The Musical Score: a challenging goal for automatic music transcription

Francesco Foscarin

le cnam

École Doctorale d'Informatique, Télécommunications et Electronique

Laboratoire Cédric - Centre d'études et de recherche en informatique et communication

THÈSE DE DOCTORAT

présentée par : **Francesco Foscarin**
soutenue le : **10 Decembre 2020**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Discipline : **Informatique**

Spécialité : **Informatique**

The Musical Score:
a challenging goal for automatic music transcription

THÈSE dirigée par

M. Rigaux Philippe	*Professeur, CNAM*
M. Jacquemard Florent	*Chargé de recherche, INRIA/CNAM*

et co-encadrée par

M. Fournier-S'niehotta Raphaël	*Maître de conférences, CNAM*

RAPPORTEURS

Mme Levé Florence	*Maître de conférences, UPJV*
M. Rohrmeier Martin	*Professeur, EPFL*

PRÉSIDENT DU JURY

Mme Doucet Anne	*Professeur, Sorbonne Université*

EXAMINATEURS

M. Chouvel Jean-Marc	*Professeur, Sorbonne Université*
Mme Liem Cynthia	*Maître de conférences, TU Delft*

Acknowledgements

I would like to thank my supervisors, Philippe Rigaux and Florent Jacquemard, and my co-supervisor Raphaël Fournier-S'niehotta for their guidance throughout my PhD. The goal-oriented mindset of Philippe Rigaux and his ability to navigate through the intricated administration domains provided me with an extremely favorable environment where I could focus on my research and avoid organizational and economical issues that are a burden for many PhD students. The insightful discussions on theoretic approaches with Florent Jacquemard challenged my ability to see research problems from different perspectives and helped me to formalize detailed framework descriptions. Practical suggestions on tools and techniques and the constant support on everyday tasks from Raphael Fournier-S'niehotta were foundamental for the development of practical implementations. All three of them made me grow as a researcher and as a person. From the first moment to the very last, they never stopped supporting me, sometimes even at the cost of personal time and research. I had an amazing research visit experience at Nagoya University (Japan) for which I must thank Masahiko Sakai. I am very grateful to Mathieu Giraud and Carlos Agon for being members of my "comité de suivi" and to Martin Rohrmeier and Florence Levé for reviewing my thesis.

Abstract

A musical score is a complex semiotic object that excels at conveying musical information in a human readable format. Nowadays, a lot of music is available exclusively as recorded performances, so systems which can automatically transform those performances into musical scores would be extremely beneficial for performers and musicologists. This task, called automatic music transcription (AMT), comprises a large number of subtasks which generically transform the performance input into higher level representations, such as unquantized and quantized MIDI files. Despite the score being the ultimate goal, most of the approaches in the literature stop short of such an output, and only consider simpler representations. Indeed, the handling of musical scores notably constitutes a great challenge for automatic approaches. We believe that a clear model of the information contained in a musical score would help the development and the evaluation of such "full" AMT systems. In particular we advocate for a clear separation between the music which the score encodes (*i.e.,* the musical content) and the set of notation symbols which is employed to represent it (*i.e.,* the score engraving).

Our main contributions in this thesis are presented in four parts: the first details a dedicated model for AMT and related tasks. The second describes a system for monophonic MIDI performance parsing, which jointly performs note quantization and metrical alignment, to build the musical content level of the score. The third part focuses on the graphical symbols in the score, with a tool which computes and displays the differences at graphical level between two musical scores. Finally, we present a new dataset of annotated classical music performances aligned with musical scores; to speed up the tedious task of manual annotation, we employ a semi-automatic workflow which relies on the information in the musical content of the score.

Keywords : musical score, score model, automatic music transcription, music information retrieval.

Résumé

Les partitions musicales sont des artefacts d'une grande complexité sémiotique visant à transmettre toutes les informations relatives à la structure et au contenu d'une œuvre musicale de manière compacte, lisible et cohérente. De nos jours, la musique est la plupart du temps diffusée sous forme enregistrée. Un système capable de transformer automatiquement une interprétation musicale en partition serait donc extrêmement bénéfique pour les interprètes et les musicologues. Cette tâche, appelée "automatic music transcription" (AMT), comprend plusieurs sous-tâches qui impliquent notamment la transformation de représentations intermédiaires comme le MIDI, quantifié ou non. Bien que la partition soit le but final de ces transformations, la plupart des approches de la littérature s'arrêtent au stade des documents MIDI. En effet, la production au stade final d'une partition musicale constitue un défi important et difficilement automatisable. Nous défendons dans cette thèse l'idée qu'un modèle robuste et bien structuré des informations contenues dans une partition musicale aiderait au développement et à l'évaluation de tels systèmes AMT dits "complets". En particulier, nous préconisons une séparation claire entre le contenu musical encodé dans la partition et l'ensemble des signes et des règles de notation utilisés pour représenter graphiquement ce contenu.

Nos contributions sont présentées dans cette thèse en quatre parties : la première détaille un modèle de données dédié à l'AMT et aux tâches connexes. La seconde décrit un système d'analyse d'une interprétation musicale encodée en MIDI monophonique, qui effectue conjointement la quantification des notes et l'alignement métrique pour construire le contenu musical de la partition. La troisième partie se concentre sur les symboles graphiques de la partition, avec un outil qui calcule et affiche les différences au niveau graphique entre deux partitions musicales. Enfin, nous présentons un nouveau jeu de données contenant des interprétations de musique classique, annotées et alignées avec les partitions correspondantes ; nous définissons notamment une chaîne de traitements semi-automatique qui s'appuie sur les informations du contenu musical pour accélérer la tâche d'annotation manuelle.

Mots-clés : partition musicale, modèle de partition, transcription musicale automatique, recherche d'information musicale.

Contents

List of Tables

List of Figures

Introduction

A musical score is a complex semiotic object that convey the composer's intents to performers. Those intents can be resumed as: *who* is going to play, *when* he is going to play, *what* he is going to play and *how* he is going to play, and can be expressed in musical terms respectively as: instruments list, pitches, rhythmical information and playing techniques/dynamics/articulations. Those information are encoded in a graphical compact representation through the usage of several symbols, including staves, key signatures, clefs, note positions, alterations, note heads, rests, tempo marks, time signature, barlines, beamings, tuplets, dots, ties, ornaments, dynamics marks, slurs, expression marks, technique marks and repetitions marks (see Figure 1 for an example).

Despite this semiotic complexity, a musical score can be efficiently interpreted by a trained musician. Alternative representations have been proposed, but none has reached a similar level of popularity to be widely known and employed. Musical scores have been, and continue to be for a large part of the music production, the most complete and accurate way to encode music in a human readable format[1]. Not only they are employed to actually perform the music, but they are also extensively used by musicologist as the basis for music analysis.

A huge amount of music is available nowadays as audio recording of music performances. An audio file enables a very detailed representation of sounds, but do not offer the same functionalities of a musical score. This motivated the research on the *automatic music transcription* (AMT) task, *i.e.,* on systems that are able to produce musical scores from audio performances. The task is typically addressed with a workflow of transformations between the following formats: the audio waveform, a time-frequency representation, a piano-roll representation of pitches over time, and a musical score (for a recent survey on this topic see [1]).

A lot of attention has been given on the first tasks of this workflow, but few researchers have tar-

[1]We limit the scope of this thesis to standard western music

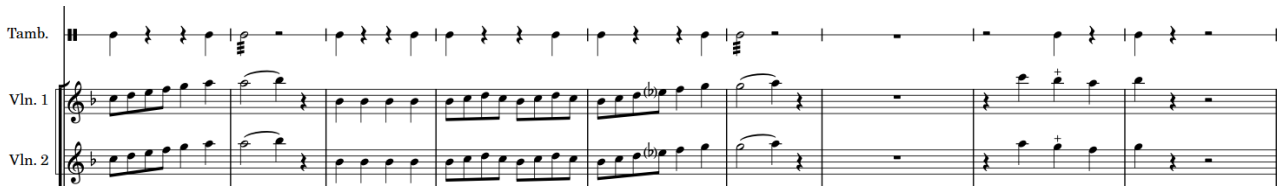


Figure 1: Extract of a musical score, from "Petite suite - Sarabande et Tambourin", Jean-Philippe Rameau. It encodes the parts of three instruments (two violins and a tambourine) in two five-line staves and one single-line staff for a duration of 9 measures, specified by barlines. Among the symbols of this score we find clefs and key signature alterations (first elements on the staves), beamings (*e.g.,* first four notes in the Vln.1 part), slurs (*e.g.,* first two notes in the Vln.1 second measure) rests (*e.g.,* last elements on the second measure), alterations (*e.g.,* in the Vln.1 5th measure) and ornaments (*e.g.,* second note in the Vln.1 8th measure).

geted the production of a fully structured musical score, especially for polyphonic and multi-instrument music. Several software packages, including Finale [2], Sybelius[3], Dorico[4] and MuseScore[5], provide the functionality of converting a piano-roll representation (in MIDI format) into music notation. However, there is a general agreement that the results are often not very satisfying. While the musical score format evolves over centuries to be efficiently read by musician, automatic machine-based approaches still struggle to handle its complexity, making it largely unresearched, compared to other formats such as audio an MIDI files.

We believe that, in part, the development of a fruitful full AMT system has been hampered by a failure to distinguish clearly among the several aspects of the music score itself. What is, indeed, a musical score from a machine perspective? Sure a score can be simply scanned and encoded as an image, but it contains symbols with musical meanings, and employing a data structure that explicitly model such meanings would be a better option. Following the proposal in [2] we can formalize this reasoning as follow:

A subject S uses a model M to represent the object O for purposes P

For example, considering a book page, one could see a collection of colored dots in a two-dimensional space. Or one could connect those dots to form lines and shapes. In this example O is the book page S a young child, M are lines and shapes and P is simply enjoying the view of shapes and colors.

[2] www.finalemusic.com
[3] www.avid.com/sibelius
[4] new.steinberg.net/dorico
[5] musescore.org/en

Those four variable are highly correlated in the statement: if one changes the others must adapt to maintain the statement valid. If we change S for a older person, M will include a meaning to each shape, or even something as complex as word and sentences. If we force the purpose P to be layout checking, the model will have to include information such as margins, columns and orientation and the subject S will be changed to a person with some competences in this field. This approach is also at the foundation of all scientific and engineers work [3]. Computer science is not an exception: all the approaches, from low level models encoded as bits in memory cells, to high level structures described by object oriented languages, relational databases or modeling languages (UML), always work by defining family of entities, and describing how they work, *i.e.,* their meaning. This is fundamental in order to efficiently target the purpose P.

In the case of musical scores, a printed paper, or its scanned image are efficient models of the score, as long as the purpose P is to be read by musicians. However, in computer music applications, the objective is to make the computer handle the score. And as P changes, the model M has also to be updated. In particular we believe that the AMT task and all related problems could be improved by a dedicated score model. In this thesis we propose such a model, and we detail some particular applications that have been enabled by it.

Overview

In Chapter 1 we cover the background information that are necessary for the understanding of this thesis. We give notions of musical theory, present the different storage formats for musical scores and formally define generic trees and context-free-grammars. The four successive chapters describe the main contributions of the thesis.

Chapter 2 contains the core of our work: we present a novel score model designed specifically for the AMT problem and the related subtasks. The model encodes with separate entities the *musical content* of the score, from the graphical symbols employed to represent it. We detail the data model and discuss about its specific usage and advantages for the problem of full AMT. Since we are dealing with such a complex structure, we also reflect about the quality issues that could arise and propose a structured and extendable approach to address them.

In Chapter 3 we present a way to build a part of this model from a monophonic MIDI performances,

according to the proposal in the previous chapter. This is equivalent to jointly perform the quantization and metrical alignment of a performance. The approach relies on a generative grammar and compute solutions optimal with respect to both the fitness of the quantization to the input performance, and a measure of complexity of the rhythmical content. We address the problem of learning such grammar from a dataset of musical scores.

Chapter 4 describes a tool that relies on our model data structures for computing the differences between two musical scores at graphical level. This is similar to the Unix diff model for text files. The algorithms for the computation rely on a combination of sequence- and tree-edit distances. We also propose a tool to visualize the differences between two scores side-by-side.

Finally in Chapter 5 we present a dataset of 222 digital musical scores aligned with 1068 performances (more than 92 hours) of Western classical piano music. The dataset itself is, to our knowledge, the largest that includes an alignment of music scores to MIDI and audio performance data. As such, it is a useful resource for targeting the complete audio-to-score AMT task, or other subproblems (*e.g.,* key signature estimation and beat or downbeat tracking from both MIDI and audio representations). The alignment between scores and performances annotation (downbeat, beat, time signature changes, and key signature changes) is obtained through a new annotation workflow that extracts information from the musical content of the score and project them onto the performances, with the goal of reducing the time for manual annotation.

Chapter 1

Preliminary Concepts

This chapter covers the background information that is necessary for the understanding of this thesis. It contains a heterogeneous disjointed sequence of notions and it is not meant to constitute a self-contained section, or to be a comprehensive explanation of the many topics addressed. The goal is to present notions, definitions and disambiguation of concepts that are used along different chapters and are not an original contribution, in order to ease the presentation in the reminder of this thesis.

The content of this chapter is organized in three main topics. Section 1.1 details some mathematical structures and algorithms, Section 1.2 contains basic concepts of music and music notation, and Section 1.3 gives an overview on the digital formats that are used for storage and distribution of musical scores.

1.1 Mathematical Background

This section contains definitions of mathematical objects such as trees, semirings and context-free grammars, and a description of edit-distance algorithms.

1.1.1 Trees

A tree is a mathematical object that we will extensively use during this thesis to model hierarchical structures, *e.g.,* recursive time divisions and metrical stresses. We give a formal definition in the following.

Let us assume given a finite set of symbols Σ, where each symbol $a \in \Sigma$ has a fixed arity. We denote as Σ_k the subset of Σ of symbols of arity k. We can recursively define a Σ-*labelled tree* t (called

tree for simplicity) as either:

- a single node labeled with a symbol a in Σ_0,
- one node labeled with a symbol a in Σ_k ($k > 0$), and an ordered sequence of k trees $t_1, \ldots, t_k$.

The node labeled by a is called *root* of the tree and denoted by $root(t)$. In the second case, t is denoted $a(t_1, ..., t_k)$, $a = root(t)$ is called the *parent* of $root(t_1), \ldots, root(t_k)$, and the latter are called *children* of $root(t)$. For all i, $1 < i \leq k$, $root(t_{i-1})$ is called the previous *sibling* of $root(t_i)$. A node $a \in \Sigma_0$ (*i.e.,* with no children) in a tree t is called a *leaf* of t.

The set of trees built over Σ is denoted $\mathcal{T}(\Sigma)$. The definition domain of a tree $t \in \mathcal{T}(\Sigma)$, denoted by $dom(t)$, is the set of nodes of t ordered according to a depth-first-search traversal. The set of nodes with arity 0 (leaves) of a tree t is denoted $dom_0(t)$. The depth of a node $a \in dom(t)$, denoted by $depth(a)$ is defined as the number of nodes (including a) between a and $root(t)$. The set of nodes with the same depth are said to be at the same *level* of the tree.

1.1.2 Semirings

A semiring is an algebraic structure that we will employ to define the domains of our functions. We use it instead of more concrete domains such as probabilities and costs, because it allows a more abstract definition of our problems. In practice we are interested in formulating our problems as both probabilities models and cost models, because each formulation brings its own advantages and interpretations. The usage of semirings allows us to give a single explanation that is valid for both domains.

Formally, a *semiring* $\mathcal{S} = \langle \mathbb{S}, \oplus, \mathbb{0}, \otimes, \mathbb{1} \rangle$ is a structure with a domain $\mathbb{S} = dom(\mathcal{S})$, two associative binary operators $\oplus$ and $\otimes$, and their respective neutral elements $\mathbb{0}$ and $\mathbb{1}$; $\oplus$ is commutative, $\otimes$ distributes over $\oplus$: $\forall x, y, z \in \mathbb{S}$, $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$. $\mathbb{0}$ is absorbing for $\otimes$: $\forall x \in \mathbb{S}$, $\mathbb{0} \otimes x = x \otimes \mathbb{0} = \mathbb{0}$. Components of a semiring $\mathcal{S}$ may be subscripted by $\mathcal{S}$ when needed. We simply write $x \in \mathcal{S}$ to mean $x \in \mathbb{S}$.

Intuitively, in the applications presented in this thesis, $\oplus$ selects an optimal value amongst two values and $\otimes$ combines two values into a single value.

A semiring $\mathcal{S}$ is *commutative* if $\otimes$ is commutative. It is *idempotent* if for all $x \in \mathcal{S}$, $x \oplus x = x$. It is

semiring	domain	$\oplus$	$\mathbb{0}$	$\otimes$	$\mathbb{1}$	natural ordering
Viterbi	$[0,1] \subset \mathbb{R}_+$	max	0	.	1	$x \leq_{\mathcal{S}} y$ iff $x \geq y$
Tropical	$\mathbb{R}_+ \cup \{+\infty\}$	min	$+\infty$	$+$	0	$x \leq_{\mathcal{S}} y$ iff $x \leq y$

Table 1.1: The two commutative, idempotent and total Semirings that we will consider in this thesis.

monotonic w.r.t. a partial ordering $\leq$ iff for all x, y, z, $x \leq y$ implies $x \oplus z \leq y \oplus z$, $x \otimes z \leq y \otimes z$ and $z \otimes x \leq z \otimes y$. Every idempotent semiring $\mathcal{S}$ induces a partial ordering $\leq_{\mathcal{S}}$ called the *natural ordering* of $\mathcal{S}$ and defined by: for all x and y, $x \leq_{\mathcal{S}} y$ iff $x \oplus y = x$. It holds then that $\mathcal{S}$ is monotonic w.r.t. $\leq_{\mathcal{S}}$. $\mathcal{S}$ is called *total* if it is idempotent and $\leq_{\mathcal{S}}$ is total *i.e.,* when for all x and y, either $x \oplus y = x$ or $x \oplus y = y$.

In practice, in this thesis we will use two kinds of total semirings presented in Figure 1.1: the Viterbi semiring defining *probability models*, and the tropical semiring, defining *cost models*.

1.1.3 Weighted Context Free Grammars and parse trees

We will set up the automatic construction of trees using the parse trees defined on a context free grammar. Semirings are employed to rank those trees.

Formally, let $\mathbb{F}$ be an alphabet of symbols. A *Weighted Context-Free Grammar* (WCFG) over a semiring $\mathcal{S}$ and an alphabet $\mathbb{F}$ is a tuple $\mathcal{G} = \langle Q, \mathit{init}, R, \mathit{weight} \rangle$ where:

- Q is a finite set of non-terminal symbols (nt),
- $\mathit{init} \in Q$ is an initial non-terminal,
- R is a set of *production rules* in one of the following forms:

  ($k-$div) $q \to \langle q_1, \ldots, q_k \rangle$ with $q, q_1, \ldots, q_k \in Q$, and rank $k > 1$, or

  (term) $q \to \eta$ with $q \in Q$ and $\eta \in \mathbb{F}$ (η is called *terminal* symbol).

- *weight* is a mapping that assigns to each production rule in R a weight value in $\mathcal{S}$,

The components of a WCFG $\mathcal{G}$ may be subscripted by $\mathcal{G}$ when needed. We use the notations $q \xrightarrow{w} \langle q_1, \ldots, q_n \rangle$ and $q \xrightarrow{w} \bar{e}$ respectively for ($k-$div) and (term) productions rules of weight $w \in \mathcal{S}$.

For a WCFG $\mathcal{G}$ over a semiring $\mathcal{S}$, $\mathcal{T}(\mathcal{G})$ denotes the set of *parse trees*, labelled by production rules of $\mathcal{G}$. More precisely, $\mathcal{T}(\mathcal{G})$ is defined as the smallest set such that

- for all production rule ρ of type (term) in $\mathcal{G}$, $\rho \in \mathcal{T}(\mathcal{G})$, with root ρ,
- for all production rules $\rho = q \xrightarrow{w} \langle q_1, \ldots, q_k \rangle$ of type (div) in $\mathcal{G}$, and all $t_1, \ldots, t_k \in \mathcal{T}(\mathcal{G})$ whose respective roots have heads $q_1, \ldots, q_k$, $\rho(t_1, \ldots, t_k) \in \mathcal{T}(\mathcal{G})$ with root ρ.

The weight, in $\mathcal{S}$, of a parse tree $t \in \mathcal{T}(\mathcal{G})$ is the product with $\otimes$ of all the weights of production rules labeling the nodes of t. Formally, given a tree t with $root(t) = \rho$:

$$weight(t) := \begin{cases} weight(\rho) & \text{if } \rho \text{ is of type } (\mathsf{term}) \\ weight(\rho) \otimes \bigotimes_{i=1}^{k} weight(t_i) & \text{if } \rho = q \xrightarrow{w} \langle q_1, \ldots, q_k \rangle \text{ is of type } (k\text{-}\mathsf{div}) \end{cases} \tag{1.1}$$

1.1.4 Edit Distance

An *edit distance* is the distance between two sequential objects as measured by the minimum cost sequence of *edit operations* needed to change one object into the other [4]. We will employ it as the basis for our algorithms to compute the distance between two musical scores.

This complicated problem is approached by using *dynamic programming*: both a mathematical optimization and a computer programming method that aims at solving complex problems by breaking them down into simpler sub-problems in a recursive manner. Many variations of it exist, depending on the specific application, but they all defines a set of edit operations $\alpha \to \alpha'$ associated to a value in a semiring $\mathcal{S}$. Given two sequential objects s and s', the edit distance $D(s, s')$ between them is the semiring product ($\otimes$) of the values of an edition sequence transforming s into s'.

The most important of those measures of similarity is the *Levenshtein distance* that targets sequences of characters (*i.e.,* strings) [5], and considers the following edit operations (n represents a character, and ε denotes the absence of a character):

$$\begin{array}{lcll} \varepsilon & \to & n & \text{insertion} \\ n & \to & \varepsilon & \text{deletion} \\ n & \to & n' & \text{substitution} \end{array} \tag{1.2}$$

It can be computed using the following dynamic programming equations, where $|s|$ denotes the

length of s, and $n.s$ represents a string made of the character n followed by the subsequence s.

$$\begin{array}{lcll} D(\varepsilon,\varepsilon) & = & 0 & \\ D(\varepsilon,n'.s') & = & D(\varepsilon,s')\otimes\delta(\varepsilon\to n') & \textsf{(ins)} \\ D(n.s,\varepsilon) & = & D(s,\varepsilon)\otimes\delta(n\to\varepsilon) & \textsf{(del)} \\ D(n.s,n'.s') & = & \bigoplus \left\{ \begin{array}{lcll} D(s,s') & \otimes & \delta(n\to n') & \textsf{(subst)} \\ D(n.s,s') & \otimes & \delta(\varepsilon\to n') & \textsf{(ins)} \\ D(s,n'.s') & \otimes & \delta(n\to\varepsilon) & \textsf{(del)} \end{array} \right. & \end{array} \tag{1.3}$$

1.2 Musical Background

In this Section we review some basic concepts of music theory related to frequencies (*i.e.,* pitch) and temporal positions (*i.e.,* rhythm) of musical events. It is out of our scope to consider non-Western music traditions or to elaborate on the many exceptions that can be found within the western music production. We acknowledge their existence, but we will generically refer to "music" and "musical scores" as temporal and frequency organized structures, as they are defined in the common western tradition.

1.2.1 Pitches

Pitches can be defined, at a low, physical level, by their frequency, or with names that convey higher level musical information.

Frequency-based pitch classification

A *Pitch* is a perceptual property of sounds that allows their ordering on a frequency-related scale [6]. Even an untrained listener can easily tell if two sounds have the same pitch or some specific relation. An emblematic example is the *octave*: the distance between a musical pitch and another with half or double its frequency. The octave relationship is important for our perception of music, because the human ear tends to hear two pitches at distance of one or more octaves, as being essentially the same pitch. This phenomenon is called *octave equivalence*.

Most of the western music has been based for some centuries on the equal temperament tuning system, that identify 12 exponentially spaced pitches in the frequency range of an octave, according

	Octave 0	Octave 1	Octave 2	Octave 3	Octave 4	Octave 5	Octave 6	Octave 7	Octave 8
C	16.35	32.70	65.41	130.81	261.63	523.25	1046.50	2093.00	4186.01
C#	17.32	34.65	69.30	138.59	277.18	554.37	1108.73	2217.46	4434.92
D	18.35	36.71	73.42	146.83	293.66	587.33	1174.66	2349.32	4698.64
D#	19.45	38.89	77.78	155.56	311.13	622.25	1244.51	2489.02	4978.03
E	20.60	41.20	82.41	164.81	329.63	659.26	1318.51	2637.02	5274.04
F	21.83	43.65	87.31	174.61	349.23	698.46	1396.91	2793.83	5587.65
F#	23.12	46.25	92.50	185.00	369.99	739.99	1479.98	2959.96	5919.91
G	24.50	49.00	98.00	196.00	392.00	783.99	1567.98	3135.96	6271.93
G#	25.96	51.91	103.83	207.65	415.30	830.61	1661.22	3322.44	6644.88
A	27.50	55.00	110.00	220.00	440.00	880.00	1760.00	3520.00	7040.00
A#	29.14	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31	7458.62
B	30.87	61.74	123.47	246.94	493.88	987.77	1975.53	3951.07	7902.13

Table 1.2: The scientific pitch notation (SPN) for pitches, assigns a frequency to each pitch in a specific octave. Pitches are notated with a couple (pitch-class,octave number), *e.g.,* C4 or G#2.

to the formula:

$$f_p = 2^{1/12} f_{p-1} \tag{1.4}$$

The distance between two pitches is called *interval*, and is defined by a frequency ratio. The smallest interval, called *semitone*, is defined by the ratio

$$\frac{f_p}{f_{p-1}} = 2^{\frac{1}{12}} \tag{1.5}$$

The ensemble of all pitches that are n (with n an integer) octaves apart constitute a *pitch-class*, commonly indicated by a number $\in [0, 12[$ or by a letter. Specifically only 7 pitch-classes (0,2,4,5,7,9,11) are named by a letter: {'C', 'D', 'E', 'F', 'G', 'A', 'B'}. The names of the remaining pitch-classes is obtained either adding the symbol ♯ to the letter of the leftmost pitch-class or adding the symbol ♭ to the letter of the rightmost pitch-class. For example both F♯ and G♭ correspond to the pitch-class 6. Other symbols are used in a similar way, such as 𝄪 and 𝄫 that alter the pitch class defined by the letter respectively adding and subtracting 2 semitones.

In this framework, it's sufficient to specify the frequency of one sound to be able to calculate the frequency of all pitches. The most common choice is to set the pitch-class 'A' at 440Hz (and all multiple and divisors). Some widely used systems for naming pitches are the *Scientific pitch notation* (SPN), that use the couple (pitch-class, octave-number) (see Table 1.2), and the *MIDI number* system, that identify each pitch with an integer $\in [0, 127]$, as shown in Figure 1.1.

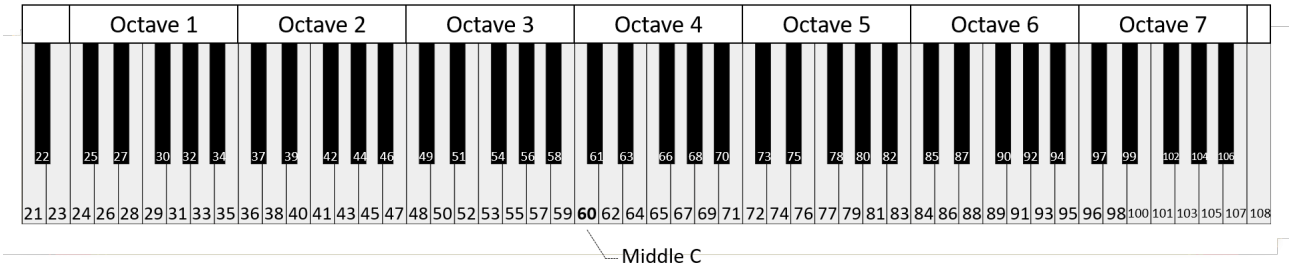


Figure 1.1: MIDI note numbers mapped on a 88-keys piano.

Pitch spelling and representation in a musical score

As shown in the previous section, pitches in the equal tempered tuning systems can have multiple names. For example, F♯3, G♭3, E𝄪3, all identify a sound at 185Hz. Pitches of the same frequency but denoted by different pitch names are said to be enharmonically equivalent [7]. Even though the choice of a specific name over the others (*i.e.,* the pitch spelling) is an information that is not available to the listener, in some kind of music it assigns a higher level musical meaning to the note. For example, it has been claimed to be more efficient for some musical tasks such as harmonic analysis, melodic pattern matching and motivic analysis [8].

This elaborate representation of pitches is used in the musical score through a set of graphical elements. A *staff* (plural *staves*) is a set of horizontal lines. All symbols in a score are related to a specific area of the staff. The most used is the five-line staff, also called pentagram. The pitches are displayed in a pentagram through three elements: the clef, the vertical position (its position over a line or between two lines) and a (optional) accidental symbol. The clef specifies what pitch is assigned to each space and line of the pentagram, while the accidental augments (in case of ♯ or 𝄪) or diminishes (in case of ♭ or 𝄫) the pitch by a semitone or a tone.

In case of repeated accidentals (*e.g.,* all the F of a piece are F♯), it is convenient to set a *key signature*: we write some accidentals at the beginning of the line, right after the clef, meaning that all the corresponding pitches are altered, unless differently specified by the symbol ♮. The key signature has to follow specific rules regarding the order of alterations, as it is tightly related to other high level musical concepts.

	$\frac{2}{X}$	$\frac{3}{X}$	$\frac{4}{X}$	$\frac{6}{X}$	$\frac{9}{X}$	$\frac{12}{X}$
1st Level	2	3	4	2	3	4
2nd Level	2	2	2	3	3	3

Table 1.3: Divisions at a different metrical levels encoded by the time signature.

1.2.2 Rhythm

Rhythm is the cyclic timing of acoustic events with some repetitive structure [9]. It is perceived as a repeating series of identical yet distinct periodic short-duration stimuli [10] to which we have the ability to synchronize. Hereafter we provide a review of several musical concepts that concern rhythm, such as beat, tempo, meter and time signature.

Beats and Tempo

Music is temporally organized with respect to a quasi-periodic pulse, called *beat*. This is a fundamental aspect of music perception; generally even an untrained listener can locate the beat with relative ease. The frequency of the beats is called *tempo* and is expressed in Beats Per Minute (BPM). Its inverse is the time-span between two adjacent beats, known as *Inter-Beat Interval* (IBI). For example, at 120 BPM there are two beats every second and the IBI corresponds to 0.5 seconds.

Metrical Hierarchy and Time Signatures

Rhythm has a hierarchical organization: beats are divided into equal units (*i.e.,* sub-beats) or compounded into longer units (*i.e.,* the *measures*). From the point of view of measures, this corresponds to two levels of division, that are encoded in the *time signature* as depicted in Table 1.3. Further recursive divisions often occur, generating a hierarchy of pulses called *metrical structure*. Different intensities are assigned to each pulse; in general the first unit generated by a division is considered "stronger" with respect to the others [11]. Groups of measures can also be identified, but they tend to be less regular [12] and we will not consider them in this thesis.

