

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

**Creative Support Musical Composition
System:
a study on Multiple Viewpoints
Representations in Variable Markov
Oracles**

Nádia de Sousa Varela de Carvalho



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Gilberto Bernardes de Almeida

July 23, 2020

**Creative Support Musical Composition System:
a study on Multiple Viewpoints Representations in
Variable Markov Oracles**

Nádia de Sousa Varela de Carvalho

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. João Jacob

External Examiner: Prof. Maria Navarro Cáceres

Supervisor: Prof. Gilberto Bernardes de Almeida

July 23, 2020

Abstract

The mid-20th century witnessed the emergence of an area of study that focused on the automatic generation of musical content by computational means. Early examples focus on offline processing of musical data and recently, the community has moved towards interactive online musical systems. Furthermore, a recent trend stresses the importance of assistive technology, which promotes a user-in-loop approach by offering multiple suggestions to a given creative problem.

In this context, my research aims to foster new software tools for creative support systems, where algorithms can collaboratively participate in the composition flow. In greater detail, I seek a tool that learns from *variable-length musical data* to provide real-time feedback during the composition process.

In light of the multidimensional and hierarchical structure of music, I aim to study the representations which abstracts its temporal patterns, to foster the generation of *multiple* ranked solutions to a given musical context. Ultimately, the subjective nature of the choice is given to the user to which a limited number of ‘optimal’ solutions are provided.

A symbolic music representation manifested as *Multiple Viewpoint Models* combined with the *Variable Markov Oracle* (VMO) automaton, are used to test optimal interaction between the multi-dimensionality of the representation with the optimality of the VMO model in providing both style-coherent, novel, and diverse solutions. To evaluate the system, an experiment will be conducted to validate the tool in an expert-based scenario with composition students, using the creativity support index test.

Keywords: Musical Composition, Pattern Analysis, Variable Markov Oracle, Context-dependent, Variable-length, Multiple Viewpoints, Musical Representations

Resumo

Em meados do século XX, assistiu-se ao surgimento de uma área de estudo focada na geração automática de conteúdo musical por meios computacionais. Os primeiros exemplos concentram-se no processamento offline de dados musicais mas, recentemente, a comunidade tem vindo a explorar maioritariamente sistemas musicais interativos e em tempo-real. Além disso, uma tendência recente enfatiza a importância da tecnologia assistiva, que promove uma abordagem centrada em escolhas do utilizador, oferecendo várias sugestões para um determinado problema criativo.

Nesse contexto, a minha investigação tem como objetivo promover novas ferramentas de software para sistemas de suporte criativo, onde algoritmos podem participar colaborativamente no fluxo de composição. Em maior detalhe, procuro uma ferramenta que aprenda com *dados musicais de tamanho variável* para fornecer feedback em tempo real durante o processo de composição.

À luz das características de multidimensionalidade e hierarquia presentes nas estruturas musicais, pretendo estudar as representações que abstraem os seus padrões temporais, para promover a geração de *múltiplas* soluções ordenadas por grau de optimização para um determinado contexto musical. Por fim, a natureza subjetiva da escolha é dada ao utilizador, ao qual é fornecido um número limitado de soluções 'ideais'.

Uma representação simbólica da música manifestada como *Modelos sob múltiplos pontos de vista* combinada com o autómato *Variable Markov Oracle* (VMO), é usada para testar a interação ideal entre a multidimensionalidade da representação e a idealidade do modelo VMO, fornecendo soluções coerentes, inovadoras e estilisticamente diversas. Para avaliar o sistema, será realizado um teste para validar a ferramenta num cenário especializado com alunos de composição, usando o modelo de testes do índice de suporte à criatividade.

Keywords: Composição Musical, Análises de Padrões, Variable Markov Oracle, Representações Múltiplas, Dependentes de contexto, Tamanho variável, Representações Musicais

Acknowledgements

I would like to express my deep gratitude to Professor Gilberto for his patient advice, enthusiastic incentives and valuable critiques of this research work.

I would also like to extend my thanks to the composers who have enthusiastically answered my request and participated in the evaluating process.

Finally, I wish to thank my family and, in particular, my parents for their constant support and encouragement throughout my studies, both in the musical and informatic fields.

Nádia de Sousa Varela de Carvalho

“Music can never have enough of saying over again what has already been said, not once or twice, but dozens of times; hardly does a section, which consists largely of repetition, come to an end, before the whole story is happily told all over again.”

Zuckerlandl, 1956

Contents

1	Introduction	1
1.1	Context and Motivation	1
1.2	Problem Definition and Objectives	2
1.3	Methodology	2
1.4	Document Structure	3
1.5	Publication	3
2	Knowledge Support Systems for Music: A literature review	5
2.1	Knowledge Support Music Systems	6
2.1.1	Important Concepts in Music Informatics	6
2.1.2	An Overview of CAAC Systems	7
2.1.2.1	CAAC Systems that use Statistical Modeling	7
2.1.2.2	CAAC Systems that use Deep Learning	9
2.2	Computational Representation of Symbolic Musical Structures	9
2.2.1	String Modeling	10
2.2.1.1	String Modeling of Monophonic Music	10
2.2.1.2	String Modeling of Polyphonic Music	11
2.2.2	Multiple Viewpoint Models	12
2.2.3	Geometric Modeling	13
2.2.4	Multidimensional Point Sets	14
2.2.5	Models based on Formal Grammars and Syntax Trees	15
2.2.6	Graph-based representations	18
2.3	Generative Algorithms	19
2.3.1	Compression algorithms	19
2.3.1.1	Lempel-Ziv 77 and 78	19
2.3.1.2	Burrows–Wheeler transform	20
2.3.1.3	SIA-based algorithms	20
2.3.2	Genetic Algorithms	21
2.3.3	Markov Models	22
2.3.3.1	N -grams	22
2.3.3.2	Hidden Markov Models	23
2.3.3.3	Variable Order Markov Models	23
2.3.4	Factor Oracle	24
2.3.5	Variable Order Markov Oracle	26
2.3.6	Deep Learning Techniques	26
2.4	Summary	27

3	Encoding Information from Symbolic Music Manifestations: A Multiple Viewpoint Model Approach	29
3.1	Unpacking a Musical Score	30
3.2	MusicXML	32
3.3	Music21	33
3.4	Abstracting multiple viewpoints from parts	33
3.4.1	Part Segregation: Separation of Voices	33
3.4.2	The Part Events' Viewpoints	34
3.4.2.1	Metadata Viewpoints	34
3.4.2.2	Basic Viewpoints	34
3.4.2.3	Duration Viewpoints	36
3.4.2.4	Pitch Viewpoints	36
3.4.2.5	Expression Viewpoints	36
3.4.2.6	Time Viewpoints	37
3.4.2.7	Key Viewpoints	37
3.4.2.8	Phrase Viewpoints	37
3.4.2.9	Derived Viewpoints	38
3.5	Abstracting multiple viewpoints from inter-part dependencies	39
3.5.1	Metadata and Duration Viewpoints	40
3.5.2	Key Viewpoints	40
3.5.3	Chord Viewpoints	40
3.6	From part events to a new Musical Score	42
4	My Musical Suggester	45
4.1	Architecture	45
4.2	Representation Sub-Module	47
4.2.1	Multiple Viewpoint Models	47
4.2.2	Segmentation of events	49
4.2.3	Multiple Viewpoint Weights	50
4.3	Generation Sub-Module	51
4.3.1	Variable Order Markov Oracle	52
4.3.2	Generation of sequences	55
4.3.2.1	Single-Part Generation	56
4.3.2.2	Generation of synchronized Multiple Parts	58
4.3.2.3	Ordering the Generated Sequences	62
4.4	Interface Module	62
4.4.1	Standalone Application	62
4.4.2	The Database Menu	63
4.4.3	The Viewpoints Menu	64
4.4.4	The Generation Menu	66
5	Evaluation and Results	69
5.1	Evaluation protocols	69
5.1.1	Task-Oriented Test	69
5.1.2	Creative Support Index	71
5.2	Results	73
5.2.1	Task-Oriented Test	73
5.2.2	Creative Support Index	73

6	Conclusions	77
6.1	Summary	77
6.2	Contribution	77
6.3	Future Work	78
	References	79
A	Table of Viewpoints	85
B	Creativity Support Index Form	95
B.1	Part I	96
B.2	Part II	97

List of Figures

2.1	The first four bars of J. S. Bach's Courante of his Suite No. 1 in G major, BWV 1007	10
2.2	Geometric Modeling for the first four bars of J. S. Bach's Prelude Courante of his Suite No. 1 in G major, BWV 1007	15
2.3	Multidimensional Point Sets projection of onset time and chromatic pitch for the first four bars of J. S. Bach's Prelude Courante of his Suite No. 1 in G major, BWV 1007	16
2.4	Multidimensional Point Sets projection of onset time and diatonic pitch for the first four bars of J. S. Bach's Prelude Courante of his Suite No. 1 in G major, BWV 1007	17
3.1	Organization of All Viewpoints. Dotted arrows represent relations of Viewpoint categories to events and full arrows represent dependencies between Viewpoint categories.	30
3.2	Part division of the first six measures of J. S. Bach's Fugue I in C Major, BWV 846 from the Well Tempered Clavier (First Book).	31
3.3	Part division of the first two measures of J. M. Whyte's We shall hear a voice, an immortal voice (1890).	31
3.4	Part division of the first four measures of the 3rd Movement (Rondo Alla Turca) of Mozart's Piano Sonata No. 11, K.331.	32
3.5	Voice making in a passage with 2-note chords that should be divided in two voices	33
3.6	Organization of Part Viewpoints, by category.	35
3.7	Original and Chordified Versions of the first five measures of J.S. Bach's Cantata, "Schwingt freudig euch empr", BWV 36	40
3.8	Organization of Inter-Part Viewpoints.	41
4.1	General Architecture of My Musical Suggester	45
4.2	Structure of My Musical Suggester 's Representation Sub-Module.	46
4.3	Structure of My Musical Suggester 's Generation Sub-Module.	47
4.4	Structure of My Musical Suggester 's Interface Module.	47
4.5	Process of Extracting the Multiple Viewpoints for both Part and Inter-Part Events, starting from Part extraction to normalization of features.	48
4.6	VMO in 4.6b constructed with the duration length information of the notes in the score in 4.6a.	53
4.7	VMOs constructed with different threshold values from the same sequence and same feature weights.	55

4.8	Three Conditions in a generation from a single oracle, that started with a threshold LRS of 2 and probability of taking forward links of 26%. The algorithm used to choose sfx is the one that chooses uniformly among all the possible suffix (or reverse suffix) links given the current state. The current state is annotated by a green box and the next real state by a blue circle. In the jump by suffix links, a red box is used to annotate the state to which the jump occurs.	59
4.9	A multiple part model, in which the last VMO, surrounded by a green box, matches the artificial line extracted from inter-part information.	60
4.10	In Figure 4.10a, we can observe a block identification for a forward transition, where the current state is the one in the green box and the transition chosen was the one made by the blue arrow in the artificial line. The block stored is the one in the blue box. In Figure 4.10b, we can see the respective musical score.	61
4.11	General Features of the GUI.	63
4.12	The Database Menu with music on the database path, and a few music selected to parse.	64
4.13	The Viewpoints Menu after clicking on <i>View Automatic Weights</i>	65
4.14	The Generation Menu after clicking on <i>Create Oracle</i>	66
5.1	Instructions of First Task.	70
5.2	Instructions of Second Task.	72
5.3	CSI average score by factor, measured from 0 (less relevance of the software in this factor) to 20 (higher relevance of the software in this factor).	74
5.4	Average values of factors importance, measured from 0 (less interest) to 5 (higher interest).	75
B.1	Part I of Creativity Support Index	96
B.2	Part II of Creativity Support Index	97

List of Tables

2.1	Different possible <i>Pitch Encodings</i> for the example in Figure 2.1, using the various possible techniques presented in Section 2.2.1.1, S.1.	11
2.2	Different possible <i>Rhythm Encodings</i> for the example in Figure 2.1, using the various possible techniques presented in Section 2.2.1.1, S.2.	12
2.3	List of possible basic and derived viewpoints and their description	13
2.4	List of some basic and derived viewpoints for the first two bars' events of Figure 2.1	14

Abbreviations

AO	Audio Oracle
CAAC	Computer-aided algorithmic composition
CSI	Creativity Support Index
FO	Factor Oracle
VMO	Variable Markov Oracle

Chapter 1

Introduction

1.1 Context and Motivation

The mid-20th century witnessed the emergence of an area of study that focus on the automatic creation of musical content by computational means. Early examples focus on offline processing of musical data. Gradually, new communities have been established with their focus on musical informatics in societies such as Sound and Music Computing (SMC)¹, the International Computer Music Association (ICMA)² and the International Society for Music Information Retrieval (ISMIR)³. Recently, those communities have expanded towards interactive online musical creation. Furthermore, a recent trend stresses the importance of computer-assisted technology, which promotes a human-in-loop approach by offering multiple suggestions to a given creative problem [78].

