico, directamente, o convirtiéndolo previamente en monofónico.

El término *similitud musical* es ambiguo o al menos se puede evaluar desde distintos puntos de vista (Barthelemy and Bonardi, 2001; Byrd and Crawford, 2002; Hofmann-Engl, 2001; Novello et al., 2006; Selfridge-Field, 1998). Se puede referir al parecido entre dos líneas melódicas, las coincidencias entre dos patrones rítmicos, o incluso en la similitud en el contenido armónico. Con el objetivo de desambiguar este concepto de aquí en adelante, consideraremos que las canciones más parecidas son las distintas interpretaciones de una misma pieza musical o aquellas producidas por la forma musical *variación*. Esto implica que la similitud se debe medir en función de un compromiso entre las dimensiones melódica, rítmica, e incluso armónica de las obras comparadas.

En la literatura (Barthelemy and Bonardi, 2001; Grachten et al., 2004; Rolland, 1998; Selfridge-Field, 1998) se acepta ampliamente que la codificación de la música afecta en gran medida las capacidades y calidad de los métodos de comparación. Tradicionalmente en MIR la música simbólica se representa mediante conjuntos de cadenas de tuplas, donde cada tupla típicamente contiene información sobre altura, duración y tiempo de ataque de las notas. Tanto la recuperación de información como la comparación han sido abordadas mayoritariamente usando técnicas de reconocimiento de formas estructural sobre cadenas. Han habido otras propuestas, como la geométrica, en la cual se transforma la melodía en una gráfica obtenida trazando una línea entre notas sucesivas, de forma que la comparación melódica se convierte en un problema geométrico. En los últimos años se han introducido nuevas propuestas como la representación mediante grafos o la comparación usando medidas estadísticas.

En esta disertación, presentamos una representación no lineal de la melodía basada en árboles que expresan la información métrica y rítmica de la música de una manera natural. Nuestra propuesta para la construcción de árboles se basa en el hecho que las distintas figuras musicales están diseñadas siguiendo una escala logarítmica: una redonda tiene una duración del doble de una blanca, la cual dura el doble que una negra, etc. Esta representación nos dota de una riqueza de posibilidades que los métodos basados en cadenas y los geométricos no tienen: como la representación implícita del ritmo, más significado musical y el enfatizado de notas relevantes. Además, los árboles abren un nueva potencia en la representación musical que permite la fusión en la misma estructura de datos todas las dimensiones involucradas en la similitud musical: altura, ritmo y armonía.

Antecedentes

Las obras musicales se pueden comprar desde diferentes perspectivas de descripción y representación. Se pueden comparar solamente el nombre de las canciones usando técnicas de comparación de cadenas. Se pueden emplear metadatos (Pérez-García et al., 2009) más detallados como descriptores de alto nivel como el autor, compás, tonalidad o la instrumentación. Finalmente, se puede usar el propio contenido musical para medir la similitud entre canciones. Esta tesis se ciñe a este último caso, la comparación utilizando solamente contenido musical representado de manera simbólica.

Dependiendo del dominio de la aplicación, se dice que el contenido musical contiene atributos distintos. En el dominio de la psicoacústica, Levitin (1999) usa altura, ritmo, tempo, contorno, timbre, presión sonora y localización espacial. En el dominio MIR, Downie (1999) considera siete facetas: altura, temporal, armónica, tímbrica, editorial, relativa a la textura y bibliográfica. Otros autores incluyen también características más elaboradas como la información temática obtenida a partir de datos en bruto (Hsu et al., 1998). Para la comparación basada en contenido musical, las propiedades más disponibles directamente son aquellas de altura y ritmo. En esta tesis se usarán estos dos atributos, incluyendo también en algunos casos información armónica.

En la comparación automatizada de piezas musicales en formato simbólico han habido dos trabajos que pueden ser considerados como hitos. El primero, por Mongeau and Sankoff (1990), usaba técnicas de comparación de cadenas (Wagner and Fischer, 1974) empleadas en bioinformática adaptadas a la comparación musical. El segundo, editado por Selfridge-Field (1998), revisaba en profundidad los métodos de representación y técnicas de comparación a la fecha de su edición, mostrando que los algoritmos de distancia de edición de cadenas eran los más usados para medir la similitud musical, además de algunos sistemas incipientes basados en representaciones geométricas (Maidín, 1998). Desde entonces se han propuesto muchos métodos para comparar la música en formato simbólico. Sin embargo, el número de representaciones o algoritmos realmente nuevos es limitado. La aportación de la mayoría de los métodos se fundamenta en el estudio del gran número de posibles variaciones de aquellos métodos originales, tanto en la forma de representar simbólicamente la música como en los algoritmos de reconocimiento de patrones usados.

La forma mediante la que se codifica la música determina la clase de búsquedas que se pueden llevar a cabo como se establece en algunos trabajos como (Selfridge-Field, 1998) o (Barthelemy and Bonardi, 2001). A la fecha de la redacción de este documento, hay básicamente siete tendencias en la representación y comparación de música en formato simbólico:

- distancias de edición y alineamiento de cadenas usando una variedad de representaciones (Grachten et al., 2005; Lemström, 2000; Mongeau and Sankoff, 1990)
- algoritmos basados en n-grams (Doraisamy, 2004; Downie, 1999; Uitdenbogerd, 2002)
- codificaciones de grafos (Pinto and Tagliolato, 2008)

- medidas de comparación estadística (Bernabeu et al., 2009; Engelbrecht, 2002; Habrard et al., 2008)
- sistemas geométricos (Aloupis et al., 2006; Lemström et al., 2008; Tanur, 2005; Typke, 2007)
- modelos ocultos de Markov (Pardo et al., 2004; Pickens, 2004)
- representaciones mediante árboles (Rizo et al., 2003, 2008)

Comparación musical con árboles

> "Uno de los hecho más obvios sobre la experiencia de escuchar prácticamente cualquier obra musical es que se percibe no meramente una secuencia arbitraria de notas, sino una especie de estructura temporal en la que se agrupan las notas en varias clases de unidades." (Lee, 1985)

> "Las reducciones ofrecen una solución tentadora a la comparación aproximada porque eliminan las diferencias en los detalles de la superficie. Pero, ¿eliminan los detalles más apropiados?" (Selfridge-Field, 1998)

Estas dos citas soportan la hipótesis en la que basamos nuestra propuesta. El tipo abstracto de datos *árbol* parece ser el más adecuado para capturar la estructura temporal y jerárquica, además de estar preparado para representar la reducción de una obra musical. Por tanto, los árboles parecen ser el tipo de datos más adecuado para codificar y procesar la música en formato simbólico para la evaluación de la similitud. A pesar de lo obvio de esta asunción, no conocemos ningún trabajo previo que haya usado árboles para comparar música hasta nuestra primera publicación en (Rizo and Iñesta, 2002).

Usos previos de árboles en informática musical

Los árboles sí han sido usados en informática musical para otros fines. El uso más evidente es la representación de la arquitectura de una obra (Linster, 1992). En el campo de la percepción musical, con el objeto de describir formalmente la percepción de ritmos complejos, se han propuesto una serie de gramáticas (Longuet-Higgins, 1978; Martin, 1972; Sundberg and Lindblom, 1992). A pesar de ser un enfoque interesante, ninguno de estos trabajos especifican una especificación computable. En cambio, podemos encontrar dos propuestas diseñadas justamente desde un punto de vista computacional (Bod, 2002) y (Gilbert and Conklin, 2007). En la primera se intenta inferir una gramática para aprender segmentaciones de un corpus de melodías con frases separadas manualmente. En la segunda, las melodías monofónicas se analizan en árboles de análisis sintáctico usando una gramática probabilística libre de contexto que se usa para realizar reducciones melódicas, y este análisis se evalúa a través de un problema de segmentación musical.

Como medio de representar conceptualmente la música para composición, los árboles se han usado en (Smaill et al., 1993) y (Balaban, 1996). En (Höthker et al., 2001) se emplean como medio de describir la naturaleza jerárquica de las codificaciones de altura. En composición automática, Högberg (2005) propone el uso de transductores de árboles para generar música.

Durante el final del siglo XIX y el inicio del XX, Heinrich Schenker (1868-1935) desarrolló un método para analizar composiciones tonales basadas en la estructuración jerárquica de la música (en (Beach, 1977) se puede encontrar un compendio de sus publicaciones). Argumentaba que un trabajo musical puede ser en último término entendido como una serie de elaboraciones que llevan desde una estructura armónica y melódica básicas a la secuencia real de notas que componen la pieza musical. Para ello se apoyó en la estructura jerárquica árbol. Sin embargo, dado que sus teorías no fueron desarrolladas desde un punto de vista formal, no hay una metodología computable que conduzca el análisis de una pieza. En cambio, se deben llevar a cabo varios principios no deterministas para derivar los sucesivos niveles de reducción del análisis schenkeriano. Han habido varios intentos para crear sistemas automáticos que realicen este tipo de análisis: Smoliar (1979), Kassler (1988), Kirlin and Utgoff (2008), y los trabajos excelentes de Marsden (2001, 2004, 2005, 2007). Sin embargo, aparte de lo ambiguo en la resolución de ciertos situaciones en las que hay que tomar una decisión para elegir entre varios análisis posibles, ninguno de estos trabajos han tenido como objetivo la comparación musical, y por lo que sabemos, no hay un conjunto de datos que garantice que los análisis realizados con estos métodos valgan para el objetivo de la evaluación de la similitud.

Una forma similar a Schenker de explicar jerárquicamente las relaciones internas de una obra musical es la Teoría Generativa de Música Tonal o GTTM (Lerdahl and Jackendoff, 1983). Inspirada en las gramáticas aplicadas en lingüística, los autores del GTTM escribieron: "nuestro objetivo es una teoría musical que sea un formato de descripción de las intuiciones musicales que una persona experimenta al escuchar música". Aunque no se define ninguna gramática formal en ese trabajo, sí se especifica una lista extensa de reglas que intentan explicar lo que se percibe al escuchar una obra musical, cubriendo desde la percepción del tempo a cómo se estructura una obra completa, pasando por las dependencias individuales entre notas. Pero como ocurre con las teorías de Schenker el GTTM no es directamente computable. La mayor parte de las reglas son ambiguas y entran en colisión en muchos casos. Han habido intentos de implementación (Hamanaka et al., 2007; Jones et al., 1993), siendo el trabajo de este último (denominado *ATTA*) el más activo y prometedor hasta el momento. Sin embargo, estos sistemas tienen una gran cantidad de parámetros para ajustar, y están lejos de poder trabajar con piezas completas.

En el contexto de la composición musical asistida por ordenador hay una herramienta que destaca sobre el resto: OpenMusic (Assayag et al., 1999). Utiliza árboles como una manera natural de representar la naturaleza jerárquica de la subdivisión de las figuras musicales y grupetos. Está implementado en LISP donde todo se representa con listas

o árboles, en la forma de listas de listas. Una secuencia musical se representa mediante una lista de alturas y un árbol representando duraciones.

Finalmente, los árboles han sido usados no como una forma de representar música, sino como una estructura de datos intermedia para otros propósitos como la estructuración de documentos para búsqueda (Blackburn, 2000; Drewes and Högberg, 2007; Skalak et al., 2008).

Representación monofónica de árboles

Una melodía tiene dos dimensiones principales: ritmo y altura. En las representaciones lineales, estas dimensiones se codifican mediante símbolos explícitos. Los árboles ordenados son capaces de representar el tiempo en la propia estructura, haciendo uso del hecho de que una pieza completa se subdivide jerárquicamente en compases, y que las duraciones de las notas son múltiplos de unidades básicas de tiempo, principalmente en subdivisiones binarias y en algunos casos ternarias. El orden de izquierda a derecha de los nodos de los árboles describe la paso del tiempo. De esta forma, los árboles son menos sensibles a los códigos usados para representar melodías, pues sólo es necesario establecer la codificación de altura implicando un número menor de grados de libertad en cuanto a la codificación que las cadenas. Es más, esta organización jerárquica permite añadir información adicional como descriptores armónicos a grupos de notas de una manera natural. Denominamos a esta representación *árboles métricos*.