1.2.3 Time units and quantization

Depending on the representation of music there are two ways of representing a specific time position τ [13]. The first expresses it in *real-time unit* (*rtu*) such as seconds and it is employed to date events in

a music performance (*e.g.,* an audio file). The second uses *musical-time unit* (*mtu*) and it's common in higher level music representations, such as the musical score. Time positions in *mtu* are defined in relation to a rhythmical entity, for example the beat or the measure. For both units, the domain is $\mathbb{Q}_+$. Time intervals (and durations of musical events) are defined analogously.

The transformation of *rtu* into *mtu* is defined by a monotonically increasing continuous function $\theta : \mathbb{Q}_+ \to \mathbb{Q}_+$. This function is invertible and we denote its inversion as θ^{-1}. A version of θ^{-1} is defined in BPM by the tempo, as presented in Section 1.2.2. However a much more complex definition could be given, that consider also time modifications such as "accelerando" and "ritardando" and small time deviations.

The positions of events in a score (in *mtu*), is fixed to a small subset of $\mathbb{Q}_+$ corresponding to rational numbers where both numerator and denominator are "small". We say that the musical events are *quantized* to those time positions.

1.3 Digitized Musical Notation storage formats

Numerous formats for storage and interchange of musical symbolic data have been deployed and we present the main ones in this section.

1.3.1 MIDI

The oldest widely used format is the Standard MIDI File (SMF), also called MIDI file or MIDI: a 8-bit binary format that stores a sequential stream of MIDI events. Many kinds of MIDI events exists [14], but we are interested mainly in: *Note-On* and *Note-Off* events, encoding respectively the starting of a sound (pressed note on a keyboard) and the ending of a sound (released note on keyboard). Those MIDI events also contain information about the pitch (encoded as a MIDI note number) and the *velocity*, *i.e.,* the intensity at which the note is played.

The MIDI events are grouped in *tracks*, that represent the notes that an instrument have to play in a piece. Specifically, a track is a stream of MIDI events, each preceded by delta-time values, *i.e.,* the amount of time before the following event. For example if the first event in a track occurs at the very beginning of a track, or if two events occur simultaneously, a delta-time of 0 is used. Delta-times can be expressed in *mtu* as fractions of a beats or in *rtu* as seconds.

Generic information about the entire MIDI file (*e.g.,* the considered time units and the file organization) are given in the header. Two possible organizations of the file exist (a third is defined but not really used in practical applications):

- `type 0`: the file contains a single track, *e.g.,* a piece for violin solo,
- `type 1`: the file contains one or more simultaneous tracks, *e.g.,* all the parts of an orchestra, that are played simultaneously.

MIDI files in *rtu* excels at representing performances in an high level format because they can explicitly encode the pitch of each note. When *mtu* are used, they can represent many information of a musical score. A set of metadata allows to encode information about the time signature, tempo, key signature and lyrics. The tempo and time signature should be specified (the bar position is not explicitly encoded, but inferred from this information). If they are not, the time signature is assumed to be 4/4, and the tempo 120 beats per minute. In format 0, these meta-events should occur at least at the beginning of the single track and in format 1, they should be contained at least in the first track. The key signature is encoded as the number of flats or sharps as respectively negative and positive numbers; for example -1 encodes a key with 1 flat (F major or D minor) and 3 encodes a key with 3 sharps (E major or C# minor). A symbol to differentiate between major and minor keys is also present. If no key signature is specified, a C major is assumed. The time-lag of *mtu* MIDI files are often quantized to certain durations and for this reason they are also called *quantized MIDI files*.

Quantized MIDI files are often employed in MIR as a "easy-to-handle" substitute for musical scores. However, compared to the information conveyed by a score, a MIDI file has severe limitations. It has no discrete note element (they have to be inferred by note-on and note-off events), ties, dots and rests are not represented, there is no distinction between two different pitch spellings of the same note (*e.g.,* D-sharp from an E-flat), there is no representation of beams, tuplets or clefs, and there is not separation of different voices, as all the voices' events are flattened into the same piano roll.

1.3.2 Text-based formats

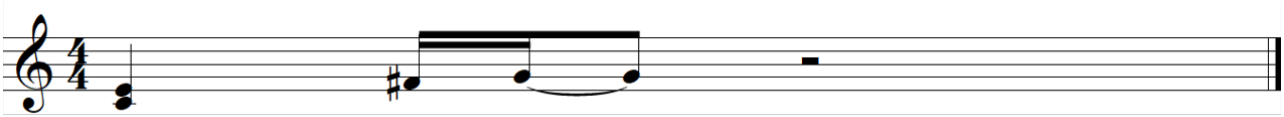
Another category of formats is text-based. It contains formats such as ABC[15], Musedata [16], Humdrum **Kern [17, 18], LilyPond [19], Guido [20] and MusicTeX [21]. The text-based formats brings the advantage of being easy to read and create manually. Different approaches are explored

in each format to cope with the difficulties of keeping a readable notation while being able to encode complex elements of a musical score[1]. As the score size and complexity increase, however, those formats shows limits as parsing a big text file is less efficient compared to other file formats (*e.g.,* XML).

1.3.3 XML-based formats

The more recent approaches, built upon the experience with the aforementioned formats, are XML-based: MusicXML [23], MEI [24] (and MNX[2] in a close future). Their common characteristic is to trade the compactness of the representation for a complete description of the score sheet, including graphic specifications that dictate how the notation content has to be visually rendered. While the initial motivation was (notably for MusicXML) to address interoperability concerns, a side benefit of these encodings is an easy access to the components of a score, at a very detailed level. Differences exist in the syntax (Figure 1.2): generically MusicXML tends to be more detailed but also more verbose. One particularity of MEI is the presence of an unique identifier associated to each element of the score, allowing an easier implementation of visualization and score interaction applications. From a compatibility point of view, MusicXML is more supported from commercial music notation software, as all the modern ones support both input and output to and from MusicXML, while only Sibelius handles MEI files. MEI (the analog of TEI for music score) was designed for the purpose of corpora preservation and analysis and is more complete regarding notation systems. Outside of standard Common Western Notation, it supports mensural (Renaissance-era) and neume (Medieval) notations. In addition, MEI is philology-oriented as it can encode notational variants and relationships between notational features and digital page images or audio recordings.

[1] An exhaustive presentation of those formats can be found in [22].
[2] https://w3c.github.io/mnx/overview/



MEI

```
...
    <measure xml:id="measure-0000000232735099" right="end" n="1">
        <staff xml:id="staff-0000001963390883" n="1">
        <layer xml:id="layer-0000000738522925" n="1">
            <chord xml:id="chord-0000001488662809" dur.ppq="4" dur="4" stem.dir="up">
                <note xml:id="note-0000000224872045" oct="4" pname="c" />
                <note xml:id="note-0000001902316584" oct="4" pname="e" />
            </chord>
            <beam xml:id="beam-0000001404430694">
                <note xml:id="note-0000002146957621" dur.ppq="1" dur="16" oct="4" pname="f" stem.dir="up">
                    <accid xml:id="accid-0000000545435759" accid="s" accid.ges="s" />
                </note>
                <note xml:id="note-0000000588407389" breaksec="1" dur.ppq="1" dur="16" oct="4" pname="g" stem.dir="up" />
                <note xml:id="note-0000001646010113" dur.ppq="2" dur="8" oct="4" pname="g" stem.dir="up" />
            </beam>
            <rest xml:id="rest-0000000011007280" dur.ppq="8" dur="2" />
        </layer>
        </staff>
        <tie xml:id="tie-0000000227382387" startid="#note-0000000588407389" endid="#note-0000001646010113" />
    </measure>
...
```

MusicXML

```
...
    <measure number="1" width="983">
      <note default-x="87">
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <voice>1</voice>
        <type>quarter</type>
        <stem default-y="-5">up</stem>
      </note>
      <note default-x="87">
        <chord/>
        <pitch>
          <step>E</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <voice>1</voice>
        <type>quarter</type>
        <stem>up</stem>
      </note>
      <note default-x="308">
        <pitch>
          <step>F</step>
          <alter>1</alter>
          <octave>4</octave>
        </pitch>
          <duration>1</duration>
        <voice>1</voice>
        <type>16th</type>
        <accidental>sharp</accidental>
        <stem default-y="5">up</stem>
        <beam number="1">begin</beam>
        <beam number="2">begin</beam>
      </note>
      <note default-x="393">
        <pitch>
          <step>G</step>
          <octave>4</octave>
        </pitch>
        <duration>1</duration>
        <tie type="start"/>
        <voice>1</voice>
        <type>16th</type>
        <stem default-y="5">up</stem>
        <beam number="1">continue</beam>
        <beam number="2">end</beam>
        <notations>
          <tied type="start"/>
        </notations>
      </note>
      <note default-x="477">
        <pitch>
          <step>G</step>
            <octave>4</octave>
        </pitch>
        <duration>2</duration>
        <tie type="stop"/>
        <voice>1</voice>
        <type>eighth</type>
        <stem default-y="5">up</stem>
        <beam number="1">end</beam>
        <notations>
          <tied type="stop"/>
        </notations>
      </note>
      <note default-x="613">
        <rest/>
        <duration>8</duration>
        <voice>1</voice>
        <type>half</type>
      </note>
      <barline location="right">
        <bar-style>light-heavy</bar-style>
      </barline>
    </measure>
...
```

Figure 1.2: A small example of MusicXML and MEI encoding of one measure.

Chapter 2

A multi-level score model for AMT

While musical scores have historically been written on paper or similar physical supports, with the advent of the digital era, ways of digitally representing musical scores have been proposed. A first approach is to obtain a scanned version of the paper score in a standard graphical format, in most cases a raster image. Many digital score libraries offer this kind of format (*e.g.,* the International Music Score Library Project - IMSLP[1]). The nature of these documents makes them easy to produce from a physical document, but does not bring any benefit beyond their improved accessibility. They remain mainly restricted to traditional usages, as a support for either performance, or visual, human-based analysis, and any further interaction would require complex systems able to work directly on images. Many more tasks could be performed starting from the high level musical information encoded in a digital score, such as automatic musicological analysis, search, score generation, (re)edition and playback. However it is clear that, to be able to efficiently perform such actions, we need a different representation of the score, *i.e.,* a dedicated model for those tasks.

In this thesis we focus on the problem of score generation from a musical performance, *i.e.,* the last step in a automatic music transcription pipeline. This is a difficult task, because of the complex information embedded in a musical score. Typical AMT systems usually stop short of such an output, producing a MIDI-like representation. In recent years, only a handful of systems have been designed to output a more complete musical score [25, 26, 27]. In the case of [27] producing it is more a proof of concept than an actual method, as the score generation step in the workflow is left to the import function of a musical notation software (MuseScore[2]) over which they have no control. [25, 26] present

[1] https://imslp.org/
[2] https://musescore.org/

machine learning techniques that output directly a musical score, respectively in Lilypond and **kern. While both of these works show promises on this difficult task, both of them consider only a simplified scenario, where time signatures and key signatures are not considered, and the number of voices is fixed. As they state, they are more focused on the transcription of a correct musical content than on the details of music representation.

A machine learning approach to AMT that generates directly a complete musical score (*e.g.,* a MusicXML file or a score image) would require a massive number of parameters and an enormous training set to work properly. Moreover, all the supervised machine learning techniques rely on the definition of a cost function measuring the difference between a ground truth and the result produced; the formulation of this cost function poses huge problems: should we consider separately *musical* differences from *graphical* ones? How to distinguish between what is graphical and what is not?

It is clear that directly targeting a complete musical score poses huge challenges, and we propose an alternative more structured approach that will reduce the complexity of the problem. Our approach sees the musical score as containing two disjoint sets of information: the *musical content*, *i.e.,* everything that concerns the production of sounds, independently from any visualization concern; and the *score engraving*, *i.e.,* all graphical artifacts used to present the musical content in a clear and human readable way. By definition the latter does not depend on the performance, but rather on a mix of principles and best practices that have been developed over time [28]. For this reason we propose to partition the "end-to-end AMT" problem in three independent processes.

1. The transcription of the musical content from a performance. This step contains most of the "partial" AMT techniques in the literature, *e.g.,* pitch detection, note quantization, metrical alignment, voice separation, etc.

2. The generation of a score engraving from the musical content. The graphical symbols are still represented in musical terms, for example we consider note-heads and beams, not geometrical shapes or pixels.

3. The generation of an image from the score engraving. A low level representation of the image is produced. We do not model such representation since formats such as vector or raster image already fit this purpose.

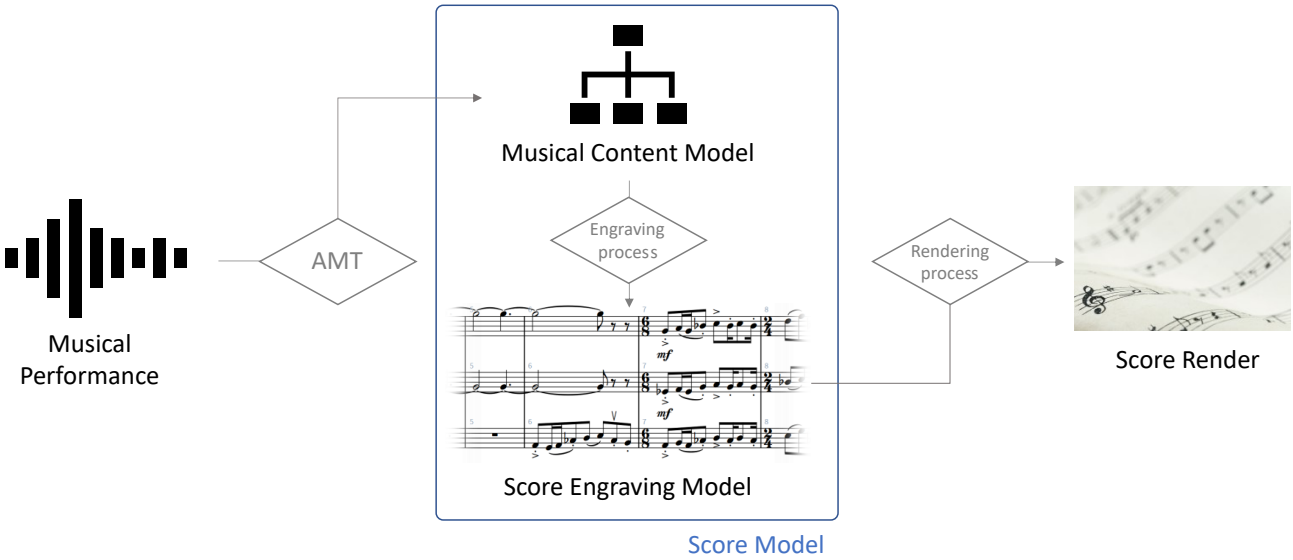


Figure 2.1: The full AMT workflow from a performance to the musical score.

In this thesis we refer to the three steps respectively as *AMT*, *engraving process* and *rendering process* (see Figure 2.1). We will refer to the ensemble of the three steps as *full AMT*.

In this chapter we first describe the current state of the art in score modeling, then we present our original contributions to this topic. In Section 2.2 we detail our original score model designed for the task of automatic music transcription, that decouples the musical content of the score from any visualization and rendering information; a part of this score model have already been published in [Foscarin et al., 2018b]. In Section 2.3.1 we conclude the chapter by discussing the problem of score quality: as any complex data structure, musical scores are prone to contain quality issues. We propose a way to deal with them, with a structured and extensible approach that extends the content of our paper [Foscarin et al., 2018a].

2.1 State of the art

To the best of our knowledge, no musical score model has been proposed with the explicit goal of score structuring from a performance. However many proposal in the literature contain interesting approaches and solutions that we can reuse and adapt to our goal. This state of the art is structured in two parts. The first one presents various score models, organizing them according to their objective, and the second details some methods that have been used to represent rhythm in music. While the

latter is not necessarily related to musical notation, it is worth exploring, since rhythm is one of the most complex components to represent in musical scores.

2.1.1 Score Models

With the goal of modeling the composition process, Buxton [29] proposes a hierarchy of *musical events*, showing how a tree structure can be employed to organize the different parts of the score. At the lowest level, events are notes, rests and chords whereas the top level coincides with the entire musical score. The object oriented approach simplifies the framework, as it allows the definition of functions that work on any level, such as for example a *sonification* function, able to play a single note, a small section or an entire score. An interesting approach is followed for repetitions, which are treated as two instances of the same event. "Similar" sections (*e.g.,* the same melody in two different keys) are also encoded as two instances of the same event where different *transformation functions* have been applied, *e.g.,* the transposition of the melody. This model deals also with the *sounds* that can be produced from the score (*i.e.,* the performance), by defining a workflow for its production: the user defines a palette of instruments that can be coupled with the score (*i.e.,* the *orchestration* process) to produce a performance. The Smalltalk Music Object Kernel (SMOKE)[30] is a similar model that relies on *events*, *i.e.,* basic objects with a certain duration that can be extended with attributes such as timbre, pitch, loudness, etc. A score consists in one or more event lists (but events in a list can be event lists themselves, thus enabling a hierarchical structure). Again the "sound" is considered, but modeled in a different entity from the score. The encoding of the score content and the performance as two clearly separated entities, where the latter is generated from the former through a workflow, has inspired our sequential approach for the score production.

A field that deals with musical scores from another perspective is Optical Music Recognition (OMR). Researchers in this field have produced score models with the purpose of being able to detect and correct problems caused by a faulty image recognition. Couasnon [31, 32] defines a grammar for full polyphonic scores, that encodes elements such as different voices on a single staff, chords on a voice, accents, tuplets, pauses, dynamic markings, phrasing slurs, rhythmic slurs and appoggiaturas. Using the grammar, full scores and isolated components can be recognized and interpreted. The model enables in particular to check consistency properties such as the number of beats in a bar (per voice) according to the time signature and the vertical alignment of notes in a system. An

automatic correction of those errors is possible in certain cases. With a similar model formulation, Szwoch proposes the Guido system [33] that automatically recognizes the symbols of a scanned musical score. This approaches uses a context-free grammar, that associates to each symbol a set of syntactic, geometrical and logical rules about the symbol itself (*e.g.,* the relation between the stem and the head of note) and about the relation with other symbols (*e.g.,* the notes composing a chord). A set of verification rules exploits local and global contextual dependencies to find and partially correct errors related to the incorrect duration of notes and rests in a measure. We employ a similar system to detect quality issues, but we apply it to a broader set of problems.

Rossant and Bloch [34] consider a rule-based system combined with fuzzy modeling of monophonic scores. The goal is to improve decision making during the OMR process by indicating possible errors (that have to be then corrected manually by a user). In this work we find an explicit separation between the musical content of a score (that they call *syntactical* information) and its graphical representation. They divide the rules into two classes: the first one containing syntactic rules, involving tonality and metre information (number of beats corresponding to the global metric indication, notes grouped into beats to improve rhythm structure understanding, accidental of the key signature are applied to each note with the same name, a duration dot modifies the length by a factor 1.5,...). The second class contains graphical rules, describing the relative positions of symbols (an accidental has to be before the note and at the same height, a duration dot has to be after the note head, a staccato is below or above the note head). Again from the OMR community, but with the goal of using scores as (sequential) input for Neural Network models, the authors of [35] introduce two representations of a monophonic score: the first contains symbols with musical meaning, e.g., a D Major key signature; the second consists of musical symbols without musical meaning (*e.g.,* lines, circles, etc.) tagged according to a catalog of pictograms and located in a position in the staff (*e.g.,* third line, first space). The division in musical content and engraving proposed in our model is inspired by these works, but we consider full polyphonic scores with multiple voices.

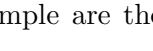
With the goal of "analyzing, searching, and transforming music in symbolic forms", we find the model implemented in the python library music21 [36]. At the core of music21 there is a grouping of musical information into Streams: nested containers for representing and manipulating collections of complex musical structures. This is a very complete practical tool for many tasks that involve musical score and, while it does not explicitly separate the musical content information from its graphical

representation, it gives an easy access to all this information. It also contains a function `makeBeams()` that performs part of the operations necessary to create the engraving information from a musical content.

2.1.2 Models of Rhythm

Rhythm, *i.e.,* the description of the temporal dimensions of musical events, is one of the fundamental aspects of music. While extensive debates exist on its precise definition and psychological implications [37, 38], we consider here the elements of rhythm that are relevant for musical notation: duration and position of musical events.

One way to represent this information is to use a sequential representation (*e.g.,* [39, 40, 41, 42, 43, 35]); however, inspired by studies in computational linguistic, many researchers claim that musical processes cannot be adequately described by a sequential model. Simon [44] pioneered drawing parallelisms with psychological works on non-musical sequences of symbols and the perception of musical phrases. According to his work, the listener extrapolates a hierarchical structure ("deep structure") from the music that he perceives ("surface structure"). A similar conclusion was reached by Lindblom [45] that proposes a rhythmical model of the measures in a melody formalized in a series of grammars rules. Later this approach was applied at a finer granularity, to notes and rests of a melody [46, 47, 48, 49]. These approaches take a sequence of notes and rests in a given time interval and recursively divide the time interval into k regular sub-intervals, where $k = 2$ or 3 (other prime divisions such as 5 or 7 are rarely considered), in order to align each note (or rest) at the beginning of a sub-interval. This generates a tree structure where each internal node encodes a certain duration and the leaves contain notes, rests and other special symbols for dots and ties (Figure 2.2). Since then, similar hierarchical models have been applied for tasks such as melodic similarity computation [50, 51], and rhythm quantization [52, 53].

While those models have been widely used to model rhythm, their usage to represent rhythmic notation (*e.g.,* [54]) shows some limitations. A simple example are the two measures  and  as they are are indistinguishable in their hierarchical representation. Agon et al. [55] present a dedicated tree model for rhythmic notation, implemented in OpenMusic [56], which allows for non-equal division of a time interval in sub-intervals. This representation can encode complex beamings and tuplet groupings while still remaining compact, but it requires complex algorithms to retrieve

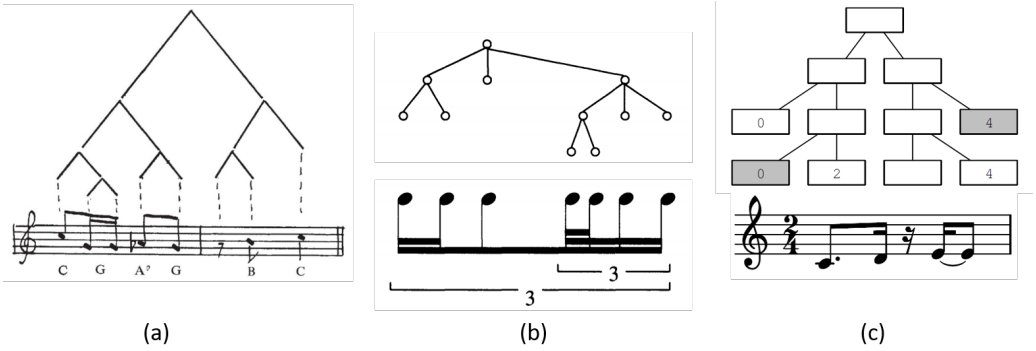


Figure 2.2: Tree representations of rhythm proposed (from left to right) in [46, 52, 50].

the actual notation symbols (note heads, number of beams, tuplet groupings, etc.). Moreover the authors do not detail the representation of chords, grace notes and the distinction between dots and tied notes.

Focusing on polyphonic music, Kameoka et al. [57] model with a tree structure both the temporal position of notes and their separation in multiple voices with the goal of obtaining a model for automatic music transcription. Practically it uses two kinds of internal nodes: one divides a temporal interval into k sub-intervals (as presented above) and the other divides a voice into k voices. The problem of this approach is that the space of possible trees is extremely large. A solution is proposed in [58] by encoding in the tree leaves frequently occurring rhythm patterns instead of single musical artifacts, following the idea that humans recognize rhythm not necessarily by perceiving each inter-onset interval as a particular note value, but rather by perceiving a set of inter-onset intervals as a particular rhythm pattern. While this technique shows promising results for a small corpus of selected scores, we believe it is not flexible enough for a general representation of rhythm.

A different approach is explored in music21 [59]: instead of building a structure on top of musical artifacts (note, rests, chords), the tool builds a hierarchy of time intervals in a measure. This hierarchy is shared among different measures, and it is possible to couple it with a list of musical artifacts (associated to a specific time position) in order to retrieve the position in the hierarchy of a specific artifact. For each measure three different hierarchies are defined: beaming information, beat positions, and accents. This approach is very interesting, because it brings attention on different aspects of the rhythmic notation that are usually overlooked or simplified. However, as intended by the authors, this is a model of rhythmic conventions tied to a specific time signature, rather than a model of the

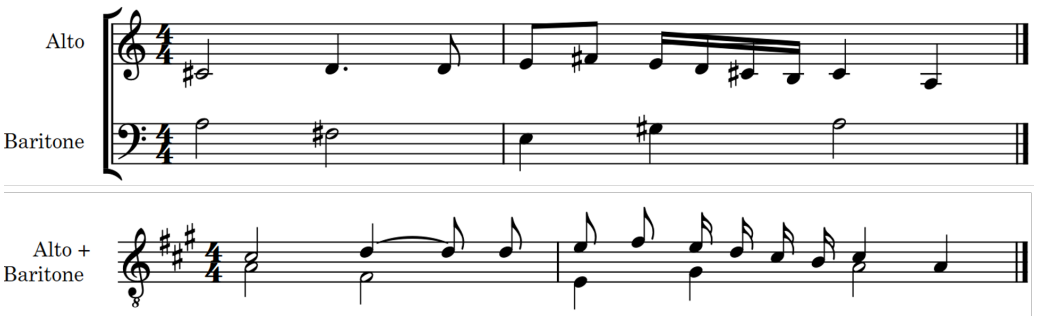


Figure 2.3: Two different engravings of the same musical content.

rhythm itself. This means that this model will not give us any information on some musical events, if they are not aligned to the temporal intervals defined by the hierarchy.

2.2 Model architecture

The musical score is a complex object that conveys the composer's intents in a compact graphical representation. In order to make this representation optimized to be read by musicians, a complex graphical representation (the *score engraving*) is employed, that involves elements such as staves, clefs, beamings, tuplets, etc. As an example, we can see in Figure 2.3 a short composition represented in two different ways. In the upper one each voice is assigned to a different staff, with a different clef, while in the bottom engraving, the two voices are displayed in the same staff. There are many other differences between the two scores: different choices made about the key signature, the usage of dots and ties and the beaming of notes; but none of those choice really influence the music that is supposed to be produced from that score. This separation between the content and the presentation is already widely used for descriptive markup languages (e.g. Latex, HTML) where the content of the document is decoupled from any particular treatment or rendition (style sheets).

If we assume an ideal sonification function, we can distinguish between the *musical content*, *i.e.,* the part of the score that contains the sufficient and necessary information to produce the intended music and the *score engraving*, *i.e.,* all the graphical symbols that are used to represent it. We propose a model of the musical score that distinguishes between these two components and encodes them in dedicated data structures.

In Table 2.1 we show some typical elements of a score, positioning them according to our model.

Musical Content	Score Engraving
Pitch Number	Pitch Spelling Key Signature Clef
Durations Metrical information	Note head Tie/Dots Beamings Tuplets

Table 2.1: Different elements in the musical score classified according to our score model.

Note that all these elements are present in a MusicXML file, yet all mixed in a redundant representation that can create consistency problems (*e.g.,* duration that does not correspond to the note figure and beaming specifications).

2.2.1 Musical Content Model

The *musical content model* focuses on the aspects of a digital score representation that describe the intended production of sounds, and is independent from any visualization concern. While at first it may seem that a representation like a quantized piano-roll, with information about notes onsets, offsets, velocities and different channels for different instruments, would be enough for this task, one can easily argue that there are other less obvious factors that influence the production of sounds from a score. Metrical information (time signature and division in measures) convey a hierarchy of "stronger" and "weaker" time positions and the division in voices results in differences in dynamics and articulation to distinguish between different melodic lines or between a melody and its accompaniment.

We model the musical content with a hierarchical structure. Musical artifacts (*e.g.,* notes, chords and rests) are grouped into voices, that are then organized into parts. We describe this organization with a bottom-up approach.

Modeling a voice musical content

Intuitively a *Voice* contains musical artifacts and their rhythmic information. A voice is *gapless*, *i.e.,* each ending date (except the last) coincide with the starting date of another musical artifact (remember that we also consider rests as musical artifacts) and *not-overlapping*, *i.e.,* musical artifacts that share the starting point, must also share the ending point. For example, a chord can be part of

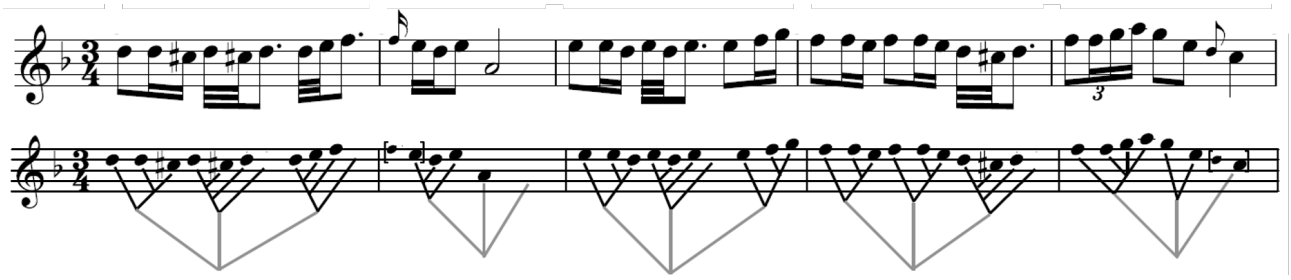


Figure 2.4: Example of rhythm trees, from Bach BWV Anh II 128. In this example we represent the symbols in $\mathbb{E}$ as note-heads in the pentagram. Note that grace-notes result in a leaf $\in \Sigma_0$ containing multiple symbols $\in \mathbb{E}$ (*e.g.,* first note in the 2nd measure) and continuations are encoded in leaves with empty sequences of symbols $\in \mathbb{E}$ (*e.g.,* last note in the 2nd measure).

a voice, but we don't allow two sequential short notes to happen over the time-span of a longer note. Sometimes the term "monophonic" is used in the literature, but we prefer to avoid it, since we can produce multiple sounds within a chord.

To encode the rhythmical information of a voice, we employ a hierarchical structure, that has been shown by many researchers to be a more expressive model for rhythm (*e.g.,* [47, 52, 51, 54, 53]). In particular, for each musical artifact, we encode its duration and its metrical stress (see Figure 2.4).

A voice in the musical content is represented with a sequence of *Rhythm Trees* (RT), one tree for each measure (see Figure 2.4). Formally, given a tree $t \in \mathcal{T}(\Sigma)$ (as defined in Section 1.1.1), we associate to each node $a \in dom(t)$ a time interval I_a of duration $dur_t(I_a)$ (expressed in *mtu*), defined as follows:

$$dur(I_a) = \begin{cases} 1 & \text{if } a = root(t) \\ \frac{dur_t(a')}{k} & \text{otherwise} \end{cases} \tag{2.1}$$

where a' is the parent of a in t and k is the arity of a'.