In light of the creative challenge most authors face when addressing a new work, independent of the support they use to express their creativity, is a possible lack of creative flow once they stare at a blank page. Unlocking this initial process with some initial co-creative brainstorming between human and computer can help a fluid craft of a narrative. In this context, our research aims to foster new software tools for creative support systems, where algorithms can collaboratively participate in the composition flow and assist in the process. As composers, we often turn to support tools that help us in achieving new ideas in order to overcome blockages and frustration. We feel that there is not yet a tool that does it in an efficient manner, as most that can be encountered in the actuality have not reached the artistic community in the form of applications, just small research systems, and those that have, do not allow for transparent solutions, adapted to the needs of the user at different times, without requiring a great degree of learning involvement. In that sense, we propose to develop a co-creative tool that unlocks the creative potential of a composer, offering real-time feedback, through the suggestion of musical sequences that can foster new ideas to populate and disrupt the creative process.

As the creative process is individual and subjective, as a composer writes using their own interests, language and techniques, the system has to learn from variable-length data, as a user

¹<http://www.smcnetwork.org/>

²<http://www.computermusic.org/>

³<https://ismir.net/>

may want to learn from a small number of notes or measures or a database of musical pieces (either their own or from other composers) and it cannot only learn from a certain style, as each composer has its manner of creating.

1.2 Problem Definition and Objectives

Following that view, our work aims to answer the following research question: *how to find a balance between the universes of computational representations of a variable musical context, abstract the patterns that best define it and generate both style-coherent, novel, and diverse solutions from them.*

To answer this question, and in a search for supporting the unlocking of the creative composition process, we defined the following three objectives for our research:

- To develop a system for creative support that learns from musical data of variable size and structure and that provides real-time feedback during the composition process;
- To study the representations that abstract temporal patterns of hierarchical and multidimensional musical structures, and
- To implement a system that generates multiple solutions, e.g. variations and continuations, for a given musical context, ordered by their similarity to the original, so that the user can make his choice or even, from time to time, provoke the composer to come up with new ideas.

1.3 Methodology

To this end, we intend to develop a knowledge-support system that uses the *multiple viewpoint models* suggested by Conklin and Witten [21] to encode information from symbolic musical surfaces. We choose this representation structure as they are the ones that allow for the best coding of structures with flexible information, due to the manner in which they incorporate derivative information from the basic data with a higher level of abstraction and hierarchy (for example, the beginning and end of musical phrases and metrics).

We will then combine these representations with the *Variable Markov Oracle* automata to study an optimal balance between the amount and typology of information encoded in the multiple viewpoints and its implications in the model's generative capabilities. Adapting the amount and typology of musical information is sought have important creative bias in the generative output.

The resulting application, **MyMusicalSuggester** is planned to be manifested as a standalone software or as a plugin for the MuseScore⁴ notation software, given the familiarity of the target audience with this environments. That way, the users can ask for and receive suggestions to continue their compositions directly in their notation software. Due to the subjective nature of

⁴<https://musescore.org/en>

the composition process and the possible lack of existing content that suits a given composer, the use of a style-driven datasets from which **MyMusicalSuggester** learns musical structures is not mandatory. Instead, the composer can use a stylistically varied database that they can compile by parsing a collection of works of their choice. Alternatively, composers can use the datasets supplied with the application. The user has the possibility of guiding the whole process, in order to achieve results better adapted to their needs.

Finally, we will evaluate **MyMusicalSuggester** in the field, by developing and realizing tests with composers and analysing their comments, considering the creative potential of the application as a support system for the composition activity, as well as applying the Creative Support Index test for comparison of results.

1.4 Document Structure

This dissertation includes six chapters.

Chapter 1 presents an introduction to the context, the motivation and objectives of the research documented in this dissertation.

Chapter 2 comprises a review of related literature, divided into three topics: 1) an historical perspective of knowledge support musical systems (primarily ones that endorse computer-aided algorithmic composition) that have been implemented with similar objectives, 2) representations that abstract information from musical sequences, and, lastly, 3) algorithms that extract temporal patterns from these representations, from which generation of new musical sequences can be done.

Chapter 3 details the encoding of multiple-viewpoints from symbolic music manifestations along with the integration with existent libraries for element-extracting.

Chapter 4 presents a detailed architecture of **MyMusicalSuggester**, expanding on its various modules and the connections between them.

Chapter 5 explains the evaluation process and discusses the results, as well, as the potential of the system in the creative musical field.

Finally, Chapter 6 presents a summary of the research, its main conclusions and highlights its original contributions. Moreover, it also promotes future challenges and open questions related to the work.

1.5 Publication

This research led to a paper that was accepted for presentation and publication in the proceedings of the International Conference on Computational Creativity, ICCCC 2020⁵ as:

- Carvalho, N., Bernardes, G. (2020). Balanced Tunes: A review of symbolic representations and the modeling techniques that capture temporal hierarchy of musical surfaces. Coimbra, 2020.

⁵<http://computationalcreativity.net/iccc20/>

Chapter 2

Knowledge Support Systems for Music: A literature review

Music informatics has been historically exploring the notion that musical structure can be modeled and predicted algorithmically, taking advantage of information theory principles and the postulate of music as a low entropy phenomenon. Modeling and predicting or generating musical structures is built upon the temporal and hierarchical nature of those structures, by typically informing the algorithms using a sequence of past events [20]. Different degrees of inter-dependency can be captured by these models across the *adopted representation* of the constituent elements of the musical surface as discrete and finite alphabets. Algorithmic models capture different traits from the musical surface, depending on the adopted intra- and inter-opus musical material. These range from tonal music principles to stylistic idiosyncrasies of a composer or the recurrent patterns in a composition.

The balance between familiarity to known compositional traits, captured by these algorithmic methods and the introduction of varied, unfamiliar and unpredictable structures is of utter importance in the design of generative systems. This balance can be represented by the Wundt curve, a hedonic function which relates the levels of novelty and expectation to the 'pleasantness' of creative work [9].

As Roads states in [72, p.3], "a central task of composition has always been the management of the interaction amongst structures on different time scales." Roads [72] identifies the following nine time scales of music: *Infinite, Supra, Macro, Meso, Sound Object, Micro, Sample, Subsample* and *Infinitesimal*. From an operational standpoint, we will be working on the *Meso* and *Sound Object* scales, as we are interested in the divisions of musical form that group the basic units of musical structure into hierarchies of phrase structures. In greater detail, we seek a representation at the *Sound Object* level in order to abstract patterns formed in the *Meso* time scale.

The review of the literature for this research has been divided into three main sections. In

Section 2.1, we review existing knowledge support systems in the domain of computer-aided algorithmic composition. In Section 2.2, we introduce the problem of analyzing and representing symbolic musical structures, where a piece of music is transformed into a higher-level description derived from the basic surface representation [20]. Finally, in Section 2.3, we present ways to automatically discover recurrent musical patterns from the representations, to generate style-coherent, novel and diverse musical sequences.

A large bulk of the literature review relative to the representation of symbolic musical structures is also presented in Chapter 3 due to its instrumental role in this research.

2.1 Knowledge Support Music Systems

The first experiments in the field of music informatics date from the middle of the 20th-Century, about a decade after the first electronic digital computers. Early research in the field of gained special momentum in 1956 when the first work composed by a computer system was presented on the television [4]. Over the last seventy years, this area of study has expanded greatly across the following two major lines of research:

- Statistical models combined with discrete-event systems, such as Markov chains.
- Deep Learning techniques, provided by the introduction of tools such as Magenta, a library for python and javascript that interfaces with Tensorflow models, specifically for music.

Before reviewing these systems, some important concepts on music informatics should be presented to better introduce the reader to the context of this dissertation.

2.1.1 Important Concepts in Music Informatics

Generative music was a term coined by Brian Eno in 1995 to describe any music created by a system, that is ever-different and changing [31]. This concept has been interpreted in different manners: from a linguistic approach, one can create structures using analytic theoretical constructs that are explicit enough such as generative grammars; from an interactive field, it refers to the non-use of discernible musical inputs during the generation process; and, from a biological perspective, it means a non-deterministic, non-repeatable music [91].

Procedural music refers to musical structures generated from abstract set of rules set in motion by a human composer but without its direct involvement. Audible outcomes are typically less relevant as the algorithms behind them, i.e., the generative process [72]. This is especially interesting within contexts where it is necessary to create music that can change or respond to different states or events at varying degrees, usually in real-time.

Computer-aided composition (CAC) refers to the generation of musical content by computational means and it allows composers to design computer processes in order to generate musical structures. It typically manifests as "offline" rendering processes [12]. However, these systems

lack the specificity of using generative algorithms, and we can include in its examples notation or sequencing software, as they aid the composers in their creative process by computational means.

Computer-aided algorithmic composition (CAAC) systems are inspired by algorithmic processes, used in Western music since several centuries ago, and implement them in computer software that facilitates the generation of new music by means other than the manipulation of direct music representation [3]. These systems expand on compositional design models by computational resources, allowing for the (semi-) automatic employment of various algorithmic techniques at different levels of the composition process [4].

Since 1995, a great number of CAAC systems have been proposed with the aim of facilitating the generation music by means other than the manipulation of direct music representations. They can be characterized by seven properties [3]:

1. The level of musical structures produced by the system;
2. The relationship between the computation of musical structures and their output;
3. The proximity of a system to a particular musical idiom, style, genre, or form; 4) its possibility for extension;
4. The proximity of a system to a particular musical idiom, style, genre, or form; 4) its possibility for extension;
5. The type of event production that it offers (either generation of events from indirect music representations or transformation of direct music representations);
6. The sources for producing sound and event data; and
7. The environment that exposes the system's abstractions to the user, by presenting an artificial language to their design and configuration, only allowing the user to provide input data for processing, without more involvement or an interactive system that allows the user to issue commands and receive responses to which they can answer with a new command.

2.1.2 An Overview of CAAC Systems

In this section, we will be reviewing CAAC systems, taking into account the techniques they use for abstracting musical structures and their temporal models.

2.1.2.1 CAAC Systems that use Statistical Modeling

A representative first example of CAAC is Hiller and Isaacson's (1958) *Illiac Suite*, composed using rule systems and Markov chains. Inspired by this work, a library providing standard implementations of the various methods used for composing was created by Baker in 1963, called MUSICOMP [2].

Starting in the early 1960s, Xenakis, renowned for his use of manually-drafted stochastic algorithms in his compositions, starts adopting computers to make these methods automatic [2].

Koenig followed similar trends and implemented some techniques, such as Markov chains to automate the generation of music, which were lately compiled in a collection of tools to facilitate various aspects of algorithmic composition, in the AC Toolbox software [2].

In 1980, Cope developed Experiments in Musical Intelligence (EMI). The system was based on generative models to analyze existing music and create new pieces based on them [38].

AthenaCL, the system of Ariza [4] was developed in 2005 as a software tool for creating musical structures, using Markov transitions.

CACIE emerged in 2007 by the hand of Daichi and Iba [25] to be a system for helping composers in the process of composing atonal works. It uses Tree representations of musical phrase of music in midi format and an evolutionary (genetic) system for generating the new pieces. It allowed for the direct manipulation of the trees by the users as well as the editing of the chromosomes.