En la música occidental, el paso del tiempo se suele dividir regularmente en compases, tiempos y en subdivisiones de éstos. La duración de los compases depende del tempo y la métrica. En nuestro modelo, cada compás se representa por un árbol. Cada nota o silencio se encuentra en un nodo hoja. El orden de izquierda a derecha de las hojas preserva el orden temporal. El nivel de la hoja en el árbol determina la duración de la nota representada de forma que el raíz del árbol representa la duración de un compás completo (una redonda en el caso de un compás $\frac{4}{4}$), los nodos hijos de la raíz representan una subdivisión de esa duración (una blanca en el caso del $\frac{4}{4}$), etc. Para esa métrica, si sólo encontramos una redonda en el compás, el árbol consistirá solamente en un nodo raíz, pero si tiene dos blancas, ese nodo se partirá en dos nodos hijos. Así, recursivamente, cada nodo del árbol se dividirá en dos (o tres en el caso de las partes de los compases compuestos) hasta alcanzar el nivel correspondiente a la duración de la nota a codificar.

Durante la construcción del árbol, se crean nodos internos cuando son necesarios para alcanzar el nivel de hoja apropiado, rellenando el árbol de izquierda a derecha. Inicialmente, sólo las hojas contienen una etiqueta. Las etiquetas contienen la representación de altura en cualquiera de las codificaciones posibles (nota MIDI, intervalo, etc.) pudiendo incorporar incluso información armónica. Una vez el árbol está construido, se realiza un proceso de propagación de abajo hacia arriba para etiquetar completamente el árbol.

Un ejemplo de este esquema se representa en la figura F.1. Nótese que en este caso el compás es un $\frac{2}{4}$ y la duración de la raíz es equivalente a una blanca. En el árbol, el hijo izquierdo de la raíz se ha partido en dos subárboles hasta alcanzar el nivel 3 que corresponde a la duración de la primera nota (la altura Si se representa por un 11). Para

representar las duración del silencio y el Sol, codificado por un 7 (ambos duran la octava parte del compás), se necesita un nuevo subárbol para el árbol a la derecha del anterior, generando dos nuevas hojas en las que pondremos una etiqueta vacía para el silencio y un 7 para el Sol. El Do negra (0) se inicia en el segundo tiempo del compás, por lo que es representado en el nivel 2 de acuerdo a su duración.

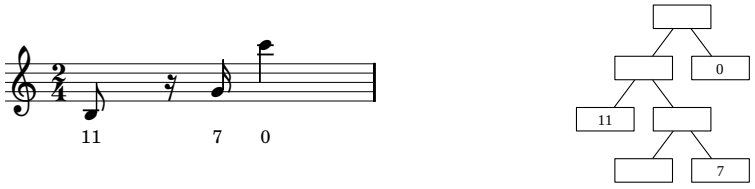


Figure F.1: Ejemplo de construcción del árbol. Las etiquetas son intervalos en número de semitonos desde Do.

.

La figura F.1 describe cómo se respeta el orden temporal de las notas en la partitura mediante el recorrido de izquierda a derecha en el árbol. Véase también cómo los tiempos de ataque y duraciones son representados implícitamente en el árbol, en contraposición a la codificación explícita del tiempo cuando se emplean cadenas. Además, esta representación es la misma aunque se cambie la escala de tiempo o la figuración usada.

Para permitir la representación de ritmos no binarios, como aquellos que encontramos en compases compuestos y grupetos, simplemente subdividimos los nodos de acuerdo a esas subdivisiones.

La representación de melodías completas nos lleva a emplear un subárbol para cada compás que es agrupado respetando el orden secuencial de compases en un bosque ordenado.

Propagación

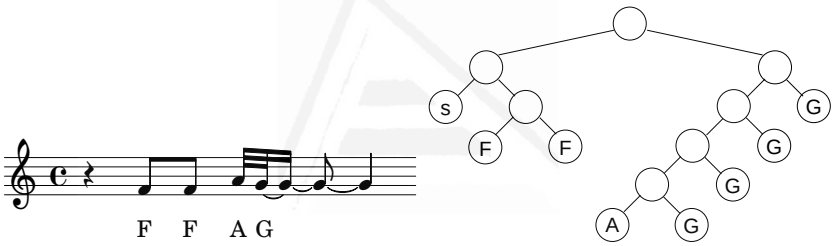
Dos causas motivan el etiquetado total de los nodos del árbol por medio de la propagación. Primero, la mayor parte de los algoritmos de comparación de árboles necesitan que todos los nodos (internos y hojas) estén etiquetados, pero en el proceso de construcción de árboles métricos solamente las hojas se etiquetan. Usamos un conjunto de reglas para propagar las etiquetas de las hojas hacia arriba, etiquetando así los nodos internos. La propagación de una etiqueta se decide en base a que la nota en un nodo es más importante que aquella de su nodo hermano. La importancia de una nota es relativa a su capacidad para contribuir a la identidad de la melodía.

Segundo, la alta complejidad temporal de las distancias de edición de árboles usadas para comparar los árboles y de esa forma las melodías, requiere que los árboles sean lo más pequeños posible. Cuando aparecen notas muy cortas o no se ajustan exactamente a las subdivisiones binarias o ternarias, los árboles resultantes son muy profundos. Así, las reglas de propagación se acompañan con una acción de poda para borrar ramas poco

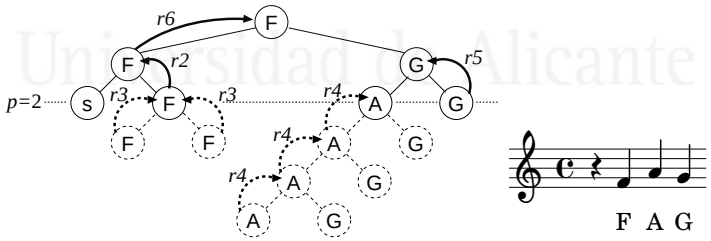
significativas a partir de un nivel dado. Este proceso ayuda a eliminar información poco relevante que haría la comparación más difícil, obteniendo árboles reducidos capaces de mantener las principales características de la melodía.

La decisión de qué es relevante y qué no es complicada, véase cómo tanto el análisis schenkeriano como el GTTM tienen problemas para tomar de manera determinista esta decisión. Nuestro objetivo no es analizar obras, sino decidir qué notas son más importantes para comparar las canciones de manera computable. Siguiendo estos principios presentamos dos esquemas de propagación: *heurístico* y *análisis-melódico*. La primera se basa en reglas empíricas encontradas tras analizar visualmente obras junto a variaciones y distintas interpretaciones de las mismas. La segunda usa un análisis melódico basado en las reglas del análisis tonal para determinar las notas más importantes. Hemos incluido además dos esquemas simples que emplean sólo la información de acentos métricos para propagar notas que atacan en tiempo o parte fuerte *propagación-izquierda* o débil *propagación-derecha*.

La figura F.2 muestra un ejemplo de propagación heurística.



(a) Una melodía de un compás y su representación empleando etiquetas de altura (sólo las hojas por ahora) antes de podar y propagar etiquetas.



(b) Árbol final propagado y con nodos podados (en líneas discontinuas tras aplicar las reglas de propagación). Se muestra también la melodía equivalente al árbol podado

Figure F.2: Efecto de la propagación heurística en una melodía.

Comparación de árboles

Una vez representadas las melodías mediante árboles con las técnicas comentadas anteriormente, la comparación de esas melodías se lleva a cabo midiendo la similitud de los árboles que las codifican. Para ello se han usado las distancias clásicas de edición de árboles (Zhang and Shasha, 1989), distancias de edición restringidas (Selkow, 1977),

distancias de alineamiento (Jiang et al., 1995), y distancias de abajo-arriba (Valiente, 2001). Para la distancia de alineamiento proponemos una implementación alternativa a la propuesta por el autor que mejora sustancialmente los tiempos de cálculo. Adicionalmente, presentamos una nueva distancia diseñada para trabajar con árboles etiquetados solamente en las hojas (Rizo and Iñesta, 2010) que hace no necesario el proceso de propagación.

Representación de música polifónica

La metodología que acabamos de introducir es capaz de trabajar solamente con música monofónica. Si las obras de entrada son polifónicas se pueden seguir dos estrategias: reducir la polifonía a monofonía en alguna de las técnicas existentes como es quedarse siempre con la nota más alta en cada instante, o tratar directamente el contenido polifónico. Nuestra propuesta usa esta última opción.

Para representar música polifónica, se insertan todas las voces en el mismo árbol siguiendo las reglas de la representación monofónica. Las etiquetas de los nodos son ahora conjuntos de notas (véase la figura F.3).

Esta representación es extendida para representar duraciones relativas de notas mediante el uso de cardinalidades en multisets. Además se desarrolla un mecanismo de propagación de etiquetas de abajo hacia arriba basado en la unión de estos multisets.

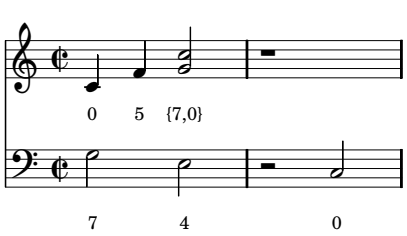
La comparación de árboles polifónicos se realiza de la misma forma que los árboles monofónicos con la única diferencia que ahora las operaciones de edición deben tener en cuenta que las etiquetas son un tipo multiset en lugar de contener datos escalares. Para evaluar el coste de sustitución entre multisets proponemos una serie de distancias entre vectores como la distancia euclídea, Manhattan, Helliger, etc. que detallamos en el apéndice A.

Experimentos

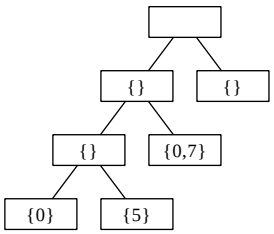
Los experimentos han sido diseñados para comprobar la idoneidad de la representación de árbol propuesta para la tarea de medir la similitud entre obras musicales. Las distintas formas de codificar la altura de las notas, las propagaciones, formas de comparar los árboles, y los distintos niveles de poda nos llevan a tener un gran número de maneras distintas de representar y comparar obras musicales mediante árboles. En los experimentos, se han probado todas las posibilidades con la intención de descubrir el impacto de cada componente en el sistema para la tarea prevista.

Dado que no hay una base de datos suficientemente grande en la comunidad investigadora para comparar los distintas propuestas disponibles en la literatura, se han compilado varios corpora, tanto monofónicos como polifónicos, enfocados en diferentes aspectos de la similitud musical.

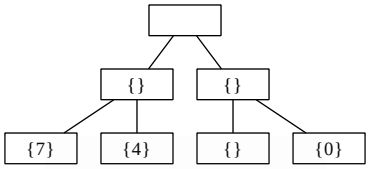
Nuestra propuesta ha sido comparada con los sistemas que se pueden considerar los más representativos actualmente. Estos sistemas también tienen un número elevado de posibles configuraciones que han sido evaluadas exhaustivamente.



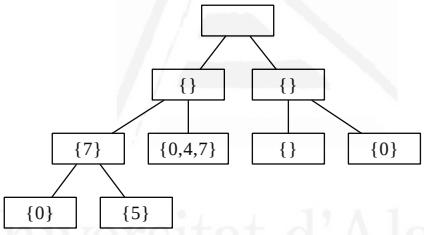
(a) Partitura a representar



(b) Árbol representando la voz superior



(c) Árbol representando la voz inferior



(d) Árbol polifónico representando ambas voces

Figure F.3: Ejemplo de música polifónica y su árbol correspondiente

Los experimentos diseñados han seguido los siguientes pasos:

1. El primer paso ha sido seleccionar aquellas configuraciones, tanto de los árboles como del resto de métodos (cadenas, grafos, etc.), que han obtenido un buen compromiso entre tiempos de proceso y tasas de acierto comportándose robustos para todos los tipos de similitud musical representados en los distintos corpora. Para ello se ha diseñado una metodología para la comparación de resultados teniendo en cuenta el significancia estadística, desequilibrio en el número de prototipos en los distintos corpora y los diferentes tipos de contenidos.