The same tree structure associates to each node a metrical stress $\mathcal{M}(a) \in \mathbb{Z}_{\leq 0}$ such that higher numbers encode stronger metrical stresses with 0 being the maximum:

$$\mathcal{M}(a) = \begin{cases} 0 & \text{if } a = root(t) \\ a & \text{if } a \text{ is leftmost child of } a' \\ \mathcal{M}(a') - 1 & \text{otherwise} \end{cases} \tag{2.2}$$

The symbols in Σ_0 (labelling tree leaves) encode all musical artifacts happening in the interval specified by the tree structure. To encode musical artifacts we consider a finite set of symbols, the

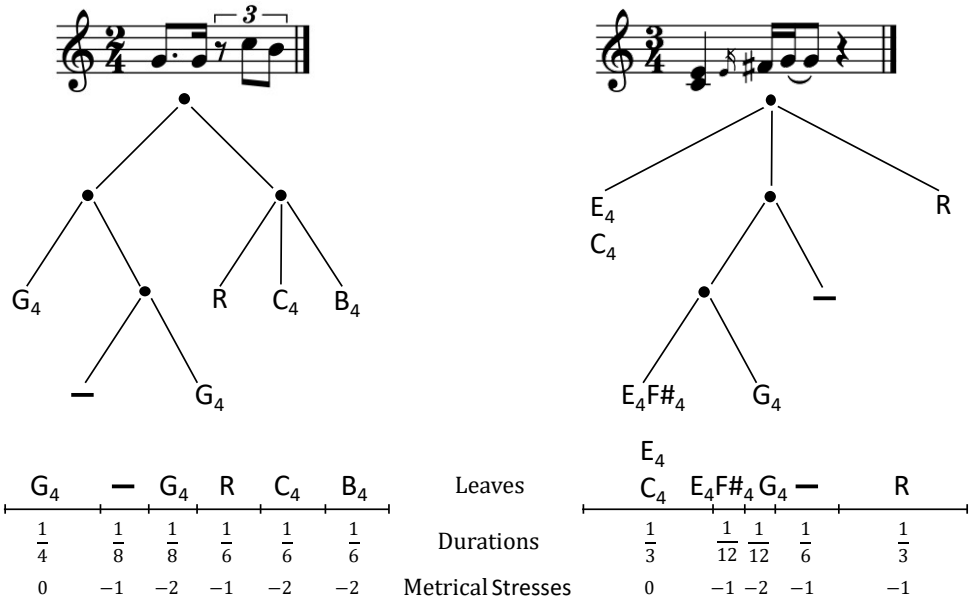


Figure 2.5: Examples of rhythm trees, with the duration and metrical stress displayed for each leaf. In this example we consider a $\mathbb{E}$ where each symbol encodes a list of pitches in scientific pitch notation (pitch-class$_{\text{octave}}$). A *cont* is graphically represented by the symbol $-$.

notational alphabet $\mathbb{E}$, where each symbol represents a list of n pitches (with $n \in \mathbb{N}$). In musical terms we call a symbol with $n = 0$ a rest, $n = 1$ a note and $n > 1$ a chord. When a symbol $a \in \Sigma_0$ contains multiple musical artifacts, all of them except the last are interpreted as grace notes. When a leaf contains no musical artifacts, we have a *continuation*, *i.e.,* an augmentation of the duration of the musical artifact in the previous leaf in a deep first traversal of the tree. Continuations are a fundamental concept in all hierarchical models of rhythm, since they allow to split a musical artifact whose duration exceeds the duration of the leaf where it is placed, in multiple parts that span multiple leaves. Other works (*e.g.,* [47]) use the term "tied note"; while a tied note result often in a continuation, we prefer to avoid this term, as we also may need to employ a continuation in other cases, *e.g.,* dotted eight-notes of measure 1 or the half-note of measure 2 in Figure 2.4.

Each voice specifies a *voice-start* attribute in *mtu*. The ending of the voice is not explicitly encoded, but can be derived from the cumulative sum of the duration of the RTs it contains.

Organization of parts

Voices are also grouped in a hierarchical structure. This organization, however, does not convey any rhythmic information, but serves uniquely the organizational aspect of the score.

Voices are grouped into *Single Parts*, that encapsulate the musical content assigned to an individual instrument (or vocal part), and single parts are grouped in *Group Parts* following a typical musical organization (*e.g.,* orchestra, Choir, Strings, Woods, Rhythmic Session). The group part at the root of the hierarchy represents the entire score. We can formalize this organization using a context-free grammar, where the symbol "*" allows for zero or more instances of the element and "|" allows a choice between different exclusive options:

```
Score := GroupPart*
GroupPart := GroupPart*|SinglePart*
SinglePart := Voice*
Voice := RT*
```

Figure 2.6 represents the part organization of a musical score for piano and (a small) orchestra. The hierarchy of group parts divides the instruments according to a typical musical organization: piano vs orchestra, winds vs strings. Finally each single part contains one ore more voices.

Our model consider also a *global* elements that is not part of this hierarchy: the time signature. Time signature is assigned to a specific time interval and will affect the portion of all voices that fall into that time interval.

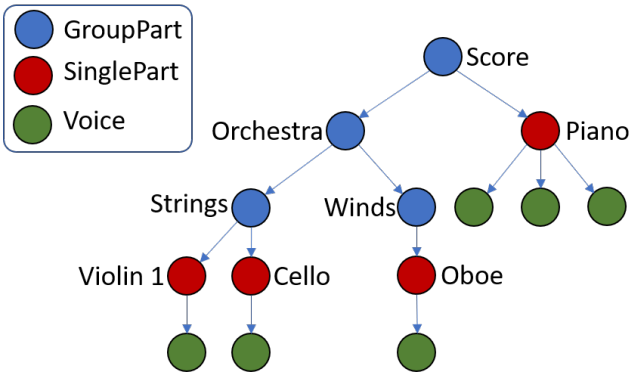


Figure 2.6: Parts organization in the musical content model.

2.2.2 Score Engraving Model

The engraving model contains information that allows human performers to efficiently parse the musical content. Elements such as the number and type of staves, the allocation of the parts to staves, clefs, key signatures and pitch spelling are encoded at this level.

Like for the previous section, we first describe the representation of voices and then we detail how they are organized in the complete score.

Modeling a voice engraving

Efficiently displaying a voice involves a series of choices in order to comply with the "rich and complex mix of rules, principles and best practices refined during centuries" [28]. We focus on the representation of two elements: pitch and rhythm. The former is displayed in the score with clefs, key signatures, vertical positions in the staff and (optionally) accidentals (the last two going under the name of *pitch spelling*); the latter uses graphical elements such as note-head, dots, ties, beamings and tuplets. The structure of beamings and tuplets is clearly hierarchical as both of them group notes in nestable containers (Figure 2.7). To represent a voice we employ two kinds of trees: a *beamings tree* $BT \in \mathcal{T}(\Sigma^B)$ and *tuplet tree* $TT \in \mathcal{T}(\Sigma^T)$. We generically refer to those trees as *Notation Trees* (NT). One voice is represented by a set of NTs, specifically one BT and one TT for each bar. The symbols that encode the leaves are shared (*i.e.,* $\Sigma_0^B \equiv \Sigma_0^T$ called for simplicity Σ_0) and given a measure, the two trees that represent it shares the same set of leaves (*i.e.,* $dom_0(BT) \equiv dom_0(TT)$).

In leaves we encode the pitch spelling (*i.e.,* note name, octave and accidental), dots, ties and grace-note information of notes, chords and rests. Formally, a symbol $a \in \Sigma_0$ consists of a quadruple that contains the following information (see Figure 2.7 for an illustration):

- *pitches*: either a rest or a chord: a sequence where each element contains a *natural pitch position*[3] (a chord can contain only a single note), an *alteration value* encoding values among 'none' and 𝄫, ♭, ♮, ♯, 𝄪 and a *tie* flag indicating if the corresponding note is tied to the closest left leaf. The sequence is ordered from the lowest to the highest pitch; in case of a single note (not a chord), *pitches* is a singleton sequence;
- *note-head*: a value representing the type of note head, *i.e.,* 𝅝, 𝅗𝅥, and ♩ for a note (or chord), and and 𝄻 , 𝄼 , 𝄽 for a rest;
- *dots*: a value that specify the number of dots;
- *grace-note*: a boolean value that specify if the note is a grace note.

[3]It represents the vertical position on the score, not the sounding pitch name, in order for it to be independent from the clefs and transposing instruments. For the same reason it does not consider alterations.

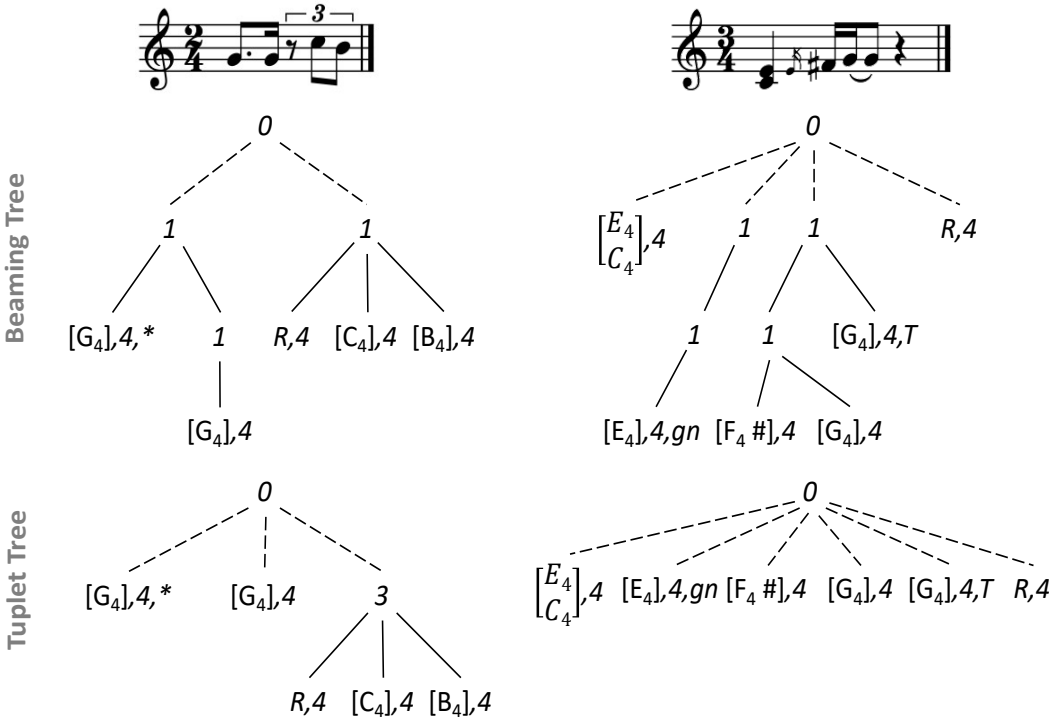


Figure 2.7: Two examples of notations trees (beaming trees and tuplets trees). To graphically represent the model, we use the symbols specified in the Example 1. Dotted or unbroken lines are used to graphically enforce the symbols in the internal nodes, *i.e.*, the absence of presence of beams (for BTs) and of tuplet number (for TTs).

A *beaming tree BT* structure represents two information: the number of beams over a note and the number of beams connecting two notes. Every internal node in Σ_k^B (with $k > 0$) encodes with a Boolean value the presence or the absence of beams between their children. Formally, let the *weight* of a node a be the number of nodes with a beam on the path from a to $root(BT)$ (including n itself). Let a_1 and a_2 be two successive leaves in $dom_0(BT)$ and a' be their least-common-ancestor in the tree (it is always defined and unique). The number of beams between a_1 and a_2 is either $weight(a')$, if in the two paths $a \to n$ and $\ell \to n'$ all inner nodes have a beam, or it is 0 otherwise.

The *tuplet trees TT* structure represents the numbers and brackets that are used in a score define the nature of the tuplets. Every internal node $\in \Sigma_k^T$ (with $k > 0$) is either unlabeled, or labeled by a pair of integers $i : j$, with $i, j \geq 2$, meaning *'i elements in the time of j'*, or by a single integer i if j can be omitted according to music notation practices [28].

Example 1. The formal model we presented can be practically implemented in multiple ways. We give in the following an example of the symbols that we can employ to represent the many information encoded in the NTs node labels. We represent rests with the letter R, pitches position in standard pitch notation, alterations with the common musical symbols (♯,♭, ♮, etc.), the tie flag with the letter T and dots with the symbol $*$ (that can be repeated multiple times). Note heads are represented with integers indicating the fraction of whole note: 1, 2, 4, represent respectively 𝅝, 𝅗𝅥, and ♩ for a note (or chord), and and 𝄻 , 𝄼 , 𝄽 for a rest. The presence or absence of beamings in the internal nodes of BTs are indicated respectively with the numbers 1 and 0. Tuplet numbers in TTs are encoded with integers $i > 0$ and unlabeled nodes are represented with the number 0. Dots and grace-notes are encoded respectively with the symbol "*" and "gn". ◇

Organization of parts

The engraving models define a set of *staves* grouped in *staff groups* and a mapping L that assigns leaves of the Notation Trees (*i.e.,* every note, chord or rest in the voice) to a staff. Every staff has a *staff name* to be displayed. L is surjective and can be non-injective, *i.e.,* every leaf is mapped to a staff, but multiple leaves can be mapped to the same staff.

If we consider a 4-part Bach fugue for instance, we may choose to place the notes of each voice in a different staff, or to use only 2 staves to have a more compact representation. In Figure 2.8 (left),

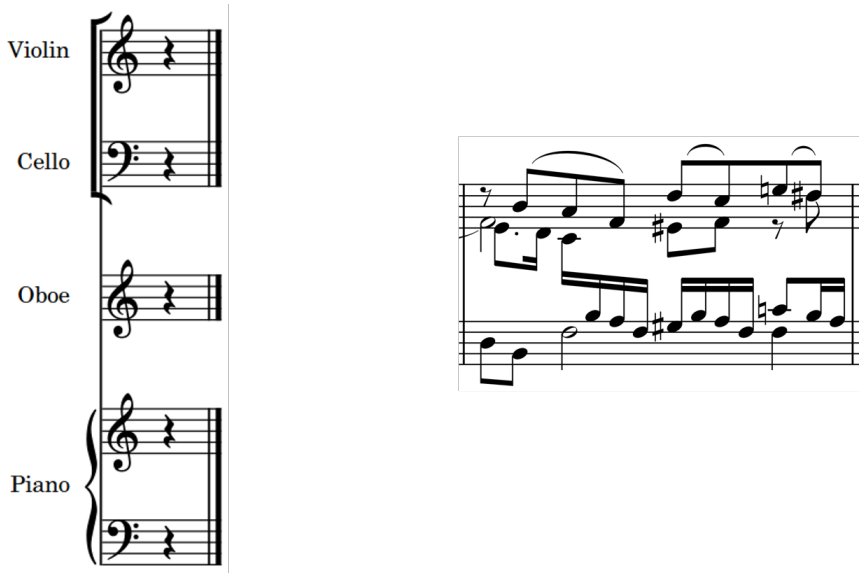


Figure 2.8: On the left: different staves and staff groups. On the right: the mapping of notes from 4 voices to two staves, from Fugue No 24 in B minor.

we see the definition of 5 staves and two staff groups, one for piano and one for the strings. This setting can be used to engrave the voices in Figure 2.6 if we want to assign each note of a voice to its dedicated staff. On the right we see an example of the notes of four voices mapped to only two staves, with a voice spanning between the two staves. This situation, not so rare in piano music, motivated us to define mapping as notes-to-staves, instead of voices-to-staves.

The *time-signature symbols*, are placed to all the staves at a specific time positions.

2.3 Musical Score Quality

The score model which we defined in the previous section is a complex data structure and, as such, it may contain quality problems. In this section we discuss on those issues and propose an approach to handle them.

Nowadays, most automatic music transcription systems use supervised or semi-supervised machine learning techniques that rely on annotated data to learn the model parameters. This means that the performances of those approaches are only as good as the quality of the training data [60]. Musical scores contain plenty of information that can be used as ground truth in full AMT tasks and on specific sub-tasks, such as pitch detection, beat and downbeat detection, note quantization and voices sepa-

ration. Unfortunately, musical scores often present many quality issues. The currently used formats for storage and distribution (see Section 1.3, page 29) don't help in this direction, as they are quite permissive in the accepted notation. This well fit their goal, since they ambition to encode as many different kinds of music and notation styles as possible, but it is detrimental for other applications, such as their usage for the full AMT task.

Defining and measuring the quality of a score encoding is not easy. Indeed, when one starts to figure out all the problems that can affect a score, it quickly seems that they are quite overwhelming. Even worse, their apparent heterogeneity seems to prevent an organized approach. Some aspects are purely syntactic (do all ties have a start/end note? Are those notes with the same pitch?), others are specific to the score layout (symbol overlapping, appropriate position of clefs and staves). Moreover the musical content itself has to be correct regarding the source, and the engraving consistent with respect to editors choices.

Some proposals were made in the field of OMR [31, 32, 33] but they are limited in scope as the goal is mainly to detect an incorrect number of beats per bar. A more structured approach, involving a classification of quality problem is proposed in [34]: the authors divide the problems as graphical and syntactic rules. In order to obtain a extensible framework that can avoid a large, messy catalogue of case-by-case quality rules, we need a similar, disciplined approach.

We exploit the organization of our score model to develop a structured and extensible catalogue of quality issues that can be used to automatically evaluate and visualize the problems in a musical score.

An automatic analysis of the musical content is especially useful since some problems could be hidden in the graphical representation and thus impossible to find just by visual inspection of the score. It is the case of notation tricks that human editors often use to obtain a good engraving, at the expenses of a good musical content (*e.g.,* hidden rests, hidden tuplets, grace notes represented as normal notes with smaller fonts). If we target specific transcription sub-problems, we can focus more on some indicators than others, *e.g.,* a beat tracker application would be more affected from musical content issues like incorrect duration than from a bad engraving of beamings. With the same techniques we can as well analyze the output of music transcription systems. This is particularly useful for approaches such as deep learning, where the output is generally unconstrained and can easily contain faulty music notation.

2.3.1 Modeling the quality

The taxonomy of quality rules that we propose is a forest where each tree corresponds to a part of the score model. We add a third component to our quality hierarchy: the *metadata* (*e.g.,* data that describe our data: title, composer, year etc.) as we believe that those are fundamental information when dealing with large collections of musical scores, for example to avoid multiple instances of the same piece. We describe it below:

1. Score content issues:
   (a) *Structural issues* concern the structure of the score: the length of its elements and their alignment. As an example of structural quality indicator, we check that all single parts are composed by the same number of measures.
   (b) *Voices issues* concern the events contained in a voice. For example all events in a measure must sum up to the expected measure length, according to the time signature (except pickup and mid-score pickup measures) and pitches should not be out of range.
2. Score engraving issues:
   (a) *Staves organisation issues* relate to the mapping that associates each leaf to a staff. While it is common in orchestral scores to have some voices merged in a unique staff to save space (*e.g.,* two horns), if the pitches of the two voices span a too big interval (*e.g.,* a piccolo and a double bass), the score would become impossible to read. Also the order of staves can be wrong: instruments must be grouped by family and ordered from higher (at the top) to lower pitch.
   (b) *Staff parameter issues* covers quality problems related to the rendering of music content on a staff. For example, all key signatures must be consistent, including a correct transposition for transposing instruments, or the clef should be chosen to ensure that the majority of notes lies inside the staff's range.
   (c) *Staff layout issues* concern the symbol that are used to display a voice on a staff. For example a note duration represented by some rhythmic notation symbols (note heads, beamings, dots, ties) could not match the duration of the note-event in the musical content; in practise this may happen in a MusicXML score, since the duration is encoded independently from

the head type and beaming information. Another example involve the beaming of notes, that must follow common practice in order to improve score readability and emphasize the meter[28].

3. Metadata issues: metadata bring information concerning the production context of the document (title, composer, provenance, data of creation, etc.). and are particularly important for the management of the document, for instance for classifying and retrieving data over a digital score library. Quality requirements address the availability of metadata information, and their accuracy.

Each rule defined in the quality model above can be easily expressed using our score model, as a set of constraints expressions on its concepts and structures.

2.3.2 Implementation

The quality framework that we developped (named *GioQoso*) is embedded in the Neuma platform[4], a digital score library devoted to the preservation and the dissemination of symbolic music content (scores). It can be used to analyze the scores in the Neuma library, or as an independent web service available at `http://neuma.huma-num.fr/quality`. At the moment it is able to analyze on-the-fly any MusicXML or MEI score accessible at a public URL annotating them with indicators of quality problems. Each annotation is an instance of a quality indicator, and the indicators themselves are organized as a forest, based on the taxonomy presented in Section 2.3.1. The visualization of the indicators is based on a procedure that involves:

1. a parsing of the MEI documents encoding the score (using music21 [36] for parsing and utility functions)

2. a call to the functions that find quality issues,

3. the usage of Verovio to display the score and highlight the quality issues.

The taxonomy of the quality model is extensible. Figure 2.9 shows the current status of the GUI of the GioQoso quality tool. In the user interface, the indicators are displayed in the top-right part;

[4] `http://neuma.huma-num.fr`

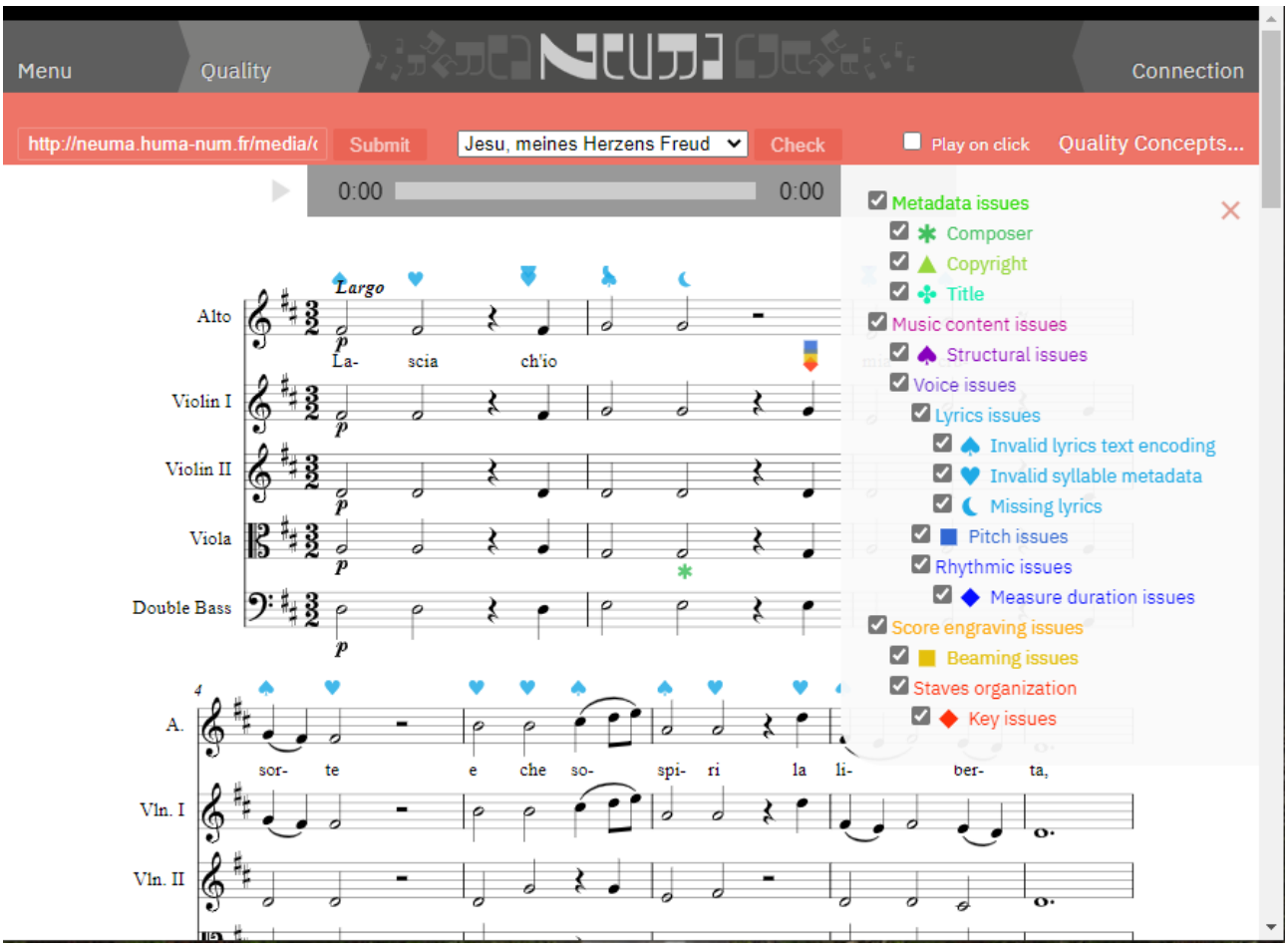


Figure 2.9: The GioQoso user interface.

each indicator comes with a description that can be highlighted by dragging the mouse over its name. Every annotation is displayed with a small icon above the elements or groups of elements where quality issues have been found. The user can hide/show a set of annotations by clicking on any level of the model tree, in order to focus on a particular aspect, or to ignore altogether some indicators if they are deemed irrelevant for the user intended application.

2.4 Conclusion

Our main contribution in this chapter is a novel score model, created for the task of automatic music transcription. Its main characteristic is its organization in two levels: the first encoding the musical content of the score and the second encoding the score engraving, *e.g.*, the symbols used to

encode the musical score. This allows us to model also the AMT process, decoupling the extraction of the information that is contained in a performance, from the specific notation symbol used to represent it, that depends on a series of notation "best practices".

When considering such complex structures, it is necessary to reflect about the quality issues that they can contain. In our settings of AMT, those can be produced during the transcription process, for example by deep-learning techniques that are difficult to constrain, or can be simply present in a dataset of musical scores that we want to use to train the transcription systems. Our score model allows us to easily define rules to detect quality problems, and groups them according to the part of the score they are affecting. We implemented the score model and the quality checking in the Neuma library.

In this chapter we focused on two musical aspects: pitch and rhythm. While those are usually the information produced by AMT systems, many other elements are encoded in a musical score, such as ties, dynamics, expression marks, articulations and embellishments (except for grace-notes that we already consider). Even if the research in this direction is limited, those elements influence the music that is produced and can in theory be extracted from a performance. We plan to extend our model in order to encode them.

A standalone implementation of this model as a Python library is under development, with the goal of giving a tool to produce musical scores to researchers that are more focused on "lower level" AMT tasks. For the same reason, we also work on a set of functions to transform the musical content into the score engraving, based on a set of "defaults" sets by the engraving best practices [28] and on some user-defined parameters, *e.g.,* how many staves to produce. Ideally this system could be used as last step for a full AMT pipeline (*e.g.,* [27]).

Chapter 3

A Parse Based Approach to Note Quantization and Metrical Alignment

In this chapter we focus on the production of the score musical content from a monophonic MIDI performance, *i.e.,* the quantization of notes and their alignment to a metrical structure.

As highlighted in Chapter 2, the set of rhythmical information that describes the production of a performance is encoded in the musical score across multiple elements: notes and rests durations are represented by symbols representing "simple" rational numbers (*e.g.,* quarter note, eighth note, triplets), and the temporal position (in *mtu*[1]) of each musical event is implicitly defined by the cumulative sum of durations of previous events. To convert those note symbols into a performance (in *rtu*), the score defines a *mtu* to *rtu* mapping using the following elements. The *time signature* defines a hierarchy of "strong" and "weak" temporal positions in *mtu* (called downbeats, beats, sub-beats) and specifies which note (and rest) symbol correspond to the interval between two beats. This structure is aligned with the notes through the subdivision of the score in *measures* and mapped to time positions in *rtu* through the *tempo* information. In general the tempo specifies the length in BPM of one beat (see Figure 3.1). To further complicate the situation, the time position of a note in the performance does not adhere perfectly to its specification in the score: musicians introduce intentional and unintentional deviations from those quantized time positions that are natural aspects of an expressive performance [13].

Different works in MIR have focused on the estimation of each of those information individually (quantized durations, tempo, time signature, beats, downbeats) from a performance [62, 63, 64, 65, 66],

[1] see Section 1.2.3 for a description of time units.

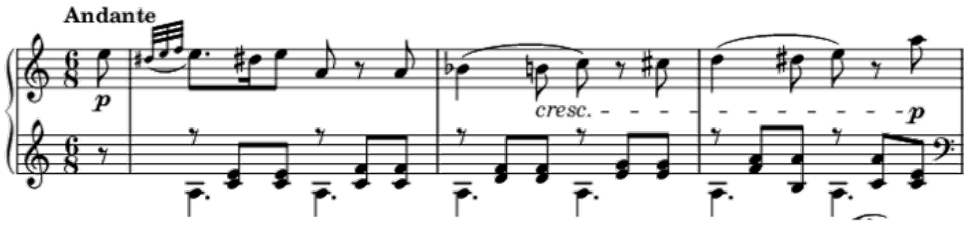


Figure 3.1: A score example from Mozart, Rondo in A minor K511. Different note (and rest) symbols represents different durations. The time signature $\frac{6}{8}$ specifies that the duration of one beat is encoded by the symbol ♩. (or multiple symbols with the same cumulative duration). The tempo indication *Andante* encodes a value between 76 and 108 BPM [61].

but, as they are highly correlated with each other, many researchers choose to consider multiple tasks at the same time. While there exist many different approaches, we can roughly group the works that target the retrieval of multiple rhythmical aspects in two main classes. The first (*e.g.,* [67, 68, 69]) focuses on metrical alignment, *i.e.,* the definition of a metrical hierarchy (generally measures, beats and sub-beats) and their alignment with time positions in the performed music. The second (*e.g.,* [70, 71, 72]) focuses on note quantization[2], *i.e.,* the operation of reverting the performing time deviations to retrieve the original quantized time position in the score.

However, none of the two approaches contains all the necessary information for the generation of a musical score. Metrical alignment systems don't produce any information on notes that don't fall on the temporal positions specified by the metrical structure, *e.g.,* sequences of notes shorter than sub-beats or defined by tuplets. At the same time, a list of quantized note durations (*e.g.,* [72]) poses considerable problems of notation[52]. Some quantization approaches produce a small set of metrical information (*e.g.,* [70, 71]), that can be used to produce a metrical structure by further post-processing the output (*e.g.,* [27]). While this may seem trivial for the notation of simple rhythms, it can lead to problems such as the misidentification of tempo scale (*e.g.,* halved or double tempos), metre (*e.g.,* confusion between 4/4 and 3/4), and positions of bar lines [75].

In this chapter we propose a parse-based approach for monophonic MIDI performances that jointly performs rhythm quantization and metrical alignment to a hierarchical structure, relying on an expressive and intuitive set of rules that can be set manually or learned from a dataset of quantized music. The output of our algorithm is a list of rhythm trees (see Section 2.2.1) encoding both the quantized duration and the position in a metrical structure for each note contained in the performance.