Morpheus is a music generation system proposed in 2016 by Herremans and Chew [37] that uses pattern detection techniques (COSIATEC) to find repeated patterns in a template piece, which are then used for applying constraints to the generation process, guided by an efficient optimization algorithm based on evolutionary systems for a new polyphonic composition. They conclude that the quality of the musical output could be improved by imposing more constraints such as those related to playability and to the statistical properties of a style of music.

FlowComposer [63] uses Markov chain models for automated composition of musical lead sheets including harmonization. The musical structures are encoded by formal string representations captured from MIDI information.

Another typical use of the statistical modeling CAAC systems is as improvisation mechanisms that interact and follow a musician in real-time.

GenJam [10], developed in 1994 was proposed as a Jazz Improviser that builds choruses of MIDI events on the tune of the music, by means of genetic algorithms.

Other project that builds upon human-machine co-improvisation is The OMax Project. It consisted of a system to learn, in real-time, features of the style of a musician. The study for the system started in 1998, but it was implemented in OpenMusic and Max and made available to the public in 2004 by Gerard Assayag, Shlomo Dubnov, M. Chemillier and G. Bloch at IRCAM. Since its inception, OMax has evolved in two directions: ImprobeK, which adds a notion of narrative to the improvisation and SoMax that extends on a context-guided improvisation. The DYCI2: Dynamics of Creative Improvised Interaction) is now trying to compile the three versions on a single program for Max/Msp [6].

The Continuator [61] was proposed in 2003 as a system that "bridges the gap between interactive musical systems, limited in their ability to generate stylistically consistent material, and music imitation systems, which are fundamentally not interactive" [61].

Freely Improvising, Learning and Transforming Evolutionary Recombination (FILTER) by [60], uses both FOs and HMMs to learn temporal structures and a fitness function to define the balance of novelty vs. repetition of the generated sequences.

2.1.2.2 CAAC Systems that use Deep Learning

The use of deep learning has been growing in the last few years in the musical community. One particular application is in the musical creation field, where it builds on the more recent techniques, applied to available corpora, to automatically learn musical styles for generating new musical content, proportioned by the introduction of tools such as Magenta [?].

AIVA, created also in 2016, proposes a system that can compose emotional soundtracks for ads, video games or movies by using deep learning techniques.

BachBot, implemented in 2016 by Feynman Liang proposes the use of LSTMs to create a system in python able to compose choral music such as J. S. Bach [45]. The results were good enough that just half the time a user distinguished the original Bach compositions from the generated by the system.

2.2 Computational Representation of Symbolic Musical Structures

Representing the content of musical structures in a symbolic format is not a menial task. Firstly, pieces of music are not physical objects, and so have no single ‘ground’ manifestation [50]. Secondly, we have to consider that the attributes that define musical content are diverse and vary in the application domain, even when considering only symbolic musical structures. For example, the information to learn from a musical score of an unpitched percussion instrument differs from that of a pitched one. In the context of pattern finding, music is typically a low entropy phenomena, from an information standpoint, as repeating content is typically adopted, such as similar intervals and melodic contour, repeating rhythmic values or chord progressions, confined registers, timbre and instrumentation, dynamics, pitch-class sets, and so on [20].

Two generative and reductive theories have been the basis for guiding previous works on creating structure-rich representations for music: The Schenkerian Analysis (1906-1935) and the Lerdahl and Jackendoff’s Theory (1983). Schenkerian analysis aims at finding structural dependencies among the events of a musical composition, instead labeling individual objects in a musical score by examining "interrelationships among melody, counterpoint, and harmony" in a hierarchical manner, identifying and extracting the notes with more structural significance. [40, 16] In the Schenkerian theory, the *Ursatz* is the fundamental structure, as it appears over most of the work being analyzed and in various levels of its hierarchy.

On the other hand, Lerdahl and Jackendoff attempt to describe the structure of music from a linguistic approach by abstracting preference-rule systems (grammars) that describe a musical work. As these rules lack weight properties that serve to distinguish the importance of each rule and are mainly focused on vertical segmentation of the musical surface, this system is described by its authors as an incomplete representation. In this section, multiple computational methods for encoding attributes from symbolic musical structures are reviewed and discussed. Provided examples encode the musical excerpt shown in Figure 2.1.



Figure 2.1: The first four bars of J. S. Bach's Courante of his Suite No. 1 in G major, BWV 1007

2.2.1 String Modeling

Encoding the component objects of a music manifestation as strings is one of the oldest computational techniques for representation symbolic music structures. We address monophonic and polyphonic encode techniques separately, in Section 2.2.1.1 and Section 2.2.1.2 as they present different difficulties, specially the second, because of their properties.

2.2.1.1 String Modeling of Monophonic Music

A common string-based encoding combines pitch and rhythm information as linear sequences, when encoding monophonic music. In his thesis, Rizo [70] defines a formal string representation framework for monodies in which strings are defined over a finite alphabet Σ and the empty symbol is denoted by $\lambda \notin \Sigma$. Each symbol in a string X is notated for its i -th position in the string as X_i and a substring as $X_{[i,j]}$, with i and j the positions of the start and end of the substring. In these representations, pitch and rhythm are encoded as explicit symbols, belonging to the respective alphabets, Σ_p and Σ_r . Each note event includes a combination of two symbol, one per alphabet. Rizo [70] equally summarizes the possible alphabet encoding techniques for pitch and rhythm as follows:

S.1 Pitch Encoding

The pitch encoding alphabet, Σ_p can follow different approaches such as the Common Music Notation (p_{cmn}), where the note is encoded as a tuple in which the first element is the name of the note (C-B notation), the second is the accidental and the third is the octave. Other encoding approaches are base- n representations (the most used n values are 7, 12, 21 and 40) that encode notes as integers in the $[0, n]$ interval, with 0 being considered the rest in most of them, Interval and Interval from tonic (p_{itv} and p_{ift}) that use the relative intervals between successive notes or to a specific note (the tonic) to map the integer value referent to a certain note and relative pitch contour (normal or extended) that only represents the motion direction of the pitches in the three possible simple ways (unison, ascending, or descending) or using a discrete interval space for better definition. The differences in pitch encoding for the chosen example are presented in Table 2.1.

S.2 Rhythm Encoding

For the rhythm encoding alphabet, all temporal dimensions of notes are considered. They can be represented by taking into account the absolute value of time (from 0 to end of the piece) or duration, for which it is used a quantized resolution of time in which '16th' is the

Pitch Encoding	Encoded Sequence
Common music notation (p_{cmn})	(G, \sharp ,3) (G, \sharp ,3) (D, \sharp ,3) (G, \sharp ,2) (B, \sharp ,3) (C, \sharp ,4) (D, \sharp ,4) (D, \sharp ,4) (B, \sharp ,3) (A, \sharp ,3) (B, \sharp ,3) (D, \sharp ,3) (G, \sharp ,2) (G, \sharp ,3) (A, \sharp ,3) (B, \sharp ,3) (G, \sharp ,3) (E, \sharp ,3) (C, \sharp ,3) (C, \sharp ,2) (A, \sharp ,3) (B, \sharp ,3) (C, \sharp ,4) (B, \sharp ,3) (A, \sharp ,3) (G, \sharp ,3) (F, \sharp ,3) (D, \sharp ,3) (D, \sharp ,2) (D, \sharp ,3) (E, \sharp ,3) (F, \sharp ,3) (G, \sharp ,3) (A, \sharp ,3) (B, \sharp ,3)
Base-12 (p_{12})	8 8 3 8 12 1 3 1 12 10 12 3 8 8 10 12 8 5 1 1 10 12 1 12 10 8 7 3 3 3 5 7 8 10 12
Base-21 (p_{21})	13 13 4 13 19 1 4 1 19 16 19 4 13 13 16 19 13 7 1 1 16 19 1 19 16 13 11 4 4 4 5 11 13 16 19
Base-40 (p_{40})	26 26 9 26 38 3 9 3 38 32 38 9 26 26 32 38 26 15 3 3 32 38 3 38 32 26 21 9 9 9 15 21 26 32 38
Interval (p_{iv})	0 0 7 0 4 5 7 5 4 2 4 7 0 0 2 4 0 9 5 5 2 4 5 4 2 0 11 7 7 7 9 11 0 2 4
Interval from tonic (p_{ift})	In this case, it is the same as p_{iv} because the first note is also the tonic.
Contour (p_c)	0 0 -1 -1 1 1 1 -1 -1 -1 1 -1 -1 1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 1 1 1 1 1 1
HD-Contour (p_{hdc})	0 0 -3 -3 4 1 1 -1 -1 -1 1 -4 -3 4 1 1 -2 -2 -2 -4 4 1 1 -1 -1 -1 -1 -2 -4 4 1 1 1 1

Table 2.1: Different possible *Pitch Encodings* for the example in Figure 2.1, using the various possible techniques presented in Section 2.2.1.1, S.1.

most common quantization event and '1' usually corresponds to the elementary 'time' event (r_{tabs} and r_{dabs} , respectively). Contour can also be used with its normal or high definition form, encoding the duration of a note as longer, equal, or shorter than the previous (or next) note. The differences in rhythm encoding for the chosen example are presented in Table 2.2.

2.2.1.2 String Modeling of Polyphonic Music

Lemström and Pienimäki [22] propose three different methods for adopting string modeling in polyphonic textures (two or more concurrent) voices: a *Non-Interleaved*, an *Interleaved* and an *Onset-Based* representations.

The *Non-Interleaved* representation divides the polyphonic texture into several monophonic lines according to composed or perceived voices and then simply represents each by an associated (monophonic) string and stores them sequentially. This representation is most effective when the patterns to be searched resemble an excerpt within one of the voices (Solid Occurrence), but meaningful extraction patterns across all voices cannot be achieved from this representation.

The *Interleaved* representation [68] consists of sorting the notes in lexicographical order, firstly by their onset times and then by pitch values. The method is more flexible than the non-interleaved, yet the strict ordering enforced is not natural. It can lead to big gaps between contiguously perceived notes when using edit distances to quantify how similar two musical segments

Rhythm Encoding	Encoded Sequence
Absolute time (r_{tabs})	0 1/2 1 3/2 2 9/4 5/2 11/4 3 13/4 7/2 4 9/2 5 21/4 11/2 6 13/2 7 15/2 8 33/4 17/2 35/4 9 37/4 19/2 10 21/2 11 45/2 13/2 47/4 12 49/4
Absolute duration (r_{dabs})	1/2 1/2 1/2 1/4 1/4 1/4 1/4 1/4 1/4 1/2 1/2 1/2 1/4 1/4 1/2 1/2 1/2 1/2 1/2 1/4 1/4 1/4 1/4 1/4 1/4 1/4 1/2 1/2 1/2 1/4 1/4 1/4 1/4 1/4 1/4
Contour (r_c)	0 0 0 0 -1 0 0 0 0 0 1 0 0 -1 0 1 0 0 0 0 -1 0 0 0 0 0 1 0 0 -1 0 0 0 0 0
HD-Contour (r_{hdc})	In this case, it is the same as r_c because there is only changes between close rhythm durations.

Table 2.2: Different possible *Rhythm Encodings* for the example in Figure 2.1, using the various possible techniques presented in Section 2.2.1.1, S.2.

are. If transposition invariance is required, this representation is highly inefficient as it requires computing the brute-force solution of all possible transpositions.

The *Onset-Based* representation [42] may be used if notes having simultaneous onsets are modeled instead of individual notes. Careful algorithmic planning is required to avoid the combinatorial explosion while aligning the monophonic voices based on the onsets. When they happen simultaneously, the notes are grouped. Comparison of musical sequences can be done straightforwardly with this approach, in cases where transposition invariance is not required. As the duration information is lost, some authors have attempted to solve this problem by fragmenting long notes into notes of fixed duration connected by ties as in [36].

2.2.2 Multiple Viewpoint Models

The multiple viewpoint models emerged from a necessity of extending the application of statistical modeling of music to domains where the internal structure of the events is an important factor. They are adaptive, in the sense that a representation of a particular piece will change as that piece progresses [21]. Pearce implemented a multiple viewpoint model in Lisp, called IDyOM (Information Dynamics of Music) [66] [65].