2. Tras seleccionar la lista de mejores configuraciones por método, se han aplicado a un escenario realista.

3. Dados los resultados obtenidos en los anteriores pasos, por cada método se ha seleccionado la que parece ser la mejor configuración

4. Finalmente, se ha estudiado el impacto de cada forma de construir y comparar árboles en la calidad de los resultados anteriores.

Conclusiones

En esta tesis se ha introducido una representación de árboles de música codificada en formato simbólico para la evaluación de la similitud, y se ha mostrado que ofrece un buen compromiso entre tiempos de proceso y calidad entre todos los métodos presentados en los corpora explorados.

Con el propósito de enmarcar nuestra propuesta en la literatura relacionada, se han revisado tanto los métodos punteros de comparación musical como los usos del tipo abstracto de datos árbol para el procesado de música simbólica en otros dominios.

Retomemos la hipótesis introducida al principio de este documento:

> El tipo abstracto de datos árbol parece ser el más idóneo para capturar la estructura temporal y jerárquica, y es adecuado para representar la reducción de una obra musical. Por tanto, los árboles se postulan como una estructura de datos válida para codificar la música en formato simbólico para la evaluación automática de la similitud.

Los experimentos presentados parecen corroborar esta hipótesis. La representación propuesta se ha mostrado capaz de codificar la información rítmica de una obra musical de manera implícita basada en la estructura métrica, siendo las principales ventajas respecto estructuras lineales su simplicidad en la representación de música monofónica y polifónica en el mismo tipo de estructura, y su versatilidad para incorporar información más elaborada como la forma musical o la armonía.

La estructura árbol propuesta se ha usado con éxito en otras tareas. Por ejemplo, la identificación de la tonalidad siguiendo un recorrido de abajo hacia arriba en árboles polifónicos (Rizo et al., 2006b). Este método ha sido usado en los experimentos de esta tesis para obtener la tonalidad para aquellos ficheros MIDI cuya tonalidad estaba ausente, y por los resultados obtenidos se puede deducir que este proceso se ha comportado lo suficientemente consistente para la tarea para la que fue diseñada. Otra aplicación ha sido la composición automática (Espí et al., 2007), donde los árboles métricos son usados como una manera de representar la música en un sistema genético, que emplea el intercambio de árboles en las operaciones de cruce.

Nuestro enfoque tiene dos inconvenientes: su fuerte dependencia de la estructura métrica de la fuente de entrada, y su dificultad para representar ligaduras, puntillos y síncopas. El primer problema se puede solucionar a través de un análisis métrico de la obra a priori (Eck and Casagrande, 2005; Meudic, 2002b). En cualquier caso, los resultados experimentales han mostrado que la información métrica es importante para la evaluación de la similitud. El segundo inconveniente, desde el punto de vista de la representación, se ha solventado con la adición de un símbolo especial que codifica el concepto de continuación. No obstante, para la tarea de comparación musical, esto

no se puede considerar como desventaja como ha habían apuntado previamente otros autores (Hanna et al., 2008; Mongeau and Sankoff, 1990; Pardo and Sanghi, 2005).

Un problema derivado que surge de la estructura métrica fija es el excesivo crecimiento de los árboles cuando aparecen notas muy cortas e imprecisiones en la interpretación. Este problema ha sido tratado mediante el proceso de propagación propuesto. Sin embargo, es una línea de investigación abierta que podría solucionarse mediante la aplicación de algoritmos avanzados de cuantización (Agon et al., 1994; Cemgil et al., 2000) o convirtiendo la estructura métrica fija en una más flexible, capaz de respetar de alguna forma la estructura de compás, tiempos y tactum, y de trabajar tanto con música monofónica como polifónica.

El último aspecto que puede ser mejorado desde el punto de vista de la representación es la sustitución del agrupamiento actual de los árboles que representan compases por una estructura jerárquica codificando la estructura formal o de arquitectura de la obra musical.

La comparación de los árboles se ha realizado utilizando una serie de distancias de edición y se ha propuesto una nueva medida de similitud entre árboles que actualmente también estamos comenzando a aplicar en otros dominios diferentes al musical. También se ha propuesta una nueva implementación de una distancia de alineamiento clásica que hace el algoritmo practicable en el tiempo. Aunque se podrían probar más distancias, y los tiempos de proceso podrían mejorar, por los resultados obtenidos parece que la calidad de recuperación musical no tiene muchas más posibilidades de mejora con el cambio de distancia. Sí se puede estudiar más en profundidad la adición de la información de la propagación para ponderar el coste de sustitución.

En esta tesis no hemos empleado ningún sistema de ajuste automático de costes de edición. En (Habrard et al., 2008) mostramos que los resultados se pueden mejorar sustancialmente con la aplicación de sistemas de ajuste de parámetros como son los sistemas genéticos, o con el uso de distancias estocásticas tanto en cadenas como en árboles, capaces de aprender de un conjunto de entrenamiento la matriz de costes de edición.

El prefiltrado de los nodos en los árboles polifónicos previa a la comparación de los árboles, y el uso de una versión adaptada a éstos del algoritmo de comparación de árboles parcialmente etiquetados propuesta es otra línea de trabajo a explorar.

Otro área de trabajo abierta es la aplicación de las técnicas de comparación de árboles que hemos aplicado a nuestros árboles métricos a otras representaciones arborescentes como aquellas obtenidas tras el análisis schenkeriano o el GTTM.

Hemos realizado una exploración exhaustiva del espacio de parámetros de algunos de los principales métodos de evaluación de la similitud existentes hoy en día, con un total de 4088 configuraciones distintas. La combinación de éstos es una opción prometedora a seguir para alcanzar definitivamente los mejores resultados. En (Rizo et al., 2008, 2009a,b) presentamos un procedimiento para seleccionar la mejor combinación de clasificadores para la tarea de la similitud polifónica basada en la elección de los mejores métodos que además generan resultados más dispares.

Para el proceso de música polifónica con métodos monofónicos que además es integrable en una combinación de clasificadores, el algoritmo que se queda con las notas más altas en cada momento se puede reemplazar por un preproceso más sofisticado: la extracción de la pista de la melodía siguiendo procesos estadísticos propuesto en (Rizo et al., 2006a).

Finalmente, la mayoría de los trabajos existentes que evalúan la similitud musical usan un solo corpus, que seguramente tiene una versión sesgada de lo que la similitud musical. Es más, en muchos casos, la evaluación se realiza de manera subjetiva. En esta tesis hemos propuesto una forma diferente de evaluar los distintas paradigmas de comparación musical. Por una parte hemos recopilado 7 corpora distintos, tanto monofónicos como polifónicos, representando formas diferentes de concebir lo que es la similitud musical Por otra parte, hemos diseñado una metodología para extraer conclusiones sobre el comportamiento de los distintos algoritmos en copora de distintos tamaños y naturaleza.

Bibliography

Répertoire international des sources musicales (rism) series a/ii: Music manuscripts after 1600 on cd-rom, 2005. (Cited on page 3).

Norman H. Adams, Mark A. Bartsch, Jonah Shifrin, and Gregory H. Wakefield. Time series alignment for music information retrieval. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2004. (Cited on page 48).

Carlos Agon, Gérard Assayag, J. Fineberg, and Camilo Rueda. Kant: A critique of pure quantification. In *Proc. of the 1994 International Computer Music Conference*, pages 52–59, Aarhus, Denmark, 1994. (Cited on pages 152 and 214).

Carlos Agon, Karim Haddad, and Gérard Assayag. Representation and rendering of rhythm structures. In *Proceedings. Second International Conference on Web Delivering of Music*, pages 109–113, 2002. (Cited on page 68).

Julien Allali, Pascal Ferraro, Pierre Hanna, and Costas S. Iliopoulos. Local transpositions in alignment of polyphonic musical sequences. In *SPIRE*, pages 26–38, 2007. (Cited on pages 35 and 47).

Greg Aloupis, Thomas Fevens, Stefan Langerman, Tomomi Matsui, Antonio Mesa, Yurai Nuñez, David Rappaport, and Godfried T. Toussaint. Algorithms for computing geometric measures of melodic similarity. *Computer Music Journal*, 30(3):67–76, 2006. ISSN 0148-9267. doi: http://dx.doi.org/10.1162/comj.2006.30.3.67. (Cited on pages 12, 38, 39, 47, and 206).

F. J. Anscombe. The validity of comparative experiments. *Journal of the Royal Statistical Society, Series A (general)*, 111(Part 3):181–211, 1948. (Cited on page 130).

Gérard Assayag, Camilo Rueda, Mikael Laurson, Carlos Agon, and Olivier Delerue. Computer-assisted composition at ircam: From patchwork to openmusic. *Computer Music Journal*, 23(3):59–72, 1999. ISSN 0148-9267. doi: http://dx.doi.org/10.1162/014892699559896. (Cited on pages 68 and 207).

David Bainbridge, Craig G. Nevill-manning, Ian H. Witten, Lloyd A. Smith, and Rodger J. McNab. Towards a digital library of popular music. In *In Proceedings of the Fourth ACM International Conference on Digital Libraries*, pages 161–169. ACM, 1999. (Cited on page 20).

Mira Balaban. The music structures approach to knowledge representation for music processing. *Computer Music Journal*, 20(2):96–111, 1996. (Cited on pages 63 and 207).

Harold Barlow and Sam Morgenstern. *A dictionary of musical themes*. E. Benn (London), 1978. ISBN 0510355013. (Cited on pages 3 and 21).

J. Barthelemy and Alain Bonardi. Similarity in computational music: A musicologist's approach. In *WEDELMUSIC '01: Proceedings of the First International Conference on WEB Delivering of Music (WEDELMUSIC'01)*, page 107, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1284-4. URL http://portal.acm.org/citation.cfm?id=882508.885251. (Cited on pages 6, 7, 11, 15, 20, 21, 26, 204, and 205).

David Beach. *Readings in Schenker Analysis and Other Approaches: A Schenker Bibliography*. Yale University Press, yeston, maury edition, 1977. (Cited on pages 64 and 207).

Pierfrancesco Bellini, Ivan Bruno, and Paolo Nesi. Assessing optical music recognition tools. *Computer Music Journal*, 31(1):68–93, 2007. ISSN 0148-9267. doi: http://dx.doi.org/10.1162/comj.2007.31.1.68. (Cited on pages 2 and 4).

Juan Pablo Bello. Audio-based cover song retrieval using approximate chord sequences: Testing shifts, gaps, swaps and beats. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR'07)*, Vienna, Austria, September 2007. (Cited on page 48).

José Francisco Bernabeu, Jorge Calera-Rubio, José Manuel Iñesta, and David Rizo. A probabilistic approach to melodic similarity. In *Proceedings of MML 2009*, pages 48–53, 2009. (Cited on pages 12, 145, and 206).

Philip Bille. A survey on tree edit distance and related problems. *Theorical Computer Science*, 337(1-3):217–239, 2005. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/j.tcs.2004.12.030. (Cited on pages 98, 101, 104, and 105).

Philip Bille. *Pattern Matching in Trees and Strings*. PhD thesis, ITU University of Copenhagen, 2007. (Cited on pages 98 and 104).