[2]Sometimes called "rhythmic transcription" in the literature (*e.g.,* [73, 70, 74, 72]).

According to our formulation in Chapter 2.2, this is a sufficient representation of the musical content of the score, and for simplicity during this chapter we will refer to it as the *score*. We name our process *midi-to-score* (M2S).

The remaining of the chapter is structured as follows. First, in Section 3.1, we present the state of the art for note quantization and metrical alignment. In Section 3.2 we set up our framework and formally define the M2S problem. Our main contributions of this chapter, the CFG-based parsing algorithm of and a technique to learn the CFG from a dataset of scores, are presented respectively in Section 3.3 and 3.4. We partially published the content of those chapters in [Foscarin et al., 2019c, Foscarin et al., 2019b]. Finally, Section 3.5 contains details on the implementation and on the experiments that we conducted to test our approach.

3.1 State of the art

Previous studies on transcription of rhythmical information from a MIDI-like representation are reviewed in this section. While there exist multiple studies that target a single rhythmical feature (*e.g.,* beat tracking, time signature estimation, tempo inference) we limit our attention to methods that build a more complete output w.r.t. the automatic music transcription task.

First we give an overview on the approaches for metrical alignment and note quantization that have been proposed in the literature. Then we review some works that share our goal of jointly targeting both problems, and we highlight the ideas that pose the basis of our work, and the limits that we address with our proposition.

3.1.1 Metrical alignment systems

Metrical alignment, *i.e.,* the task of producing a metrical hierarchy (usually downbeat, beat and sub-beat) and its alignment with temporal positions in a performance, have been performed almost exclusively on MIDI data. To our knowledge, an approach was attempted by Whiteley et al. [76]: they target audio live performance data by jointly modelling tempo and meter with a probabilistic approach. However, the evaluation is very brief, only testing the model on 3 bars of a single Beatles piano performance, and the idea was not used further on MIDI data [77].

Some metrical alignment systems work only on quantized MIDI files. The earliest approach is

proposed by Longuet-Higgins and Steedman [78]: note-onsets are examined one by one in a left-to-right fashion, and beat levels are gradually built up in a manually defined rule-governed procedure. With a more mathematical based approach, Eck and Casagrande [79] introduce a way to compute autocorrelation such that the distribution of energy in phase space is preserved in a matrix and use it to detect the meter and align the hierarchical structure with the music. While they obtained promising results for metrical estimation, they show only preliminary results on the alignment task, working on a small set of artificial patterns. De Haas and Volk [68] successfully target the problem of metrical alignment on quantized data with the inner metric analysis (IMA) model, that analyses periodicities in the MIDI signal and determines a list of strong and weak metrical positions.

Other researchers target unquantized MIDI files, generated from performances. Those systems essentially enumerate a list of possible solutions and then proceed to select the one that gives the best score, that can be computed with different techniques. In [80, 81] Temperley uses a "preference-rule" approach with the following rules: prefer a structure that aligns strong beats with onset of longer events, prefer a structure that aligns strong beats with more simultaneous notes, prefer beats to be maximally equally spaced. While this approach gives a formulation very close to the musical intuition, the parameters are tuned manually and there isn't any principled basis for setting them other than inefficient optimisation methods like grid search. In a later work [67] Temperley targets those limitations by proposing a technique based on the probabilistic framework of Bayesian modelling. He models the probability of a note onset occurring, given a metrical hierarchy (a three-leveled structure). This, combined with a simple model of tempo changes, allows to jointly perform meter detection and alignment of monophonic MIDI performances, later extended for polyphonic performances [82]. McLeod and Steedman [77] improve the performances on this task, proposing an HMM model that incorporates a rhythmic grammar. This approach is then optimized in [83] on unseen rhythmic patterns using a dynamically learned piece-specific grammar that exploits musical repetition and self-similarity, enabling the model to recognize repeated rhythmic patterns, even when a similar pattern was never seen in the training data.

3.1.2 Quantization systems

Traditional quantization approaches, implemented in musical notation softwares, work by quantizing the notes to the closest point in a grid. The user has to define the quantization step through a series

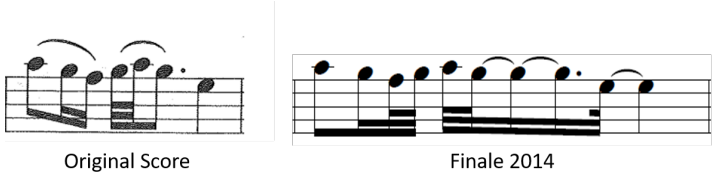


Figure 3.2: A comparison between the original score and the grid-quantized version of a performance generated from Finale2014, with a quantization step of a 32nd note.

of parameters, generally the tempo (in BPM), the time signature and the shortest considered note. A similar approach is described in [73], where the grid is specified by a list of user-defined rules that set constraints on the note durations. The grid approaches techniques have several limitations [52]: if the quantization step is too big, it always favors certain pulse speeds distorting others (*e.g.,* 8th notes vs 8th note triplets). The alternative of using a quantization step that allows to represent all note position in which we are interested would perform poorly, because the increment is too fine, even shorter than the shortest note (*e.g.,* in order to represent durations of 1/6 and 1/4, we need a grid step of 1/12), thus augmenting the risk of overfitting. Moreover, those models are focused only in the minimization of distance of performed notes from the grid, not on the evaluation of the note pattern that is obtained after the alignment; this often results in unnatural and complex patterns (see Figure 3.2).

A more advanced approach is the connectionist approach developed by Desain and Honing [84] that tries to limit the complexity of the output by iteratively converting note durations so that adjacent durations tend to have simple integral ratios. Another class of solution is offered by HMM models of the note duration[74] and [72]. Those approaches are very flexible and robust and the latter obtained state-of-the-art results in the quantization problem, especially for performances that contain polyrhythms. However, as those systems model sequences of notes durations without considering any metrical information, their output can contain syntactically incorrect sequences of notes (*e.g.,* an incomplete triplet) or even voices with different durations in the polyphonic. This makes them hard to employ for the production of a score.

Other approaches model with a Markov process the generation of note position in the measure. Raphael in [70] quantises performed rhythms to a grid using a Bayesian model that simultaneously estimates of the quantized note positions and tempo variables given the observed note positions from a MIDI performance. The probability of a note time positions an the tempo value are interconnected.

However the inference technique used to numerically solve the quantization problem only works on simple scenarios: the set of possible onset locations for notes is known a priori (and changes based on the time signature of the underlying piece). Cemgil [71] works on a similar model and he surveys different techniques for the numerical approximation of the quantization problem. He formulates his quantization problems as the problem of balancing the score complexity and the tempo deviations. However the definition of "score complexity" is based on a simple heuristic that assigns probabilities to events proportional to the bits necessary to encode the fractional part of their time position. For example in his settings, the notes in the score ♩♪♪♪ are encoded at positions $[0, 1, 1.5, 1.75]$; their fractional part (or the position $mod1$) are $[0, 0, 0.5, 0.75]$, or $[0, 0, 0.1, 0.11]$ in base 2, generating a "score complexity" of respectively $0, 0, 1$ and 2. This is equivalent to considering the depth of the time position in hierarchical structure of binary divisions and very similar to the heuristic of "preferring notes on stronger metrical positions".

A grammar based approach is proposed by Tsuchiya et al. [58]: they model with a tree structure both the temporal position of notes and their separation in multiple voices with the goal of obtaining a generative model for automatic music transcription. Practically they use two kinds of internal nodes: one divides a temporal interval in k sub-intervals (as presented above) and the other divides a voice in k voices. To reduce the search space of possible trees they encode in the leaves frequently occurring rhythm patterns instead of single musical events, following the idea that humans recognize rhythm not necessarily by perceiving each inter-onset interval as a particular note value, but rather by perceiving a set of inter-onset intervals as a particular rhythm pattern. Unfortunately, a tractable inference algorithm could not be obtained for the model. In the evaluation performed by [72] with a dataset of classical pieces from different composers, every piece took more than 24 hour for the computation and some were interrupted after one week of processing.

3.1.3 Jointly Note Quantization and Metrical Alignment

Few works in the literature share our goal of targeting both quantization and the alignment problem in the same process.

Longuet-Higgins's approach [46] parses a monophonic MIDI performance and builds a tree on top of its note that conveys both the quantized duration of notes and a metrical structure. The number of beats and the initial tempo are considered given. He employs a rule-based technique where

he recursively examines a temporal interval and split it in sub-intervals if a note happened "in the middle" of the interval. The final result is a tree partitioning the performance in intervals such that every note is "close" to the beginning of an interval. The choice of whether a note has to be considered close to the beginning or in the middle of an interval is left to a tolerance parameter. He also accounts for the modification of the tempo during a performance: "if, in the course of a binary split interval, a note which terminates the first sub-interval begins a little less than half way through, then the expected further duration of the unit is reduced in magnitude, to the mean of its original value and the value implied by the time of onset of the note in question" (page 647). Similar considerations are applied for late notes or ternary divisions.

The approach of Nauert [52] is motivated by a particular goal. As a contemporary composer, he develops a Markov chain for the automatic generation of notes in time, in particular with "non standard" durations. While he seeks this complexity in the listener experience, he still needs his music to be performed, thus researching for the less complex rhythmic notation. In other words, his goal is to produce a simple notation, while moving the notes as little as possible, thus making this equivalent to our goal of rhythmic transcription from a performance. His approach is based on hierarchical division of time intervals by prime numbers (that he call *Q-grid*) and relies on a set of preference-rule for evaluating the complexity of the result. The rules are very articulated, but can be generally resumed as a preference towards: slower notes, divisions by smaller prime, repetitions of previously occurred patterns and tied notes on weaker metrical positions.

Ycart et al. [53] propose a system to generate a musical score from a monophonic MIDI performance. Many similarities exist between this system and our approach as they are both grammar based with an analogous approach for the computation of the complexity of the score. However, the goal of Ycart et al. is fundamentally different: as proposed by Agon et al. [85], the transcription process is considered as an interactive creative process between the user and the software. In this scenario, there is no need for any parameter learning, since the user will manually tune them to reach its fixed goal or explore new possibilities.

While those three papers pose a good basis for our work, in a MIR scenario of score transcription from a performance, they all suffer from the same problems (the first because it is a rather old approach, and the other two because they have a different goal): the absence of an automatic and efficient technique to set the model parameters. In our paper we address this deficiencies by proposing

a technique for setting the parameters from a dataset of quantized music.

3.2 Framework definition

In order to formally define the problem of music transcription we introduce some key concepts and notation that extend the content of Chapter 1. We develop an abstract explanation and propose practical implementation choices in the examples.

3.2.1 Tempo curves

We call *tempo curve* a monotonically increasing function $\theta : \mathbb{Q}_+ \to \mathbb{Q}_+$, converting *rtu* values into *mtu* values as defined in Section 1.2.3 (page 28).

Let $\bar{\tau} = \langle \tau_0, \ldots, \tau_m \rangle$ be a sequence of rtu values with $\tau_0 = 0$ (typically the timestamps of input events). A *tempo model* $\mathcal{M}$ compatible with $\bar{\tau}$ is a set of tempo curves piecewise linear, with slope changing at the timestamps of $\bar{\tau}$. More precisely, every $\theta \in \mathcal{M}$ is defined by a sequence $\langle T_0, \ldots, T_{m-1} \rangle$ such that for all $0 \leq i \leq m-1$, the restriction of θ to $[\tau_i, \tau_{i+1}[$ is a line of slope $T_i \in \mathbb{Q}_+$ (expressed in bars per second in our case, instead of bpm). We employ $\mathcal{M}$ to express restrictions on the changes of T_i, according to the real durations $\tau_{i+1} - \tau_i$, in order to ensure a certain smoothness of the tempo curves.

3.2.2 Timelines

A *time interval* is a right-open interval $I = [\tau, \tau'[\subset \mathbb{Q}_+$. I is called *unbounded* when $\tau' = +\infty$ and *bounded* otherwise. The left bound τ and τ' are respectively called the *start* and the *end* of I, and denoted as $start(I)$ and $end(I)$.

We call *partition* of a time interval I a sequence of disjoint time intervals $I_1, \ldots, I_k$, with $k \geq 1$, such that $\bigcup_{j=1}^{k} I_k = I$. We also write $I = I_1 + \ldots + I_k$ in this case. The *k-split* of a bounded time interval I (for $k > 0$) is a partition of I of size k such that each component has an equal duration $\frac{dur(I)}{k}$. Each sub-interval I_j (with $1 \leq j \leq k$) is defined as:

$$I_j = \Big[start(I) + \frac{(j-1) \cdot dur(I)}{k}, start(I) + \frac{j \cdot dur(I)}{n} \Big[. \tag{3.1}$$

We denote the first resulting sub-interval I_1 as $left(I)$.

We assume given a *notational alphabet* $\mathbb{E}$, which is a finite set of symbols η to encode musical artifacts, *i.e.,* notes and rests.

An *event* e is a pair $\langle \eta, \tau \rangle$ made of a symbol $\eta \in \mathbb{E}$, denoted $symb(e)$ corresponding to a musical artifact, and a *timestamp* $\tau \in \mathbb{Q}$, denoted $date(e)$.

A *timeline* $\mathcal{I}$ is a pair $\langle I, \bar{e} \rangle$, where I is a time interval denoted $carrier(\mathcal{I})$ and $\bar{e} = e_1, \ldots, e_m$ is a finite sequence of events denoted $events(\mathcal{I})$, with increasing timestamps and such that $date(e_i) \in I$ for all $1 \leq i \leq m$. Note that we don't explicitly encode the ending dates of events, as they can be inferred from the available information. Specifically, given a timeline $\mathcal{I}$ with m events, the ending date of an event e_j, with $1 \leq j < m$ is $date(e_{j+1})$ and the ending date of $e_m = end(\mathcal{I})$. A timeline with timestamps in *rtu* (resp. *mtu*) is called a *real*-timeline (resp. *musical*-timeline). Operations defined on time intervals (*e.g.,* $+$,*end*,*k-split*) are extended to timelines as expected.

Example 1. A possible implementation choice for $\mathbb{E}$ is the set of MIDI note numbers $[0, 127]$ extended with the number 128 for the explicit representation of rests. ◇

Example 2. Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be timelines defined by: $carrier(\mathcal{I}_1) = [0, 1[$ and $events(\mathcal{I}_1) = \langle e_1, e_2, e_3 \rangle$, with respective timestamps $0.07, 0.72, 0.91$; $carrier(\mathcal{I}_2) = [1, 2[$ and $events(\mathcal{I}_2) = \langle e_4, e_5, e_6 \rangle$ with respective timestamps $1.05, 1.36, 1.71$. ◇

3.2.3 Weighted Context-Free Grammars for Rhythm

In order to automatically produce the Rhythm Trees (defined in Section 2.2.1, page 41) from a performance, we employ a context free grammar introduced in Section 1.1.3 (page 23). Each component of the grammar assumes a specific musical meaning.

Non-terminal symbols $\in Q$ represent musical intervals, ($k-\mathsf{div}$) rules define k-splits of time intervals and recursive application of ($k-\mathsf{div}$) rules represents nested time divisions. (term) rules define the musical artifacts in $\mathbb{E}$ occurring at the beginning of a certain time interval. In principle, every terminal symbol in $\mathbb{F}$ is a sequence of musical artifacts symbols (*i.e.,* $\mathbb{F} \equiv \mathbb{E}^*$) and we notate terminal symbols as $\bar{\eta} = \langle \eta_1, \cdots, \eta_n \rangle$. An empty sequence represents the *continuation* of an event started before and not yet released – notated with a tie or a dot in a score. A sequence of length $n > 1$, encodes $n - 1$ grace notes followed by one note. However, since our goal is to represent rhythm and we are not interested

in melodic details, we use as set of terminal symbols $\mathbb{F}$ a finite abstraction of $\mathbb{E}^*$,that considers only the number of musical artifacts in the leaf. This has the advantage of avoiding the definition of different rules for each possible sequence of musical symbols, keeping the size of $\mathcal{G}$ reasonably small.

The values defined by the *weight* function specify a degree of *preference* for each rule. We employ them to encode the *complexity* of a certain rule, *e.g.,* the presence of two grace notes in a certain interval may be seen as more complex than a single note, or a division in ternary division of a time interval as more complex that a binary division.

Example 3. The following $(2-, 3-\mathsf{div})$ production rules, with weight values in a tropical semiring, define two possible divisions of a bounded time interval represented by the *nt* q_0, into respectively a duplet and a triplet.

$$\rho_1 : q_0 \xrightarrow{0.06} \langle q_1, q_2 \rangle,\ \rho_2 : q_0 \xrightarrow{0.12} \langle q_1, q_2, q_2 \rangle.$$

In those rules, q_1 represents the first event in a division, and q_2 the others. Further binary divisions of time sub-intervals are possible with:

$$\rho_3 : q_2 \xrightarrow{0.1} \langle q_3, q_3 \rangle,\ \rho_4 : q_3 \xrightarrow{0.11} \langle q_4, q_4 \rangle. \qquad \diamond$$

Example 4. The set $\{0, 1, 2, 3\}$ is a finite abstraction of $\mathbb{E}^*$ where the symbols 0, 1, 2 represent resp. a continuation, one and two symbols of $\mathbb{E}$ and 3 represents a sequence of $\mathbb{E}^*$ of length ≥ 3. In the following, we assign a weight value (in a tropical semiring) to (term) productions rules of a grammar $\mathcal{G}$,

$$\begin{array}{cccccccccccccccc}
\rho_5: & q_0 & \xrightarrow{0.15} & 0, & \rho_6: & q_0 & \xrightarrow{0.01} & 1, & \rho_7: & q_0 & \xrightarrow{0.79} & 2, & \rho_8: & q_0 & \xrightarrow{1.02} & 3, \\
\rho_9: & q_1 & \xrightarrow{0.02} & 0, & \rho_{10}: & q_1 & \xrightarrow{0.01} & 1, & \rho_{11}: & q_1 & \xrightarrow{0.25} & 2, & \rho_{12}: & q_1 & \xrightarrow{0.64} & 3, \\
\rho_{13}: & q_2 & \xrightarrow{0.02} & 1, & \rho_{14}: & q_3 & \xrightarrow{0.04} & 0, & \rho_{15}: & q_3 & \xrightarrow{0.01} & 1, & \rho_{16}: & q_4 & \xrightarrow{0.01} & 1
\end{array}$$

The *nt* q_0 represents a whole bar, with a single event in ρ_6 (*e.g.,* a whole note in a $\frac{4}{4}$ bar), as a continuation in ρ_5 or preceded by 1 or 2 grace notes in ρ_7 and ρ_8; q_1 represents a first note in a division with a rule of Example 3 (preceded by 1 or 2 grace notes in ρ_{11} and ρ_{12}); q_2 represents the next notes in the same divisions and q_3 and q_4 further levels of divisions (grace notes are not allowed for q_2, q_3, q_4, ties are not allowed for q_4). $\diamond$

parse tree t	$[\![t]\!]_{[0,1[}$	$weight(t)$	notation
$t_1 = \rho_1(\rho_{10}, \rho_3(\rho_{14}, \rho_{15}))$	$0, \frac{3}{4}$	0.22	
$t_2 = \rho_1(\rho_{10}, \rho_3(\rho_{14}, \rho_4(\rho_{16}, \rho_{16})))$	$0, \frac{3}{4}, \frac{7}{8}$	0.34	
$t_3 = \rho_2(\rho_{10}, \rho_{13}, \rho_{13})$	$0, \frac{1}{3}, \frac{2}{3}$	0.17	
$t_4 = \rho_2(\rho_{11}, \rho_{13}, \rho_{13})$	$0, 0, \frac{1}{3}, \frac{2}{3}$	0.41	

Figure 3.3: Examples of parse trees and their serialization. A possible notation is given for a clearer understanding. The rules are defined in the Example 5.

3.2.4 Parse Trees and Serialization

Similarly to the Rhythm Tree, a parse tree t yields both a sequence of events and a hierarchical structure for those events. Its advantage compared to the rhythmic tree (and the reason why we employ it), is the possibility of computing a complexity value for the entire hierarchical structure, by composing the *weight* on the rules on its nodes (see Equation 1.1).

We extend the definition of the *weight* function on parse trees $t \in \mathcal{G}$ with $root(t) = \rho$ and children as follows:

$$weight(t) = \begin{cases} weight(\rho) & \text{if } \rho = q \xrightarrow{w} \bar{\eta} \ (\mathsf{term}) \\ weight(\rho) \otimes weight(t_1) \otimes \ldots \otimes weight(t_k) & \text{if } \rho = q \xrightarrow{w} q_1 \ldots q_k \ (k{-}\mathsf{div}) \end{cases} \tag{3.2}$$

To obtain a sequence of events from a parse tree we define the *serialization* function. Formally, the *serialization* is the association to every parse tree t of $\mathcal{T}(\mathcal{G})$ and every *mtu* interval I of a musical-timeline denoted $[\![t]\!]_I$ and defined as follows:

$$[\![\rho]\!]_I = \begin{cases} \langle I, \langle \eta_1, start(O) \rangle, \ldots, \langle \eta_n, start(O) \rangle \rangle & \text{if } \rho = q \xrightarrow{w} \bar{\eta} \ (\mathsf{term}) \\ [\![t_1]\!]_{I_1} + \ldots + [\![t_k]\!]_{I_k} & \text{if } \rho = q \xrightarrow{w} q_1 \ldots q_k \ (k{-}\mathsf{div}) \end{cases} \tag{3.3}$$

where $\bar{\eta} = \langle \eta_1, \ldots, \eta_n \rangle$ and $I_1, \ldots, I_k$ is a k-splits of I.

Example 5. Taking the rules of Examples 3, 4, Figure 3.3 presents parse trees and their serialization in a 1 bar *mtu*-interval. Note that t_1 has 3 leaves, but $[\![t_1]\!]_{[0,1[}$ contains only 2 events, because its second leaf is a continuation with 0 event. ◇

Example 6. We extend the grammar of Example 5 with $(2{-}\mathsf{div})$ rules $\rho_0 = q \xrightarrow{\mathbb{1}} \langle q_0, q_0 \rangle$, $\rho'_0 = q \xrightarrow{\mathbb{1}} \langle q_0, q \rangle$ for partitioning a *mtu* interval $O = [\tau, \tau'[$ into one bar $(nt\ q_0)$ and its right part $(nt\ q)$. These binary rules can be used to recursively divide a musical-time interval into several bars. The function $mus(\rho'_0)$ maps O to the partition made of $[\tau, \tau + 1[$ (first bar) and $[\tau + 1, +\infty[$ (rest of O), providing

that $\tau' > \tau + 1$ or $\tau' = +\infty$. For O of duration 2 bars, the serialization $[\![\rho_0(t_2, t_3)]\!]_O$ is a timeline with 6 events, with timestamps $0, \frac{3}{4}, \frac{7}{8}, 1, \frac{4}{3}, \frac{5}{3}$. This tree corresponds to the notation  , assuming a time signature of $\frac{1}{4}$. ◇

3.2.5 Input-Output Fitness

We model expressive timing of human performance by a composed application of a *global tempo curve* θ and *local time-shifts* for individual events. The *distance* δ measures time shifts between written and played events. It is computed in a semiring $\mathcal{S}$ based on a given $\delta_0 : \mathbb{Q}_+ \times \mathbb{Q}_+ \to \mathcal{S}$.

$$\delta(e_1, e_2) = \begin{cases} \delta_0(\mathit{date}(e_1), \mathit{date}(e_2)) & \mathit{if}\, \mathit{symb}(e_1) = \mathit{symb}(e_2) \\ \mathbb{0} & \mathit{otherwise} \end{cases} \tag{3.4}$$

We extend δ to sequences of events $\bar{e} = \langle e_1, \ldots, e_m \rangle$ and $\bar{e}' = \langle e'_1, \ldots, e'_n \rangle$ by $\delta(\bar{e}, \bar{e}') = \bigotimes_{i=1}^{m} \delta(e_i, e'_i)$ if $m = n$ and $\mathbb{0}$ otherwise, and to timelines by $\delta(\mathcal{I}, \mathcal{O}) = \delta\big(\mathit{events}(\mathcal{I}), \mathit{events}(\mathcal{O})\big)$.

Example 7. For a tropical semiring, let $\delta_0(\tau_1, \tau_2) = |\tau_1 - \tau_2|$. Its extension δ is a measure of the accumulation of time-shifts of events, in rtu. ◇

We also use a measure γ of *tempo variations* defined by a tempo curve, based on a given $\gamma_0 : \mathbb{Q} \times \mathbb{Q}_+ \to \mathcal{S}$. Given a real-timeline $\mathcal{I}$, and a tempo curve θ in a model $\mathcal{M}$ compatible with $\mathit{events}(\mathcal{I})$, γ is defined as

$$\gamma(\theta) = \bigotimes_{i=0}^{m-1} \gamma_0(T_{i+1} - T_i, \tau_{i+1} - \tau_i)$$

where $\langle T_0, \ldots, T_{m-1} \rangle$ is the sequence of slope values defining θ and $\langle \tau_0, \ldots, \tau_m \rangle$ are the timestamps in $\mathit{events}(\mathcal{I})$.

Example 8. For a $\mathcal{S}$ tropical, we can define γ_0 as the ratio between the variation of slopes $\gamma_0(dT, d\tau) = \frac{dT}{d\tau}$ when $d\tau \neq 0$, and $\gamma_0(dT, 0) = 0$. ◇

Altogether, we define the *fitness* of a quantized musical-timeline $\mathcal{O}$ (a score) to the real-timeline $\mathcal{I}$ (a performance), w.r.t. a tempo curve θ, as

$$\mathit{fit}(\mathcal{I}, \mathcal{O}, \theta) = \delta\big(\theta(\mathcal{I}), \mathcal{O}\big) \otimes \gamma(\theta).$$

In our settings, the output timeline $\mathcal{O}$ will be the serialization $[\![t]\!]_O$ of a parse tree t of a WCFG $\mathcal{G}$ over $\mathcal{S}$ (for a given mtu time interval O).

3.2.6 Transcription Objective

Assuming an alphabet $\mathbb{E}$, a commutative, idempotent and total semiring $\mathcal{S}$ and a fitness measure based on δ_0 and γ_0 as above, the M2S problem is defined as follows.

INPUT:
- a real-timeline $\mathcal{I}$, non-empty (*i.e.,* with $\|events(\mathcal{I})\| > 0$),
- a WCFG $\mathcal{G} = \langle \mathbb{E}^*, Q, init, P, weight, mus \rangle$ over $\mathcal{S}$,
- a musical-time interval $O = [0, N[$ with $N \in \mathbb{N}_+ \cup \{+\infty\}$,
- a tempo model $\mathcal{M}$ compatible with $events(\mathcal{I})$.

OUTPUT:
- a tempo curve $\theta \in \mathcal{M}$ defined on $carrier(\mathcal{I})$,
- a parse tree $t \in \mathcal{T}(\mathcal{G})$, such that $weight(t) \otimes fit(\mathcal{I}, [\![t]\!]_O, \theta)$ is minimal w.r.t. $\leq_{\mathcal{S}}$.

Therefore, the objective of M2S is to find a parse tree t representing a score that optimizes a combination of its *rhythmic complexity* (weight w.r.t. $\mathcal{G}$), and its *fitness* to the input $\mathcal{I}$. The two criteria are antinomic, in the sense that improving the fitness generally increases the complexity of the tree and viceversa.

Let us discuss the relevance of the above combination with $\otimes$ by reviewing two concrete domains used for $\mathcal{S}$.

If $\mathcal{S}$ is a Viterbi semiring then the weight w.r.t. $\mathcal{G}$ and the fitness are probability values and we want to maximize their product with $\otimes$. One can resettle this problem as a Bayesian approach to maximize the posterior probability of a certain score given a performance, since from the Bayes theorem we know that

$$P(\text{score}|\text{perf}) \propto P(\text{perf}|\text{score})P(\text{score}) \tag{3.5}$$

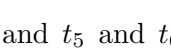
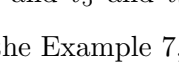
The likelihood $P(\text{perf}|\text{score})$ and the prior probability $P(\text{score})$ are respectively what we defined as *fitness* and *complexity*.

Formally, not all combinations of weights in a tropical semiring can be interpreted as probabilities.

We must also ensure that all the weights are correctly normalized, *i.e.,* for all $q_i \in Q$, $\sum_{q_i \xrightarrow{w} \alpha \in R} w = 1$

If $\mathcal{S}$ is a tropical semiring then the weight and the fitness can be seen as two unrelated quality criteria, and one can resettle M2S as a multicriteria optimization problem [86]. Let $\mathcal{P} = \mathcal{T}(\mathcal{G}) \times \mathcal{M}$ be the solution space for M2S, and let us consider the two objective functions c and d of $\mathcal{P}$ into the tropical semiring $\mathcal{S}$ defined, for $p = (t, \theta) \in \mathcal{P}$, by $c(p) = \mathit{weight}(t)$ and $d(p) = \mathit{fit}(\mathcal{I}, [\![t]\!]_O, \theta)$ (for the given $\mathcal{I}$ and O). By monotonicity of $\mathcal{S}$, we can restrict our search to so-called *Pareto-optimal* points $p \in \mathcal{P}$, *i.e.,* such that there is no $p' \in \mathcal{P}$ with $c(p') <_{\mathcal{S}} c(p)$ and $d(p') <_{\mathcal{S}} d(p)$.

Of these formulation, the probabilistic one is the most frequent in the MIR literature and offers a set of techniques to automatically set the model parameters (that we employ later in this chapter) in a context of fully automated transcription. However, considering M2S as a multicriteria optimization problem can be useful in a more supervised context (*e.g.,* an interactive transcription tool) as it exposes a parameter that allows to easily control the output. M2S is expressed as the minimization of the combination $c(p) \otimes d(p)$, where $\otimes$ is interpreted as a sum in $\mathbb{R}_+$. This is similar to a technique of scalarization by weighted sum, selecting a point p with minimal $\alpha.c(p) + d(p)$ (in $\mathbb{R}_+$) called *scalar optimal*. Intuitively, α can be seen as a user parameter setting how much one want to favour the rhythmic complexity against the fitness. In practice, one can apply the coefficient α to $\mathit{weight}(t)$ by multiplying by α all the weight values in productions of $\mathcal{G}$. This approach is correct and complete in the following sense: every scalar optimal point $p \in \mathcal{P}$ (for some α) is Pareto-optimal and for all Pareto-optimal point $p \in \mathcal{P}$ there exists a coefficient α such that p is a scalar optimal for α (Theorem 11.17 and Corollary 11.19 of [86], chapter 11.2.1).