The viewpoints use background domain knowledge to derive new ways of expressing events in a sequence by introducing types (represented as τ) as abstract properties of the musical events and for each, associating a partial function Ψ_τ that maps the events to this type. A viewpoint comprises one such function and the set of all sequences that can be represented using elements of the respective type. A multiple viewpoint model is comprised of a collection of different viewpoints.

As the viewpoints can have correlations, a new type was introduced: the product type ($\tau_x \otimes \tau_y$), whose elements are the cross product of the constituents. For inferring to which viewpoint an event

τ	description	$[\tau]$	Derived from
st	start-time of the event	$\{0, 1, 2, \dots\}$	st
pitch	pitch	Z	pitch
duration	fraction of duration	$\{1, 2, 3, 4, 6, 8, 12, 16\}$	duration
keysig	key signature	$\{-4, \dots, +4\}$	keysig
timesig	time signature	$\{12, 16\}$	timesig
fermata	is (not) in fermata	$\{T, F\}$	fermata
deltast	rest or not	$\{T, F\}$	st
accent	accentuation	Z	accent
voice	voice to which it belongs	Z	voice
timbre	instrumentation	Z	timbre
gis221	difference to start-time	$\{1, 2, \dots\}$	st
posinbar	position of event in bar	$\{0, \dots, 15\}$	st
fib	initial event in a bar	$\{T, F\}$	st
seqint	sequential melodic interval	Z	pitch
contour	same, above or under the last event	$\{-1, 0, 1\}$	pitch
hdcontour	contour with extended metric	$\{-4, \dots, 4\}$	pitch
referent	referent of piece	$\{0, \dots, 11\}$	keysig
intfref	vertical interval from referent	$\{0, \dots, 11\}$	pitch
inscale	in/not in scale	$\{T, F\}$	pitch
intfib	interval from fib	$[\text{seqint}]$	pitch
intfip	interval from fip	$[\text{seqint}]$	pitch
intphbeg	interval from beginning of phrase	$[\text{seqint}]$	pitch
thrbar	seqint at bars	$[\text{seqint}] \times Z^+$	pitch, st
lphrase	length of phrase	Z^+	fermata, st
thrph	seqint at phrases	$[\text{seqint}] \times Z^+$	pitch, st
thrqu	seqint at quarters	$[\text{seqint}] \times Z^+$	pitch, st

Table 2.3: List of possible basic and derived viewpoints and their description

belongs, it is necessary to compute the probability of belonging to that context, which involves converting the event's surface string to the sequences belonging to that viewpoint. That is done by using a function Φ_τ that is empty for an empty event and the extension of all elements in the viewpoint until that moment plus the current event if defined.

In a musical context, we can have several types of basic and derived elements. A summary of possible viewpoints can be seen in Table 2.3.

For the first 2 bars of the example of Figure 2.1, a set of multiple viewpoints that could be extracted is represented in the table 2.4.

2.2.3 Geometric Modeling

Proposed by Maidín in 1998 [48], the geometric representations for music present a technique for creating a 2-dimensional graphic representation of the pitch-duration contour of a music score by plotting the pitch of the notes versus their duration on a sequential timeline, with a similar result as a piano-roll sheet. This technique allows for the use of mathematical concepts, such

Type	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10	e11	e12	e13	e14	e15	e16	e17
st	10	12	14	16	18	19	20	21	22	23	24	26	28	30	31	32	34
pitch	55	55	50	43	59	60	62	60	59	57	59	50	43	55	57	59	55
duration	2	2	2	2	1	1	1	1	1	1	2	2	2	1	1	2	2
keysig	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
timesig	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
deltast	⊥	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
posinbar	10	0	2	4	6	7	8	9	10	11	0	2	4	6	7	8	10
fib	F	T	F	F	F	F	F	F	F	F	T	F	F	F	F	F	F
seqint	⊥	0	-5	-7	16	1	2	-2	-1	-2	2	-9	-7	12	2	2	-4
contour	⊥	0	-1	-1	1	1	1	-1	-1	-1	1	-1	-1	1	1	1	-1
hdcontour	⊥	0	-3	-3	4	1	1	-1	-1	-1	1	-4	-3	4	1	1	-2
referent	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
thrbar	⊥	0	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	4	⊥	⊥	⊥	⊥	⊥	⊥
thrqu	⊥	0	⊥	-12	⊥	⊥	19	⊥	⊥	⊥	-4	⊥	-16	⊥	⊥	16	⊥

Table 2.4: List of some basic and derived viewpoints for the first two bars' events of Figure 2.1

as area and spatial transformations, when processing music scores, which are particularly useful when computing similarities between two musical sequences. This method can be extended for representing other features of the notes by increasing the number of dimensions used.

For the example of Figure 2.1, a possible geometric representation would be as shown in Figure 2.2 where circled in red, we can observe similarities between small fragments of the melody and in blue, similarities between bigger fragments of the melody.

2.2.4 Multidimensional Point Sets

The idea of using a set of points in a multidimensional Euclidean space to represent music in the context of pattern recognition algorithms is a result of the work of Meredith [53], in order to overcome the limitations of string-based modeling methods which, as described by the author, are inefficient when dealing with polyphonic music (which produces a combinatorial explosion in the number of strings required) or trying to find highly embellished occurrences of a query pattern.

In this approach, the author defines a musical object as a tuple of 5 elements in which the first element indicates the onset time of the note and the second defines the respective chromatic pitch (MIDI pitch - 21) for the note. The third element gives the diatonic pitch, defined by an integer that indicates the position of the head of the note on the staff. The fourth element marks the duration of the note and the fifth, the voice to which the note belongs.

From these point sets, it is possible to do a 2- or 3-dimensional projection, which is particularly useful in searching for patterns as it is possible to visually discover repetitions by analyzing the relative position of the points in the space and the geometric figures they make. Choosing the variables for the projection depends on the kind of pattern we are aiming to find and the style of music we are analyzing. For Tonal Music, a 2-dimensional projection using onset time and diatonic pitch of each note is more relevant than one using chromatic pitch, as two occurrences of

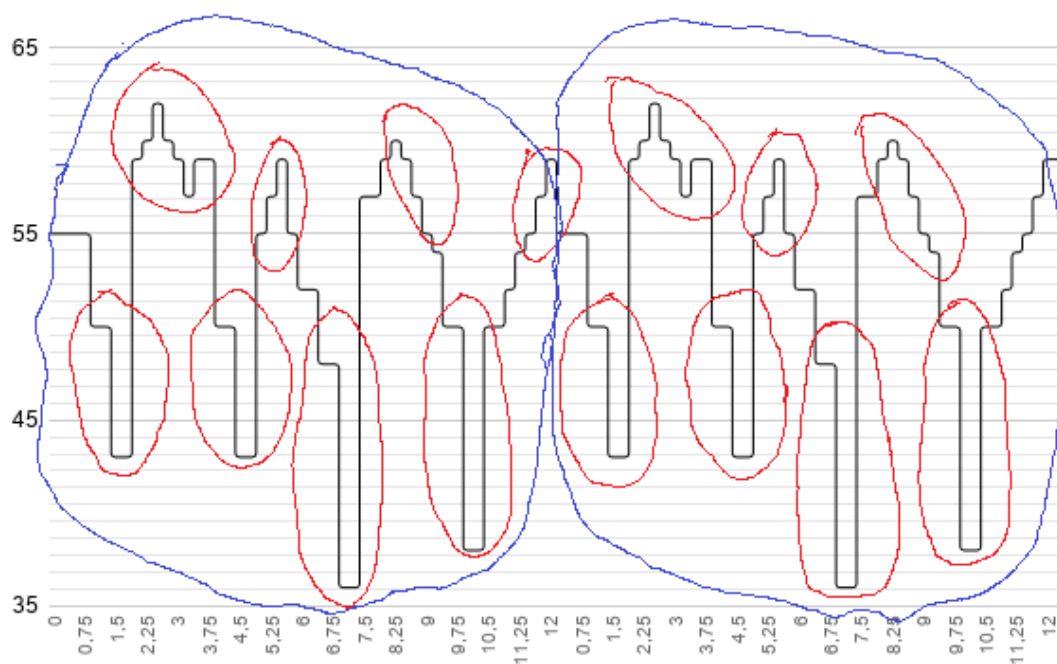


Figure 2.2: Geometric Modeling for the first four bars of J. S. Bach's Prelude Courante of his Suite No. 1 in G major, BWV 1007

a motive on different degrees of a scale might be perceived to be similar even if the corresponding chromatic intervals in the patterns differ. This conversion can be made using a pitch spelling algorithm such as *ps13*, which tries to compute the correct pitch names of the notes in a passage of tonal music when given only the onset-time, chromatic pitch and possibly the duration and voice of each note [54].

For the example of Figure 2.1, a projection of onset time and chromatic pitch and a projection of onset time and diatonic pitch would be as shown in Figures 2.3 and 2.4. The set of multidimensional points for this musical sequence would be:

$$\{ \langle 0, 34, 20, 2, 1 \rangle, \langle 2, 34, 20, 2, 1 \rangle, \langle 4, 29, 17, 2, 1 \rangle, \langle 6, 22, 13, 2, 1 \rangle, \langle 8, 38, 22, 1, 1 \rangle, \langle 9, 39, 23, 1, 1 \rangle, \langle 10, 41, 24, 1, 1 \rangle, \langle 11, 39, 23, 1, 1 \rangle, \langle 12, 38, 22, 1, 1 \rangle, \langle 13, 36, 21, 1, 1 \rangle, \langle 14, 38, 22, 2, 1 \rangle, \langle 16, 29, 17, 2, 1 \rangle, \langle 18, 22, 13, 2, 1 \rangle, \langle 20, 34, 20, 1, 1 \rangle, \langle 21, 36, 21, 1, 1 \rangle, \langle 22, 38, 22, 2, 1 \rangle, \langle 24, 34, 20, 2, 1 \rangle, \langle 26, 31, 18, 2, 1 \rangle, \langle 28, 37, 16, 2, 1 \rangle, \langle 30, 15, 09, 2, 1 \rangle, \langle 32, 36, 21, 1, 1 \rangle, \langle 33, 38, 22, 1, 1 \rangle, \langle 34, 39, 23, 1, 1 \rangle, \langle 35, 38, 22, 1, 1 \rangle, \langle 36, 36, 21, 1, 1 \rangle, \langle 37, 34, 20, 1, 1 \rangle, \langle 38, 33, 19, 2, 1 \rangle, \langle 40, 29, 17, 2, 1 \rangle, \langle 42, 17, 10, 2, 1 \rangle, \langle 44, 29, 17, 1, 1 \rangle, \langle 45, 31, 18, 1, 1 \rangle, \langle 46, 33, 19, 1, 1 \rangle, \langle 47, 34, 20, 1, 1 \rangle, \langle 48, 36, 21, 1, 1 \rangle, \langle 49, 38, 22, 1, 1 \rangle, \}$$

2.2.5 Models based on Formal Grammars and Syntax Trees

In 1979, Curtis Roads [71] presents a small survey on several theories for using formal grammars for representing musical sequences until that moment, such as in [79] where Smoliar implements