William P. Birmingham, Roger B. Dannenberg, Gregory H. Wakefield, Mark A. Bartsch, David Bykowski, Dominic Mazzoni, Colin Meek, Maureen Mellody, and William Rand. Musart: Music retrieval via aural queries. In *Proceedings of the 2nd Annual International Symposium on Music Information Retrieval (ISMIR)*, pages 73–81, 2001. (Cited on pages 31 and 35).

Steven G. Blackburn. *Content Based Retrieval and Navigation of Music Using Melodic Pitch Contours*. PhD thesis, University of Southampton, Southampton, UK, September 2000. URL http://www.beeka.org/research/papers/index.html. (Cited on pages 70 and 208).

Rens Bod. A general parsing model for music and language. *Music and Artificial Intelligence*, pages 77–90, 2002. URL http://dx.doi.org/10.1007/3-540-45722-4_3. (Cited on pages xii, 61, 63, and 206).

H.. Bunke and J. Csirik. Inference of edit costs using parametric string matching. In *Pattern Recognition, 1992. Vol.II. Conference B: Pattern Recognition Methodology and Systems, Proceedings., 11th IAPR International Conference on*, pages 549–552, Aug-3 Sep 1992. doi: 10.1109/ICPR.1992.201838. (Cited on page 122).

Donald Alvin Byrd and Tim Crawford. Problems of music information retrieval in the real world. *Information Processing and Management*, 38(2):249–272, 2002. ISSN 0306-4573. doi: http://dx.doi.org/10.1016/S0306-4573(01)00033-4. (Cited on pages 6 and 204).

Emilios Cambouropoulos, Maxime Crochemore, Costas S. Iliopoulos, Manal Mohamed, and Marie-France Sagot. A pattern extraction algorithm for abstract melodic representations that allow partial overlapping of intervallic categories. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 167–174, 2005. (Cited on page 25).

Domenico Cantone, Salvatore Cristofaro, and Simone Faro. On tuning the (δ, α)-sequential-sampling algorithm for δ-approximate matching with alpha-bounded gaps in musical sequences. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 454–459, 2005a. (Cited on page 35).

Domenico Cantone, Salvatore Cristofaro, and Simone Faro. Solving the (δ, α)-approximate matching problem under transposition invariance in musical sequences. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 460–463, 2005b. (Cited on page 35).

Gail A. Carpenter, Stephen Grossberg, and John H. Reynolds. Artmap: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Networks*, 4(5):565–588, 1991. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/0893-6080(91)90012-T. (Cited on page 43).

Ali Taylan Cemgil, Bert Kappen, and Peter Desain. Rhythm quantization for transcription. *Computer Music Journal*, 24(2):60–76, 2000. ISSN 0148-9267. doi: http://dx.doi.org/10.1162/014892600559218. (Cited on pages 152 and 214).

Wei Chai. *Automated Analysis of Musical Structure*. PhD thesis, Massachusetts Institute of Technology, MA, USA, September 2005. URL http://alumni.media.mit.edu/~chaiwei/papers/whole0622.pdf. (Cited on page 76).

Elaine Chew. *Towards a Mathematical Model for Tonality*. PhD thesis, Massachusetts Institute of Technology, MA, USA, February 2000. URL https://dspace.mit.edu/handle/1721.1/9139/. (Cited on page 51).

G. Sayeed Choudhury, Tim DiLauro, Michael Droettboom, Ichiro Fujinaga, Brian Harrington, and Karl MacMillan. Optical music recognition system within a large-scale digitization project. In Donald Alvin Byrd and J. Stephen Downie, editors, *Proceedings of International Symposium on Music Information Retrieval: Music IR 2000*, page

[4] page numbers unknown, Amherst, MA, 2000. University of Massachusetts at Amherst. URL http://ciir.cs.umass.edu/music2000/papers/choudhury_paper.pdf. (Cited on page 2).

Michael Clausen, Ronald Engelbrecht, D. Meyer, and J. Schmitz. Proms: A web-based tool for searching in polyphonic music. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2000. (Cited on pages 23, 26, 43, 53, and 55).

R. Clifford and Costas S. Iliopoulos. Approximate string matching for music analysis. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 2004. URL http://www.springerlink.com/content/4hh82ba1j696vqqx/fulltext.pdf. (Cited on pages 32 and 33).

Raphael Clifford, Manolis Christodoulakis, Tim Crawford, David Meredith, and Geraint A. Wiggins. A fast, randomised, maximal subset matching algorithm for document-level music retrieval. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2002. (Cited on page 55).

Darrell Conklin and Christina Anagnostopoulou. Representation and discovery of multiple viewpoint patterns. In *Proceedings of the 2001 International Computer Music Conference, La Habana, Cuba*, pages 479–485. International Computer Music Association, 2001. (Cited on page 28).

Maxime Crochemore, Costas S. Iliopoulos, Thierry Lecroq, and Yoan J. Pinzon. Approximate string matching in musical sequences. In *In Proceedings of the Prague Stringology Conference*, pages 26–36, 2001. (Cited on pages 32 and 33).

C. Cronin. The music plagiarism digital archive at columbia law library; an effort to demystify music copyright infringement. In *Proceedings. Second International Conference on Web Delivering of Music*, pages 133–139, 2002. (Cited on page 3).

Roger B. Dannenberg and Ning Hu. Understanding search performance in query-by-humming systems. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2004. (Cited on page 32).

Roger B. Dannenberg, William P. Birmingham, George Tzanetakis, Colin Meek, Ning Hu, and Bryan Pardo. The musart testbed for query-by-humming evaluation. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2003. (Cited on pages 19 and 25).

Michel Marie Deza and Elena Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, 2009. doi: 10.1007/978-3-642-00234-2_1. (Cited on pages 118 and 159).

Shyamala Doraisamy. An approach towards a polyphonic music retrieval system. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2001. (Cited on page 36).

Shyamala Doraisamy. *Polyphonic Music Retrieval: The N-gram approach*. PhD thesis, Imperial College London, London, UK, 2004. (Cited on pages 2, 12, 19, 23, 25, 36, 50, and 205).

Shyamala Doraisamy and Stefan M. Rüger. A comparative and fault-tolerance study of the use of n-grams with polyphonic music. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2002. (Cited on page 36).

Shyamala Doraisamy and Stefan M. Rüger. A polyphonic music retrieval system using n-grams. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2004. (Cited on page 36).

W. J. Dowling. Two components of a theory of memory for melodies. *Psychological Review*, 85(4):341—354, 1978. (Cited on page 20).

J. Stephen Downie. Informetrics and music information retrieval: An informetric examination of a folksong database. *Proceedings of the 26th Annual Conference of the Canadian Association for Information Science*, 1998. (Cited on page 20).

J. Stephen Downie. *Evaluating a Simple Approach to Music Information Retrieval: Conceiving Melodic N-Grams as Text*. PhD thesis, University of Western Ontario, Canada, July 1999. URL http://people.lis.uiuc.edu/~jdownie/mir_papers/thesis_missing_some_music_figs.pdf. (Cited on pages 11, 12, 36, and 205).

J. Stephen Downie. Toward the scientific evaluation of music information retrieval systems. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2003. (Cited on page 1).

J. Stephen Downie. The music information retrieval evaluation exchange (mirex. *D-Lib Magazine*, 12(12), 2006. ISSN 1082-9873. (Cited on page 123).

J. Stephen Downie, Mert Bay, Andreas F. Ehmann, and M. Cameron Jones. Audio cover song identification: Mirex 2006-2007 results and analyses. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008)*, September 2008. (Cited on page 48).

Frank Drewes and Johanna Högberg. An algebra for tree-based music generation. In *CAI*, pages 172–188, 2007. (Cited on pages 70 and 208).

Serge Dulucq and Laurent Tichit. Rna secondary structure comparison: exact analysis of the zhang-shasha tree edit algorithm. *Theoretical Computer Science*, 306(1-3):471 – 484, 2003. ISSN 0304-3975. doi: DOI:10.1016/S0304-3975(03)00323-2. URL http://www.sciencedirect.com/science/article/B6V1G-48TK62R-4/2/86634c04f5f06b3e0194246fd2ee7a58. (Cited on page 102).

Adriane Swalm Durey and Mark A. Clements. Melody spotting using hidden markov models. In J. Stephen Downie and David Bainbridge, editors,

Proceedings of the Second Annual International Symposium on Music Information Retrieval: ISMIR 2001, pages 109–117, Bloomington, IN, 2001. Indiana University. URL http://ismir2001.indiana.edu/pdf/durey.pdf,andhttp://music-ir.org/gsdl/ismir2001/pdf/durey.pdf(1022k). (Cited on page 41).

Douglas Eck and Norman Casagrande. Finding meter in music using an autocorrelation phase matrix and shannon entropy. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 504–509, 2005. (Cited on pages 26, 152, and 213).

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. doi: 10.1016/0364-0213(90)90002-E. URL http://www.isrl.uiuc.edu/~amag/langev/paper/elman90findingStructure.html. (Cited on page 43).

Ronald Engelbrecht. Statistical comparison measures for searching in melody databases. Technical report, Communication Systems and Algorithms Group, University of Bonn, 2002. (Cited on pages 12, 39, and 206).

D. Espí, P.J. Ponce de León, Carlos Pérez-Sancho, David Rizo, José Manuel Iñesta, F. Moreno-Seco, and Antonio Pertusa. A cooperative approach to style-oriented music composition. In *Proc. of the Int. Workshop on Artificial Intelligence and Music, MUSIC-AI*, pages 25–36, Hyderabad, India, 2007. (Cited on pages 152 and 213).

Pascal Ferraro and Pierre Hanna. Optimizations of local edition for evaluating similarity between monophonic musical sequences. In *In RIAO 2007: Proceedings of the 8th International Conference on Information Retrieval - LargeScale Semantic Access to Content (Text, Image, Video and Sound*, 2007. (Cited on pages 15, 17, 19, 20, 21, 28, and 31).

Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974. (Cited on page 153).

Allen Forte and Steven E. Gilbert. *Introduction to Schenkerian Analysis*. W.W.Norton, 1982. (Cited on page 64).

Klaus Frieler. Beat and meter extraction using gaussified onsets. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2004. (Cited on pages 24 and 26).

Joe Futrelle and J. Stephen Downie. Interdisciplinary research issues in music information retrieval: Ismir 2000–2002. *Journal of New Music Research*, 32:121–131, 2003. (Cited on page 1).

Jörg Garbers, Peter van Kranenburg, Anja Volk, Frans Wiering, Remco C. Veltkamp, and Louis P. Grijp. Using pitch stability among a group of aligned query melodies to retrieve unidentified variant melodies. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2007. (Cited on page 56).

Asif Ghias, Jonathan Logan, David Chamberlin, and Brian C. Smith. Query by humming: musical information retrieval in an audio database. In *In ACM Multimedia*, pages 231–236, 1995. (Cited on pages 2 and 20).

E. Gilbert and D. Conklin. A probabilistic context-free grammar for melodic reduction. *International Workshop on Artificial Intelligence and Music at IJCAI-07. Twentieth International Joint Conference on Artificial Intelligence, Hyderabad, India*, 2007. (Cited on pages xii, 61, and 206).

Carlos Gómez, Soraya Abad-Mota, and Edna Ruckhaus. An analysis of the mongeau-sankoff alogorithm for music information retrieval. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2007a. (Cited on page 32).

Carlos Gómez, Soraya Abad-Mota, and Edna Ruckhaus. An analysis of the mongeau-sankoff algorithm for music information retrieval. MIREX 2007, 2007b. (Cited on pages 15, 17, and 35).

Emilia Gómez. *Tonal Description of music audio signals*. PhD thesis, Universitat Pompeu Fabra, 2006. (Cited on page 48).

Michael Good and Geri Actor. Using musicxml for file interchange. *Web Delivering of Music, International Conference on*, 0:153, 2003. doi: http://doi.ieeecomputersociety.org/10.1109/WDM.2003.1233890. (Cited on pages 4, 119, and 203).