Example 9. Let $\mathcal{I} = \mathcal{I}_1 + \mathcal{I}_2$ (see Example 2), $\mathcal{G}$ be the WCFG from the Ex. 3,4,6, a musical-time interval $O = [0, 1[$ and a tempo model containing a single tempo curve θ_0 mapping 1 seconds to 1 measure. Two possible solutions to M2S are the parse trees $t_5 = \rho_0(t_2, t_3)$ (Ex. 6) and $t_6 = \rho_0(t_1, t_4)$. Their serialization $[\![t_5]\!]_O$ and $[\![t_6]\!]_O$ have both six events with respective timestamps $(0, \frac{3}{4}, \frac{7}{8}, 1, \frac{4}{3}, \frac{5}{3})$ and $(0, \frac{3}{4}, 1, 1, \frac{4}{3}, \frac{5}{3})$; and t_5 and t_6 respectively correspond to  and  . By using the distance defined in the Example 7, we obtain $\mathit{weight}(t_5) \otimes \mathit{fit}(\mathcal{I}, [\![t_5]\!]_O, \theta_0) = 0.51 + 0.265 = 0.775$ and $\mathit{weight}(t_5) \otimes \mathit{fit}(\mathcal{I}, [\![t_6]\!]_O, \theta_0) = 0.63 + 0.32 = 0.95$. That means that t_5 is preferred over t_6, and the M2S algorithm that we will present will return t_5 as optimal solution. The reason is that in $\mathcal{G}$ the weight for

having a grace note (rule ρ_{11}) is quite high compared to the other rules. If we lower the weight of ρ_{11} to 0.07, then $\mathit{weight}(t_6) \otimes \mathit{fit}(\mathcal{I}, [\![t_6]\!]_O, \theta) = 0.77$ and t_6 becomes the optimal solution. This illustrates the notation preferences defined in $\mathcal{G}$, *e.g.,* for favouring grace-notes or precise rhythmic notation. $\diamond$

3.3 Transcription Algorithm

We now present a transcription algorithm that works in two steps: first it computes a WCFG $\mathcal{K}$ by augmenting $\mathcal{G}$ with some information from the input $\mathcal{I}$ (Sections 3.3.2 and 3.3.3 describe two examples of construction of such $\mathcal{K}$, corresponding to different use cases) and then it solves M2S by computing an optimal parse tree for $\mathcal{K}$, using a Dynamic Programming algorithm presented in next Section 3.3.1.

Readers that are familiar with Bayesian approaches may benefit from the following intuitive description of our approach. We consider the Equation 3.5 (page 67) using two grammars: one grammar $\mathcal{G}$ (the one described in Section 3.2.3) models the score prior and a second grammar $\mathcal{G}_{\text{perf}}$ (tied to a specific performance) models the likelihood of a certain parse-tree w.r.t. the performance. A naive approach would enumerate all possible solutions, rate them according to the two grammars weights and pick the best one. However, this approach is not tractable, as the number of computations to perform is too high. Inspired by [87, 88] our proposal can be seen as considering a combination of the two grammars, generating a third one:

$$\mathcal{K} = \mathcal{G}_{\text{perf}} \times \mathcal{G} \tag{3.6}$$

The grammar $\mathcal{K}$ is then employed for the parsing algorithm. Practically we don't explicitly perform the combination of the two grammars, but we obtain an analogous result by building directly the grammar $\mathcal{K}$ with a rule-based approach. We will detail such approach and present a proof of its correctness in Section 3.3.2.

3.3.1 Viterbi 1-best algorithm

Let $\mathcal{K}$ be a WCFG with *nt* set $\mathbb{K}$ over a total semiring $\mathcal{S}$. The following recursive function $\mathit{best}_{\mathcal{K}}$ associates to every *nt* $k \in \mathbb{K}$ a parse tree $t \in \mathcal{T}(\mathcal{K}, k)$ with a weight optimal w.r.t. $\geq_{\mathcal{S}}$. By abuse of notation, we make no distinction below between a parse tree $t \in \mathcal{T}(\mathcal{K})$ and its weight value $\mathit{weight}(t) \in \mathcal{S}$. Since $\mathcal{S}$ is total, it means that $\oplus$, applied to parse trees of $\mathcal{T}(\mathcal{K})$, selects the tree with minimal weight w.r.t. $\leq_{\mathcal{S}}$.

$$best_{\mathcal{K}}(k) = \bigoplus_{\rho_0 = k \xrightarrow[\mathcal{K}]{w_0} a} \rho_0 \oplus \Big[\bigoplus_{\rho = k \xrightarrow[\mathcal{K}]{w} \langle k_1, \ldots, k_n \rangle} \rho\big(best_{\mathcal{K}}(k_1), \ldots, best_{\mathcal{K}}(k_n)\big) \Big]$$

If $\mathcal{K}$ is acyclic, then the above definition is well founded and the following results holds (*e.g.,* by induction on k following a topological sort of $\mathbb{K}$).

Remember that the weight of a parse tree is the product of all the weights of the production rules labeling its nodes. Therefore, the above formula can be understood as an alternation of sums with $\oplus$ (selection of one parse tree of optimal weight) and products with $\otimes$ (of weights of all subtrees). The function *best* can be computed by a straightforward adaptation of a Viterbi-like Dynamic Programming algorithm returning the best derivations for weighted acyclic hypergraphs [89]. With a tabulation over $\mathbb{K}$, in order to avoid recalculation of solution for subproblems, this algorithm runs in time linear in the size of $\mathcal{K}$.

We apply this algorithm to a WCFG $\mathcal{K}$ built on the top of $\mathcal{G}$ from an input timeline $\mathcal{I}$. Two particular computations of $\mathcal{K}$ corresponding to different case studies are presented below.

3.3.2 Constant Tempo

In this first case study, we assume a constant tempo. This case study, illustrated in Example 9, corresponds to performances recorded with a metronome. The tempo model is $\mathcal{M} = \{\theta_0\}$, where a single tempo curve θ_0 represents the constant tempo value T which, for the sake of simplicity, we assume below to be the identity ($T = 1$).

In this case, the purpose of M2S transcription is essentially to correct local time-shifts of events. A parse tree t defines a partition of a given time interval O, by recursive application of the division rules labeling the nodes of t (see the definition of $[\![t]\!]_O$). This partition can be seen as a "*grid*" containing the time positions of the bounds of the sub-intervals. M2S then consists in the *realignment* of the events of $\mathcal{I}$ to the nearest bound in the grid. The cost of this alignment is then the distance, with δ, between $\mathcal{I}$ and the score represented by t.

$\mathcal{K} = \langle \mathbb{E}^*, \mathbb{K}, init, P, weight \rangle$ is built to represent all the time sub-intervals defined from $carrier(\mathcal{I})$ by recursive divisions, according to the rules of $\mathcal{G}$. For this purpose, every *nt* $k \in \mathbb{K}$ contains two components accessible with the following attributes:

- $k.nt$: a nt of $\mathcal{G}$,
- $k.car$: a real-time interval embedded in $carrier(\mathcal{I})$.

(div) productions rules are of the form $\rho = k \xrightarrow{w} \langle k_1, \ldots, k_n \rangle$, where

1. $\rho' = k.nt \xrightarrow[\mathcal{G}]{w} \langle k_1.nt, \ldots, k_n.nt \rangle$ is a rule of $\mathcal{G}$,
2. $k_1.car, \ldots, k_n.car$ is the application of k-split to $k.car$,
3. $events(\mathcal{I}|_{k.car}) \neq \emptyset$, meaning that k is *inhabited*[3].

The last condition drastically reduces the size of $\mathcal{K}$ in practice, *wlog* since it is useless to divide empty intervals.

(term) rules of P deal with the realignement of the events of $\mathcal{I}$ to the bounds defined by a parse tree of $\mathcal{K}$, and compute the input-output distance with the Equation 3.4 (page 66). For a nt k, we know the events of $\mathcal{I}$ inside $C = k.car$. Some of these events (those in the first half of C, called *early*) will be aligned to the left bound of C. The others (those in the second half of C, called *late*) will be aligned to the right bound of C, *i.e.,* actually to the left bound of the interval defined by the next leaf in a parse tree. To deal with this situation correctly, we add two components to every nt $k \in \mathbb{K}$, accessible with the following attributes:

- $k.post$: the late events of $\mathcal{I}$ in $k.car$ (*i.e.,* those in its second half),
- $k.pre$: a buffer memorizing the *post* for the previous leaf.

And we add the following conditions for every (div) production rule:

4. $$\begin{cases} k_1.pre &= k.pre \\ k_{i+1}.pre &= k_i.post \quad \forall 1 \leq i \leq n \\ k_n.post &= k.post \end{cases}$$

This ensures a correct bottom-up propagation of *pre* and *post* values.

The other rules of P have the form (term) $k \xrightarrow{w} \bar{e}$, where $\bar{e} = k.pre + events(\mathcal{I}|_{left(k.car)})$ (concatenation of the *pre* buffer and the vector of events of $\mathcal{I}$ early in $k.car$), $events(\mathcal{I}|_{right(k.car)}) = k.post$ and $w = w_0 \otimes \left[\bigotimes_{i=1}^{\|\bar{e}\|} \delta_0(start(k.car), date(e_i)) \right]$, with $q \xrightarrow{w_0} \bar{e}$ in $\mathcal{G}$.

[3]The restriction $\mathcal{I}|_C$ of $\mathcal{I}$ is such that by $carrier(\mathcal{I}|_C) = C$ and $events(\mathcal{I}|_C)$ is the sub-sequence of $events(\mathcal{I})$ inside C.

The weight of the above (term) rule combines with $\otimes$ the weight w_0 of the corresponding rule in $\mathcal{G}$ (*i.e.,* a complexity value) with the distance for the realignment of the points of $\bar{e}$. Moreover, $k_0 = init_{\mathcal{K}}$ is defined by $k_0.nt = init_{\mathcal{G}}$, $k_0.car = carrier(\mathcal{I})$, $k_0.pre = k_0.post = 0$.

Let us now sketch a proof that from $best_{\mathcal{K}}(k_0)$, one can build a solution for M2S conform to the definition in Section 3.2.6 (page 67). First, one can observe that every parse tree $t \in \mathcal{T}(\mathcal{K})$ can be projected onto a parse tree $\pi_{\mathcal{G}}(t) \in \mathcal{T}(\mathcal{G})$. Indeed, by condition (1.) for (div) rules of $\mathcal{K}$, it is sufficient to replace, in every label of t, every $nt\ k$ by $k.nt$ and, for (term) rules, to replace the rule's weight by the weight defined in $\mathcal{G}$. Next, for $k \in \mathbb{K}$, let use us define $\mathcal{I}|_k$ by $carrier(\mathcal{I}|_k) = carrier(\mathcal{I}|_{k.car})$ and $events(\mathcal{I}|_k) = k.pre + events(\mathcal{I}|_{left(k.car)})$ (using the above notations).

By induction on t, using the above conditions (2-4) for inductions steps, we can see that for all $k \in \mathbb{K}$ and $t \in \mathcal{T}(\mathcal{K}, k)$ $weight(t) = weight(t') \otimes fit(\mathcal{I}|_k, [\![t']\!]_{k.car}, \theta_0)$, where $t' = \pi_{\mathcal{G}}(t)$.

From this and the fact that $\mathcal{I}|_{k+0} = \mathcal{I}$ it follows that that $(\pi_{\mathcal{G}}(best_{\mathcal{K}}(k_0)), \theta_0)$ is a solution of M2S for $\mathcal{G}$ and $\mathcal{I}$.

3.3.3 Coupled Tempo Inference and Rhythm Quantization

In this second case, we sketch a construction of grammar $\mathcal{K}$ that permits a joined evaluation of a tempo curve θ and a parse tree t. Instead of realigning late events to the right bound of the current real-time interval, we move the right bound of the interval, such that the modified interval contains only early events. More precisely, in an interval $[\tau, \tau'[$ obtained by k-splitting, we modify the right bound τ' into τ'' in a way that all the events inside $[\tau, \tau''[$ are closer to τ than to τ''. The new interval length induces the definition of a piecewise linear tempo curve θ defined in Section 3.2.1 (page 62).

In addition to the attributes $k.nt$ and $k.car$ defined in the previous Section, every $nt\ k$ of $\mathcal{K}$ has a third attribute $k.mod$ for a real-time interval modified from $k.car$. The latter is defined in (term) production rules $k \xrightarrow{w} \bar{e}$, such that events $\bar{e}$ in $k.mod$ are all early in this interval. The propagation of *mod* values for (div) rules is similar to that of *pre* and *post* in the previous section.

Moreover, the non-terminals of $\mathcal{K}$ also store the slope values induced by the $k.mod$, and the (div) rules also verify that the corresponding tempo curve θ belongs to the given model $\mathcal{M}$.

3.3.4 Metrical inference

In this section we discuss how our system can be used to perform the task of metrical inference. Let's suppose we have n grammars $G_1, \cdots, G_n$ that are disjointed *i.e.,* for the sets of non-terminals $Q_1, \cdots, Q_n$ it holds that $Q_1 \cap \cdots \cap Q_n = \emptyset$. We merge them in a single grammar G by setting an initial non-terminal that is shared between all the grammars, *i.e.,* we substitute the symbols $init_1, \cdots, init_n$ in the rules of all grammars with an unique symbol *init*. By using G in our parsing process, we will generate a best solution tree that contains rules from only one of the initial grammars (except *init*).

If we create those grammars in order to be related to a specific time-signature (more details on grammar learning in the next section), this process will output the best time-signature for each measure. The generalization to pieces where time signature does not change, can then be made simply by selecting the initial grammar that generated the best tree for the majority of measures or by using more complex approaches that consider also the probability of the time signature in a given context. Some optimizations based on heuristics can then be applied, for example discarding some of the initials grammars if they are never selected after a few measures.

3.4 Learning the grammar from a corpus of scores

We have proposed a grammar-based approach to solve the M2S problem. The grammar rules have a specific and intuitive musical meaning, so one can manually define them. However, in the context of automatic music transcription, a more modern and flexible approach is to automatically learn them from a dataset.

It may seem straightforward to base the learning step on the many corpora of existing (quantized) MIDI files or digital scores. However, while these datasets provide sequences of quantized events, they contain no explicit information about the metrical structure in which the notes are grouped. This information have to be inferred (like a musician reading the score would do) from the time signature and the position of barlines.

There is mostly a general consensus in music theory regarding the first divisions of the metrical structure encoded by a time signature (see Table 1.3) and those information have already been used to set model probabilities (*e.g.,* [67]). However, considering lower metrical levels, there is no direct mean to chose between the multiple possible structures (see Figure 3.4). This phenomenon of *rhythmic*

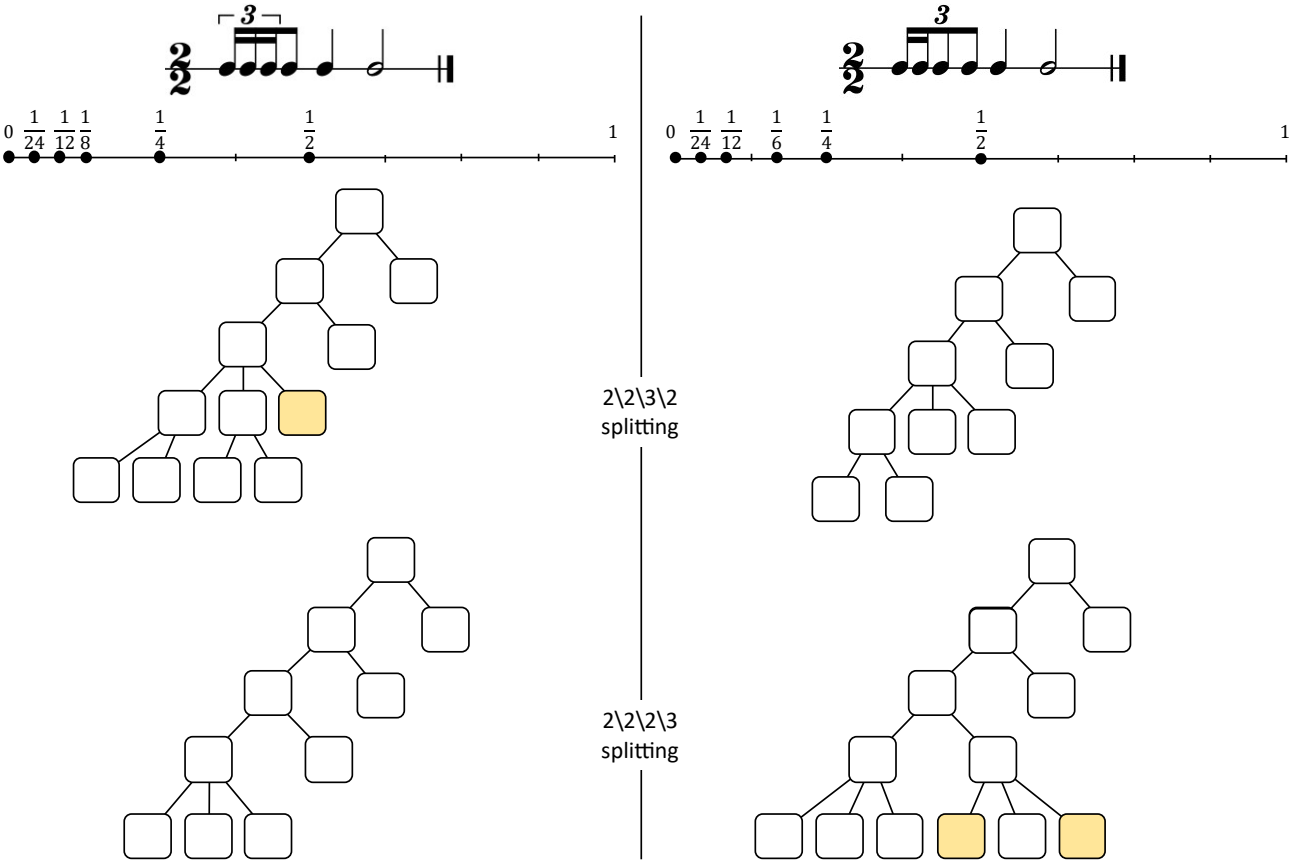


Figure 3.4: An example of different metrical structures that can be build on quantized timelines (the musical notation is displayed only for an easier understanding). For simplicity we do not display the rules in the parse-tree, but we mark with a different colour the rules that encode a continuation.

ambiguity can be extended to the whole metrical structure if we consider the point of view of a listener that has no access to the time signature information. However, in general listeners agree on the rhythmic structure of a given sequences of note values. Longuet-Higgins and Lee worked on this topic, trying to find the perceptive rules that govern this sort of "natural" interpretation of rhythm. They conclude that "when a sequence of notes can be interpreted as the realization of an unsyncopated passage, then the listener will interpret the sequence this way" ([47] page 431). This can be easily translated in our framework as the choice of the tree with the minimal number of leaves. Since the number of leaves has to be at least the number of notes, this can be seen equivalently as the minimization of continuations in the leaves. In the example of Figure 3.4, this select the 2/2/2/3 splitting on the left and the 2/2/3/2 on the right.

In the following we detail the application of this technique for building a training set of parse

trees, and we show that, as expected, our minimization criteria lead to the same high metrical division specified by the time signature symbol for a significant majority of the dataset. We then consider the problem of computing weight values in the rules of a PCFG using such training set.

3.4.1 Training Set Construction from a Score Corpus

We produce a training set of parse trees from a dataset of quantized sequences. This applies to a wide range of input scores, from monophonic (quantized) MIDI files to XML scores.

In our case we chose a corpus of XML scores, exploiting the explicit division in voices offered by this format. From each score in the corpus, each part in the score, each voice in the part, and each measure in the voice, we extract a timeline (of carrier $[0, 1[$) from the list of event durations. We use this dataset $\mathcal{D}$ of extracted timelines as input to build parse trees. The resulting set of parse trees is the training set C used for learning a grammar.

Formally, let us assume given a grammar $\mathcal{G}$ whose weights are initially unknown (we call such $\mathcal{G}$ *unweighted*). We define a function *rep* that returns for a non terminal symbol $q \in Q$ and a timeline $\mathcal{I} = \langle I, \bar{e} \rangle$, a parse tree $t \in \mathcal{T}(\mathcal{G})$ such that: $root(t)$ is headed by q, $[\![t]\!]_I = \mathcal{I}$ and t has the minimum number of leaves:

$$rep(q, \mathcal{I}) = \begin{cases} \rho = q \to \bar{e} & \text{if } \bar{e} = \emptyset \\ & \text{or } date(e) = start(I)\ \forall\ e \in \bar{e} \\ min_{\rho = q \to (q_1, \cdots, q_k)} \Big(\rho\big(rep(q_1, \mathcal{I}_1), \ldots, rep(q_k, \mathcal{I}_k)\big) \Big) & \text{otherwise} \end{cases} \tag{3.7}$$

where $\mathcal{I}_1, \ldots, \mathcal{I}_k$ is the k-split of $\mathcal{I}$, and min of a set T of trees is the tree with a minimum number of leaves.

The proof that this definition of *rep* generates the tree with the minimal number of leaves is straightforward, because the function that associates to a tree its number of leaves is monotonic, *i.e.,* if t_i has more leaves than t'_i, then $\rho(t_1, \ldots, t_i, \ldots, t_k)$ has more leaves than $\rho(t_1, \ldots, t'_i, \ldots, t_k)$. It follows that we can build a best representative t for a timeline $\mathcal{I}$ (w.r.t. this criteria) from best representatives for sub-timelines of $\mathcal{I}$ (which are sub-trees of t).

The representative of a 1-measure timeline $\mathcal{I}$ in the dataset $\mathcal{D}$ is $rep(q_{init}, \mathcal{I})$. It may however be undefined, either because there is no parse tree $t \in \mathcal{T}(\mathcal{G})$ yielding $\mathcal{I}$, or, on the contrary, because there

are more than one such parse tree with an equivalent minimum number of leaves. For now we assume that we can build a minimal tree for each 1-measure timeline, and we will discuss the case of undefined trees in Section 3.4.3.

Example 10. Let us present some steps of the computation of *rep* for the grammar in Example 11 (page 77) (ignoring the weight values), and the timeline $\mathcal{I} = \langle [0, 1[, (0, \frac{3}{4})\rangle$.

$$rep(q_1, \mathcal{I}) = \min \begin{cases} \rho_1(rep(q_{\frac{1}{2}}, \mathcal{I}_{2,1}), rep(q_{\frac{1}{2}}, \mathcal{I}_{2,2})), \\ \rho_2(rep(q_{\frac{1}{3}}, \mathcal{I}_{3,1}), rep(q_{\frac{1}{3}}, \mathcal{I}_{3,2}), rep(q_{\frac{1}{3}}, \mathcal{I}_{3,3})) \end{cases}$$

where

$$\mathcal{I}_{2,1} = \langle [0, \tfrac{1}{2}[, (0)\rangle,\ \mathcal{I}_{2,2} = \langle [\tfrac{1}{2}, 1[, (\tfrac{3}{4})\rangle,\ \mathcal{I}_{3,1} = \langle [0, \tfrac{1}{3}[, (0)\rangle,\ \mathcal{I}_{3,2} = \langle [\tfrac{1}{3}, \tfrac{2}{3}[, (\ )\rangle,\ \mathcal{I}_{3,3} = \langle [\tfrac{2}{3}, 1[, (\tfrac{3}{4})\rangle$$

Following (3.7), $rep(q_{\frac{1}{2}}, \mathcal{I}_{2,1}) = \rho_{11}$, $rep(q_{\frac{1}{3}}, \mathcal{I}_{3,1}) = \rho_{14}$, and $rep(q_{\frac{1}{3}}, \mathcal{I}_{3,2}) = \rho_{13}$. For $rep(q_{\frac{1}{2}}, \mathcal{I}_{2,2})$ and $rep(q_{\frac{1}{3}}, \mathcal{I}_{3,3})$, more computation steps are needed. ◇

3.4.2 Use-Cases of Unweighted Grammar

During the definition of *rep*, we assumed given an unweighted grammar. In theory, such grammar must be as complete as possible; however, to have a grammar of a reasonable size, we need to adopt some restrictions. In particular we allows $k-\mathsf{div}$ rules only with k prime number, up to a Other $k-split$ can be obtained by sequential splits by the prime-number factors of k, *e.g.,* to represent a 6-tuplet, we split by 2 and 3. This choice is common in tree representations of rhythm (*e.g.,* [46, 52, 67]).

In the following, we propose three different families of unweighted acyclic grammars acting as input to the construction of the training set.

Grammar 1 This grammar contains rules that encode intervals of equal size in the same non terminal symbol, not considering their positions in the measure. For example, $[0, \frac{1}{2}[$ and $[\frac{1}{2}, 1[$ are represented by the same non terminal $q_{\frac{1}{2}}$. This grammar allows to select possible metrical divisions at each level, while being very small in size.

Grammar 2 This grammar contains a large set of non-terminal symbols that can distinguish intervals according to both duration and temporal position. For example, a $3-\mathsf{div}$ rule $\rho = q_1 \to \langle q_2, q_3, q_4\rangle$

produces three different symbols that can be treated separately to encode very specific behaviors, *e.g.,* allowing grace notes only on q_2.

Grammar 3 The third grammar we propose has the goal of reducing the size of Grammar 2 while retaining most of its versatility. Considering a $k-$div rule, this grammar uses the same symbol for all non-terminals except for the first. For example, a $3-$div rule $\rho = q_1 \rightarrow \langle q_2, q_3, q_3 \rangle$ produces only two different symbols. The musicological intuition behind this choice is that the first interval is often seen as "stronger" compared to the next, while no well defined musical conventions exists that distinguish the successive intervals from each other. This grammar can easily encode the perception principle "infer stronger beats on longer notes" widely employed by works on the topic of metrical alignment (*e.g.,* [12, 80, 90, 67]), by assigning a higher probability to term rules headed by non terminals on stronger metrical position than to term rules headed by non terminal on weaker metrical positions (Grammar 2 can encode this behavior as well, but using a bigger size grammar). For example, considering the rule $\rho = q_1 \rightarrow \langle q_2, q_3, q_3 \rangle$, the product of all term probabilities on leaves headed by q_2 will be grater than the product of all term probabilities headed by q_3.

Example 11. A possible implementation of Grammar 1. The maximum prime for division is $K_{max} = 3$, the maximum depth is $D_{max} = 2$ and the maximum number of grace notes is $gn_{max} = 1$, *i.e.,* terminal symbols are in $[0, 2]$. Rules 1 to 9 are $k-$div rules and from 10 to 24 they are term rules.

$$
\begin{array}{llll}
\rho_1 : q_1 \xrightarrow{0.6} \langle q_{\frac{1}{2}}, q_{\frac{1}{2}} \rangle, & \rho_2 : q_1 \xrightarrow{0.2} \langle q_{\frac{1}{3}}, q_{\frac{1}{3}}, q_{\frac{1}{3}} \rangle, & \rho_3 : q_{\frac{1}{2}} \xrightarrow{0.1} \langle q_{\frac{1}{4}}, q_{\frac{1}{4}} \rangle, & \rho_4 : q_{\frac{1}{2}} \xrightarrow{0.7} \langle q_{\frac{1}{6}}, q_{\frac{1}{6}}, q_{\frac{1}{6}} \rangle \\
\rho_5 : q_{\frac{1}{3}} \xrightarrow{0.6} \langle q_{\frac{1}{6}}, q_{\frac{1}{6}} \rangle, & \rho_6 : q_{\frac{1}{3}} \xrightarrow{0.3} \langle q_{\frac{1}{9}}, q_{\frac{1}{9}}, q_{\frac{1}{9}} \rangle & & \\
\rho_7 : q_1 \xrightarrow{0.05} 0, & \rho_8 : q_1 \xrightarrow{0.1} 1, & \rho_9 : q_1 \xrightarrow{0.05} 2, & \rho_{10} : q_{\frac{1}{2}} \xrightarrow{0} 0, \\
\rho_{11} : q_{\frac{1}{2}} \xrightarrow{0.1} 1, & \rho_{12} : q_{\frac{1}{2}} \xrightarrow{0.1} 2, & \rho_{13} : q_{\frac{1}{3}} \xrightarrow{0.05} 0, & \rho_{14} : q_{\frac{1}{3}} \xrightarrow{0.05} 1, \\
\rho_{15} : q_{\frac{1}{3}} \xrightarrow{0.0} 2, & \rho_{16} : q_{\frac{1}{4}} \xrightarrow{0.1} 0, & \rho_{17} : q_{\frac{1}{4}} \xrightarrow{0.8} 1, & \rho_{18} : q_{\frac{1}{4}} \xrightarrow{0.1} 2 \\
\rho_{19} : q_{\frac{1}{6}} \xrightarrow{0.3} 0, & \rho_{20} : q_{\frac{1}{6}} \xrightarrow{0.7} 1, & \rho_{21} : q_{\frac{1}{6}} \xrightarrow{0} 2, & \rho_{22} : q_{\frac{1}{9}} \xrightarrow{0.5} 0, \\
\rho_{23} : q_{\frac{1}{9}} \xrightarrow{0.3} 1, & \rho_{24} : q_{\frac{1}{9}} \xrightarrow{0.2} 2 & &
\end{array}
$$

An informal notational rule about metrical level divisions like: "prefer to divide in 2 parts at measure level and subsequently in 3 parts" (*e.g.,* for the time signature $\frac{6}{8}$) can be easily represented with this grammar setting $weight(\rho_1) \geq weight(\rho_2)$ and $weight(\rho_4) \geq weight(\rho_3)$. $\diamond$

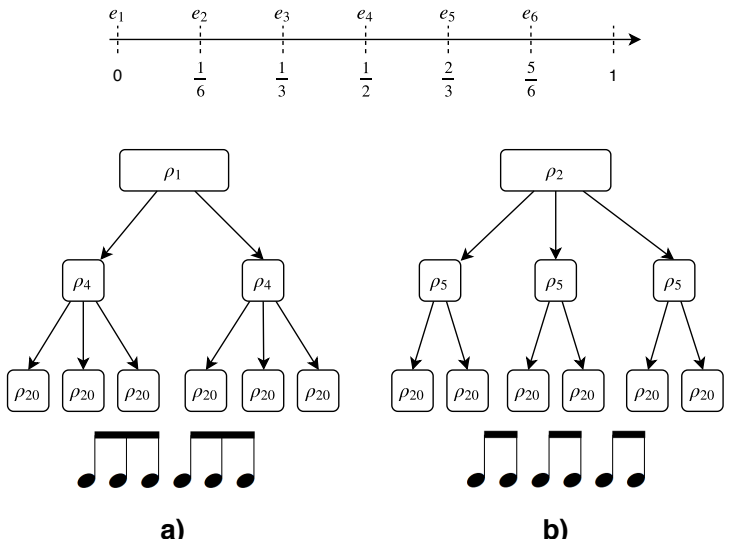


Figure 3.5: Two parse trees from the grammar of Example 11 yielding the same timeline. Since both have a minimal number of leaves 6, the function *rep* is undefined and cannot choose a representative between them.

3.4.3 Computation of Grammar's Weights

Let C be our training set of parse trees. We compute the weight values for the rules of an unweighted grammar $\mathcal{G}$ with a *Maximum Likelihood Estimator* [91]. For rule $\rho = q \to \alpha$ in R, the weight value is defined as

$$weight(\rho) = \frac{C_{\mathcal{T}}(\rho)}{\sum_{q \to \beta \in R} C_{\mathcal{T}}(q \to \beta)} \tag{3.8}$$

where $C_{\mathcal{T}}(\rho)$ is the number of occurrences of ρ in the trees of $\mathcal{T}$.