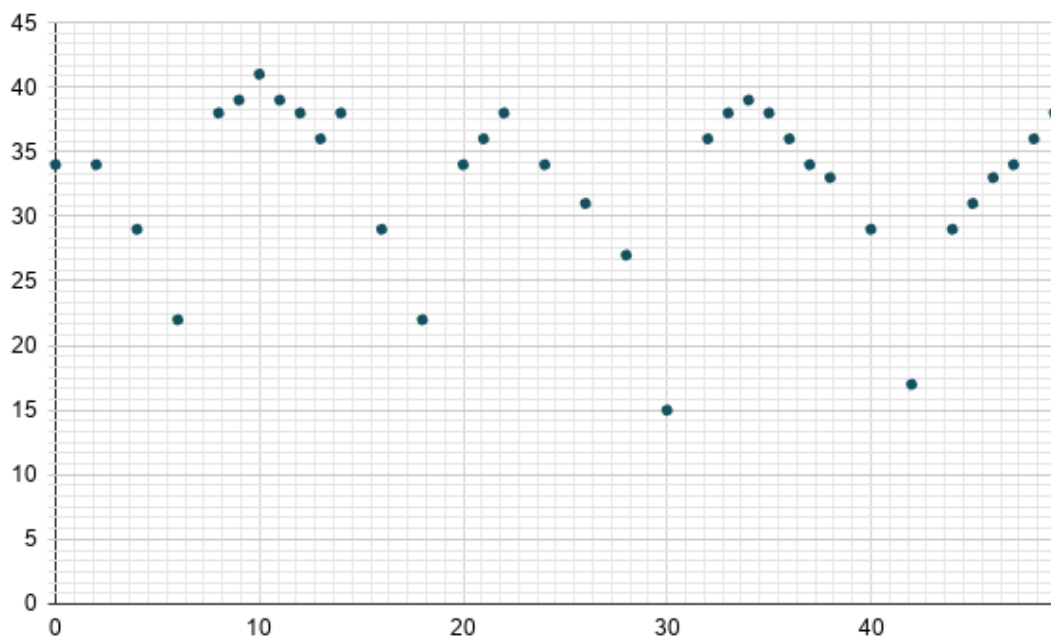


Figure 2.3: Multidimensional Point Sets projection of onset time and chromatic pitch for the first four bars of J. S. Bach's Prelude Courante of his Suite No. 1 in G major, BWV 1007

a language for musical analysis using a tree structure, [57] in which Moorer suggests that context-free grammars may be augmented with procedures to perform transformations to enhance the context-sensitivity and in the Generative Theory of Tonal Music system of Lerdahl and Jackendoff, a grammar attempts to model a "formal description of the musical intuitions of a listener who is experienced in a musical idiom", focusing on four hierarchical systems: Grouping (related to motives, phrases, periods and sections), Metric (regular alternation of strong and weak beats), Time-span reduction (TSR), derived from metrical and grouping structures to at all temporal levels of a work and Prolongational reductions (PR) which describe the moments of tension and relaxation along a piece [43].

He concludes that grammars and parse trees had been found useful formal models for representing the syntax of musical structures as they allow for a very concise manner of modeling as the production rule of a grammar is a powerful abstraction but that they were always considered insufficient by itself, particularly in the context-free form, as the context in music is both parallel and sequential whereas, in the formal grammars, it is only sequential.

Some more experiences were undergone in trying to apply formal grammars for representing musical contexts, as in [51], where Mardsen and Pople proposed a grammar, implemented in the Bol Processor, to model improvisations in North Indian tabla drumming as it is very similar to speech. This type of music is rule-based and its rhythmic patterns are very similar to those encountered in formal grammars. The major limitation presented for this system was that "expert systems represent knowledge at a low (non-hierarchical) theoretical level, and so it was impossible

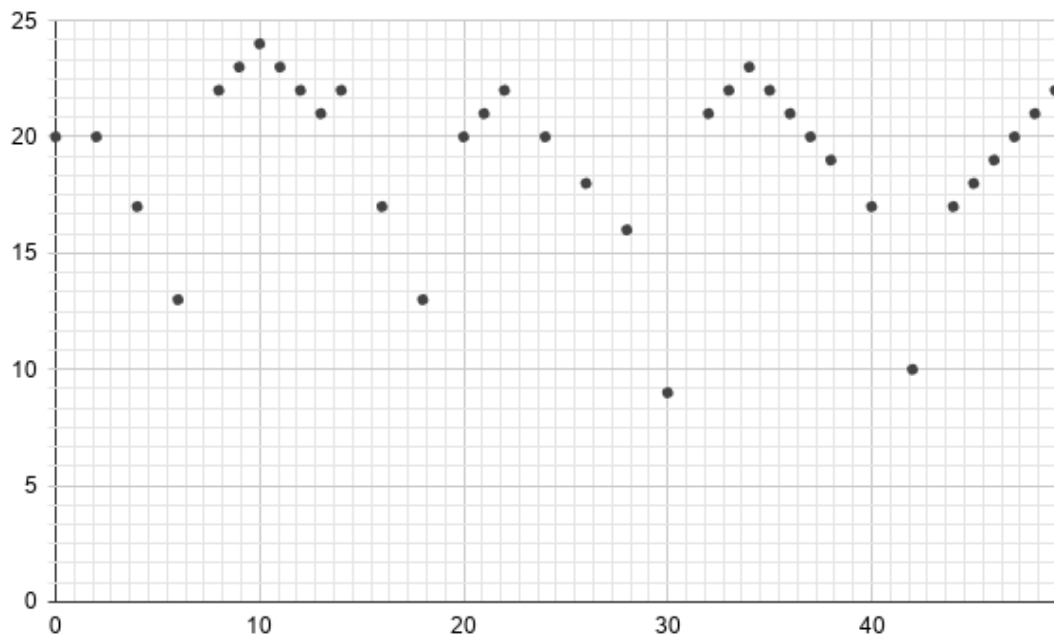


Figure 2.4: Multidimensional Point Sets projection of onset time and diatonic pitch for the first four bars of J. S. Bach's Prelude Courante of his Suite No. 1 in G major, BWV 1007

to separate general analytical statements from specific instances of facts" [51].

As noted by Rizo in [70], the hierarchy of subdivisions of the figure duration as always been represented in theory books, such as in [46], like a tree. For that reason, as a means to understand and model perceived rhythms from notation, several grammars were proposed to, assuming that a listener unconsciously performs this kind of parsing. These can be encountered in a survey done by C. S. Lee. in [41]. However, none of those implemented the grammars in a computational algorithm.

The OpenMusic tool, implemented at IRCAM [7], uses trees for representing the hierarchical nature of subdividing the musical figures and groupings, allowing for the representation of tied figures.

In [11], a grammar learns the segmentation of melodies from the ESSEN corpus, with manually separated phrases, using Data-Oriented Parsing (DOP) and in [33], the parsing of simple monophonic melodies into tree structures using a probabilistic context-free grammar allows for their melodic reduction.

David Rizo, in his dissertation thesis in 2010 [70], presented a novel tree representation for both monophonic and polyphonic music, suitable for doing similarity computation of musical sequences. The construction of the three followed a metrical based hierarchical subdivision of time, similar to that proposed by Lee [41]. The notes are only represented in the leaves of the tree, so a set of rules were proposed for propagating the labels from the leaves upwards, labeling the internal nodes. For polyphonic music, all voices are simply merged in the same tree with

node labels being represented by sets of pitch classes when there are various notes played in the different voices at the same time (chords) and using multi-sets to encode cardinalities of the labels for representing their duration. He concluded that this tree structure is very versatile as it is able to fit more elaborate information than simply notes or rhythm, as the musical form or harmony but presents two main drawbacks related to its tight dependency on the meter structure of the input source, and its difficulty to represent ties, dots, and syncopations.

2.2.6 Graph-based representations

In 2001, Marsden [49] presented his work on describing a melody as the product of successive elaborations of a simple outline. He called his structure an E-Graph, which consisted of places, elaborations and links. The structure is intended as a sequence of at least two places and a pair of places can be connected by an elaboration, generating a new intermediate place, without crossing links, making it interpretable as an acyclic graph, which can be easily converted in a tree representation.

A place (representation of a note) is constituted by time (expressed in terms of a particular metre), pitch (distinguishes between enharmonically equivalent pitches) and articulation (tied or untied, to describe if the pitch was already sounding or not). Some particular functions were developed for dealing with pitch and time, such as `chromaticUp/Down(pitch)`, `stepUp/Down(pitch)`, `arpeggioUp/Down(pitch)`, `octaveUp/Down(pitch)` and `timeDivision(time1, time2)`.

Elaborations are composed of characteristics in metre (relation between metres of places connected by the elaboration to the new place resulting from this elaboration), time (same as metre but relating to time) and pitch, and can be simple or accented. This is most important, as the kind of elaborations allowed will, to a certain degree, define a musical language. There are various kinds of simple elaborations, where a new note is simply inserted between two existing notes, such as shorten (the new place is a rest), repetition (the pitch of the new place is the same as the pitch of the first parent place), anticipation (the pitch of the new place is the same as the pitch of the second parent place), and relating to the style of the music, (chromatic and not) passing notes, arpeggios in the key harmony and octave jumps. The accented elaboration types refer to cases in which the time of the original notes is unchanged, such as delays, suspensions of notes from one time to the other and accented passing notes.

Although this representation was presented as having several potential uses, particularly for pattern recognition in cases when the same pattern can generate different sequences of pitches or intervals because of harmonic or pitch context differences or when a pattern is not on a top-level, it was subsequently abandoned as excessively complex in computational terms, especially considering that it only could represent well single-line melodies and a mechanism for their parsing was required.

Another approach to a graph representation was posteriorly presented in [80], in which the graph intends to model a global, time-independent signature of the melody, with its pitches encoded in the 12-tone system, capturing its global structure while being invariant under transformations such as inversions and retrogradations. New weights were proposed for the edges of the

representative graph, containing information on basic rhythmic features and order of events.

2.3 Generative Algorithms

In their theory of GTTM, Lerdahl and Jackendoff [43] state that

"the importance of parallelism [i.e., repetition] in musical structure cannot be overestimated. The more parallelism one can detect, the more internally coherent an analysis becomes, and the less independent information must be processed and retained in hearing or remembering a piece."

The discovery of repeated patterns is a known problem in different domains, including computer vision, bioinformatics, and music information retrieval (MIR).

As explained in the introduction to this chapter, "discovering the important repetitions in a passage of music is an essential step towards achieving a rich understanding of it" [53]. However, most of the patterns that occur in the musical pieces were not planned by the composers or are not even perceived when listening to it. Thus, a tool for discovery of repeated patterns in a musical surface must be capable of discovering those interesting, relevant patterns.

In this section, we will present the algorithms that have been studied to find the redundancy in musical structures and in particular, those that were later applied in the generation of new musical sequences based on those patterns. Most of these algorithms also apply to text, as the treatment of musical sequences is similar to the treatment of the text.

2.3.1 Compression algorithms

General-purpose lossless compression algorithms use the redundancy of input sequences for compressing them in order to reduce their size, maintaining the maximum possible information from the original. For that reason, it was suggested that they might be useful for finding musically relevant patterns. In the next few sections, some of these algorithms will be explained as well as their use in pattern discovery for music.

2.3.1.1 Lempel-Ziv 77 and 78

The algorithms *Lempel-Ziv 77* (LZ77) [93] and *78* (LZ78) [94] were developed by Abraham Lempel and Jacob Ziv in 1977 and 1978, respectively and are some of the most popular lossless data-compression algorithms.

Both the algorithms work as a dictionary-based approach, creating a dictionary of tokens that appear in the original input sequence. LZ77 does it by creating a search buffer where it stores the offset of positions to move backward to reach the start of a pattern, its length, and the event after the pattern by finding the longest match of a string that starts at the current position with a pattern available in the search buffer. This buffer is constructed over a sliding window so the encoder and decoder only have this data to create and get the references. The larger the sliding window is, the

longer back the encoder may search but all data can be encoded and decoded by incorporating a flexible and easy form of run-length encoding, repeating a single copy of data multiple times until necessary [93]. LZ78 creates an explicit dictionary in the format dictionary[...] = index, event and encodes the input stream by replacing repeated occurrences of data with references to the dictionary which is constructed by searching in it for a match and if exists, replacing the index (that refers to the last match found for that event) for the index of the event being processed. If a match is not found, then a new dictionary entry is created and the index of the last match is reset for the new event being processed. When the end of the input stream is reached, the algorithm outputs the index of the last match.

In the musical domain, they were mostly used in genre-classification contexts such as in [44] or music analysis by encountering patterns as in [47], where various general-purpose compression algorithms were explored in pattern discovery in music for classifying folk song melodies. One of the conclusions taken from this research was the correlation between compression factor and score for all the algorithms, as the best classifications were achieved from the shortest representations.