Maarten Grachten, Josep Lluis Arcos, and Ramon López de Mántaras. A comparison of different approaches to melodic similarity. In Christina Anagnostopoulou, M. Ferrand, and Alan Smaill, editors, *Music and artificial intelligence: Second international conference, ICMAI 2002, Edinburgh, Scotland, UK, September 12-14, 2002. Proceedings*. University of Edinburgh, Scotland, University of Edinburgh, Scotland, 2002. (Cited on pages 17, 19, 20, and 35).

Maarten Grachten, Josep Lluís Arcos, and Ramon López De Mántaras. Melodic similarity: Looking for a good abstraction level. In *Proc. ISMIR*, 2004. (Cited on pages 7, 19, 20, 25, 32, 35, 36, and 204).

Maarten Grachten, Josep Lluís Arcos, and Ramon López De Mántaras. Melody retrieval using the implication/realization model. In *6th Inter. Conf. On Music information retrieval. ISMIR 2005. (First prize of the MIREX Symbolic Melodic Similarity Contest)*., 2005. (Cited on pages 12, 32, 35, 36, and 205).

Bas De Haas, Remco C. Veltkamp, and Frans Wiering. Tonal pitch step distance: A similarity measure for chord progressions. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008)*, September 2008. (Cited on pages 47 and 48).

Amaury Habrard, José Manuel Iñesta, David Rizo, and Marc Sebban. Melody recognition with learned edit distances. *Lecture Notes in Computer Science*, 5342: 86–96, 2008. (Cited on pages 12, 15, 34, 145, 154, 206, and 214).

Masatoshi Hamanaka, Keiji Hirata, and Satoshi Tojo. Atta: Implementing gttm on a computer. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR'07)*, Vienna, Austria, September 2007. (Cited on pages 68 and 207).

Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 1st edition, September 2000. ISBN 1558604898. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1558604898. (Cited on pages 118 and 159).

Pierre Hanna and Pascal Ferraro. Polyphonic music retrieval by local edition of quotiented sequences. In *International Workshop on Content-Based Multimedia Indexing, CBMI*, pages 61–68, June 2007. doi: 10.1109/CBMI.2007.385393. (Cited on pages 15, 23, 34, 45, and 47).

Pierre Hanna, Matthias Robine, Pascal Ferraro, and Julien Allali. Improvements of alignment algorithms for polyphonic music retrieval. In *Computer Music Modeling and Retrieval 2008*, Copenhague, 2008. URL http://www.labri.fr/publications/mabiovis/2008/HRFA08. (Cited on pages 15, 24, 44, 45, 47, 75, 152, and 214).

Howard Hanson. *Harmonic materials of modern music*. Appleton-Century-Crofts (New York), 1960. (Cited on page 19).

S. Harford. Automatic segmentation, learning and retrieval of melodies using a self-organizing neural network. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2003. (Cited on pages 42 and 43).

Walter B. Hewlett. A base-40 number-line representation of musical pitch notation. *Musikometrika*, 4, 1992. (Cited on page 15).

Ruben Hillewaere, Bernard Manderick, and Darrell Conklin. Melodic models for polyphonic music. In *Proc. Second International Workshop on Machine Learning and Music (MML 2009)*, pages 19–24, Bled, Slovenia, September 2009. (Cited on page 36).

Tin Kam Ho, Jonathan J. Hull, and Sargur N. Srihari. Decision combination in multiple classifier systems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(1):66–75, 1994. ISSN 0162-8828. doi: http://dx.doi.org/10.1109/34.273716. (Cited on page 130).

Ludger Hofmann-Engl. Towards a cognitive model of melodic similarity. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2001. (Cited on pages 6 and 204).

Johanna Högberg. Wind in the willows - generating music by means of tree transducers. In *CIAA*, pages 153–162, 2005. URL http://www.springerlink.com/content/1m337277h72511lm/fulltext.pdf. (Cited on pages 63, 65, and 207).

Holger H. Hoos, Keith A. Hamel, Kai Renz, and Jürgen Kilian. The guido notation format: A novel approach for adequately representing score-level music. In *Proceedings of the International Computer Music Conference, International Computer Music Conference*, pages 451–454, 1998. URL http://www.cs.ubc.ca/~hoos/Publ/icmc98b.pdf. author's preprint. (Cited on pages 4 and 203).

Karin Höthker, Dominik Hörnel, and Christina Anagnostopoulou. Investigating the influence of representations and algorithms in music classification. *Computers and the Humanities*, 35(1):65–79, February 2001. URL http://www.springerlink.com/content/u0464766750p6445/fulltext.pdf. (Cited on pages 63, 64, and 207).

Jia-Lien Hsu, Arbee L. P. Chen, and C.-C. Liu. Efficient repeating pattern finding in music databases. In *CIKM '98: Proceedings of the seventh international conference on Information and knowledge management*, pages 281–288, New York, NY, USA, 1998. ACM. ISBN 1-58113-061-9. doi: http://doi.acm.org/10.1145/288627.288668. (Cited on pages 11 and 205).

Ning Hu and Roger B. Dannenberg. A comparison of melodic database retrieval techniques using sung queries. In *JCDL '02: Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 301–307, New York, NY, USA, 2002. ACM. ISBN 1-58113-513-0. doi: http://doi.acm.org/10.1145/544220.544292. (Cited on page 31).

Ning Hu, Roger B. Dannenberg, and Ann L. Lewis. A probabilistic model of melodic similarity. In *International Computer Music Conference (ICMC), Goteborg, Sweden*, pages 16–21, 2002. URL http://citeseer.ist.psu.edu/590349.html. (Cited on pages 34 and 35).

David Huron. Music information processing using the humdrum toolkit: Concepts, examples, and lessons. *Computer Music Journal*, 26(2):11–26, 2002. ISSN 0148-9267. doi: http://dx.doi.org/10.1162/014892602760137158. (Cited on pages 4, 17, and 203).

David Huron and Matthew Royal. What is melodic accent? what is melodic accent?. converging evidence from musical practice. *Music Perception*, 13(4):489–516, 1996. (Cited on page 26).

Plácido R. Illescas, David Rizo, and José Manuel Iñesta. Harmonic, melodic, and functional automatic analysis. In *Proceedings of the 2007 International Computer Music Conferrence*, volume I, pages 165–168, 2007. (Cited on pages 26 and 27).

Carsten Isert. The editing distance between trees. Technical report, Institut für Informatik, Technische Universität München, 1999. URL citeseer.ist.psu.edu/isert99editing.html. (Cited on page 100).

Tristan Jehan. *Creating Music by Listening*. PhD thesis, Media Arts and Sciences, MIT, 2005. (Cited on page 49).

Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of trees – an alternative to tree edit. *Theoretical Computer Science*, 143(1):137 – 148, 1995. ISSN 0304-3975. doi: DOI:10.1016/0304-3975(95)80029-9. URL http://www.sciencedirect.com/science/article/B6V1G-475914W-9/2/6801af9da317211542072be4138934af. (Cited on pages 98, 104, 105, 146, and 211).

Jacqueline Jones, Don Scarborough, and Benjamin Miller. Gtsim a computer simulation of music perception. *Computers and the Humanities*, 27(1):19–23, 01 1993. URL http://dx.doi.org/10.1007/BF01830713. (Cited on pages 68 and 207).

M.R. Jones. Dynamic pattern structure in music: recent theory and research. *Perception and Psychophysics*, 41(6):621–634, 1987. (Cited on page 26).

T. Kadota, M. Hirao, A. Ishino, M. Takeda, A. Shinohara, and F. Matsuo. Musical sequence comparison for melodic and rhythmic similarities. *International Symposium on String Processing and Information Retrieval*, 0:0111, 2001. doi: http://doi.ieeecomputersociety.org/10.1109/SPIRE.2001.10016. (Cited on page 32).

M. Kassler. Apl applied in music theory. In *APL '88: Proceedings of the international conference on APL*, pages 209–214, New York, NY, USA, 1988. ACM. ISBN 0-89791-253-5. doi: http://doi.acm.org/10.1145/55626.55654. (Cited on pages 64 and 207).

Jürgen Kilian and Holger H. Hoos. Musicblast - gapped sequence alignment for mir. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2004. (Cited on page 31).

Youngmoo E. Kim, Wei Chai, Ricardo Garcia, and Barry Vercoe. Analysis of a contour-based representation for melody. In *Proc. International Symposium on Music Information Retrieval*, 2000. (Cited on page 20).

Phillip B. Kirlin and Paul E. Utgoff. A framework for automated schenkerian analysis. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008)*, September 2008. (Cited on pages 64 and 207).

Carol L. Krumhansl. *Cognitive Foundations of Musical Pitch*. Oxford University Press, New York, NY, USA, 1990. (Cited on pages 51, 53, and 153).

William H. Kruskal and W. Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 7(260):583–621, December 1952. (Cited on page 130).

Tetsuji Kuboyama, Kilho Shin, and Tetsuhiro Miyahara. A hierarchy of tree edit distance measures. *Theoretical Computer Science and its Applications*, 2005. URL http://hdl.handle.net/2433/47289. (Cited on page 98).

Jan LaRue. *A catalogue of eighteenth-century symphonies: Thematic-identifier*. Bloomington: Indiana University Press, 1988. (Cited on page 3).

C. S. Lee. The rhythmic interpretation of simple musical sequences: Towards a perceptual model. In R. West, P. Howell, and I. Cross, editors, *Musical Structure and Cognition*, pages 53–69. Academic Press, London, 1985. (Cited on pages 59, 60, 61, 62, 73, 77, and 206).

Joon Ho Lee. Analyses of multiple evidence combination. In *SIGIR '97: Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 267–276, New York, NY, USA, 1997. ACM. ISBN 0-89791-836-3. doi: http://doi.acm.org/10.1145/258525.258587. (Cited on page 37).

Kjell Lemström. *String Matching Techniques for Music Retrieval*. PhD thesis, University of Helsinki, Finland, November 2000. URL `http://www.cs.helsinki.fi/u/klemstro/THESIS/`. (Cited on pages 12, 17, 31, 35, and 205).

Kjell Lemström and Pauli Laine. Musical information retrieval using music parameters. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 341–348, Ann Arbor, USA, October 1-6 1998. (Cited on page 20).

Kjell Lemström and Veli Mäkinen. On minimizing pattern splitting in multi-track string matching. *J. Discrete Algorithms*, 3(2-4):248–266, 2005. (Cited on page 44).

Kjell Lemström and Sami Perttu. Semex-an efficient music retrieval prototype. In *In 1st International Conference on Music Information Retrieval (ISMIR)*, 2000. (Cited on page 20).

Kjell Lemström and Anna Pienimäki. Approaches for content-based retrieval of symbolically encoded polyphonic music. In *9th International Conference on Music Perception and Cognition (ICMPC9)*, August 2006. (Cited on pages 44, 45, and 46).

Kjell Lemström and Anna Pienimaki. On comparing edit distance and geometric frameworks in content-based retrieval of symbolically encoded polyphonic music. *Musicae Scientiae*, 11:135–152, 2007. (Cited on page 38).

Kjell Lemström and Jorma Tarhio. Transposition invariant pattern matching for multi-track strings. *Nordic J. of Computing*, 10(3):185–205, 2003. ISSN 1236-6064. (Cited on pages 44 and 45).

Kjell Lemström and Esko Ukkonen. Including interval encoding into edit distance based music comparison and retrieval. In *In Proceedings of the AISB'2000 Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science*, pages 53–60, 2000. (Cited on page 35).

Kjell Lemström and Geraint A. Wiggins. Formalizing invariances for content-based music retrieval. In *ISMIR'09, 10th International Society for Music Information Retrieval Conference*, pages 591–596, Kobe, Japan, October 26-30 2009. (Cited on page 13).