One can check that the grammar $\mathcal{G}'$ defined from $\mathcal{G}$ with these weight values is a PCFG (see [91]).

Undefined tree for a timeline

Given a timeline $\mathcal{I} \in \mathcal{D}$, it may happen that its representative is undefined because there is more than one parse tree yielding $\mathcal{I}$ with minimum number of leaves, see the example in Figure 3.5. In this case, it is not possible to choose a unique representative, and we will initially discard such $\mathcal{I}$. However, to reduce the loss of information from those timeline, we propose the following two step procedure:

1. compute the training set $\mathcal{T}$ as we have proposed (see Section 3.4.1, page 75), using only the

timelines of $\mathcal{D}$ with a defined (unique) representative, and define weight values for $\mathcal{G}$ from $\mathcal{T}$ as above, resulting in a PCFG $\mathcal{G}'$.

2. for every timeline $\mathcal{I} \in \mathcal{D}$ discarded at step 1, compute a representative t which is a parse tree of $\mathcal{G}'$ build with a modification of the function in Equation 3.7 (page 75) where the min w.r.t. the number of leaves of t is replaced by the max of $weight(t)$ computed with the weights of $\mathcal{G}'$, *i.e.,* build an unique tree for each timeline, using the probabilities learn at step 1 to choose when in doubt.

3. Compute new weight values with these representatives, resulting in a new PCFG $\mathcal{G}''$.

3.5 Implementation and Experiments

The M2S algorithms of Section 3.3.2 have been implemented in C++. The computation of "minimal trees" for the grammar learning is also computed in C++, while the dataset parsing and grammar learning step are performed in Python. The sources, examples and pretrained grammars are available online.[4]

Regarding the M2S algorithms, two optimisations permitted to gain efficiency while searching solutions in a large space. First, instead of fully computing the augmented grammar $\mathcal{K}$ of Section 3.3.2 (their set of NTs would be extremely large as they need to represent the time deviations between the input timeline and the serialization of all possible trees), we performed an online construction of the rules of K during the parsing process, and generalized the algorithm of Section 3.3.1 to compute 1-bests in this scenario. Second, the representation of sequences of bars as binary trees (instead of tuples), like in Example 6, permits to process successive bars monotonically (online), in a greedy way, without having to compute an exponential number of bar sequences.

Regarding the learning process, we implemented the search of the minimal *rep* with Dynamic Programming through a tabulation procedure similar to the 1-best parsing algorithm for weighted grammars [92].

For the experiments we built grammars of type 1 and 3, setting the maximum depth as $D_{max} = 3$ and the maximum number of grace notes as $gn_{max} = 2$. The maximum prime for division K_{max} was

[4]See `https://qparse.gitlabpages.inria.fr` for the M2S algorithms and `github.com/fosfrancesco/Modeling-Rhythm-Structure` for the grammar learning algorithms.

	2/X	3/X	4/X	6/X	9/X	12/X
Lamarque dataset	69	55	45	38	6	0
GMMT dataset	75	91	92	34	3	4
music21 dataset	4233	2180	2887	2268	175	6

Table 3.1: The distribution of the musical scores in the three considered datasets, according to the time signature.

set as 3 for the first level and 7 for the others.

We considered three datasets for the grammar learning process. The first two, the *GMMT dataset* and the *Lamarque dataset* consists of monophonic extracts from the classical repertoire coming respectively from the GTTM database,[5] and from a progressive textbook for learning rhythmic reading [93], that we transformed in MusicXML format, through an OMR system and manual corrections. The third dataset was built from the music21 corpus[6]: a miscellaneous collection of pieces with different genre. The pieces in this corpus are not monophonic, so we extracted the voices and considered each of them as a monophonic part. From those datasets, we kept only the pieces with an unique time signature. Their distribution is detailed in Table 3.1. We learned two grammars (type 2 and 3) for time signatures with numerator $2, 3, 4, 6, 9$. We excluded the pieces with time signature $12/X$ due to the small amount of training data.

An systematic evaluation of our M2S approach is tricky to perform: similar methods in the literature that perform both quantization and metrical alignment (*e.g.,* [46, 52]) are quite old and the code is not available. Moreover a metric that allows to evaluate the differences between the musical content of two scores is also something that still have to be developed (some proposals have been made (*e.g.,* [69, 94] but they consider graphical elements as well). To evaluate our approach we propose a visual comparison between our quantized scores and those obtained via the import function of the commercial softwares Finale and MuseScore (we set the parameters of their input function in order to emulate the same grid that was encoded in our grammar). Note that our algorithms produce RTs, not music notation, but we deployed a simple engraver to produce a musical score, since the visual comparison of trees is not very practical. We considered extracts from the Lamarque dataset that have a length of about 15 bars on average, different time signatures, and contain rhythmic notations of various style and complexity. Performances were recorded based on those scores with a MIDI keyboard by several

[5] Available at `gttm.jp/gttm/database/`

[6] Available at `web.mit.edu/music21/doc/about/referenceCorpus.html`

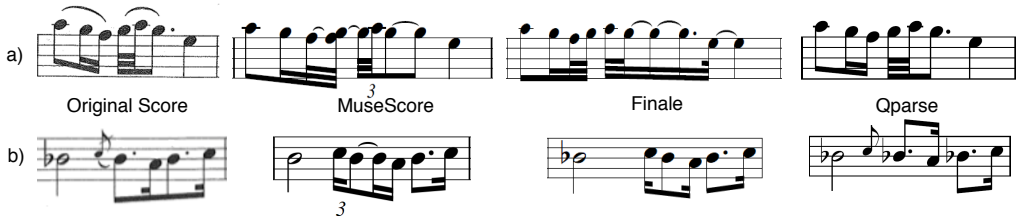


Figure 3.6: Two extracts of transcription experiments; the output of our framework is compared with the output of commercial softwares.

players of different levels. We considered only the simplified scenario where the time signature and the length of the measure are given.

These experiments gave promising results, especially in cases that are traditionally complex to handle (see Figure 3.6), *e.g.,* complex rhythms with mixed tuplets or alternations between short and long notes, grace notes, *etc.* The grammar 3 yielded the best results in almost all the cases. The transcription time for each extract was below 100ms. The complete scores for comparison are available at `qparse.gitlabpages.inria.fr/docs/examples/`.

3.6 Conclusion

In this chapter we presented a system that jointly performs score quantization and metrical alignment of a monophonic MIDI performance. Our approach is based on a context free grammar, that define a set of possible rhythmical divisions with an easy musical interpretation. The output is the score musical content that we defined in Section 2.2.1. We detailed a parsing algorithm that compute the best solution to our problem with respect to two criteria: the similarity with the input performance and the complexity of the rhythmical divisions in the output.

We proposed three possible sets of grammar rules with different advantages and disadvantages and a technique to train the grammar weights from a dataset of musical scores or quantized monophonic MIDI. We tested the training procedure on three dataset: two composed of small excerpts of classical music and one miscellaneous collection of scores of different musical styles.

Our system works "online", with a left to right parsing, measure by measure. This makes it less adapted to target challenges such as big tempo variations, but more suited to other transcription scenarios such as interactive systems where the user wants an immediate feedback.

The system was evaluated through the visual comparison of the scores produced by our system with the results of the input function of Finale and Musescore, in the simplified scenario where the time signature and measure length is given. We described an approach to time signature estimation and the handling of tempo variation, but we leave the evaluation of those elements as future work. A more systematic evaluation would require the development of a metric to compute the differences between the RTs we produce and the musical content of the target scores. We leave this for future work, along with a complete evaluation of the tempo tracking scenario we sketched in Section 3.3.3.

Other extensions of our approach, such as the handling of polyphonic performances, are also under development, with a technique similar to [58] where the grammar encode also voice information.

Chapter 4

Computing differences between musical scores

In this chapter we present a system which relies on our engraving model of Section 2.2.2 (page 44) to compute the list of differences between two musical scores. Those differences are displayed in a graphical tool that represents the two scores side-by-side, highlighting the notation differences between them.

Similar systems that target text have been around for more than forty years (Unix diff [95],[96]) and are widely applied nowadays, for software development, collaborative edition and version control systems. Their purpose is to identify differences between two text files relatively similar (typically two versions of the same file), at the granularity level of lines with the *Longest Common Subsequence* (LCS) algorithm [4], or at the granularity level of characters, with *edit distances*. They output a list of differences, called *patch* or *edit script*, between the two files that corresponds to our intuitive notion of difference.

Comparing musical scores is relevant for several applications: at a global level, it helps to define metrics capable of grasping the *distance* between two scores, for example to evaluate the quality of a full AMT system. At a detailed level, it is valuable for musicologists and developers of version control systems to get precise clues on the *locations* of the differences between scores (*e.g.,* between two editions of the same score).

One difficulty that immediately arises for defining a *diff* tool for musical scores is that, as presented in Chapter 2, a score contains multiple levels that can be compared: the musical content and the engraving level (see Figure 4.1). Most of the literature focuses on the musical content, or even on

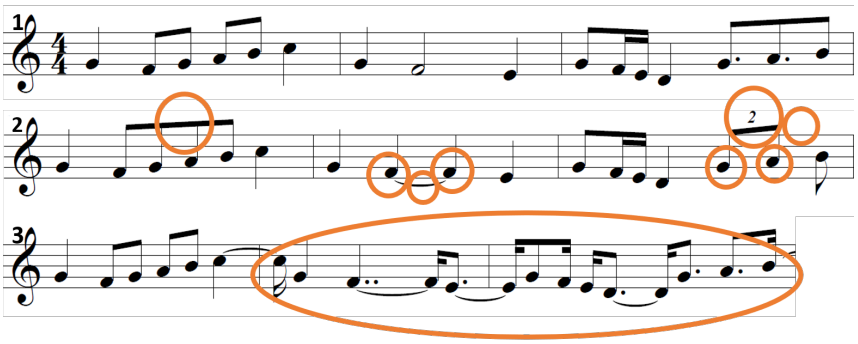


Figure 4.1: Three staves showing different possible comparisons of simple monophonic excerpts. The staves (1,2) have the same music content but differ in their engraving (orange circles highlight those differences). The staves (1,3) differ only in one note (a longer C at the end of the first bar); if we consider a sequential note representation this is a rather small difference, but both the musical content and the engraving (in the orange circle) change considerably.

sequential representations without any metrical information, and the comparison is made with some kind of *edit distance* [39, 97, 98, 94, 99], or employing tree models [50, 51]. In this chapter, instead, we focus on the engraving level; in particular we consider 7 graphical elements: note-heads (and rest type), beams, tuplets, ties, dots, note position and accidentals.

Antila et al. [100], following the work of Knopke and Byrd [101], observe that it does not make sense to apply directly the standard Unix `diff` utility to XML score files. The problem is mainly a disambiguation problem, since the same score can be encoded in multiple different XML scores, *e.g.,* by interchanging the order of notes in a chord. To overcome this problem, we will compare our engraving model of the score, by using an original dedicated *tree-edit distance* based on tree nodes operations [102, 103]. The edit distance algorithm yields a list of differences between two scores, considering first the bar level, and then a finer granularity level, similarly to text diff tools that first find the different lines and then explore the differences between the words of each line. The resulting distance is the size of the smallest list of modifications required to transform one score into the other, and it reflects visual differences between two scores.

Our graphical tool represents two scores side-by-side and highlights the notation differences between the two scores, with coloured annotations, similarly to a visual diff tool for text. While it is currently based on our algorithm only, it may be extended to present other metrics and should also be easy to integrate into other programs.

The remainder of this chapter starts with a review of related works. Then we present our two

main contributions: a new set of algorithms that yields a list of differences (in Section 4.2) and the implementation of a graphical tool to compare two scores side-by-side (in Section 4.3). Finally, Section 4.4 contains experiments on the algorithms and the visualization tool.

The content of this chapter has been partially published in [Foscarin et al., 2019a].

4.1 Related works

Our aim is to create for music notation a tool similar to the `diff` program by Douglas McIlroy [4], which compares texts. Besides its practical usage, it has motivated theoretical studies which led to efficient algorithms for edit distances computation (*e.g.,* [104, 105]). We draw from this work the process of computing the differences with dynamic programming, as introduced in Section 1.1.4 (page 24).

The problem of comparing musical scores has been studied in previous works with different objectives, such as collaborative score editing, and the evaluation of optical music recognition (OMR) and automatic music transcription (AMT) systems. Knopke and Byrd [101] pioneered with their "musicdiff" program, focusing on comparing several OMR outputs. They show that a traditional XML `diff` tool can't be used *as is* on MusicXML: a *normalization* step is required to avoid the problem of having the same music notated in two different xml structures, *e.g.,* the same notes in a chord, but encoded in a different order. We employ a similar workflow, but our objective is different since we focus only on the graphical content of the score, not on its XML encoding. To reach this goal we consider as *normalized* representation of the score, our engraving model.

Other works compare two scores, or part of them, without focusing on obtaining an alignment or a list of differences, but rather producing a similarity index that can be used for automatic tasks such as querying a database (*e.g.,* [97, 98, 50]), evaluating the performances of a transcription technique (*e.g.,* [94, 99]) or plagiarism detection (*e.g.,* [50]).

The similar problem of comparing MIDI files is also studied in MIR. In [106, 107, 108, 109] the authors address the problem of computing an alignment between two MIDI performances or between a score and a MIDI performance. The differences between the two files to compare are caused by performance mistakes, *e.g.,* wrong pitch, notes missing or notes inversion. In [107] a tool to graphically visualize the alignment between two piano-roll representations of the MIDI files is also presented.

Aiming at improving collaborative editing of scores, Antila et al.[100] introduce the idea of comparing hierarchical representations (that we will extensively use), by employing the Zhang-Shasha tree-edit distance [102]. They bring advances from theoretical computer science (*e.g.,* [110]) into the music notation community and show, on a MEI-encoded file, that a diff tool that work directly on XML musical scores suffers from the notation ambiguity problems identified by Knopke et al. [101]. To overcome this issue we will employ our engraving model.

Our last contribution, the graphical tool to visualize differences between scores, is inspired by the MIDI comparison tool of Nakamura et al. [107] and by similar tools for texts. The latter are now ubiquitous, either being standalone dedicated programs like Meld[1] (cross-platform) or FileMerge/opendiff (MacOS), or integrated into IDEs (Eclipse, IntelliJ). To the best of our knowledge, our work is the first proposition in the context of musical scores.

4.2 Diff Algorithms

In this section we present the algorithms that are used to compute the differences between the engraving parts of two musical scores.

As presented in Chapter 2, we model the score as a nested structure: on the top level we have a list of parts (usually one for each instrument), each part consists in a list of voices and each voice is a list of beaming and tuplet trees (one BT and one TT for each bar). For the sake of presentation, we will initially focus on scores with a single part and a single voice. We will explain later in Section 4.2.4 how our technique generalizes to polyphonic scores with multiple voices and parts.

Exploiting the tree representation of the score engraving, we proceed in two stages:

1. *comparison at the bar level* with an alignment of identical bars (*i.e.,* bars that have the same trees), with a Longest-Common-Subsequence (LCS) algorithm [4].

2. *comparison inside-the-bar* a more in-depth processing of unaligned bars, using purposely designed tree-edit distances algorithms that explore bar representations in order to identify fine-grained differences.

[1] http://meldmerge.org/

4.2.1 Longest Common Subsequence of Parts

Let's assume that we are given two voices extracted from two different scores. According to our score model, we represent each voice (at the engraving level) by a sequence of pairs $p = \langle bt, tt \rangle$, one pair per bar, where bt is a beaming tree and tt is a companion tuplet tree.

Similarly to the line-by-line comparison procedures for text files [95], and by considering a bar as the analogue of a line in a text file, we align identical bars from the two voices, by computing their longest common subsequence (LCS). Its size is defined by the following recursive Dynamic Programming equations (the notation is presented in Section 1.1.4, page 24):

$$\begin{array}{lcl} LCS(\varepsilon, s') & = & 0 \\ LCS(s, \varepsilon) & = & 0 \\ LCS(p.s, p'.s') & = & \begin{cases} 1 + LCS(s, s') & \text{if } p \equiv p' \\ \max(LCS(p.s, s'), LCS(s, p'.s')) & \text{otherwise.} \end{cases} \end{array} \tag{4.1}$$

In the third line, $p \equiv p'$ means that the trees in pairs $p = \langle bt, tt \rangle$ and $p' = \langle bt', tt' \rangle$ are identical, *i.e.,* that $bt = bt'$ and $tt = tt'$. According to our model the corresponding bars can be considered identical.

The above equation can be adapted in order to compute a maximal *alignment* of identical pairs between the two sequences. Formally, denoting the two sequences by $s = p_1, \ldots, p_m$ and $s' = p'_1, \ldots, p'_n$, the alignment is made of two strictly increasing sequences of indices of same length, $(i_0 = 0, i_1, \ldots, i_k)$ and $(i'_0 = 0, i_1, \ldots, i'_k)$ such that for all $0 \leq j < k$,

(i) $p_{i_{j+1}} \equiv p'_{i'_{j+1}}$, and

(ii) the pairs in the respective sub-sequences $b_j = p_{i_j+1}, \ldots, p_{i_{j+1}-1}$ and $b'_j = p'_{i_j+1}, \ldots, p'_{i_{j+1}-1}$ are pairwise distinct (each of these sequences might be empty if $i_{j+1} = i_j + 1$ or $i'_{j+1} = i'_j + 1$).

We call each pair $\langle b_j, b'_j \rangle$ as above a *difference block* (see Figure 4.2) and the objective of the next two subsections is to perform a comparison between blocks.

4.2.2 Inside the bar comparison

In the following, for a tree t (either a beaming tree or tuplet tree), $\|t\|$ denotes the sum of the *sizes* of all the nodes a in t. The size of a node a, denoted as $\|a\|$, is 0 for unlabelled internal nodes, 1 for

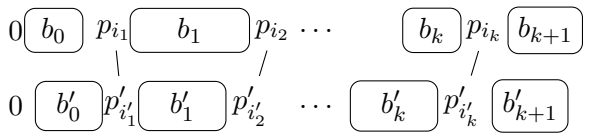


Figure 4.2: LCS and difference blocks.

labeled internal nodes and for leaves it corresponds to the number of graphical symbols they contain. Specifically, given a leaf a, its size $\|a\|$ is a natural number ≥ 0 that is computed by summing the number of: note-heads, rests, ties, dots and alterations. For example a chord with two notes [D4,F4♯] will have size 3 and a note [G3] with 1 dot will have size 2.

For a sequence $s = t_1, \ldots, t_k$ of trees, $\|s\| = \sum_{i=1}^{k} \|t_i\|$.

Comparison of Beaming Trees

The following recursive equations define an edit distance (denoted by B) between sequences of beaming trees. The distance between two trees is defined by the particular case of singleton sequences. In these equations, $t.s$ and $t'.s'$ denote two beaming tree sequences, with $t = a(s_0)$ and $t' = a'(s'_0)$, where s_0 and s'_0 might be empty.

We call a tree *atomic* if it is composed of a single leaf node (*i.e.,* it represents an event). We call a tree $t = a(s_0)$ *unary* if s_0 is a singleton sequence (*i.e.,* the root has a single child).

We define the distance B between two sequences of beaming trees as:

$$
\begin{array}{lcl}
B(\varepsilon, s') & = & \|s'\| \\
B(s, \varepsilon) & = & \|s\| \\
B(t.s, t'.s') & = & \min \left\{ \begin{array}{llll}
B(s, t'.s') & + \|t\| & \text{(del-tree)} & \\
B(t.s, s') & + \|t'\| & \text{(ins-tree)} & \\
B(s_0.s, t'.s') & + 1 & \text{(del-node)} & \\
\multicolumn{4}{l}{\text{if } t \text{ non-atomic}, t' \text{ atomic, or } t \text{ unary}, t' \text{ not unary}} \\
B(t.s, s'_0.s') & + 1 & \text{(ins-node)} & \\
\multicolumn{4}{l}{\text{if } t \text{ atomic}, t' \text{ non-atomic, or } t \text{ not unary}, t' \text{ unary}} \\
B(s_0.s, s'_0.s') & + \delta(a, a') & \text{(descend)} & \\
 & & \text{if } t, t' \text{ non-atomic} & \\
B(s, s') & + A(t, t') & \text{(leaf)} & \\
 & & \text{if } t, t' \text{ atomic} &
\end{array} \right.
\end{array}
\tag{4.2}
$$

where $\delta(a, a')$ is a defined by $\delta(a, a') = 0$ if $a = a'$, and $\delta(a, a') = 1$ otherwise.

The cases (del-tree) and (ins-tree) correspond to the deletion or insertion of a whole subtree. The

cases (del-node) and (ins-node) with one unary tree correspond to a difference of one beam between the trees. The other cases descend down to leaves and then compare atomic trees t and t' with the distance $A(t, t')$ defined as follows:

$$A(t, t') = D(p(t), p(t')) + \delta(n(t), n(t')) + |d(t) - d(t')| \tag{4.3}$$

where p, n, d stand respectively for the attributes *pitches*, *note-heads*, *dots* detailed in Section 2.2.2 (page 44), and the first literal $D(p(t), p(t'))$ is the Levenshtein distance defined in Section 1.1.4 (page 24), using the following operations costs:

$$\begin{array}{rcl} \delta(\varepsilon \to pitch) & = & \delta(pitch \to \varepsilon) = \|pitch\| \\ \delta(pitch \to pitch') & = & \delta(n, n') + \delta(alt, alt') + \delta(tie, tie') \end{array} \tag{4.4}$$

where n, n', alt, alt' and tie, tie' are the respective name, alteration and tie values of $pitch$ and $pitch'$.

Comparison of Tuplet Trees

Tuplet trees have a simpler structure than beaming trees (*e.g.,* no unary nodes). The algorithm for beaming trees presented in the previous section is thus not needed in this case. We compute the distance between tuplet trees (denoted by T) with the (less costly) Zhang-Sasha tree edit distance algorithm [102]:

$$\begin{array}{rcl} T(\varepsilon, s') & = & \|s'\| \\ T(s, \varepsilon) & = & \|s\| \\ T(t.s, t'.s') & = & \min \left\{ \begin{array}{lll} T(s_0.s, t'.s') + 1 & & \text{(del-node)} \\ T(t.s, s'_0.s') + 1 & & \text{(ins-node)} \\ T(s, s') + T(s_0, s'_0) + \delta(a, a') & & \text{(subst-node)} \end{array} \right. \end{array} \tag{4.5}$$

where for inner nodes $\delta(a, a') = 0$ if $a = a'$, and $\delta(a, a') = 1$ otherwise. For leaf labels $\delta(a, a') = A(a, a')$ as defined for beaming trees in Equation 4.3.

4.2.3 Comparison of different blocks

We recall that a block is a sequence of pairs (bar representations) of the form $p = \langle bt, tt \rangle$, where bt and tt are respectively a beaming tree and a tuplet tree. We compute the distance between two difference blocks b, b' using the following recursive equations, similar to the Levenshtein distance of

Section 1.1.4 (page 24).

$$\begin{array}{lcl} D(\varepsilon, b') & = & \|b'\| \\ D(b, \varepsilon) & = & \|b\| \\ D(p.b, p'.b') & = & \min \left\{ \begin{array}{llll} D(b,b') & + & \Delta(p,p') & \textsf{(edit-bar)} \\ D(p.b,b') & + & \|p'\| & \textsf{(ins-bar)} \\ D(b,p'.b') & + & \|p\| & \textsf{(del-bar)} \end{array} \right. \end{array} \tag{4.6}$$

where, for $p = \langle bt, tt \rangle$ and $p' = \langle bt', tt' \rangle$:

$$\Delta(p, p') = B(bt, bt') + T(tt, tt') - corr(p, p') \tag{4.7}$$

is the edit distance between two bar representations; $corr(p, p')$ is a correction of $T(tt, tt')$ to avoid to count twice the same differences in the leaves (more details are given later). Finally, the distance between two parts is the sum of differences between all the blocks.

4.2.4 Multi-voice and multi-parts extension

Recall that we consider a model of scores made of a list of *parts* (usually one part for each instrument), where each bar is a list of *voices*, and each voice is a list of trees (one TT and one BT for each bar).

The algorithms described in the previous sections deal with the comparison of single-part and single-voice scores. The generalization to polyphonic scores with multiple-voices and multiple-parts is implemented as follows.

- **multiple-voices:** when comparing two bars we consider all pairs made of one voice of the first bar and one voice of the second one, we run the algorithm of Sections 4.2.2 and 4.2.3 on each pair independently and then take the pairs that yield with minimal distance. If the number of voices in the two bars is different, we return the corresponding voice-insertion and voice-deletion.

- **multiple-parts:** we first map the parts in the first score into parts of the second one using the metadata like part or instrument names. We then run our algorithms on each pair of mapped parts independently. In case of parts of score 1 not present in score 2 (or vice-versa), we return the eventual part-insertion and part-deletion annotations. An alternative would be to apply a procedure similar to the above one for multiple voices (coupling parts and taking the ones with smallest distance) but it would be very costly. Indeed, the number of parts can be high (*e.g.,*

in the case of orchestral scores), with respect to the number of voices in a bar (typically 2 or 3 voices) and the comparison of 2 parts requires to compare all the data inside these parts (bars, voices *etc*).

4.3 A graphical tool to display differences

We implemented all algorithms previously described and produced a graphical tool that displays the differences between two digitized scores in MusicXML or MEI format.

4.3.1 Algorithms implementation

The implementation of the proposed algorithms is performed in Python3. The string edit distances defined in Sections 4.2.1 and 4.2.3 (pages 87 and 89) are performed iteratively and run in time $O(m*n)$, where m, n are the length of the two sequences. The tree-edit distances of Section 4.2.2 (page 87) are implemented by recursive functions, with a worse case time complexity in $O(n^4)$. We use a table of solutions in order to avoid to recompute the function on the same inputs and hence reduce the computation time. Nevertheless, efficiency is not a crucial issue in this case, because the difference blocks are typically small (*e.g.,* in OMR evaluation), or such that $b = \varepsilon$ or $b' = \varepsilon$ (insertion or deletion of bars).

A difference in our score is modeled as a triple $(operation, t, t')$ where *operation* is a string identifying the performed operation (e.g. ins-node), and t, t' are the subtrees transformed by that operation (or ε is case of insertion/deletion). In the implementation, the recursive equations defined above are extended in order to compute, beside the edit distance, a list of differences. It means that the functions corresponding to B, T *etc.* return pairs made of a cost value along with a *diff list*. To that respect, the operator $+$ in the recursive equations makes the sum of the cost values and the concatenation of diff lists, and the *min* selects the pair with the smallest cost value. The function $corr(p, p')$ in Equation 4.7 (page 90) uses the difference lists returned by B and T, in order to detect intersection that may occur, in particular for leaves. These intersections are removed (operator "$-$" in Equation 4.7) both from the cost value and the diff list to avoid to account twice for the same difference (*e.g.,* a pitch modification must be counted only once, even if it is present in the result of both B and T).

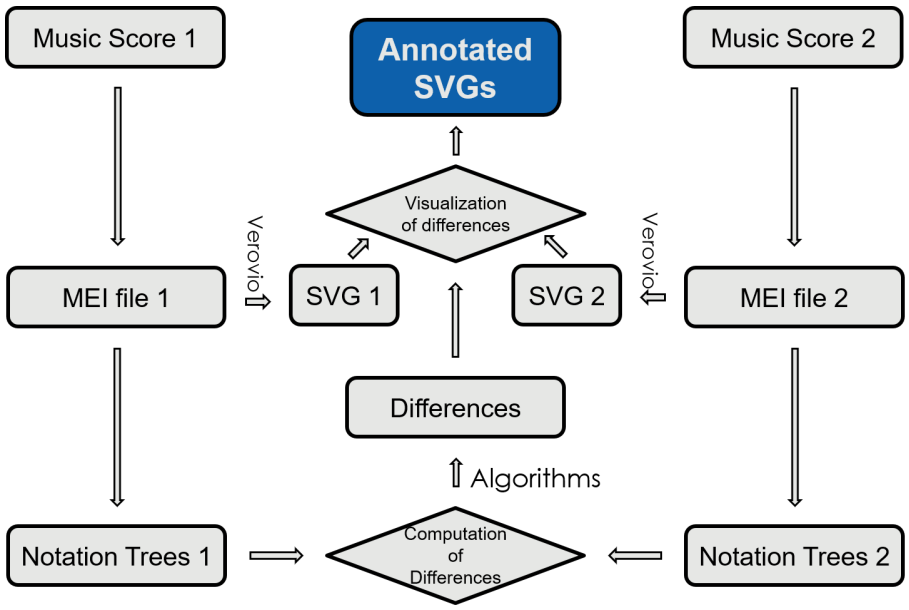


Figure 4.3: Complete process for the computation and visualization of the differences between two scores.

4.3.2 Data preparation and differences visualization

The extraction of the engraving model from the scores is performed in Python, using the Music21 toolkit [111] to parse the XML file. To display the differences on the scores, we need a direct way to point to score elements. This is provided in the MEI format in the form of an unique `id` attached to each element. The XML ids are not present in the MusicXML format, thus we transform MusicXML scores in input in MEI as a preprocessing operation. We store those ids in the leaves of our tree representation in the engraving model.

The elements of *diff lists* are pairs of ids in the two scores, plus an identifier of the edit operation. After computing the edit distances and list of differences between two given scores, we display, using Verovio [112], the two scores side-by-side, highlighting the differences. The completed workflow is displayed in Figure 4.3.

4.4 Experiments

We ran experiments in order to test the usability of our workflow for two scenarios: collaborative edition of scores, and OMR evaluation. For the first use case, we considered made-up scores with some

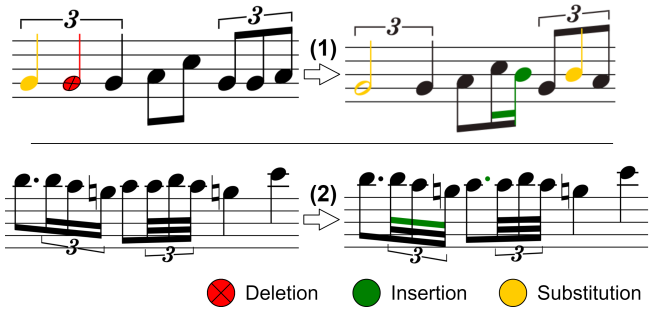


Figure 4.4: Visualization of differences between two bars. The upper example (1) is artificially created to emulate different phases of a composition process, while the lower example is a real-life example from an incorrect OMR score import, compared to the correct score.

typical modifications (bar insertions, note pitch modifications, *etc.*) in a context of versioning. For the second scenario, we used a corpus produced by the Bibliothèque nationale de France (BNF) composed of 21 *ouvertures* of Jean-Philippe Rameau, each with an *OMR*ized version (from manuscripts using PhotoScore[2]) and its manual correction, that we compare with our tool (Figure 4.5)[3].

Our algorithms showed promising results in both cases, by highlighting very fine-grained differences in the scores (*e.g.,* beams and dots), as illustrated on Figure 4.4. The evaluation of OMR results were not possible for many scores, due to quality issues in the XML files, *e.g.,* a tie between two non-consecutive notes, or syntax errors in the XML file, such as an opened beam not closed. It should be noted that our tree-based model can only handle correctly encoded scores, and that such *faulty* notations will result in an error.