2.3.1.2 Burrows–Wheeler transform

The *Burrows–Wheeler transform* [15] is an algorithm for preparing the data without loss for data-compression invented by Michael Burrows and David Wheeler in 1994. Its implementation has linear time complexity, using a suffix array for executing a permutation of the input sequence as to bring equal elements closer together, in order of increasing the probability of finding a character *c* at a point in a sequence in case of *c* already occurring near this point. Along with *Move-to-front* coding, this means a better compression effect.

The *Move-to-front* algorithm encodes the string incoming from the *Burrows–Wheeler transform* by building the alphabet of the events in the input sequence by reading it from left to right, and then constructing a vector of the indexes of the sequence events in the alphabet. To ensure reversibility, the algorithm needs to return the alphabet, the index vector and the index of the row of the *Burrows–Wheeler transform*'s resulting matrix corresponding to the original input [47].

2.3.1.3 SIA-based algorithms

The Structure Induction Algorithm (SIA) and the SIATEC (TEC stands for *translational equivalence classes*) were presented in 2001 [56] for discovering maximal repeated patterns in any set of points in Cartesian spaces of any dimensionality, in particular in musical surfaces.

SIA can be described as the computation of a structure set of all the maximal subsets from a dataset of any set of points with any number of dimensions for which can be encountered a metric that gives their total ordering, removing repetition under symmetry for helping prevent waste of effort and duplication of results but admitting unlimited gaps in the patterns found [52].

SIATEC extends SIA by calculating all the occurrences of each of the maximal repeated patterns computed by this algorithm, generating a set of translational equivalence classes. Translational equivalence between patterns happens if one can be obtained from the other only by translation. Each pattern can only be contained in a TEC, as it contains all the translationally invariant patterns [52].

Using the SIATEC algorithm, its authors design COSIATEC, a data-compression algorithm to generate a compressed or efficient representation of a dataset [53]. It works by extracting the TECs resulting from running SIATEC on a dataset and from these selecting the "best" ones that don't overlap. This compression algorithm was used in [53] to find themes and motives in a set of 15 Two-Part Inventions (BWV 772–786) by J. S. Bach. The musical sequences were represented by using the Multidimensional Point Sets and the results generated by COSIATEC for these pieces were concluded to be similar to what one might expect from a thematic analysis of an expert music analyst.

In more recent works, a new extension of these algorithms, RECURSIA-RRT (Recursive translatable point-set pattern discovery with the removal of redundant translators) was proposed, designed to increase the compression factor achieved using any TEC cover algorithm [55].

2.3.2 Genetic Algorithms

Genetic Algorithms model the evolutionary processes without needing domain-specific knowledge of the problem [59]. The general scheme of the algorithm implies the generation of a random starting population of a given number of chromosomes (binary coded symbol strings that represent an individual) and for each chromosome, the calculation of a fitness score using a fitness function that usually represents a rule-based system that examines the ability of a chromosome to fulfill the objective. If a chromosome achieves a score higher than a given threshold, the result is deemed as good enough and the process terminates. Otherwise, the chromosomes with the highest scores are sent to the next generation unmodified or are submitted to a mutation (elements of the chromosome are modified in position) or crossover (elements of the chromosome are crossed with elements of another chromosome, giving rise to new chromosomes that are a mix of the two). Those are the new population for the next phase, for which the process is repeated.

The first use of Genetic algorithms for music generation was documented by Horner [39], using a technique of thematic bridging for specifying the thematic material and delegating the development of the new melodic sequences to the genetic algorithm.

In [64], Papadopoulos and Wiggins achieve a system for the generation of melodies of variable length and rhythmic structure over chord sequences by using an automatic fitness evaluation that evaluated, among other aspects, the similarity between patterns. In the following year, Townsey [83] applied the algorithm to existing melodic material describing 21 melodic features used as the basis for a multidimensional fitness function and mutation procedures.

In 2007, Waschka [89] implemented a system called *GenDash* that employs evolutionary computation to help compose pieces with various instrumentation.

In 2016, a novel evolutionary technique combining Feasible/Infeasible two-population method (FI-2POP) and multi-objective optimization was used to create an extensible framework for music composition called *MetaCompose* [76].

A recent trend in genetic algorithms seems to be the use of artificial immune systems to optimize the objective function that encodes the musical features, by returning multiple candidate solutions, instead of the usual optimal one, which feats better when thinking about musical creation, where we require different solutions to the same problem. *ChordAIS* [58] uses Opt-aiNet (Artificial Immune Network for Optimization) with this finality in order to assist its user in generating realistic and functional chord progressions.

2.3.3 Markov Models

In music generating systems, the use of Markov models to capture the statistical occurrence of features in a particular piece or corpus to generate music having the distributions of those features similar to the ones captured is one of the most explored techniques. The first system that attempted to use Markov models was Pinkerton's "Banal Tune-Maker" [69] in 1956, in which the author analyzed simple nursery songs, taking the sequential probabilities of its notes (First-Order Markov model) and then produced similar using a random walk process. In the next year, n -grams with n in the range 1 to 8 were constructed to analyze a corpus of melodies [14] and then generate similar ones. In the discussion of the results, the authors noted that the use of elevated order values led to sequences most similar to the original, while lower values generate more random results. This problem has been examined by Papadopoulos [62], where the author studies the problems of plagiarism arising from higher-order Markov chains, proposing a mechanism to curb excessive repetitions of source material by implementing a maximum allowable subsequence order in a generated sequence.

2.3.3.1 N -grams

N -grams are a specific type of Markov model in which N corresponds to the total number of preceding symbols plus the current symbol being analyzed.

One of the limitations of these contiguous n -grams is that they offer no alternatives with the dependence of every event only on the respective contiguous neighbors (data sparsity problem) but if all neighbors were permitted, the number of associations between events would explode in combinatorial complexity with the increase of n and the length of the sequence. These limitations are overcome by introducing *skip*-grams.

The *skip*-grams are a generalization of n -grams in which the events processed do not have to be contiguous and the gaps that are skipped over. The maximum length of the skips allowed can be fixed to a threshold in the *fixed-skip* model, where the tokens are only evaluated if they are within a fixed range of skips of the event processed or can follow a *variable-skip* approach, useful for sequences of events that depend on temporal structure, where the events are evaluated if the inter-onset interval(s) (IOI) between them occur within a specified upper boundary [77].

In [77], the authors analyzed the influence of fixed length of skips allowed as well as the IOI for the *variable-skip* approach for reducing sparsity in pattern discovery and prediction tasks while modeling harmony using a corpus consisting of four datasets of Western classical music in symbolic form. They concluded that *skip*-grams significantly outperformed contiguous *n*-grams in discovering cadences.

2.3.3.2 Hidden Markov Models

The Hidden Markov Models (HMM) are from the family of the Markov models and are used to model relations between states that are partially hidden (from which we typically only know part of the information, usually not enough to determine the state). For that, they use a discrete probability distribution at each state which defines the probability of emitting a specific alphabet symbol in this hidden state. These models are defined by a tuple of five elements, (Σ, Q, a, b, π) . Σ corresponds to the finite alphabet of visible symbols, Q is the finite set of states, a and b relate to the mappings defining the probability of transitions between hidden states and emission probability of each visible symbol at a given hidden state, respectively (in range 0 to 1) and π refers to the initial probabilities of the hidden states (in range 0 to 1). For all the states, the sum of all the probabilities of transitions between hidden states and the sum of all the emission probabilities for each visible symbol have to amount to 1, as well as the sum of all the initial probabilities [75].

Some implementations of Hidden Markov Models in automatic generation of musical sequences can be observed in [32], [75] and [92]. The first consists of an HMM implementation of the Forward-Backward, Viterbi, and Baum-Welch algorithms for learning pitch, rhythm, and dynamics from a musical sequence and a generator that goes through the model to create new sequences. In [75], a melodic arc analysis calculates the parameters of an HMM of first, higher, or mixed order by using empirical counts to model the melodic arc and an accompaniment analysis learns chords in the chord progression with their duration. Generation is not achieved by calculating the most probable sequence of notes, but rather by sampling from the distributions obtained from the analysis, using a chosen style and tempo, time signature, scale, and respective instruments are adopted from that style [75]. In their last example [92], the authors consider various Markov models, with special attention to HMMs to solve the task of composing classical piano pieces in the style of the Romantic era. They conclude that the major limitation of the techniques used is that the resulting pieces lack melodic progression.

2.3.3.3 Variable Order Markov Models

Although the Hidden Markov Models provide flexible structures that can model complex sources of sequential data, as explored in the last section, to restrict possible model architectures it is necessary a considerable understanding of the problem domain and a large number of training examples.

The Variable order Markov Models (VMM) extend the Markov chain models, learning from conditional distributions where context lengths $|s|$ vary in response to the available statistics in the

training data, providing the means for capturing both large and small order Markov dependencies based on the observed data. [8]

There are many algorithms for learning VMMs, but all of them are structured using three base components: counting, smoothing (probabilities of unobserved events) and variable-length modeling. The counting component estimates the probability for events by counting the number of occurrences of that event appearing after a certain context in the training sequence. The smoothing component deals with elements that were not observed in the input sequence (zero frequency problem). The third component is where the biggest changes occur from algorithm to algorithm, as they can construct only one or various models and may include a threshold for the considered contexts or calculate the maximal context size from the data incoming. [8]

Some of these algorithms for constructing VMMs include:

- Prediction by Partial Match (PPM), a compression algorithm that needs a threshold value for the maximal Markov order of the model to create and uses mechanisms of escape (a probability for all symbols that did not appear after the context is calculated and the exceeding value is allocated to the symbols have non-zero counts for this context) and exclusion (if a symbol appears after the context it is not counted as part of the alphabet for the escape mechanism) as its smoothing component; [19]
- Context Tree Weighting Method (CTW), works by combining many VMMs of bounded order, calculating a set of probability distributions for each one, using a set of fixed weights; [90]
- Probabilistic Suffix Trees (PST) which attempts to optimize the construction of a single with a known upper-bound VMM. The objective of this algorithm is to identify a good suffix set for a PST tree and to assign a probability distribution over the alphabet for each symbol; [74]
- Bayesian Variable Order Markov Models (BVMM) create a tree of partitions, where the disjoint sets of every partition correspond to different contexts in a similar way to CTWs but the weights are updated at each time taking into account the given new observations. [27]

For the generation of music, [29] implements a variant of PST for creating models for various styles of music and generating new instances of musical sequences that respect each explicit style. The results were more interesting as new transitions were introduced and not just juxtapositions of motifs.

2.3.4 Factor Oracle

The Factor Oracle structure was introduced in 1999 by Allauzen, Crochemore, and Raffino [1] as an acyclic automaton that recognizes at least the factors of a word p , having the fewer number of states as possible (minimum is length of p plus 1) and a linear number of transitions that had its use in optimal string matching algorithms, but were easily extended for computing repeated

factors in a word and for data compression. This oracle is learned on-line, in an incremental way and has linear complexity in time and space.

A sequence of symbols is learned by the model by creating labeled forward transitions (factor links) for sequentially numbered states and non-labeled backward transitions (suffix links) for higher-numbered states to previous ones. These links are constructed by following the ones starting in the last analyzed state ($i-1$) and going backward, creating a suffix link labeled with the symbol being processed from each state to the state i , until a state is reached where there is a direct transition labeled with this symbol or there are no more suffix links to follow. A suffix link is added in i to this state, if existent, or to the initial state if not. All states are considered as accepting states.

In a pattern discovery context, the factor links represent states that produce similar patterns by continuing forward while suffix links correspond to states that share the largest similar subsequence from the input sequence.

Its first use in music dates back to 2004 in [5], where its application in a "real life" machine improviser in a complex performance situation was proposed. In this system, the machine improviser "listens" to three synchronized sources: a metric source that generates a stream of pulses, a harmonic source that sends harmonic labels, and a melodic source that sends a stream of time-tagged note events. The improviser continuously learns from the harmonic and melodic sources and aligns with the metric source. When triggered, it generates a harmonic or melodic part. [5] Two years later, and with the same finality, the authors used this model in the OMax Project as the learning and generator model. [6].