Kjell Lemström, Veli Mäkinen, Anna Pienimäki, M. Turkia, and Esko Ukkonen. The c-brahms project. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2003. (Cited on pages 55 and 127).

Kjell Lemström, Gonzalo Navarro, and Yoan J. Pinzon. Practical algorithms for transposition-invariant string-matching. *Journal of Discrete Algorithms*, 3(2–4):267–292, 2005. (Cited on pages 32 and 35).

Kjell Lemström, Niko Mikkilä, and Veli Mäkinen. Fast index based filters for music retrieval. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008)*, September 2008. (Cited on pages 38, 43, 55, 127, and 206).

M. Leppig. Musikuntersuchungen im rechenautomaten. *Musica*, 41(2):140–150, 1987. (Cited on page 53).

Fred Lerdahl. *Tonal pitch space*. New York: Oxford University Press, 2001. (Cited on page 47).

Fred Lerdahl and Ray Jackendoff. *A Generative Theory of Tonal Music*. MIT Press, 1983. (Cited on pages 67, 69, and 207).

Daniel J. Levitin. Memory for musical attributes. In *Music, cognition, and computerized sound: an introduction to psychoacoustics*, pages 209–227. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-03256-2. (Cited on pages 11 and 205).

Ming Li and Paul Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, February 1997. ISBN 0387948686. URL `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0387948686`. (Cited on page 56).

M. Ling and R. Sharp. Melody classification using a similarity metric based on kolmogorov complexity. In *Conference on Sound and Music Computing*, 2004. (Cited on pages 36, 37, and 56).

Christiane Linster. On analyzing and representing musical rhythm. *Understanding music with AI: perspectives on music cognition*, pages 414–427, 1992. (Cited on pages 59, 60, and 206).

Hugh Christopher Longuet-Higgins. The perception of music. *Interdisciplinary Science Review*, 3:148–156, 1978. (Cited on pages 60 and 206).

Anna Lubiw and Luke Tanur. Pattern matching in polyphonic music as a weighted geometric translation problem. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2004. (Cited on pages 38 and 55).

Søren Tjagvad Madsen, Rainer Typke, and Gerhard Widmer. Automatic reduction of midi files preserving relevant musical content automatic reduction of midi files preserving relevant musical content. In *6th International Workshop on Adaptive Multimedia Retrieval (AMR'08)*, June 2008. (Cited on page 154).

Donncha Ó. Maidín. A geometrical algorithm for melodic difference. In Walter B. Hewlett and Eleanor Selfridge-Field, editors, *Melodic Similarity: Concepts, Procedures, and Applications. Computing in Musicology*, volume 11, chapter 1, pages 65—72. MIT Press, 1998. (Cited on pages 11, 26, 38, 39, and 205).

Veli Mäkinen, Gonzalo Navarro, and Esko Ukkonen. Algorithms for transposition invariant string matching. In *STACS '03: Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science*, pages 191–202, London, UK, 2003. Springer-Verlag. ISBN 3-540-00623-0. (Cited on pages 29, 32, and 35).

H.B. Mann and D.R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18:50–60, 1947. (Cited on page 130).

Arpi Mardirossian and Elaine Chew. Music summarization via key distributions: Analyses of similarity assessment across variations. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 234–239, 2006. (Cited on pages 51 and 52).

Alan Marsden. Representing melodic patterns as networks of elaborations. *Computers and the Humanities*, 35(1):37–54, 2001. URL http://www.springerlink.com/content/p5r588275712171g/fulltext.pdf. (Cited on pages 64 and 207).

Alan Marsden. Extending a network-of-elaborations representation to polyphonic music: Schenker and species counterpoint. *Sound and Music Computing, Paris*, October 2004. URL http://eprints.lancs.ac.uk/23/1/SMC04.pdf. (Cited on pages 64 and 207).

Alan Marsden. Generative structural representation of tonal music. *Journal of New Music Research*, 34(4):409 – 428, December 2005. URL http://www.informaworld.com/smpp/content~content=a746297533~db=all~order=page. (Cited on pages 64, 66, 67, and 207).

Alan Marsden. Automatic derivation of musical structure: A tool for research on schenkerian analysis. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR'07)*, Vienna, Austria, September 2007. (Cited on pages 64 and 207).

J.G. Martin. Rhythmic (hierarchical) versus serial structure in speech and other behavior. *Psychol Rev.*, 79(6):487–509, 1972. (Cited on pages 60 and 206).

Colin Meek and William P. Birmingham. Thematic extractor. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2001. (Cited on page 44).

Colin Meek and William P. Birmingham. Johnny can't sing: A comprehensive error model for sung music queries. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2002. (Cited on page 19).

Massimo Melucci and Nicola Orio. Smile: a system for content-based musical information retrieval environments. In *RIAO' 2000 Conference proceedings, Vol 2*, page 1261, 2000. (Cited on page 36).

David Meredith. *Computing Pitch Names in Tonal Music: A Comparative Analysis of Pitch Spelling Algorithms*. PhD thesis, University of Oxford, UK, April 2006. (Cited on page 14).

Benoît Meudic. A causal algorithm for beat-tracking. In *2nd conference on understanding and creating music*, Caserta, Italy, Novembre 2002a. URL `http://mediatheque.ircam.fr/articles/textes/Meudic02b`. (Cited on page 49).

Benoît Meudic. Automatic meter extraction from midi files. In *Journées d'informatique musicale. Marseille: Centre National de Création Musicale*, 2002b. (Cited on pages 26, 152, and 213).

Benoît Meudic. Musical similarity in a polyphonic context : A model outside time. In *Proceedings of the XIV Colloquium on Musical Informatics, Firenze, Italy, May 8-10*, 2003. URL `http://citeseer.ist.psu.edu/636525.html`. (Cited on pages 5, 49, and 50).

The Complete MIDI 1.0 Detailed Specification. MIDI Manufacturers Association, 1996. (Cited on pages 4 and 204).

Michael Mitzenmacher and Sean Owen. Estimating resemblance of midi documents. *ALENEX*, 2001:79–90, 2001. (Cited on page 53).

Takao Miura and Isamu Shioya. Similarity among melodies for music information retrieval. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 61–68, New York, NY, USA, 2003. ACM. ISBN 1-58113-723-0. doi: http://doi.acm.org/10.1145/956863.956877. (Cited on page 53).

Marcel Mongeau and David Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24(3):161–175, 1990. (Cited on pages 11, 12, 15, 23, 28, 32, 34, 75, 126, 152, 205, and 214).

Daniel Müllensiefen and Klaus Frieler. Cognitive adequacy in the measurement of melodic similarity: Algorithmic vs. human judgements. *Computing in Musicology*, 13:147–176, 2004a. (Cited on pages 20, 21, 24, 26, 35, 48, and 51).

Daniel Müllensiefen and Klaus Frieler. Optimizing measures of melodic similarity for the exploration of a large folk song database. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2004b. (Cited on page 153).

E. Narmour. *The Analysis and cognition of basic melodic structures*. University of Chicago Press, 1990. (Cited on page 35).

Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443 – 453, 1970. ISSN 0022-2836. doi: DOI:10.1016/0022-2836(70)90057-4. URL http://www.sciencedirect.com/science/article/B6WK7-4DN8W3K-7X/2/0d99b8007b44cca2d08a031a445276e1. (Cited on page 30).

Giovanna Neve and Nicola Orio. Indexing and retrieval of music documents through pattern analysis and data fusion techniques. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2004. (Cited on pages 17, 19, 24, and 25).

Alberto Novello, Martin F. McKinney, and Armin Kohlrausch. Perceptual evaluation of music similarity. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 246–249, 2006. (Cited on pages 6 and 204).

Nicola Orio. Combining multilevel and multifeature representation to compute melodic similarity. MIREX, 2005. (Cited on pages 19, 25, and 37).

Nicola Orio. Music retrieval: A tutorial and review. *Foundations and Trends in Information Retrieval*, 1(1):1–90, 2006. (Cited on page 1).

Bryan Pardo and William P. Birmingham. Algorithms for chordal analysis. *Computer Music Journal*, 26(2):27–49, 2002a. ISSN 0148-9267. doi: http://dx.doi.org/10.1162/014892602760137167. (Cited on page 45).

Bryan Pardo and William P. Birmingham. Encoding timing information for musical query matching. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2002b. (Cited on pages 19, 21, and 25).

Bryan Pardo and Manan Sanghi. Polyphonic musical sequence alignment for database search. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 215–222, 2005. (Cited on pages 44, 45, 75, 152, and 214).

Bryan Pardo, Jonah Shifrin, and William Birmingham. Name that tune: a pilot study in finding a melody from a sung query. *Journal of the American Society for Information Science and Technology (JASIST)*, 55(4):283–300, 2004. ISSN 1532-2882. doi: http://dx.doi.org/10.1002/asi.10373. (Cited on pages 12, 25, 31, 41, 42, and 206).

Denys Parsons. *Dictionary of Tunes and Musical Themes*. Cambridge: Spencer Brown, 1975. (Cited on pages 3 and 20).

Tomás Pérez-García, José Manuel Iñesta, and David Rizo. metamidi: a tool for automatic metadata extraction from midi files. In Andrea Rauber, Nicola Orio, and David Rizo, editors, *Proceedings of the Workshop on Exploring Musical Information Spaces, ECDL 2009*, pages 36–40, Corfu, Greece, October 2009. ISBN 978-84-692-6082-1. (Cited on pages 11 and 205).

Carlos Pérez-Sancho, David Rizo, and José Manuel Iñesta. Genre classification using chords and stochastic language models. *Connection Science*, 21(2, 3):145–159, May 2009. ISSN 0954-0091. (Cited on page 153).

Antonio Pertusa and José Manuel Iñesta. Multiple fundamental frequency estimation using gaussian smoothness. In *Proc. of the IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, ICASSP 2008*, pages 105–108, Las Vegas, USA, 2008. ISBN 1-4244-1484-9. (Cited on page 4).

Jeremy Pickens. A comparison of language modeling and probabilistic text information retrieval approaches to monophonic music retrieval. In *International Symposium on Music Information Retrieval*, 2000. (Cited on page 36).

Jeremy Pickens. *Harmonic Modeling for Polyphonic Music Retrieval*. PhD thesis, University of Massachusetts Amherst, MA, USA, May 2004. URL `http://citeseer.ist.psu.edu/pickens04harmonic.html`. (Cited on pages 5, 12, 41, 51, 52, and 206).

Jeremy Pickens and Tim Crawford. Harmonic models for polyphonic music retrieval. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 430–437, New York, NY, USA, 2002. ACM. ISBN 1-58113-492-4. doi: http://doi.acm.org/10.1145/584792.584863. (Cited on page 10).

Anna Pienimäki. Indexing music databases using automatic extraction of frequent phrases. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2002. (Cited on page 45).

Anna Pienimäki and Kjell Lemström. Clustering symbolic music using paradigmatic and surface level analyses. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2004. (Cited on page 45).

Alberto Pinto and Paolo Tagliolato. A generalized graph-spectral approach to melodic modeling and retrieval. In *MIR '08: Proceeding of the 1st ACM international conference on Multimedia information retrieval*, pages 89–96, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-312-9. doi: http://doi.acm.org/10.1145/1460096.1460113. (Cited on pages 12 and 205).

Alberto Pinto, Reinier H. van Leuken, M. Fatih Demirci, Frans Wiering, and Remco C. Veltkamp. Indexing music collections through graph spectra. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR'07)*, Vienna, Austria, September 2007. (Cited on page 40).

M D Plumbley, S A Abdallah, J P Bello, M E Davies, and G Monti. Automatic music transcription and audio source separation. *Cybernetics and Systems*, 33(6):603–627, 2002. (Cited on page 4).

L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. (Cited on pages 41, 42, and 51).