4.5 Conclusion

In this chapter we presented a system that compare XML scores at the engraving level, exploiting the hierarchical organization of our score model. In contrast with most approaches that only compute a similarity measure, our methodology produces a list of differences that can be highlighted directly on the scores, and alleviates the user from the tedious task of manual scores comparison. Experiments show that our approach can correctly find and display very fine differences in the scores. Technical improvements are still needed in other stages (XML parsing, error detection and correction, complex score visualization) to improve usability.

[2]https://www.neuratron.com/photoscore.htm
[3]see our supplementary material page at `https://anonymoususer12042019.github.io/` for details.

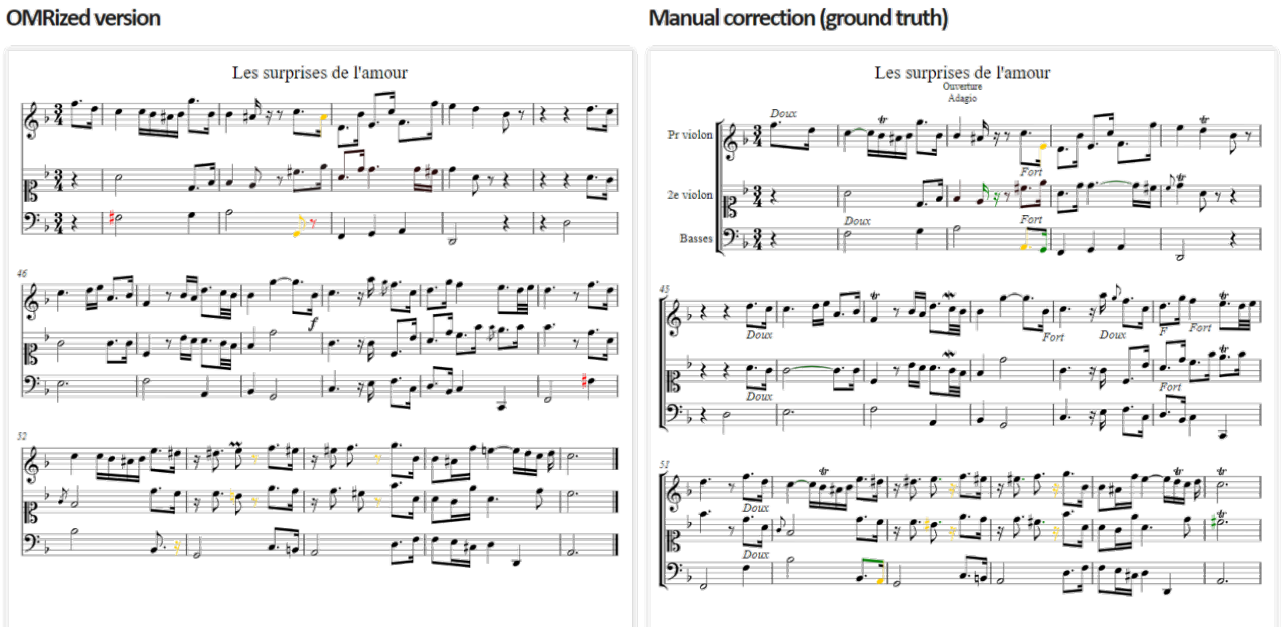


Figure 4.5: Visualization of differences between the OMRized (faulty) version of a musical score and the correct (manually edited) score.

Our algorithms also yield a similarity metric, but it is only used to compute the list of differences and was not designed to be used directly for other tasks. We leave for future work an in-depths evaluation of this metric for automatic system. In our proposed AMT workflow of Chapter 2, we don't suggest to perform the evaluation of AMT frameworks at graphical level, but some systems would still require such metric, such as holistic approaches that produce directly a musical score or optical music recognition systems.

We also leave for future work the treatment of notation symbols that are not considered in the actual implementation, such as stem directions, clefs, slurs, expression markings, ornaments and lyrics. We also plan to investigate on a more complete treatment of multi instruments polyphonic scores, that will correctly handle staves swapping and multi-staves voices.

Chapter 5

The Aligned Scores and Performances Dataset

In this chapter we introduce the *Aligned Scores and Performances* (ASAP) dataset: a large collection of digital musical scores of Western classical piano musical scores (in MusicXML and MIDI format), aligned at the beat level with audio and MIDI performances. The dataset is designed to target the full AMT problem, and the several sub-tasks that compose it (*e.g.,* beat and downbeat tracking, tempo estimation, metrical structure alignment, and rhythm quantization) from both audio and MIDI.

As data-hungry deep learning models have become more ubiquitous in the field of MIR in recent years [113, 114, 115], large, well-annotated dataset have become increasingly important. Similar trends towards deep learning methods have been seen in related fields such as natural language processing [116] and computer vision [117]. For many tasks in these fields, large datasets can be automatically scraped from the web, and annotated quickly by non-experts [118]. Unfortunately, the same can often not be said for tasks in MIR, for many reasons.

First, producing high-quality audio or MIDI data is a non-trivial task, requiring expert performers and expensive equipment that is not always available. Second, the availability of a high-quality digital ground truth is not guaranteed in many cases, particularly for tasks which require a musical score[1], e.g., for the complete audio-to-score Automatic Music Transcription (AMT) task (see [1] for a recent overview of AMT). Finally, even in the case when both audio/MIDI data and high-quality digital

[1] Although PDF scores are sometimes available, and the widely-researched field of Optical Music Recognition (OMR—see [119] for a recent overview) involves converting these into digital format, this conversion can add errors, and starting from a clean, digital score is generally better if available.

	Size		Performance		Quantized		Annotations		
Dataset	Total	Unique	Audio	MIDI	MIDI	Score	Alignment	Metrical	Key
MAPS[121]	269	52	Pseudo†	Pseudo	✓		Full	✓[122]	✓[122]
CrestMuse-PEDB[123]	411	≈100	Partial†	✓	✓	✓	Full	✓	✓
MAESTRO[124]	1282	≈430	✓	✓					
GTZAN [125]	1000	1000	✓					✓[126]	Global
Ballroom [127]	685	685	✓					Beat [128]	
Hainsworth [129]	222	222	✓					Beat	
SMC [130]	217	217	✓					Beat	
ASAP	1068	222	520	✓	✓	✓	Beat	Beat	✓

Table 5.1: An overview of the most relevant datasets for AMT (top section) and metrical tasks (middle section), compared to ASAP (bottom). Alignment refers to the level of alignment between the quantized and performance data. MAPS consists of pseudo-live performance (quantized MIDI with manually altered tempo curves). †Both MAPS (fully) and CrestMuse-PEDB (partially) include synthesized audio (not real recordings).

ground truth scores are available, acquiring an alignment between the two is non-trivial. Automatic alignment methods (e.g., [107]) are often not robust enough to deliver highly reliable results and require time-consuming post-processing and cleaning by expert musicians for advanced data usage.

In regards to those the three difficulties (data creation, digital ground truth availability, and recording-ground truth alignment), we (1) use publicly available MIDI files and audio recordings from expert human performances; (2) use publicly-available digital musical scores scraped from the web (in the form of MusicXML and quantized MIDI files); and (3) have designed a workflow to efficiently produce aligned beat, downbeat, key signature, and time signature annotations for music scores and performances with minimal human correction required. In particular, our approach exploits the precise metrical structure information encoded in the musical content of a score and projects this onto each corresponding performance. This guarantees a robust means to identify the beat and downbeat positions that drastically reduce the time required for manual annotation. Working with MIDI allows us to overcome many difficulties found in similar approaches using only audio data (e.g.,[120]).

Our contributions in this chapter, *i.e.,* the dataset itself and the annotation technique, have been published in [Foscarin et al., 2020].

5.1 Similar datasets

Some public datasets similar to ASAP (*i.e.,* containing combinations of musical scores, MIDI performances, and audio recordings, for AMT and/or beat tracking) already exist. In this Section, we describe those existing datasets highlighting specifically where ASAP addresses their deficiencies. Table 5.1 contains a summary of the largest of these datasets in comparison to ASAP. We first describe those containing musical performances (useful for AMT), and follow that with a brief discussion of available datasets for metrical structure-based tasks.

5.1.1 Performance datasets

There are two datasets which contain multiple performances of many different pieces. The Vienna 4x22 Piano Corpus [131] consists of 22 different performances of each of 4 different pieces, in both audio and MIDI format, aligned to a metrical grid. The CHARM Chopin Mazurka Project[2] dataset contains many recordings of each of 49 different Mazurkas composed by Frédéric Chopin, although the original audio recordings are only referenced, and not available online (instead, many pre-calculated features are provided). While these datasets are valuable for investigating live performance deviations and comparisons between different performances of the same piece, they are not as useful for AMT, since they each consist of a small number of different pieces, leading to likely model overfitting (only 4 different pieces for Vienna 4x22, and only pieces by a single composer in the Mazurka dataset).

The Saarland Music Data (SMD) dataset [132] contains 50 synchronized audio and MIDI recordings of human performers playing a Disklavier piano. The files are not aligned with any musical scores or beat annotations, and the dataset's size is somewhat small compared to other similar datasets. Likely because of its size, SMD has has not been used for AMT in recent work to our knowledge.

CrestMuse PEDB [123] is a dataset based on audio recordings of multiple piano performances of around 100 unique pieces. However, the original audio recordings are not included . Rather, references to commercial CDs which can be purchased, and on which the recordings can be found are given. After a PDF application and pledge are filled out and submitted, access is granted to download the database in about a week. Provided in the dataset are MIDI files, whose notes have been hand-aligned to the referenced recordings; and digital musical scores in an XML-based format, to which the notes and beats

[2] `http://www.mazurka.org.uk/`

of the MIDI files are aligned (using "deviation" XML tags in the score files). Since its initial release, some audio files have been added. However, these are different from the original score-aligned audio recordings, and in some cases are synthesized from MIDI performance. The difficulty of acquiring the audio recordings makes this database rarely used for audio-based tasks such as AMT.

The piano-midi dataset[3] contains 324 quantized MIDI files whose tempo curves have been manually altered with the goal of becoming more human-like. The MAPS dataset [121] contains 210 of these MIDI files (of 119 different pieces), without key and time signature information, each paired with an audio file—some synthesized and some actual recordings. A-MAPS [122] later augmented MAPS with MIDI files containing key signature and time signature annotations. Since the MIDI data is generated from tempo-varied quantized MIDI, rather than actual performance, the MIDI files and recordings do not contain all of the timing variance that would be present in real performance: note onsets and offsets which lie on the same beat in the original musical score also occur simultaneously in the corresponding MIDI and audio files. In real performance, such events only rarely occur simultaneously. Rather, small timing deviations introduce gaps, overlaps, and other timing variation (see e.g. [133], Figure 4), which are therefore missing (along with ornamentation such as trills, as well as performance errors). Although these datasets contain perfectly-aligned ground truth annotations (which has made MAPS a standard for AMT evaluation since its release), their modest size and the fact that they are not real live performance are drawbacks that we hope to address with ASAP.

The SUPRA dataset [134] contains 478 MIDI files of around 430 different pieces generated from an archive of piano performances in the form of physical piano rolls. SUPRA also contains synthesized audio recordings of each MIDI file, and labels each with a composer and title, but provides no metrical alignment of the pieces.

The MAESTRO dataset [124] contains 1282 real performances of around 430 different pieces from the Yamaha piano e-competition[4]. Each performance is available as a MIDI file and an audio recording with a fine alignment of around 3 ms. Metadata are available for each performance, including the composer and title of each (although 2 performances of the same piece are sometimes labeled differently). MAESTRO's size, fine alignment with ground truth, and the fact that it is real performance have made it an excellent source of training and evaluation data for AMT from recording to piano-roll.

[3] `www.piano-midi.de`
[4] `http://piano-e-competition.com/`

However, MAESTRO does not contain any note-, beat-, or even piece-level alignment with digital musical scores, required for the complete audio-to-score AMT task (and which ASAP does contain).

5.1.2 Metrical structure datasets

For metrical structure-based tasks, from live performance MIDI data, annotated datasets from the previous section (particularly piano-midi and CrestMuse-PEDB) are typically used. However, they are relatively small (especially in terms of unique pieces), and piano-midi files in particular do not contain real live performance, as mentioned. For the same tasks from audio data, large annotated datasets exist, enabling sophisticated models to be designed and trained (e.g., [135]). The largest and most widely used (where audio files are publicly available, including at least beat annotations) are: GTZAN [125] (1000 audio recordings of various genres with beat, downbeat, and 8th-note annotations), Ballroom [127] (685 audio recordings of ballroom dancing music with beat and downbeat annotations), Hainsworth [129] (222 audio recordings of Western music), and SMC [130] (217 audio recordings of Western music, specifically selected to be difficult for beat tracking). However, the music contained in these datasets tend to have a much steadier tempo than those contained in ASAP (even SMC; see Section 5.3.3 for a comparison).

5.2 Producing music annotations

Annotating a dataset the size of ASAP with ground truth for AMT and related problems such as beat tracking is a time consuming task that can, in principle, only be performed by expert musicians. This severely limits the ease with which one can produce a reliable and finely crafted dataset at the required scale.

For piano music, MIDI and audio performances can be automatically aligned if they are recorded at the same time using an acoustic piano fitted with the proper sensors such as a Disklavier. The main problem then becomes annotating the performances with metrical markings (such as beats and downbeats) and aligning those with a musical score. Holzapfel et al. [130] describe the process used to annotate the SMC dataset in detail, showing how much manual work was required for its beat annotations. ASAP contains more than 92 hours of performance data, and even a skilled musician would need to listen to each multiple times with the proper annotation software in order to annotate

it fully (and this time can increase dramatically for complicated pieces with multiple time and key signature changes). Moreover, as highlighted in [136], the manual annotations would still be affected by human subjectivity, requiring a system with multiple annotators and a reconciliation mechanism between them (e.g., [130, 137]) We believe that without a large budget and/or the ability to involve a community of expert users willing to spend a significant amount of time on the task, producing a large ground truth dataset cannot be achieved through a purely manual approach.

We therefore propose an annotation workflow that allows for the automatic production of annotations from digital musical scores and MIDI performances. The precision of the results is much higher than what would be expected from human-based annotation with a single annotator. Moreover, when the quality of the available scores is high, the workflow does not require any human intervention. Manual intervention is sometimes required to fix problems related to either the digital encoding of the scores or performance errors.

In Section 5.2.1, we describe the basic annotation workflow. Then, in Section 5.2.2, we discuss practical problems—such as poor score encoding, performance errors, and limitations of existing software—that may still require manual correction based on our experience.

5.2.1 Annotation Workflow

The annotation workflow takes a MIDI performance and a MusicXML score as input, and produces beat, downbeat, key signature change and time signature change annotations for each. The annotations are each aligned to a position (in seconds) in both the MIDI performance and a MIDI score (generated automatically from the MusicXML score), and each downbeat is further aligned with a measure number from the MusicXML score.

The annotation workflow (Figure 5.3) is as follows:

1. Expand any repetitions present in the XML score and extract time and key signature changes using music21 [138].
2. Generate the MIDI score using the MuseScore3 MIDI export function.
3. Extract the times of beats, downbeats, and key and time signature changes from the generated MIDI using pretty_midi[139].
4. Align every downbeat from the MIDI score with a measure number in the XML score.

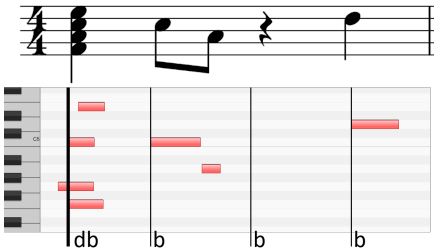


Figure 5.1: Beat and downbeat annotations produced by our workflow in different cases. The 1st (db) is the median of the onsets of the corresponding notes, the 2nd and the 4th (b) are the onset of the corresponding note, and the 3rd (b) is the mean between the two neighbor annotations.

5. Produce a Score2Performance mapping from each note in the MIDI score to each note in the MIDI performance using the algorithm presented in [107].
6. The performance annotations can then be obtained as follows. For each annotation in the MIDI score:
   (a) Take the notes with an onset within 20ms of the annotation (there can be multiple notes, e.g. for the downbeat annotation in Figure 5.1).
   (b) Using the Score2Performance mapping, obtain the onset times of the corresponding notes in the performance file.
   (c) Compute the median of the onset times of those notes, and use it as the time of the annotation in the performance.
   (d) If no notes are within 20ms of the MIDI score annotation (e.g., in case of a rest), the position is interpolated from neighboring annotations (e.g., the 3rd annotation in Figure 5.1)

As shown in [140], annotations on rests (step 6d) or multiple non-simultaneous notes (grace-notes, arpeggios, as in step 6c) are inherently problematic, even for human annotators. An inspection of the results reveals that this workflow produces good results. In particular, our use of the median increases robustness against outliers (e.g., notes that are very delayed or incorrectly aligned by the Score2Performance mapping).

5.2.2 Practical issues with erroneous input

While the mapping between the scores and performances in ASAP is such that they have a very good correspondence in general, localized problems can still occur in specific cases. These can be

caused either by encoding issues in the XML score, or by performance errors in the MIDI.

For our automatic workflow to produce good results, the *musical content* must be correctly encoded in the XML file, although we can ignore problems at the *score engraving* level (we are referring to the model presented in Section 2.2). Many of the problems that we encountered during the automatic creation of ASAP's annotations are tricks used by editors to fix problems at the engraving level at the expense of correctness at the content level: for example, grace notes entered as regular notes with a smaller font, invisible barlines inserted mid-bar, invisible notes or rests, or note heads changed for visual reasons inconsistent with their actual duration.

In some cases, editors are forced to use such tricks because the original score itself does not follow standard notation rules. Figure 5.2 shows two examples of incorrect measure duration: one from Ravel's *Ondine* (top) and another from Mozart's *Fantasie in C minor* (in 4/4; bottom left). There are many ways to handle such cases. Possibilities include having invisible tuplet markings, having a different time signature than the one displayed, and having an overflowing measure. The latter two techniques create problems for automatic beat extraction. Figure 5.2 (bottom right) is an example of a key change in the middle of a bar in Beethoven's *Sonata No.29, Op.106*, 2nd movement. One way to encode this situation is to split the bar into 2 separate bars, but this also creates problems for automatic beat extraction in the form of an extra downbeat.

Fortunately, such problems are generally easy to detect since they often result in a measure where the sum of the events does not match the duration implied by the time signature. To be able to produce correct annotations even in the case of these "faulty" scores, we introduce a manual correction step for the MIDI score annotations (Figure 5.3). This avoids the case of such problems propagating down our workflow to the performance annotations.

The alignment that we use to generate the Score2Performance mapping [107] is robust against small errors and note inversions. Nonetheless, small errors in its alignment exist. As the difference between a performance and MIDI score increases, the chance of having an incorrect alignment also increases. This can occur in the case of embellishments (e.g. long trills, or mordents that can be played from the printed note or from the note above) or major performance mistakes. We try to detect these problems automatically by measuring the inter-beat-intervals of each performance and marking the outliers as possible problems. On these outliers (which occurred in around 400 of ASAP's performances), we introduce a final manual correction step. 43 performances contained significant

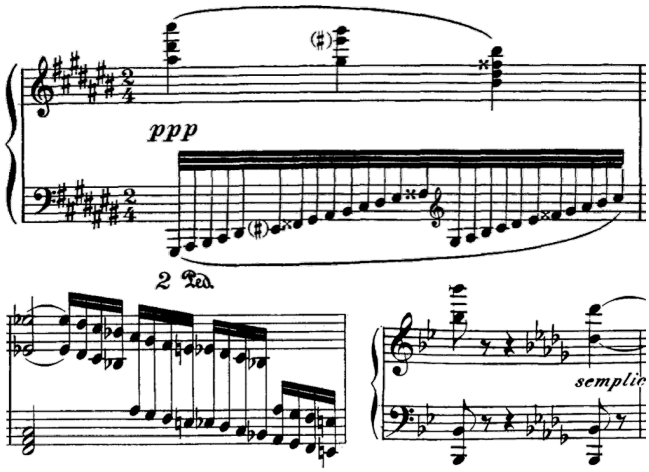


Figure 5.2: Examples of musical content problems in musical scores, including incorrect bar length (Ravel's *Ondine* (top) and Mozart's *Fantasie in C minor* (in 4/4; bottom left)) and a mid-bar key change (Beethoven's *Sonata No.29, Op.106*, 2nd movement (bottom right)).

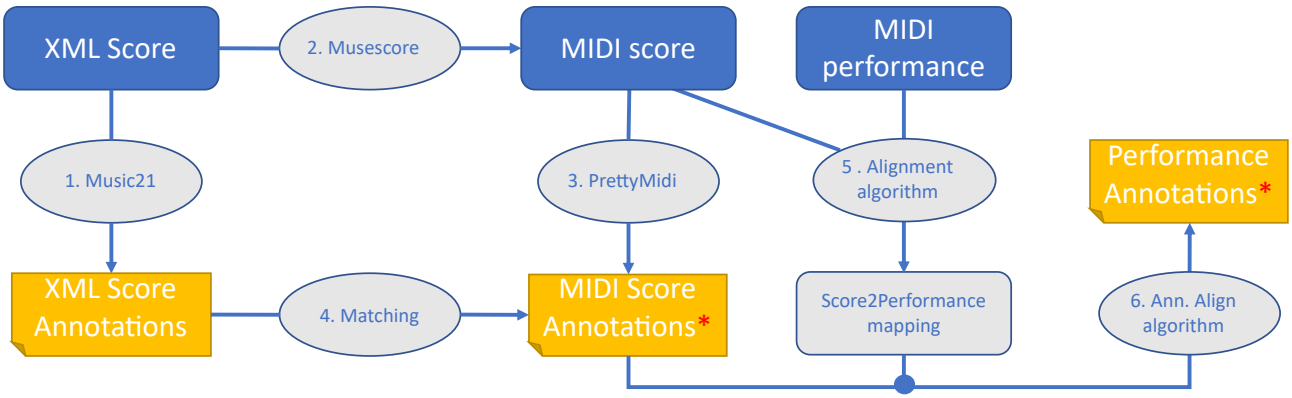


Figure 5.3: Our dataset annotation workflow. "*" indicates manual correction (see Section 5.2.2)

alignment errors that were corrected, and around 2% of annotations had to be moved by less than 1 second.

5.3 Dataset Overview

This section contains information about the file of the dataset: the format in which they are distributed, their origin and some statistics about the tempo variation.

Composer	XML/MIDI Score	MIDI Performance	Audio Performance
Bach	59	169	152
Balakirev	1	10	3
Beethoven	57	271	120
Brahms	1	1	0
Chopin	34	290	109
Debussy	2	3	3
Glinka	1	2	2
Haydn	11	44	16
Liszt	16	121	48
Mozart	6	16	5
Prokofiev	1	8	0
Rachmaninoff	4	8	4
Ravel	4	22	0
Schubert	13	62	44
Schumann	10	28	7
Scriabin	2	13	7
Total	222	1068	520

Table 5.2: The composition of ASAP.

5.3.1 Dataset content

ASAP contains 222 distinct musical scores and 1068 unique performances of Western classical piano music from 15 different composers (see Table 5.2 for a breakdown). 548 of the recordings are available as MIDI only, and all the others (520) are provided as MIDI and audio recordings aligned with approximately 3 ms precision. Each score corresponds with at least one performance (and usually more). Every score and performance in ASAP is labeled with metadata including the composer and the title of the piece. We took care to ensure that any two performances of the same piece are labeled with the same title and composer, and no two distinct pieces in ASAP share both.

Each musical score is provided in both MusicXML[5] and MIDI formats. In the MIDI score, the position of all MIDI events are quantized to a metrical grid according to their position in the MusicXML score. Grace notes are represented in MIDI as notes of very short duration. Repetitions in the score are "unfolded" in the MIDI file such that some sections of the MusicXML score may be duplicated in the MIDI score. Except for performance mistakes, there is a one-to-one correspondence between the

[5] `https://www.musicxml.com/`

notes in a MIDI performance and its associated MIDI score.

For each performance and MIDI score, ASAP provides the positions (in seconds) of all beats, downbeats, time signature changes and key signature changes. Time signature changes are annotated only on downbeats. In the case of pickup measures (and pickup measures in the middle of the score) we delay the position of the time signature change annotation to the following downbeat. Similarly, key signature changes are annotated only on beats. Each downbeat is also mapped to a specific measure number in the MusicXML score, which allows for a clear score alignment, even in the case of repetitions.

The dataset and the code used to generate annotations, along with a detailed description of the specific formatting of the dataset and usage examples are available online[6].

5.3.2 Origin of the files

The files in ASAP are drawn from multiple sources. The MusicXML scores are from the MuseScore online library[7], created and uploaded by the users of the MuseScore music notation software. They were first collected and associated to MIDI performances from the Yamaha e-piano competition[8] by the authors of [141]. We manually edited the MusicXML scores using MuseScore3 to correct any notation errors, and generated quantized MIDI files using MuseScore3's MIDI export utility. The paired audio and MIDI performances come from the MAESTRO dataset[124]. We automatically matched as many of the MAESTRO performances as we could to ones collected by [141], thus associating them with musical scores. The unmatched performances from MAESTRO are not included in ASAP. Finally, we modified 5 of the MIDI performances by inserting a missing note at the beginning, and 277 more have been cut to obtain more homogeneous pieces (e.g., the Bach Preludes are separated from the Fugues, even though they sometimes come from the same performance).

5.3.3 Dataset Statistics

ASAP contains performances of classical piano music, a style that can be challenging for beat tracking systems due to the large tempo variations that are often present. Here, we compare the tempo variation of the pieces in ASAP to that of the pieces of datasets commonly used for beat

[6] `https://github.com/fosfrancesco/asap-dataset`
[7] `https://musescore.com/sheetmusic`
[8] `http://piano-e-competition.com/`

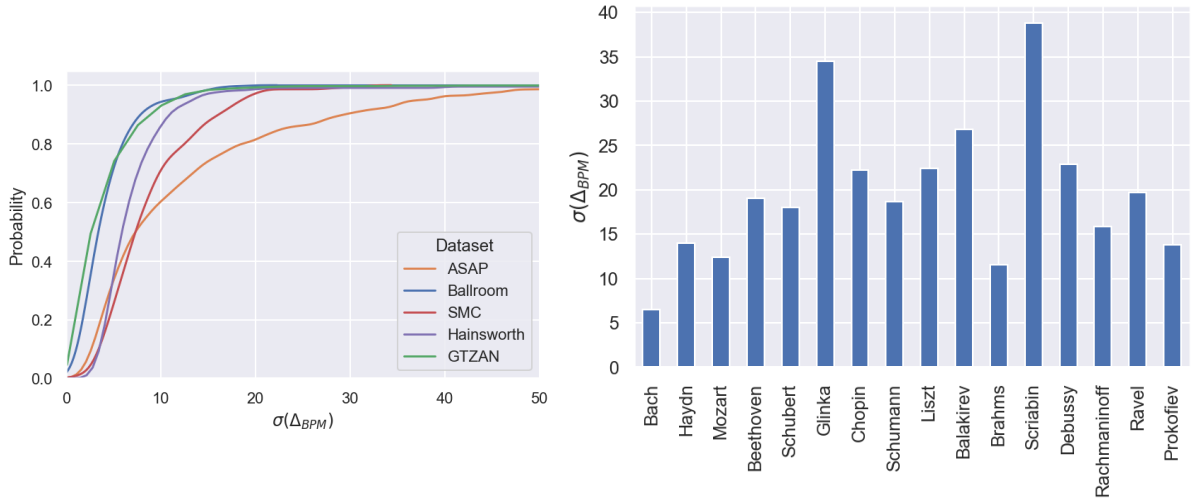


Figure 5.4: On the left: cumulative Distribution Function of $\sigma(\Delta_{BPM})$ of each piece in the compared datasets. On the right: average $\sigma(\Delta_{BPM})$ of the pieces by each composer in ASAP, by chronological order.

tracking from audio [127, 130, 129, 125]. To quantify the tempo variation for each piece, we first compute the BPM at each beat based on the amount of time between consecutive beats. Then, we compute Δ_{BPM} at each beat as the difference between consecutive BPMs. Finally, we compute the standard deviation of the set of all Δ_{BPM} values in a particular piece, which we call $\sigma(\Delta_{BPM})$. Figure 5.4 presents the distribution of these standard deviations for each dataset as a Cumulative Distribution Function, which shows the probability that a randomly chosen piece from each dataset has a $\sigma(\Delta_{BPM})$ less than the given value. From the plot, it can be seen that Ballroom and SMC have generally steadier tempos than the other datasets, and that ASAP's steadiest 40% and 50% of pieces roughly match those of Hainsworth and SMC respectively. However, a large portion of the pieces in ASAP have significantly larger tempo variation than any of the compared datasets.

Within ASAP, differences can be observed between different composers. In fact, even though the pieces in the dataset fall under the broad term "classical music", ASAP is very diverse, containing pieces of various styles written across different periods. This can be observed from differences in average $\sigma(\Delta_{BPM})$ for each composer, as is shown in Figure 5.4. The values in this plot generally match musicological intuitions about tempo variation for the different composers.

5.4 Conclusion

This chapter presented ASAP: a new dataset of aligned musical scores and performances of classical piano music. Downbeat, beat, time signature, and key signature annotations are produced using a novel workflow that exploits information present in the score musical content to drastically reduce manual annotation time compared to fully manual annotation. ASAP contains over 1000 annotated MIDI performances of classical piano music, over 500 of which are paired with audio from the MAESTRO dataset. To our knowledge, it is the largest dataset of that contains such a fine-grained alignment between scores and performances.

This work has only scratched the surface of what can be done with ASAP. Future work will present further statistical analyses on the data and baseline model performance on tasks for which it can be used: complete AMT and beat tracking as presented, as well as others such as expressive performance analysis and rendering [142]. For complete AMT in particular, the evaluation method is still an open problem, although proposals have been made (*e.g.,* [69, 94]).

Conclusions

In this thesis we have worked on musical scores, in the context of automatic music transcription. We believe that the development of fruitful AMT systems will benefit from a clear distinction among the several aspects of music notation.

We proposed a dedicated score model which encodes in separate structures the information related to the production of music (*i.e.,* the musical content) and the graphical symbols used to represent it (*i.e.,* the score engraving). We also discussed on quality issues that such a complex structure may contain and we proposed a structured and extensible approach to classify and detect them.

We presented a system which jointly performs the quantization and full metrical alignment of a monophonic MIDI performance. Our algorithm works "online", processing measure by measure with a left-right parsing. Its output is the musical content model. Our approach relies on a generative grammar and computes solutions which are optimal with respect to both the fitness to the input performance, and the complexity of the rhythmical content. We addressed the problem of learning such grammar from a dataset of musical scores.

We described a tool that relies on our model data structures for the computation and visualization of the differences between two musical scores at the engraving level. The algorithms for the computation rely on a combination of sequence- and tree-edit distances. The visualization tool mimics the interfaces for the differences between two text files: the scores are displayed side-by-side, and the music notation engraving library Verovio is used to display and highlight their differences.