Recently, some extensions have been made to this model to adapt probabilities in [82] and [26]. The first extends a version of the Factor Oracle with the Probabilistic Non-deterministic Timed Concurrent Constraint (pntcc) to decrease the probability of choosing a sequence previously improvised by assigning integers to the links of the FO with which it is possible to calculate probabilities to choose a link based on a probability distribution. These weights are calculated by decreasing the weight of the forward link when chosen and computing the length of the common suffix (context) associated with each suffix link to calculate its reward. In the second approach, they present a system that combines interpolated probabilistic models with a factor oracle, in which the probabilities are trained in a musical corpus and another model in which several factor oracles, each representing a musical dimension, can communicate through message passing, using belief propagation on a cluster graph which can use the other system. This can be applied in cases of polyphonic or multidimensional music, as well as in improvisations with various musicians.

Another important extension was presented in 2007, by Cheng-I Wang and Shlomo Dubnov that allows for the processing of continuous dataflows such as audio. [30] The great differences in the algorithm are that it does not assign symbols to transitions, instead of presenting one-to-one correspondences from states to frames in the audio buffer and it needs to calculate the degree of similarity between frame descriptions using a distance function and a threshold value associated with the metric over the feature space.

2.3.5 Variable Order Markov Oracle

The Variable Order Markov Oracle (VMO) was proposed in 2014 by Cheng-I Wang and Shlomo Dubnov [84] for clustering multivariate time series data points without specifying the number of clusters, following along the research on both Factor and Audio Oracles.

The VMO explores the AO construction by explicitly identifying the clusters of frames formed during it, maintaining the on-line nature of the algorithm by treating the cluster labels as the symbolic sequence would in the factor oracle and keeping the pointers to the events in the incoming time series according to the cluster they belong to. The threshold for calculating the degree of similarity between is calculated via a measure of information rate (IR) which describes the extent of the inherent structured quality of a time series, using the entropy of the events in the series. If this value is very low, every incoming data point is considered different from all other data points observed which translates in every data point being assigned to a different cluster, meaning no real pattern can be abstracted from the structure. A high threshold diminishes the number of different clusters encountered and in an extreme case, there will only be one cluster abstracted in which all events observed are contained, and the structure does not retain any characteristics of the original time series. [84]

The VMO was used [85] to synthesize new music audio signals by using a query music signal that provided the map to how the materials should be recombined to guide a path traversing the target one. The same experience was executed for both symbolic and audio musical sequences in [87] and shown to achieve state-of-the-art performance on the JKU-PDD dataset although, because of the use of the chroma to represent the events incoming, different occurrences of the same pattern were not always recognized as the information from some of the voices in the musical piece is discarded by this type representation.

In [88], the authors made a first attempt at establishing a statistical model for VMO by making an analogy to the Hidden Markov Models based on the inference of emission probabilities. This research was conducted to improve the machine improvisation scheme of the OMax project but did not introduce probabilities to the transitions themselves. That was proposed a year later in [86], where the authors assign a VMO for recording the location and length of the longest repeated suffixes at every time step and then extract the Markov transition probabilities from the VMO, using the lengths of longest repeated suffixes, which provide variable-length Markov transition information. The VMO-HMM was experienced in a case study on analyzing Jazz music harmonic progression and used in the same improvisation context as before. The authors conclude that this new implementation provides a more compact and abstract representation of the oracle structure while keeping its variable-length Markov properties and making it possible to unify different VMO structures belonging to different pieces in a unique one for a whole corpus.

2.3.6 Deep Learning Techniques

In later years, deep learning has become a big domain for classification and prediction tasks which can be explained by the increasing amount of available data, the increase in efficient and affordable

computing power. An area in which deep learning is growing is in the generation of content, which includes musical content. In this domain, the focus has been in applying these techniques to available corpora to automatically learn musical styles and to generate new musical content based on these.

This review on architectures used as deep learning techniques for generating musical sequences has been based on a survey done by Jean-Pierre Briot, Gaëtan Hadjeres and François-David Pachet. [13]

The main types of deep learning architectures used for music generation are:

- **Feedforward:**

The feedforward neural network is a multilayer network composed of at least three layers: input, output and hidden layers. The hidden layers can be combined with a nonlinear activation function to make the network a universal approximator.

- **Autoencoder:**

This architecture consists of a neural network composed by one hidden layer. The number of input nodes has to be equal to the number of output nodes, as the output mirrors the input. The training of this model is classified as unsupervised learning because the examples provided are not labeled but it is implemented using conventional supervised learning techniques. The decoder used to reconstruct the information compressed by the encoder presents a high-level control of content generation.

- **Restricted Boltzmann Machine (RBM):**

The RBM is a generative stochastic artificial neural network that learns a distribution of probabilities over a set of inputs. It is organized in two layers: the visible one (similar to an input layer but also works as output) and the hidden one, and connections between nodes in the same layer are not allowed. It is trained in an unsupervised learning manner by using contrastive divergence. In the context of music generation, it is especially interesting for learning (and generating) chords, as the combinatorial nature of possible notes forming a chord is large and the number of examples is usually smaller.

- **Recurrent Neural Networks (RNN):**

An RNN is a feedforward neural network extended with recurrent connexions for learning series of items, whose order is important for its comprehension. The output of the hidden layer reenters itself as an additional input for trying to predict the next element in the sequence. The long short-term memory (LSTM) can be added to a RNN for helping prevent too many repetitive generations by securing information in memory.

2.4 Summary

After reviewing the literature on models that abstract patterns from representations of music structures and because of the capacity of the VMOs for achieving a better balance between the quantity

of available information and the optimization of the generation, this algorithm will be the one explored for our system.

Also, despite the great benefits that deep learning brought to the area of generation of content, in particular, musical content, we will be focusing on the more conservative techniques that are the target of study of the first line of research that we mentioned earlier and that allow us to obtain this type of information as we want to study the implications of representation in the model's intrinsic structure, adapted to varying volumes of musical information provided, for which we need greater transparency of the model of temporal relations, which cannot be yet achieved using these novel, neural-network based techniques. Another reason for not using deep learning is that the amount of data necessary for learning a model is very big and we want to model variate quantities of data that may be quite small, which would not result in good models in this type of machine learning.

At the same time, although the representation frameworks have been developed, most of the systems that reached the artistic community only use formal string representations, mainly captured from MIDI information. This kind of representation, as seen in Section 2.2 captures only a small amount of the desired information to be extracted from symbolic musical structures. The representation that seems to lead to a more powerful modelling is the Multiple Viewpoint Models as they permit the modeling of both simple and derived information and, thus will be the ones explored during the research and further construction of our system.

Chapter 3

Encoding Information from Symbolic Music Manifestations: A Multiple Viewpoint Model Approach

This chapter presents a comprehensive explanation of the methodology used to encode information from a symbolic musical score in **MyMusicalSuggester**.

In light of the multidimensional and hierarchical structure of music, we explore a representation structure that is constructive (possible to create the music represented), derivable (possible to derive a representation from the music represented), meaningful (significant differences in musical objects should translate in different representations), decomposable (possible to segment the representation as we would divide the respective musical objects represented), hierarchical (possible to distinguish different levels of detail in the representation corresponding to different levels of significance) and generative (possible to infer relations between elements in the musical structure from the respective representation) [50]. As we detailed in Chapter 2, the representation structure that better achieves these principles is the Multiple Viewpoint Models, proposed by Conklin and Witten in 1995 [21].

The Multiple Viewpoint Models comprise a collection of different attributes of the musical structure. For a complete list of the viewpoints adopted in our work please refer to Appendix A. All the names mentioned in Sections 3.4.2 and 3.5 will be references to the ones listed there.

We will start by documenting the process of extracting and encoding the information from symbolic music manifestations encoded in the MusicXML syntax. We will then detail our implementation for abstracting the different viewpoints from the musical score. A general organization of the different viewpoints can be observed in Figure 3.1. The description of the reverse process of translating the encoded musical events, organized as viewpoints, to a score in MusicXML format will also be provided as it is relevant in our work.

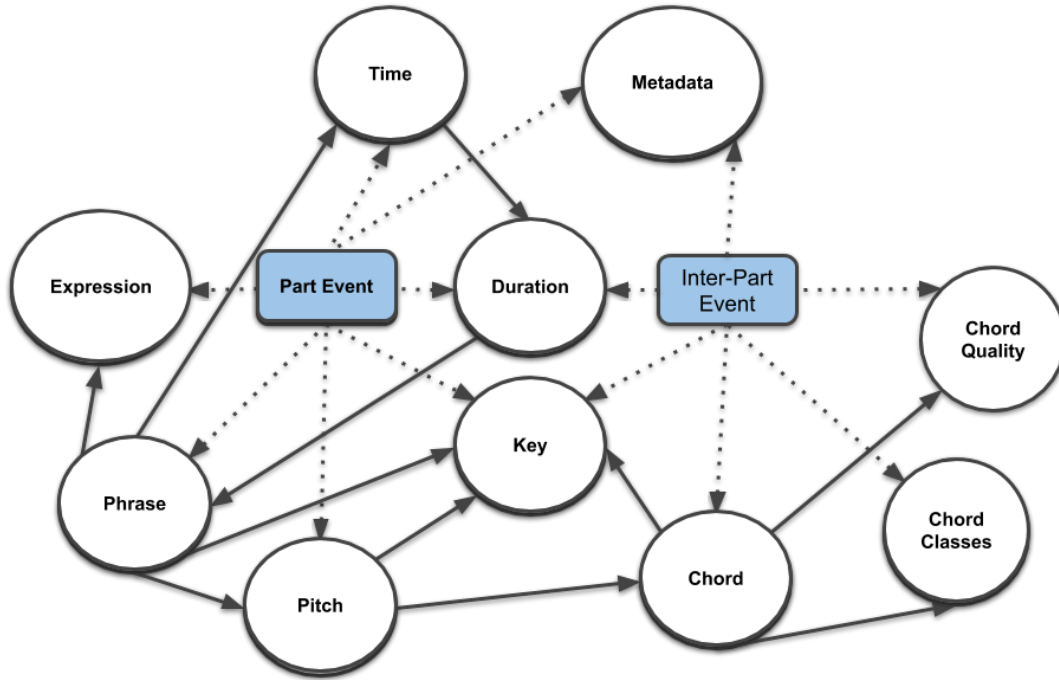


Figure 3.1: Organization of All Viewpoints. Dotted arrows represent relations of Viewpoint categories to events and full arrows represent dependencies between Viewpoint categories.

3.1 Unpacking a Musical Score

Western music notation adopts a symbolic representation which encodes both single and multi instrumental *parts*. Each part is typically attributed to a unique instrument, which can be performed by a single player or doubled by multiple players of the same instrument. However, a musical part can have multiple *voices* to be interpreted by different players. Those voices are usually explicitly written, identified as a unique voice but they can also be written as chords, if the instrument is not a polyphonic one. A single instrumentalist will not be able to play multiple notes simultaneously, unless the instrument is polyphonic or in very particular extended techniques, called as multiphonics, where the simultaneous notes played are produced either with new fingerings, by using different embouchure positions, or voicing the throat with conventional fingerings and are not intended to be listened as chords. Hereafter, in the context of this dissertation, a part linked to a single instrument (to be performed either by a single player or group of players) will be addressed as *Part*. A single hand of a piano part will also be considered as a single *Part*, and each *voice* explicitly written in the musical score will also be represented as a single *Part*. Three examples of part division can be seen in Figures 3.2, 3.3 and 3.4, where each color represents a different *part*. In the first example, a fugue from Bach, each part comes from an explicitly written voice,

that is very clearly seen on the musical score. In the second example, a choral music from Whyte, multiple vocal voices exist as chords in the same staff. As a human voice is not a polyphonic instrument, the notes overlapping (identified as two-note chords) will be identified as different *parts*. In the tenor and baritone staff, the first measure was written with two explicit voices. They will still be identified as different parts. In the third example, each hand of the piano will be identified as a *part*, but the chords will not be divided because they make sense in the same part, as they correspond to an accompaniment and not different, independent voices.



Figure 3.2: Part division of the first six measures of J. S. Bach's Fugue I in C Major, BWV 846 from the Well Tempered Clavier (First Book).