D. Rafailidis, Alexandros Nanopoulos, Emilios Cambouropoulos, and Yannis Manolopoulos. Detection of stream segments in symbolic musical data. In *Proceedings 9th International Symposium on Music Information Retrieval (ISMIR'2008)*, Philadelphia, 2008. URL http://delab.csd.auth.gr/papers/ISMIR08rncm.pdf. (Cited on pages 6, 43, 45, and 153).

Jan Ramon and Maurice Bruynooghe. A polynomial time computable metric between point sets. *Acta Informatica*, 37(10):765–780, 2001. (Cited on page 56).

J. Reiss and Mark B. Sandler. Benchmarking music information retrieval systems. The MIR/MDL Evaluation Project White Paper Collection, 2002. (Cited on page 1).

David Rizo and José M. Iñesta. Tree-structured representation of melodies for comparison and retrieval. In *Proc. of the* 2^{nd} *Int. Conf. on Pattern Recognition in Information Systems, PRIS 2002*, pages 140–155, Alicante, Spain, 2002. (Cited on pages 59, 73, 123, and 206).

David Rizo and José M. Iñesta. New partially labelled tree similarity measure: a case study. In E. R. Hancok, R. C. Wilson, T. W. Ilkay, and F. Escolano, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, volume 6218 of *Lecture Notes in Computer Science (LNCS)*, pages 296–305, Cesme, Izmir, Turkey, August 2010. Springer. ISBN 978-3-642-14979-5. (Cited on pages 109 and 211).

David Rizo, F. Moreno-Seco, and José Manuel Iñesta. Tree-structured representation of musical information. *Lecture Notes in Computer Science - Lecture Notes in Artificial Intelligence*, 2652:838–846, 2003. (Cited on pages 12, 15, 24, and 206).

David Rizo, P.J. Ponce de León, Carlos Pérez-Sancho, Antonio Pertusa, and José Manuel Iñesta. A pattern recognition approach for melody track selection in midi files. In Tindale A. Dannenberg R., Lemström K., editor, *Proc. of the 7th Int. Symp. on Music Information Retrieval ISMIR 2006*, pages 61–66, Victoria, Canada, 2006a. ISBN: 1-55058-349-2. (Cited on pages 43, 155, and 215).

David Rizo, José Manuel Iñesta, and P.J. Ponce de León. Tree model of symbolic music for tonality guessing. In *Proc. of the IASTED Int. Conf. on Artificial Intelligence and Applications, AIA 2006*, pages 299–304, Innsbruck, Austria, 2006b. IASTED, Acta Press. ISBN 0-88986-404-7. (Cited on pages 152 and 213).

David Rizo, F. Moreno-Seco, José Manuel Iñesta, and L. Micó. *Efficient search with tree-edit distance for melody recognition*, chapter 14, pages 218–244. Centre de Visió per Computador, 2006c. (Cited on page 154).

David Rizo, Kjell Lemström, and José Manuel Iñesta. Tree structured and combined methods for comparing metered polyphonic music. In *Proc. Computer Music Modeling and Retrieval 2008 (CMMR'08)*, pages 263–278, Copenhagen, Denmark, Copenhagen, Denmark, May 19-23 2008. (Cited on pages 12, 154, 206, and 214).

David Rizo, Kjell Lemström, and José Manuel Iñesta. Ensemble of state-of-the-art methods for polyphonic music comparison. In Andrea Rauber, Nicola Orio, and David Rizo, editors, *Proceedings of the Workshop on Exploring Musical Information Spaces, ECDL 2009*, pages 46–51, Corfu, Greece, October 2009a. ISBN 978-84-692- 6082-1. (Cited on pages 154 and 214).

David Rizo, Kjell Lemström, and José Manuel Iñesta. Tree representation in combined polyphonic music comparison. *Computer Music Modeling and Retrieval. Genesis of Meaning in Sound and Music. Lecture Notes in Computer Science*, 5493/2009:177–195, 2009b. URL http://dx.doi.org/10.1007/978-3-642-02518-1_12. (Cited on pages 43, 154, and 214).

Matthias Robine, Pierre Hanna, and Pascal Ferraro. Music similarity: improvements of edit-based algorithms by considering music theory. In *Multimedia Information Retrieval*, pages 135–142, 2007a. (Cited on pages 26 and 29).

Matthias Robine, Pierre Hanna, Pascal Ferraro, and Julien Allali. Adaptation of string matching algorithms for identification of near-duplicate music documents. In *Workshop on Plagiarism Analysis, Authorship Identification, and Near-Duplicate Detection*, pages 37–43, Amsterdam, ND, jul 2007b. URL http://www.labri.fr/publications/mabiovis/2007/RHFA07. simbals. (Cited on pages 7 and 28).

Matthias Robine, Pierre Hanna, Thomas Rocher, Julien Allali, and Pascal Ferraro. Structured representation of harmony for music retrieval. Demo at ISMIR'08, 2008. (Cited on page 48).

Pierre-Yves Rolland. *Automated Discovery of Regularities in Sequences and its Application to Music Analysis*. PhD thesis, Université Paris 6, 1998. (Cited on pages 7, 29, and 204).

Pierre-Yves Rolland. Discovering patterns in musical sequences. *Journal of New Music Research*, 28(4):334—350, 1999. (Cited on page 73).

Christian André Romming and Eleanor Selfridge-Field. Algorithms for polyphonic music retrieval: the hausdorff metric and geometric hashing. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2007. (Cited on page 38).

Tae-Wan Ryu, Tae-wan Ryu, and Christoph F. Eick. A unified similarity measure for attributes with set or bag of values. In *Proc. Sixth International Workshop on Rough Sets, Data Mining and Granular Computing (RSDMGrC '98*, 1998. (Cited on pages 118 and 159).

Stanley Sadie, editor. *The New Grove Dictionary of Music and Musicians*. Ed. Macmillan, 2nd ed. edition, 2000. (Cited on pages 2 and 5).

Helmut. Schaffrath. *The Essen Folk Song Collection in the Humdrum Kern Format*. Menlo Park, CA: Center for Computer Assisted Research in the Humanities., 1995. (Cited on pages 61 and 71).

H. Schenker. *Der freie Satz*. Vienna: Universal Edition. Published in English as *Free Composition*, new york: longman, 1979. edition, 1935. (Cited on page 66).

Eleanor Selfridge-Field. *Beyond MIDI: The handbook of musical codes*. MIT Press, Cambridge, Massachusetts, USA, 1997. (Cited on pages 4 and 203).

Eleanor Selfridge-Field. Conceptual and representational issues in melodic comparison. In Walter B. Hewlett and Eleanor Selfridge-Field, editors, *Melodic Similarity: Concepts, Procedures, and Applications. Computing in Musicology*, volume 11, chapter 1, pages 223–230. MIT Press, 1998. (Cited on pages 3, 6, 7, 11, 15, 16, 21, 23, 26, 59, 204, 205, and 206).

Stanley M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6 (6):184–186, 1977. (Cited on pages 98, 106, and 210).

Joan Serrà and E. Gomez. Audio cover song identification based on tonal sequence alignment. *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 61–64, 31 2008-April 4 2008. ISSN 1520-6149. doi: 10.1109/ICASSP.2008.4517546. (Cited on pages 35 and 48).

J.F. Serrano and José Manuel Iñesta. Comparison of hanson intervallic representations for music information retrieval. In A. Gelbuck and S. Suárez-Guerra, editors, *Proc. of the 15th Int. Conf. on Computing (CIC-2006)*, pages 147–153. IEEE computer society press, 2006a. ISBN: 0-7695-2708-6. (Cited on page 19).

J.F. Serrano and José Manuel Iñesta. Music information retrieval through melodic similarity using hanson intervallic analysis. *Research in Computing Science*, 20:131–142, 2006b. (Cited on page 37).

D. Shasha and K. Zhang. Fast parallel algorithms for the unit cost editing distance between trees. In *SPAA '89: Proceedings of the first annual ACM symposium on Parallel algorithms and architectures*, pages 117–126, New York, NY, USA, 1989. ACM. ISBN 0-89791-323-X. doi: http://doi.acm.org/10.1145/72935.72949. URL http://portal.acm.org/ft_gateway.cfm?id=72949&type=pdf&coll=GUIDE&dl=GUIDE&CFID=7050436&CFTOKEN=56854202. (Cited on pages 98 and 99).

Dennis Shasha and Kaizhong Zhang. Fast algorithms for the unit cost editing distance between trees. *J. Algorithms*, 11(4):581–621, 1990. ISSN 0196-6774. doi: http://dx.doi.org/10.1016/0196-6774(90)90011-3. URL http://portal.acm.org/citation.cfm?id=95994. (Cited on page 101).

Dennis Shasha and Kaizhong Zhang. Approximate tree pattern matching. In *Pattern Matching Algorithms*, pages 341–371. Oxford University Press, 1997. URL citeseer.ist.psu.edu/shasha95approximate.html. (Cited on pages 98 and 106).

Ilya Shmulevich, Olli Yli-Harja, Edward J. Coyle, Dirk-Jan Povel, and Kjell Lemström. Perceptual issues in music pattern recognition - complexity of rhythm and key finding. In *Computers and the Humanities*, volume 35, pages 23–35, 2001. (Cited on page 51).

Michael Skalak, Jinyu Han, and Bryan Pardo. Speeding melody search with vantage point trees. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008)*, September 2008. (Cited on pages 70 and 208).

Alan Smaill, Geraint A. Wiggins, and Mitch Harris. Hierarchical music representation for composition and analysis. *Computers and the Humanities*, 27(1):7–17, 01 1993. URL http://dx.doi.org/10.1007/BF01830712. (Cited on pages 63 and 207).

T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J Mol Biol*, 147(1):195–197, March 1981. ISSN 0022-2836. URL http://view.ncbi.nlm.nih.gov/pubmed/7265238. (Cited on pages 31 and 32).

Stephen W Smoliar. A computer aid for schenkerian analysis. In *ACM 79: Proceedings of the 1979 annual conference*, pages 110–115, New York, NY, USA, 1979. ACM. ISBN 0-89791-008-7. doi: http://doi.acm.org/10.1145/800177.810043. (Cited on pages 64 and 207).

Johan Sundberg and Bjorn Lindblom. *Generative theories in language and music descriptions*, pages 263–286. MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-19319-1. (Cited on pages 60 and 206).

Iman S. H. Suyoto and Alexandra L. Uitdenbogerd. Simple efficient n-gram indexing for effective melody retrieval. MIREX 2005, 2005. (Cited on page 127).

Sara Taheri-Panah and Andrew MacFarlane. Music information retrieval systems: why do individuals use them and what are their needs? In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2004. (Cited on page 1).

Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/322139.322143. (Cited on pages 98 and 101).

Luke Tanur. *A Geometric Approach to Pattern Matching in Polyphonic Music*. PhD thesis, University of Waterloo, 2005. URL http://etd.uwaterloo.ca/etd/ltanur2005.pdf. (Cited on pages 12, 24, 38, and 206).

Heinrich Taube. Automatic tonal analysis: Toward the implementation of a music theory workbench. *Computer Music Journal*, 23(4):18–32, 1999. ISSN 0148-9267. doi: http://dx.doi.org/10.1162/014892699559977. URL http://www.mitpressjournals.org/doi/pdfplus/10.1162/014892699559977. (Cited on page 44).

H. G Tekman. Interactions of perceived intensity, duration, and pitch in pure tone sequences. *Music Perception*, 14, 1997. (Cited on page 26).

D. Temperley. *The Cognition of Basic Musical Structure*. MIT Press, 2001. (Cited on page 45).

David Temperley and Daniel Sleator. Modeling meter and harmony: A preference-rule approach. *Computer Music Journal*, 23(1):10–27, 1999. ISSN 0148-9267. doi: http://dx.doi.org/10.1162/014892699559616. (Cited on page 26).

Joseph M. Thomassen. Melodic accent: Experiments and a tentative model. *The Journal of the Acoustical Society of America*, 71(6):1596–1605, 1982. doi: 10.1121/1.387814. URL http://link.aip.org/link/?JAS/71/1596/1. (Cited on page 26).