Finally, we presented ASAP: a dataset of 222 digital musical scores aligned with 1068 performances of Western classical piano music (more than 92 hours). The scores are available as quantized MIDI files in MusicXML format, and the performances as MIDI and audio files. To our knowledge, it is the largest dataset that contains a fine-grained alignment between scores and performances. Downbeat,

beat, time signature changes, and key signature changes are annotated and aligned in MIDI and audio files, and each downbeat is also linked to its corresponding measure in the XML score. Those annotations are obtained through a semi-automatic workflow which drastically reduces the time for manual annotation, by extracting information from the musical content of the score and tying them to the performances.

The source code of some of our approaches have been shared through public repositories, attached to the related publications.

5.5 Perspectives

We now summarize some of the limitations and possible extensions of the studies presented in this thesis.

Our score model (presented in Chapter 2) does not yet consider many elements of the musical score, such as slurs, dynamics, expression marks, articulations and embellishments (except grace-notes). Even if the research in this direction is limited, those elements influence the music that is produced and can be extracted from a performance, thus we plan to extend our model in order to encode them. Moreover we are currently working on producing a tool that transforms the score model into the engraving model, based on "defaults" settings that follow the engraving best practices [28], and on a set of user-defined parameters, *e.g.,* how many staves to produce. Ideally this system could be used as a final step for a full AMT pipeline.

The work on the parsing tool presented in Chapter 3 lacks of an in-depth evaluation of the results, especially time signature estimation and tempo changes. In order to perform a more systematic evaluation of the results, we plan to develop a metric for the comparison of the musical content of the scores. We also plan to work on an extension of our system that handles polyphonic performances, with a grammar encoding both rhythmical and voice information.

The diff tool presented in Chapter 4 only considers a subset of the graphical information contained in a musical score. Other symbols such as stem directions, clefs, slurs, expression markings, ornaments and lyrics would be necessary for the development of a complete tool. Technical improvements are also still needed in the XML parsing step and in the visualization interface. Another possible improvement of our work concerns the similarity metric that we employed: while it was developed focusing on

the computation of a list of differences, it could be employed for other automatic tasks targeting the graphical content of the score. We leave experiments on this usage for the future.

The work on the ASAP dataset could be expanded computing benchmark baselines that would help researchers evaluate their methods. A more in-depth statistical analysis of the dataset content (*e.g.,* note durations, spreading of notes in the same chord) would also help setting the parameters of systems that use this or similar datasets. The usage of this dataset for testing the techniques presented in this thesis is currently under development.

More generally, future works will involve the development of a "full" AMT system, able to transform a performance in a structured musical score. MIR research targeted for decades this problem, without reaching a satisfying solution, even for the most studied case of piano music. Fortunately, very promising results have been reached recently regarding the generation of piano roll representations from piano audio performances, thanks to the latest developments of deep learning models.

Advanced techniques that extract some metrical information (in general beats and downbeats) have also been developed, but they are hard to employ in a full AMT workflow for piano music, because they mostly target pop, rock and dance music, with multiple instruments and steadier tempos. Moreover beat and downbeat annotations are only a part of the rhythmical information necessary to build a musical score. Except for some very simplified scenarios, the application of similar techniques to the entire metrical structure has not been very successful yet. Their limits lie in the sequential representation of data, where complex metrical (*i.e.,* hierarchical) information is difficult to include. We believe that, to produce musical scores, it is necessary to explore systems that deal with complex hierarchical structures. A promising starting point could be the recent deep learning techniques proposed in other fields (*e.g.,* natural language processing) that target graphs and trees.

In parallel to this research in the field of machine learning, a good amount of work is still needed in score modeling, to be able both to simplify the work of supervised algorithms with efficient input representations and similarity metrics, and to build a correct and human-readable musical score from their output.

Let us conclude this thesis with a final consideration on the usage of musical scores for music analysis. While recent technological developments enabled the retrieval of high level information

directly from audio files, this kind of research requires high technical skills and it is usually hard to grasp for musicologist that are used to higher level symbolic representations of music. This has split music analysis research in two parts: the first targets large datasets of audio files and focuses on modeling the human intuition, mostly with with black box approaches that show few clues on the logic behind their choices; the second targets very little datasets of musical scores, with a great attention to details and the involvement of high level musical structures and procedures. We believe that the automatic production of musical scores could act as a bridge between these two groups of people, by enabling advanced musicological research which uses high level musical concepts on large music datasets.

Bibliography

[1] E. Benetos, S. Dixon, Z. Duan, and S. Ewert, "Automatic music transcription: An overview," *IEEE Signal Processing Magazine*, vol. 36, pp. 20–30, 2019.

[2] R. N. Giere, "How models are used to represent reality," *Philosophy of science*, vol. 71, no. 5, pp. 742–752, 2004.

[3] R. Milner, "The tower of informatic models," *From semantics to Computer Science*, 2009.

[4] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM (JACM)*, vol. 21, no. 1, pp. 168–173, 1974.

[5] G. Navarro, "A guided tour to approximate string matching," *ACM computing surveys (CSUR)*, vol. 33, no. 1, pp. 31–88, 2001.

[6] A. Klapuri, "Introduction to music transcription," *Signal Processing Methods for Music Transcription*, pp. 3–20, 2006.

[7] E. Chew and Y.-C. Chen, "Real-time pitch spelling using the spiral array," *Computer Music Journal*, vol. 29, no. 2, pp. 61–76, 2005.

[8] E. Cambouropoulos, "Pitch spelling: A computational model," *Music Perception*, vol. 20, no. 4, pp. 411–429, 2003.

[9] P. Desain and L. Windsor, *Rhythm perception and production*. Swets & Zeitlinger Publishers, 2000.

[10] R. Delone, V. L. Kliewer, M. H. Wennerstrom, H. J. Reisberg, and A. Winold, *Aspects of twentieth-century music*. Prentice Hall, 1975.

[11] G. W. Cooper, G. Cooper, and L. B. Meyer, *The rhythmic structure of music*. University of Chicago Press, 1963.

[12] F. Lerdahl and R. Jackendoff, *A generative theory of tonal music*. MIT press, 1983.

[13] H. Honing, "From time to time: The representation of timing and tempo," *Computer Music Journal*, vol. 25, no. 3, pp. 50–61, 2001.

[14] D. Back, "Standard MIDI-file format specifications," 1999, accessed September 23, 2020.

[15] I. Oppenheim, C. Walshaw, and J. Atchley, "The abc standard 2.0." `https://abcnotation.com/wiki/abc:standard:v2.0`, 2010.

[16] W. B. Hewlett, "Musedata: multipurpose representation," in *Beyond MIDI: The handbook of musical codes*, pp. 402–447, MIT press, 1997.

[17] D. Huron, "The humdrum toolkit: Software for music research." `https://humdrum.org`, 1993.

[18] D. Huron, "Humdrum and kern: selective feature encoding," in *Beyond MIDI: the handbook of musical codes*, MIT Press, Cambridge, MA, 1997.

[19] H.-W. Nienhuys and J. Nieuwenhuizen, "Lilypond, a system for automated music engraving," in *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*, vol. 1, pp. 167–171, Citeseer, 2003.

[20] H. H. Hoos, K. A. Hamel, K. Renz, and J. Kilian, "The guido notation format: A novel approach for adequately representing score-level music," in *International Computer Music Conference (ICMC)*, Citeseer, 1998.

[21] D. Taupin, R. Mitchell, and A. Egler, "Musixtex. using tex to write polyphonic or instrumental music," *EuroTEX*, vol. 92, pp. 257–272, 1993.

[22] D. Byrd, R. D. Boyle, U. Berggren, D. Bainbridge, *et al.*, *Beyond MIDI: the handbook of musical codes*. MIT press, 1997.

[23] M. Good *et al.*, "Musicxml: An internet-friendly format for sheet music," in *Xml conference and expo*, pp. 03–04, 2001.

[24] P. Roland, "The music encoding initiative (mei)," in *Proceedings of the First International Conference on Musical Applications Using XML*, vol. 1060, pp. 55–59, 2002.

[25] M. A. Román, A. Pertusa, and J. Calvo-Zaragoza, "A holistic approach to polyphonic music transcription with neural netwoks," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 731–737, 2019.

[26] R. G. C. Carvalho and P. Smaragdis, "Towards end-to-end polyphonic music transcription: Transforming music audio directly to a score," in *WASPAA*, pp. 151–155, 2017.

[27] E. Nakamura, E. Benetos, K. Yoshii, and S. Dixon, "Towards complete polyphonic music transcription: Integrating multi-pitch detection and rhythm quantization," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 101–105, IEEE, 2018.

[28] E. Gould, *Behind bars: the definitive guide to music notation*. Faber Music Ltd, 2016.

[29] W. Buxton, W. Reeves, R. Baecker, and L. Mezei, "The use of hierarchy and instance in a data structure for computer music," *Computer Music Journal*, pp. 10–20, 1978.

[30] S. Pope, "Object-oriented music representation," *Organised Sound*, vol. 1, pp. 56–68, 1996.

[31] B. Coüasnon and B. Rétif, "Using a grammar for a reliable full score recognition system," in *International Computer Music Conference (ICMC)*, 1995.

[32] B. Couasnon, "Dmos: a generic document recognition method, application to an automatic generator of musical scores, mathematical formulae and table structures recognition systems," in *Proceedings of Sixth International Conference on Document Analysis and Recognition*, pp. 215–220, 2001.

[33] M. Szwoch, "Guido: A musical score recognition system," in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 2, pp. 809–813, IEEE, 2007.

[34] F. Rossant and I. Bloch, "Robust and adaptive omr system including fuzzy modeling, fusion of musical rules, and possible error detection," *EURASIP Journal on Advances in Signal Processing*, vol. 2007, 2006.

[35] J. Calvo-Zaragoza and D. Rizo, "End-to-end neural optical music recognition of monophonic scores," *Applied Sciences*, vol. 8, no. 4, p. 606, 2018.

[36] M. S. Cuthbert and C. Ariza, "music21: A toolkit for computer-aided musicology and symbolic music data," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 637–642, 2010.

[37] P. Fraisse, "Rhythm and tempo," *The psychology of music*, vol. 1, pp. 149–180, 1982.

[38] T. L. Bolton, "Rhythm," *The american journal of psychology*, vol. 6, no. 2, pp. 145–238, 1894.

[39] M. Mongeau and D. Sankoff, "Comparison of musical sequences," *Computers and the Humanities*, vol. 24, no. 3, pp. 161–175, 1990.

[40] E. Selfridge-Field, "Conceptual and representational issues in melodic comparison," *Melodic similarity, concepts, procedures, and applications*, 1998.

[41] M. Clausen, R. Engelbrecht, D. Meyer, and J. Schmitz, "Proms: A web-based tool for searching in polyphonic music," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2000.

[42] D. Müllensiefen, K. Frieler, *et al.*, "Cognitive adequacy in the measurement of melodic similarity: Algorithmic vs. human judgments," *Computing in Musicology*, vol. 13, no. 2003, pp. 147–176, 2004.

[43] P. Hanna and P. Ferraro, "Polyphonic music retrieval by local edition of quotiented sequences," *International Workshop on Content-Based Multimedia Indexing, Proceedings*, pp. 61 – 68, 2007.

[44] H. A. Simon, "Perception du pattern musical par auditeur," *Science de l'Art*, vol. 5, no. 2, pp. 28–34, 1968.

[45] B. Lindblom and J. Sundberg, "Towards a generative theory of melody," *Svensk Tidskrift for Musikforsknin*, vol. 33, pp. 71–88, 1970.

[46] H. C. Longuet-Higgins, "Perception of melodies," *Nature*, vol. 263, no. 5579, pp. 646–653, 1976.

[47] H. C. Longuet-Higgins and C. S. Lee, "The rhythmic interpretation of monophonic music," *Music Perception*, vol. 1, no. 4, pp. 424–441, 1984.

[48] C. S. Lee, "The rhythmic interpretation of simple musical sequences: towards a perceptual model," *Musical structure and cognition Journal*, 1985.

[49] F. Lerdahl and R. Jackendoff, "Toward a formal theory of tonal music," *Journal of music theory*, vol. 21, no. 1, pp. 111–171, 1977.

[50] D. Rizo, *Symbolic music comparison with tree data structures*. PhD thesis, Universidad de Alicante, 2010.

[51] J. F. Bernabeu, J. Calera-Rubio, J. M. Iñesta, and D. Rizo, "Melodic identification using probabilistic tree automata," *Journal of New Music Research*, vol. 40, no. 2, pp. 93–103, 2011.

[52] P. Nauert, "A theory of complexity to constrain the approximation of arbitrary sequences of timepoints," *Perspectives of New Music*, vol. 32, no. 2, pp. 226–263, 1994.

[53] A. Ycart, F. Jacquemard, J. Bresson, and S. Staworko, "A supervised approach for rhythm transcription based on tree series enumeration," in *International Computer Music Conference (ICMC)*, 2016.

[54] F. Jacquemard, A. Ycart, and M. Sakai, "Generating equivalent rhythmic notations based on rhythm tree languages," in *International Conference of Technologies for Music Notation and Representation (TENOR)*, 2017.

[55] C. Agón, K. Haddad, and G. Assayag, "Representation and rendering of rhythm structures," in *Second International Conference on WEB Delivering of Music (WEDELMUSIC)*, pp. 109–116, 2002.

[56] J. Bresson, C. Agon, and G. Assayag, "Openmusic: visual programming environment for music composition, analysis and research," in *ACM international conference on Multimedia*, pp. 743–746, 2011.

[57] H. Kameoka, K. Ochiai, M. Nakano, M. Tsuchiya, and S. Sagayama, "Context-free 2d tree structure model of musical notes for bayesian modeling of polyphonic spectrograms.," in *International Society for Music Information Retrieval Conference (ISMIR)*, vol. 2012, pp. 307–312, 2012.

[58] M. Tsuchiya, K. Ochiai, H. Kameoka, and S. Sagayama, "Probabilistic model of two-dimensional rhythm tree structure representation for automatic transcription of polyphonic midi signals," in

2013 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference, pp. 1–6, IEEE, 2013.

[59] C. Ariza and M. S. Cuthbert, "Modeling beats, accents, beams, and time signatures hierarchically with music21 meter objects.," in *International Computer Music Conference (ICMC)*, 2010.

[60] A. Rogers, "Peer review in nlp: resource papers." `https://hackingsemantics.xyz/2020/reviewing-data/`, 2020.

[61] L. C. Elson, *Elson's Pocket Music Dictionary: The Important Terms Used in Music with Pronunciation and Concise Definition, Together with the Elements of Notation and a Biographical List of Over Seven Hundred Noted Names in Music*. Oliver Ditson Company, 1909.

[62] S. Dixon, "Automatic extraction of tempo and beat from expressive performances," *Journal of New Music Research*, vol. 30, no. 1, pp. 39–58, 2001.

[63] S. Böck, F. Krebs, and G. Widmer, "Joint beat and downbeat tracking with recurrent neural networks.," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 255–261, New York City, 2016.

[64] S. Durand, J. P. Bello, B. David, and G. Richard, "Robust downbeat tracking using an ensemble of convolutional networks," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 1, pp. 76–89, 2016.

[65] J. C. Brown, "Determination of the meter of musical scores by autocorrelation," *The Journal of the Acoustical Society of America*, vol. 94, no. 4, pp. 1953–1957, 1993.

[66] B. Meudic, "Automatic meter extraction from midi files," in *Journées d'Informatique Musicale (JIM)*, 2002.

[67] D. Temperley *et al.*, *Music and probability*. Mit Press, 2007.

[68] W. B. De Haas and A. Volk, "Meter detection in symbolic music using inner metric analysis," in *International Society for Music Information Retrieval Conference (ISMIR)*, p. 441, 2016.

[69] A. McLeod and M. Steedman, "Evaluating automatic polyphonic music transcription," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 42–49, 2018.

[70] C. Raphael, "Automated rhythm transcription.," in *International Society for Music Information Retrieval Conference (ISMIR)*, vol. 2001, pp. 99–107, 2001.

[71] A. T. Cemgil and B. Kappen, "Monte carlo methods for tempo tracking and rhythm quantization," *Journal of Artificial Intelligence Research*, vol. 18, pp. 45–81, 2003.

[72] E. Nakamura, K. Yoshii, and S. Sagayama, "Rhythm transcription of polyphonic piano music based on merged-output hmm for multiple voices," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 4, pp. 794–806, 2017.

[73] J. Pressing and P. Lawrence, "Transcribe: A comprehensive autotranscription program," in *International Computer Music Conference (ICMC)*, pp. 343–345, 1993.

[74] H. Takeda, N. Saito, T. Otsuki, M. Nakai, H. Shimodaira, and S. Sagayama, "Hidden markov model for automatic transcription of midi signals," in *2002 IEEE Workshop on Multimedia Signal Processing.*, pp. 428–431, IEEE, 2002.

[75] K. Shibata, E. Nakamura, and K. Yoshii, "Non-local musical statistics as guides for audio-to-score piano transcription," *arXiv preprint arXiv:2008.12710*, 2020.

[76] N. Whiteley, A. T. Cemgil, and S. J. Godsill, "Bayesian modelling of temporal structure in musical audio.," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 29–34, Citeseer, 2006.

[77] A. McLeod and M. Steedman, "Meter detection and alignment of midi performance.," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 113–119, 2018.

[78] H. C. Longuet-Higgins and M. Steedman, "On interpreting bach," *Machine intelligence*, vol. 6, 1971.

[79] D. Eck and N. Casagrande, "Finding meter in music using an autocorrelation phase matrix and shannon entropy," *Target*, vol. 300, no. 350, p. 400, 2005.

[80] D. Temperley and D. Sleator, "Modeling meter and harmony: A preference-rule approach," *Computer Music Journal*, vol. 23, no. 1, pp. 10–27, 1999.

[81] D. Temperley, *The cognition of basic musical structures*. MIT press, 2001.

[82] D. Temperley, "A unified probabilistic model for polyphonic music analysis," *Journal of New Music Research*, vol. 38, no. 1, pp. 3–18, 2009.

[83] A. McLeod, E. Nakamura, and K. Yoshii, "Improved metrical alignment of MIDI performance based on a repetition-aware online-adapted grammar," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 186–190, 2019.

[84] P. Desain and H. Honing, "The quantization of musical time: A connectionist approach," *Computer Music Journal*, vol. 13, no. 3, pp. 56–66, 1989.

[85] C. Agon, G. Assayag, F. Joshua, and C. Rueda, "Kant: A critique of pure quantification," in *International Computer Music Conference (ICMC)*, 1994.

[86] J. Jahn, *Vector Optimization. Theory, Applications, and Extensions*. Springer-Verlag, 2011.

[87] M.-J. Nederhof and G. Satta, "Probabilistic parsing as intersection," in *International Conference on Parsing Technologies (IWPT)*, pp. 137–148, 2003.

[88] A. Maletti and G. Satta, "Parsing algorithms based on tree automata," in *International Conference on Parsing Technologies (IWPT)*, pp. 1–12, 2009.

[89] L. Huang, "Advanced dynamic programming in semiring and hypergraph frameworks," in *Advanced Dynamic Programming in Computational Linguistics: Theory, Algorithms and Applications*, pp. 1–18, 2008.

[90] C. Raphael, "A hybrid graphical model for rhythmic parsing," *Artificial Intelligence*, vol. 137, no. 1-2, pp. 217–238, 2002.

[91] Z. Chi and S. Geman, "Estimation of probabilistic context-free grammars," *Computational linguistics*, vol. 24, no. 2, pp. 299–305, 1998.

[92] T. Kasami, "An efficient recognition and syntax-analysis algorithm for context-free languages," Tech. Rep. Scientific report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA, 1965.

[93] E. Lamarque and M.-J. Goudard, *D'un rythme à l'autre, vol. 1-4*. Lemoine, 1997.

[94] A. Cogliati and Z. Duan, "A metric for music notation transcription accuracy," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 407–413, 2017.

[95] J. W. Hunt and M. D. McIlroy, "An algorithm for differential file comparison," *Computing Science Technical Report*, vol. 41, 1976.

[96] P. Heckel, "A technique for isolating differences between files," *Communications of the ACM*, vol. 21, no. 4, pp. 264–268, 1978.

[97] J. Allali, P. Ferraro, P. Hanna, C. Iliopoulos, and M. Robine, "Toward a general framework for polyphonic comparison," *Fundamenta Informaticae*, vol. 97, no. 3, pp. 331–346, 2009.

[98] K. Lemström, *String Matching Techniques for Music Retrieval.* PhD thesis, University of Helsinki, Department of Computer Science, 2000.

[99] A. Mcleod and M. Steedman, "Evaluating automatic polyphonic music transcription," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 42–49, 2018.

[100] C. Antila, J. Treviño, and G. Weaver, "A hierarchic diff algorithm for collaborative music document editing," in *Third International Conference on Technologies for Music Notation and Representation (TENOR)*, 2017.

[101] I. Knopke and D. Byrd, "Towards musicdiff: A foundation for improved optical music recognition using multiple recognizers," *Dynamics*, vol. 85, no. 165, p. 121, 2007.

[102] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM journal on computing*, vol. 18, no. 6, pp. 1245–1262, 1989.

[103] G. Cobena, S. Abiteboul, and A. Marian, "Detecting changes in XML documents," in *Proceedings 18th International Conference on Data Engineering*, pp. 41–52, IEEE, 2002.

[104] E. W. Myers, "Ano (nd) difference algorithm and its variations," *Algorithmica*, vol. 1, no. 1-4, pp. 251–266, 1986.

[105] E. Ukkonen, "Algorithms for approximate string matching," *Information and control*, vol. 64, no. 1-3, pp. 100–118, 1985.

[106] E. Nakamura, N. Ono, S. Sagayama, and K. Watanabe, "A stochastic temporal model of polyphonic midi performance with ornaments," *Journal of New Music Research*, vol. 44, no. 4, pp. 287–304, 2015.

[107] E. Nakamura, K. Yoshii, and H. Katayose, "Performance error detection and post-processing for fast and accurate symbolic music alignment," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 347–353, 2017.

[108] C.-T. Chen, J.-S. R. Jang, and W. Liou, "Improved score-performance alignment algorithms on polyphonic music," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1365–1369, IEEE, 2014.

[109] B. Gingras and S. McAdams, "Improved score-performance matching using both structural and temporal information from midi recordings," *Journal of New Music Research*, vol. 40, no. 1, pp. 43–57, 2011.

[110] P. Bille, "A survey on tree edit distance and related problems," *Theor. Comput. Sci.*, vol. 337, no. 1-3, pp. 217–239, 2005.

[111] M. S. Cuthbert and C. Ariza, "music21: A toolkit for computer-aided musicology and symbolic music data," in *International Society for Music Information Retrieval Conference (ISMIR)*, International Society for Music Information Retrieval, 2010.

[112] L. Pugin, R. Zitellini, and P. Roland, "Verovio: A library for engraving mei music notation into svg," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 107–112, 2014.

[113] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, I. Simon, C. Hawthorne, N. Shazeer, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer: Generating music with long-term structure," in *International Conference on Learning Representations*, 2018.

[114] D. Stoller, S. Ewert, and S. Dixon, "Wave-U-Net: A multi-scale neural network for end-to-end audio source separation," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 334–340, 2018.

[115] J. W. Kim and J. P. Bello, "Adversarial learning for improved onsets and frames music transcription," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 670–677, 2019.

[116] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[117] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[118] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, 2016.

[119] J. Calvo-Zaragoza, J. Hajic Jr, and A. Pacha, "Understanding optical music recognition," *Computer Research Repository, abs/1908.03608*, 2019.

[120] A. Olmos, N. Bouillot, T. Knight, N. Mabire, J. Redel, and J. R. Cooperstock, "A high-fidelity orchestra simulator for individual musicians' practice," *Computer Music Journal*, vol. 36, no. 2, pp. 55–73, 2012.

[121] V. Emiya, R. Badeau, and B. David, "Multipitch estimation of piano sounds using a new probabilistic spectral smoothness principle," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, no. 6, pp. 1643–1654, 2010.

[122] A. Ycart and E. Benetos, "A-MAPS: Augmented MAPS dataset with rhythm and key annotations," in *ISMIR Late Breaking and Demo Papers*, 2018.

[123] M. Hashida, T. Matsui, and H. Katayose, "A new music database describing deviation information of performance expressions.," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 489–494, 2008.

[124] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck, "Enabling factorized piano music modeling and generation with the MAESTRO dataset," in *International Conference on Learning Representations*, 2019.

[125] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on speech and audio processing*, vol. 10, no. 5, pp. 293–302, 2002.

[126] U. Marchand and G. Peeters, "Swing ratio estimation," in *Digital Audio Effects (DAFx)*, pp. 423–428, 2015.

[127] F. Gouyon, A. Klapuri, S. Dixon, M. Alonso, G. Tzanetakis, C. Uhle, and P. Cano, "An experimental comparison of audio tempo induction algorithms," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 5, pp. 1832–1844, 2006.

[128] F. Krebs, S. Böck, and G. Widmer, "Rhythmic pattern modeling for beat and downbeat tracking in musical audio," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 227–232, 2013.

[129] S. W. Hainsworth and M. D. Macleod, "Particle filtering applied to musical tempo tracking," *EURASIP Journal on Advances in Signal Processing*, vol. 2004, no. 15, p. 927847, 2004.

[130] A. Holzapfel, M. E. Davies, J. R. Zapata, J. L. Oliveira, and F. Gouyon, "Selective sampling for beat tracking evaluation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 9, pp. 2539–2548, 2012.

[131] W. Goebl, "Numerisch-klassifikatorische interpretationsanalyse mit dem "Bösendorfer Computerflügel"," Master's thesis, Universität Wien, 1999.

[132] M. Müller, V. Konz, W. Bogler, and V. Arifi-Müller, "Saarland music data (SMD)," in *ISMIR Late Breaking and Demo Papers*, 2011.

[133] A. McLeod and M. Steedman, "HMM-based voice separation of MIDI performance," *Journal of New Music Research*, vol. 45, no. 1, pp. 17–26, 2016.

[134] Z. Shi, C. S. Sapp, K. Arul, J. McBride, and J. O. Smith, "SUPRA: Digitizing the stanford university piano roll archive," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 517–523, 2019.

[135] S. Böck, M. E. Davies, and P. Knees, "Multi-task learning of tempo and beat: Learning one to improve the other," in *International Society for Music Information Retrieval Conference (ISMIR)*, 2019.

[136] C. Weiß, V. Arifi-Müller, T. Prätzlich, R. Kleinertz, and M. Müller, "Analyzing measure annotations for western classical music recordings," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 517–523, 2016.

[137] T. Gadermaier and G. Widmer, "A study of annotation and alignment accuracy for performance comparison in complex orchestral music," *arXiv preprint arXiv:1910.07394*, 2019.

[138] M. S. Cuthbert, C. Ariza, and L. Friedland, "Feature extraction and machine learning on symbolic music using the music21 toolkit.," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 387–392, 2011.

[139] C. Raffel and D. P. W. Ellis, "Intuitive analysis, creation and manipulation of midi data with pretty_midi," in *ISMIR Late Breaking and Demo Papers*, 2014.

[140] P. Grosche, M. Müller, and C. S. Sapp, "What makes beat tracking difficult? a case study on chopin mazurkas," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 649–654, 2010.

[141] D. Jeong, T. Kwon, Y. Kim, K. Lee, and J. Nam, "VirtuosoNet: A hierarchical RNN-based system for modeling expressive piano performance," in *International Society for Music Information Retrieval Conference (ISMIR)*, pp. 908–915, 2019.

[142] C. E. Cancino-Chacón, M. Grachten, W. Goebl, and G. Widmer, "Computational models of expressive music performance: A comprehensive and critical review," *Frontiers in Digital Humanities*, vol. 5, p. 25, 2018.

Authored Papers

[Foscarin et al., 2018a] Foscarin, F., Fiala, D., Jacquemard, F., Rigaux, P., and Thion, V. (2018a). Gioqoso, an online quality assessment tool for music notation. In *International Conference on Technologies for Music Notation and Representation (TENOR)*.

[Foscarin et al., 2018b] Foscarin, F., Fournier-S'niehotta, R., Jacquemard, F., and Rigaux, P. (2018b). Évaluation de la correction rythmique des partitions numérisées. In *Journées d'Informatique Musicale (JIM)*.

[Foscarin et al., 2019a] Foscarin, F., Jacquemard, F., and Fournier-S'niehotta, R. (2019a). A diff procedure for music score files. In *International Conference on Digital Libraries for Musicology (DLfM)*, page 58–64. Association for Computing Machinery.

[Foscarin et al., 2019b] Foscarin, F., Jacquemard, F., and Rigaux, P. (2019b). Modeling and learning rhythm structure. In *Sound and Music Computing Conference (SMC)*.

[Foscarin et al., 2019c] Foscarin, F., Jacquemard, F., Rigaux, P., and Sakai, M. (2019c). A parse-based framework for coupled rhythm quantization and score structuring. In *International Conference on Mathematics and Computation in Music (MCM)*, pages 248–260. Springer.

[Foscarin et al., 2020] Foscarin, F., Mcleod, A., Rigaux, P., Jacquemard, F., and Sakai, M. (2020). Asap: a dataset of aligned scores and performances for piano transcription. In *International Conference on Music Information Retrieval (ISMIR)*.

le cnam

Francesco FOSCARIN

The Musical Score:a challenging goal for automatic music transcription

le cnam

Résumé : Les partitions musicales sont des artefacts d'une grande complexité sémiotique visant à transmettre toutes les informations relatives à la structure et au contenu d'une œuvre musicale de manière compacte, lisible et cohérente. De nos jours, la musique est la plupart du temps diffusée sous forme enregistrée. Un système capable de transformer automatiquement une interprétation musicale en partition serait donc extrêmement bénéfique pour les interprètes et les musicologues. Cette tâche, appelée "automatic music transcription" (AMT), comprend plusieurs sous-tâches qui impliquent notamment la transformation de représentations intermédiaires comme le MIDI, quantifié ou non. Nous défendons dans cette thèse l'idée qu'un modèle robuste et bien structuré des informations contenues dans une partition musicale aiderait au développement et à l'évaluation de systèmes AMT. En particulier, nous préconisons une séparation claire entre le contenu musical encodé dans la partition et l'ensemble des signes et des règles de notation utilisés pour représenter graphiquement ce contenu.

Mots clés : partition musicale, modèle de partition, transcription musicale automatique, recherche d'information musicale.

Abstract : A musical score is a complex semiotic object that excels at conveying musical information in a human readable format. Nowadays, a lot of music is available exclusively as recorded performances, so systems which can automatically transform those performances into musical scores would be extremely beneficial for performers and musicologists. This task, called automatic music transcription (AMT), comprises a large number of subtasks which generically transform the performance input into higher level representations, such as unquantized and quantized MIDI files. We believe that a clear model of the information contained in a musical score would help the development and the evaluation of AMT systems. In particular we advocate for a clear separation between the music which the score encodes and the set of notation symbols which is employed to represent it.

Keywords : musical score, score model, automatic music transcription, music information retrieval.