H. Touzet. Comparing similar ordered trees in linear-time. *Journal of Discrete Algorithms*, 2006. doi: 10.1016/j.jda.2006.07.002. URL http://dx.doi.org/10.1016/j.jda.2006.07.002. (Cited on page 104).

Wei-Ho Tsai, Hung-Ming Yu, and Hsin-Min Wang. Query-by-example technique for retrieving cover versions of popular songs with similar melodies. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 183–190, 2005. (Cited on pages 31 and 32).

Rainer Typke. *Music Retrieval based on Melodic Similarity*. PhD thesis, Utrecht University, Netherlands, February 2007. URL http://rainer.typke.org/publications/music-IR-melodic-similarity.pdf. (Cited on pages 12, 39, 54, 56, and 206).

Rainer Typke, Rainer Typke, Panos Giannopoulos, Panos Giannopoulos, Remco C. Veltkamp, Remco C. Veltkamp, Frans Wiering, Frans Wiering, and Rene Van Oostrum. Using transportation distances for measuring melodic similarity. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR) Proceedings*, pages 107–114, 2003. (Cited on pages 17, 26, and 28).

Rainer Typke, Marc Den Hoed, Justin De Nooijer, Frans Wiering, Remco C. Veltkamp, Rainer Typke, Marc Den Hoed, Justin De Nooijer, Frans Wiering, and Remco C. Veltkamp. A ground truth for half a million musical incipits, 2004. (Cited on page 123).

Rainer Typke, Frans Wiering, and Remco C. Veltkamp. A survey of music information retrieval systems. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 153–160, 2005a. (Cited on page 1).

Rainer Typke, Frans Wiering, and Remco C. Veltkamp. Evaluating the earth mover's distance for measuring symbolic melodic similarity. MIREX 2005, 2005b. (Cited on pages 40 and 56).

Rainer Typke, Remco C. Veltkamp, and Frans Wiering. A measure for evaluating retrieval techniques based on partially ordered ground truth lists. *Multimedia and Expo, IEEE International Conference on*, 0:1793–1796, 2006. doi: http://doi.ieeecomputersociety.org/10.1109/ICME.2006.262900. (Cited on page 123).

George Tzanetakis and Perry Cook. Marsyas: a framework for audio analysis. *Org. Sound*, 4(3):169–175, 1999. ISSN 1355-7718. doi: http://dx.doi.org/10.1017/S1355771800003071. (Cited on page 2).

Alexandra L. Uitdenbogerd. *Music Information Retrieval Technology*. PhD thesis, RMIT University, Melbourne, Victoria, Australia, July 2002. URL http://goanna.cs.rmit.edu.au/~alu/research/mythesis.pdf. (Cited on pages 1, 12, 17, 31, 37, and 205).

Alexandra L. Uitdenbogerd. N-gram pattern matching and dynamic programming for symbolic melody search. MIREX 2007, 2007. (Cited on page 19).

Alexandra L. Uitdenbogerd and Yaw Wah Yap. Was parsons right? an experiment in usability of music representations for melody-based music retrieval. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2003. (Cited on pages 15, 19, 20, 21, 23, 24, and 26).

Alexandra L. Uitdenbogerd and Justin Zobel. Manipulation of music for melody matching. In *ACM Multimedia*, pages 235–240, 1998. URL citeseer.ist.psu.edu/uitdenbogerd98manipulation.html. (Cited on page 43).

Alexandra L. Uitdenbogerd and Justin Zobel. Melodic matching techniques for large music databases. In *Proc. ACM Multimedia*, 1999. URL citeseer.ist.psu.edu/uitdenbogerd99matching.html. (Cited on pages 13, 15, 19, 20, 26, 28, 31, and 36).

Alexandra L. Uitdenbogerd and Justin Zobel. Music ranking techniques evaluated. In *Proc. Australasian Computer Science Conference*, pages 275–283, 2002. (Cited on pages 37 and 43).

Alexandra L. Uitdenbogerd and Justin Zobel. An architecture for effective music information retrieval. *Journal of the American Society for Information Science and Technology (JASIST)*, 55(12):1053–1057, 2004. ISSN 1532-2882. doi: http://dx.doi.org/10.1002/asi.20057. (Cited on page 43).

L. Uitdenbogerd, Alexandra, Abhijit Chattaraj, and Justin Zobel. Methodologies for evaluation of note-based music-retrieval systems. *INFORMS J. on Computing*, 18 (3):339–347, 2006. ISSN 1526-5528. doi: http://dx.doi.org/10.1287/ijoc.1050.0139. (Cited on page 37).

Esko Ukkonen, Kjell Lemström, and Veli Mäkinen. Geometric algorithms for transposition invariant content based music retrieval. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2003a. (Cited on pages 12, 38, and 55).

Esko Ukkonen, Kjell Lemström, and Veli Mäkinen. Sweepline the music! In *Computer Science in Perspective*, pages 330–342, 2003b. (Cited on page 38).

G. Valiente. An efficient bottom-up distance between trees. *International Symposium on String Processing and Information Retrieval*, 0:0212, 2001. doi: http://doi.ieeecomputersociety.org/10.1109/SPIRE.2001.10019. (Cited on pages 80, 98, 104, 107, 108, 109, and 211).

Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, January 1974. ISSN 0004-5411. doi: 10.1145/321796.321811. URL http://dx.doi.org/10.1145/321796.321811. (Cited on pages 11 and 205).

T. A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/MC.1984.1659158. (Cited on page 57).

Geraint A. Wiggins, Kjell Lemström, and David Meredith. Sia(m)ese: An algorithm for transposition invariant, polyphonic content-based music retrieval. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, 2002. (Cited on pages 12 and 54).

Feng Yahzong, Yueting Zhuang, and Yunhe Pan. Query similar music by correlation degree. In *PCM '01: Proceedings of the Second IEEE Pacific Rim Conference on Multimedia*, pages 885–890, London, UK, 2001. Springer-Verlag. ISBN 3-540-42680-9. (Cited on pages 42 and 43).

K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989. ISSN 0097-5397. doi: http://dx.doi.org/10.1137/0218082. URL http://portal.acm.org/citation.cfm?id=76082. (Cited on pages 98, 101, 102, 105, 106, and 210).

Glossary

$\mathcal{P}$	Note in common music notation sort. 13, 15, 21
α	Algorithm: setup or configuration of a method. xiv, 120, 130, 132
child	Node child. 81, 107, 111, 168
compound	True if the meter is compound. 83
children	Node children. 80, 100, 101, 107, 114
$\mathcal{D}$	Set of documents retrieved. 120, 121
depth	Node depth. 80, 101, 116
desc	Descendants of a node. 81, 83, 102, 103
h	Tree height. 80, 83, 116, 166
indel	Insertion and deletion cost. 30, 31
label	Label contained in a node. 80, 85, 100, 101, 164
leaf	True if the node is a leaf. 80, 111, 168
leaves	Set of leaves of a tree. 80, 101
l	Tree pruning level. 74, 75, 86, 90–92, 116, 125, 137, 138, 140, 142, 144–148, 183–186, 188, 189, 192–195, 198–201
mdur	Meter duration. 83, 84
nodedur	Node duration. 83
P1	C-Brahms algorithm P1. 55
p_{12}	Pitch class encoding. 14–16, 40
P2	C-Brahms algorithm P2. 55
p_{21}	Base-21 pitch encoding. 14, 16
P2v5	C-Brahms algorithm P2v5. 55
P2v6	C-Brahms algorithm P2v6. 55
P3	C-Brahms algorithm P3. 55
p_{40}	Base-40 pitch encoding. 14, 17
p_7	Base-7 pitch encoding. 15
p_{abs}	Absolute pitch encoding. 14, 17, 20, 29, 35
par	Node parent. 80, 100, 107
p_c	Pitch contour encoding. 14, 20

p_{dm}	Pitch directed-modulo-12 encoding. 14, 19
p_{hdc}	Pitch high definition contour encoding. 14, 20, 21
p_{ift}	Interval from tonic pitch encoding. 14–16, 51, 54, 118
p_{itv}	Pitch Interval encoding. 14, 19, 20, 27, 35, 41
$\mathcal{Q}$	Corpus of queries. 120
$\mathcal{R}$	Set of relevant documents. 120, 121
rank	Node arity or rank. 80, 81, 104, 105, 107, 110, 111, 168
r_c	Contour rhythm encoding. 24
r_{dabs}	Absolute duration rhythm encoding. 23–25, 27, 29
r_{dior}	IOR discretized rhythm encoding. 41
r_{hdc}	High definition contour rhythm encoding. 24, 25, 126
r_{ioi}	IOI rhythm encoding. 25, 35
r_{ior}	IOR rhythm encoding. 25, 41
rlabel	Label of the root of the tree. 80, 103, 105, 107, 110, 111, 114, 115, 168
root	Tree root. 80, 81
r_{pim}	Position in measure rhythm encoding. 26
$\mathcal{R}_r$	Set of relevant documents retrieved. 120, 121
r_{tabs}	Absolute time rhythm encoding. 23
Σ_p	Pitch alphabet. 13, 27, 28, 34
Σ_r	Rhythm alphabet. 27, 28, 34

Acronyms

ADR	Average Dynamic Recall. 123, 139, 141
DOP	Data-Oriented Parsing. 61
DP	Delta Pitch. 19
DTW	Dynamic Time Warping. 31, 48
EMD	Earth Mover Distance. 39, 56
GTTM	Generative Theory of Tonal Music. 67, 68, 77, 86, 95, 207, 210, 214
HMM	Hidden Markov Model. 41, 42, 51
HPCP	Harmonic Pitch Class Profiles. 48
I/R	Implication / Realization Narmour model. 36
IDF	Inverse Document Frequency. 37
IOI	Inter-Onset Interval. 25, 28, 35, 43, 53
IOR	Inter-Onset interval Ratio. 25, 41, 50
IORatio	Inter-Onset Interval Ratio. 25
IORr	Inter-Onset interval Ratio. 25
MIR	Music Information Retrieval. 1, 2, 4, 7, 11, 203–205
MRR	Mean Reciprocal Rank. 121, 135–138, 140–142, 144, 147
NHT	Non-Harmonic Tone. 26–29
OMR	Optical Music Recognition. 2, 4, 20
PaC	Precision-at-*class*. 121
PIT	Pitch Interval in semitones with the previous note. 19
PR	Precision. 121
QBE	Query-By-Example. 2, 20
QBH	Query-By-Humming. 2, 20, 35
QBM	Query-By-Melody. 2
QPI	Quantized, Partially Overlapping Intervals. 20
RC	Recall. 121

RR Reciprocal rank. 121

SMF Standard MIDI files. 4, 23, 28, 119, 122, 204

TF Term Frequency. 37

VLDC Variable-Length Doesn't Care. 98

Index

Reunido el Tribunal que suscribe en el día de la fecha acordó otorgar, por a la Tesis Doctoral de Don/Dña. David Rizo Valero la calificación de .

Alicante de de

El Secretario,

El Presidente,

UNIVERSIDAD DE ALICANTE
CEDIP

La presente Tesis de D. David Rizo Valero ha sido registrada con el nº _____ del registro de entrada correspondiente.

Alicante ___ de _____ de _____

El Encargado del Registro,

La defensa de la tesis doctoral realizada por D/Dª David Rizo Valero se ha realizado en las siguientes lenguas: y , lo que unido al cumplimiento del resto de requisitos establecidos en la Normativa propia de la UA le otorga la mención de “Doctor Europeo”.

Alicante, de de

EL SECRETARIO EL PRESIDENTE

Universitat d'Alacant
Universidad de Alicante
Departament de Llenguatges i Sistemes Informàtics
Departamento de Lenguajes y Sistemas Informáticos