Soprano
Alto

1. We shall hear a voice, an im - mor - tal voice, "Be -
 2. When the voice shall cry, "Go ye forth to - night, Be -
 3. Bro - ther, trim your lamp, have it burn - ing bright, "Be -
 4. Hast thou made a vow? hast - en ye to pay, "Be -

Tenor
Baritone

Figure 3.3: Part division of the first two measures of J. M. Whyte's We shall hear a voice, an immortal voice (1890).

On the other hand, if the musical score has more than one part, to fully unpack its structure, we have to examine the relations across the different parts at each moment of the musical time. These relations mainly refer to the vertical concurrent aggregates resulting from the overlaying of different parts. In music theory, these vertical studies are addressed in the study of harmony. In the context of our dissertation, these relations are an important stylistic element, which ought to be

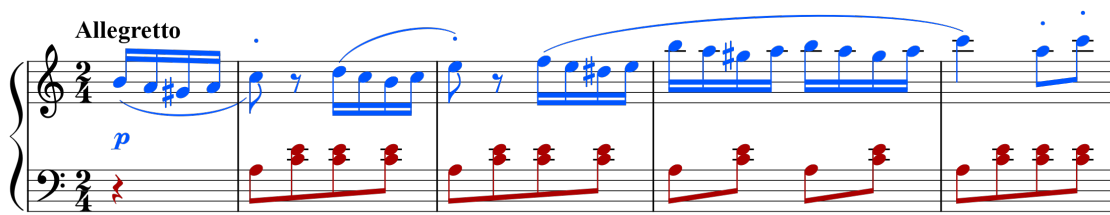


Figure 3.4: Part division of the first four measures of the 3rd Movement (Rondo Alla Turca) of Mozart's Piano Sonata No. 11, K.331.

captured from the datasets and in this sense, we denote these structures as *inter-part dependencies* as they work as a reduction of the score to a succession of events that represent everything that is happening in the score in a vertical sense.

For both types of parts, the analysis of the musical events is made from the perspective of the basic components: every note, rest or chord is identified as a single event of the same type as the part that has its characteristics and absorbs the characteristics of the what is happening in the part at the time it occurs. Each event is identified by its offset, the time of the musical score in which it occurs, measured at quarter notes from the beginning of the part and a dictionary of viewpoints, organized by category.

In what follows, the constitution of each type of events will be described, taking into account the musical meaning of each viewpoint and its connection to the other viewpoints and categories.

3.2 MusicXML

MusicXML¹ was first presented in 2004 by Michael Good (and Recordare LLC) as a XML-based file format to represent Western musical notation digitally [34]. The MusicXML format is designed to standardize a syntax across music notation software, so that digital music scores can be shared across multiple software. Currently, the version 2.0 of MusicXML has become the standard for representing music notation and has been incorporated by most score editing software, such as MuseScore², Finale³, Sibelius⁴ or Dorico⁵, but also musical sequencers like Cubase⁶ or Logic Pro⁷.

The use of XML as the basis for this musical file format translates in an easiness of parsing and manipulation for automated tools. In that sense, it is very similar to HTML but the tags are musically orientated, for instance <key>, <time> or <pitch>, which allows for the representation of essential characteristics of music that increase the level of expressiveness that can be extracted from it.

¹<https://www.MusicXML.com/>

²<https://musescore.org/en>

³<https://www.finalemusic.com/>

⁴<https://www.avid.com/sibelius>

⁵<https://new.steinberg.net/dorico/>

⁶<https://new.steinberg.net/cubase/>

⁷<https://www.apple.com/logic-pro/>



Figure 3.5: Voice making in a passage with 2-note chords that should be divided in two voices

3.3 Music21

Music21⁸ is a toolkit for Python developed by Cuthbert, launched in 2008 and since then frequently developed, that aims to bring computational musicology tools to scholars and other active listeners working in Python so that they can easily promote the creation of tools for the analysis and generation of musical content. [23]

It uses an object-oriented skeleton that makes it easier to handle complex musical data. One of its main features is the possibility to parse musical information from various types of formats, such as MusicXML, MIDI⁹, MuseData¹⁰, TinyNotation¹¹, MEI¹² or Humdrum¹³, and includes a corpus of more than 500 musics in the compressed .mxl format, which we used as a database.

3.4 Abstracting multiple viewpoints from parts

This section first describes the method for extracting the parts from a musical score in MusicXML format and then, it details the existent viewpoints that abstract the information from the musical surface of a given part.

From a single part, we can extract great amounts of information such as notes (or chords, in the case of a polyphonic instrument) and rests and every underlying aspect of those basic components, such as pitch and duration but also expressive components as, for instance, articulations and ornaments or time and key information that give us the perception of how the basic components relate between themselves.

3.4.1 Part Segregation: Separation of Voices

Music21 has both a `makeVoices` implementation that isolate all overlaps of a part into unique voices and a `voicesToParts` implementation that extracts each different voice into a new part. We found that they did not achieve the results we were expecting, in most cases, as the division did not really occur, either if used inside every measure, the parts or the musical score. The result of music21's `makeVoices` and our implementation can be observed in Figure 3.5, where we can clearly see that music21's supposed division didn't occur, even though it should have divided the chords in two voices, while our implementation correctly divided the chords in two voices.

⁸<http://web.mit.edu/music21/>

⁹<https://www.midi.org/>

¹⁰<https://musedata.org/>

¹¹https://web.mit.edu/music21/doc/usersGuide/usersGuide_16_tinyNotation.html

¹²<https://music-encoding.org/>

¹³<https://www.humdrum.org/>

Additionally, the particular instruments have idiosyncrasies in terms of overlaps in light of their capacity to perform polyphony. Therefore, we created our own algorithm for segregating the voices into parts, whenever they belong to an instrument that is not capable of performing multiple notes at the same time.

Our implementation starts by evaluating the instrument that the part is written for. If it is a monophonic instrument (i.e., if it belongs to the winds, brass families or is a human voice) with note overlaps or different voices, an ideal number of voices will be calculated from the whole score, by identifying the maximum number of different notes that exist simultaneously. Every measure will be parsed to extract the total number of voices. To this end, we either use the information encoded in the voice layer information of the MusicXML score, or divide the notes of the chords in the same number of voice layers. Otherwise, if the instrument is polyphonic and has written voice layers, the original music21 `voicesToParts` will be adopted. In case the instrument is not recognized by music21, the default instrument assigned to the part will be the piano, as typically happens in most notation software.

3.4.2 The Part Events' Viewpoints

Multiple Viewpoint Models were adopted to abstract information from the musical surface in order to have the most comprehensive set of descriptions from the surface. A set of different characteristics can achieve better results in predicting and modeling the complex surface relations of a musical work. As such, it was important for us to explore a great number of possible representations, organized by categories that we thought pertinent for the problem. Some viewpoints share relationships of dependence with elementary viewpoints (those that can be extracted directly from the score). We call them derived viewpoints. A visual graph that explains the relations between viewpoints belonging to part events and their general organization can be observed in Figure 3.6.

3.4.2.1 Metadata Viewpoints

The *metadata viewpoints* allow us to learn global information, such as title and composer, as well as number and type of instruments included. In the specif case of single work, and most noticeably, solo musical pieces, none of the viewpoints of this category will variate at all and thus, won't be of much utility for modeling. Metadata viewpoints have a greater importance when generating from a large set of works from multiple composers.

3.4.2.2 Basic Viewpoints

The *basic viewpoints* include fundamental attributes of Western music without whom it would be impossible to recreate new musical sequences. The events can be either annotated as *grace notes*, *rests* or *chords*. If an event does not include any annotation, it is assumed to be a note event. A different basic viewpoint is the *bioi*, which translates as the basic offset interval of the event to the previous one.

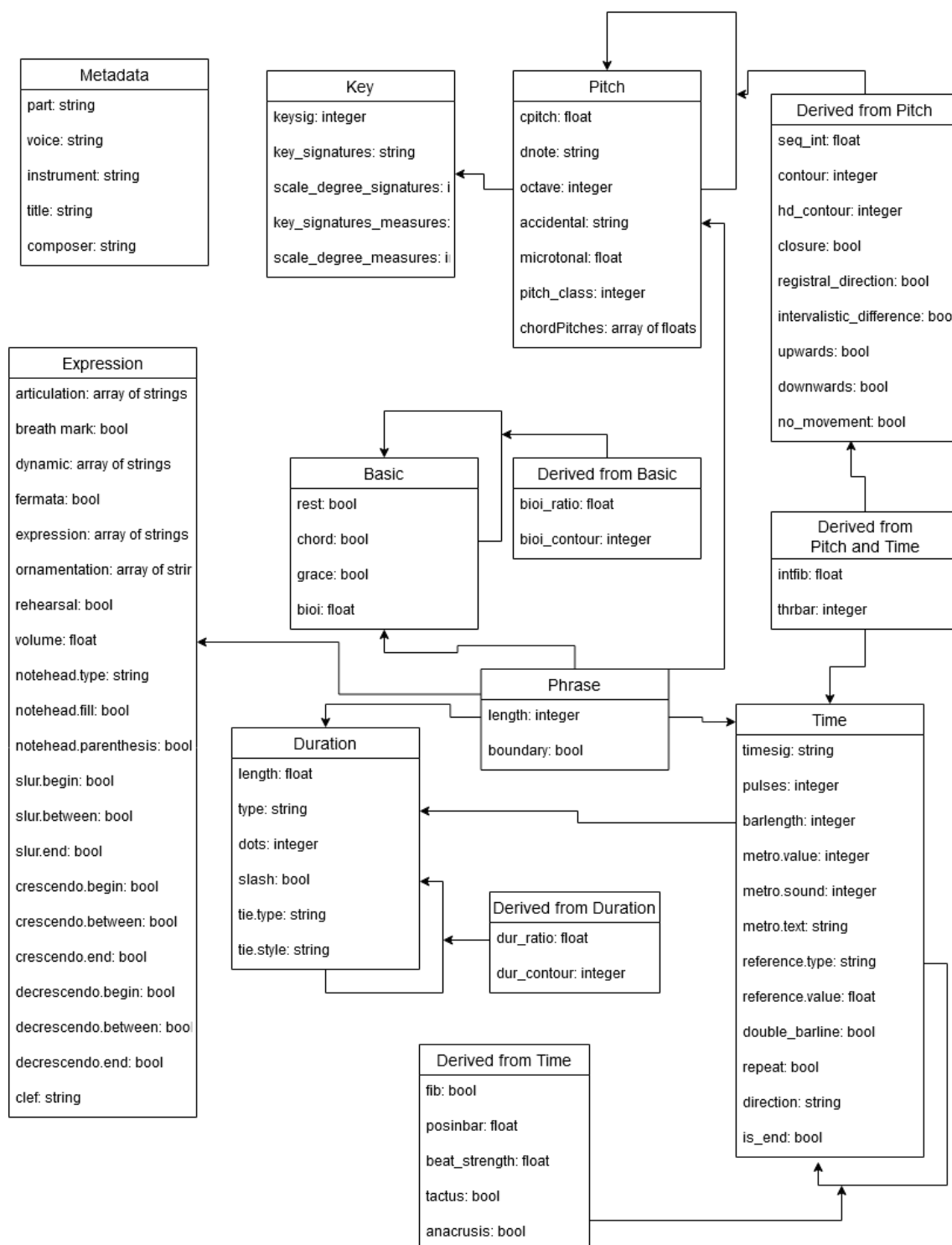


Figure 3.6: Organization of Part Viewpoints, by category.

3.4.2.3 Duration Viewpoints

The *duration viewpoints* represent the span of musical time in which a musical event exists. Although this information is also essential to the musical understanding, it is so important on its own that we felt the necessity of separating it from the basic viewpoints. The viewpoints that express duration are *length*, that is the fundamental numeric value of the duration of a single event; *type*, which relates to the musical notation name attributed to the duration, if existent. Otherwise, it is interpreted as a 'complex' duration; *dots*, as in the amount of dots applied to the type of the duration; and *slash*, that makes reference to the existent of slashes in grace notes that make its duration smaller, as a grace note parsed with music21 has no duration length or type. If an event is tied to another, the *type* and *style* of the tie are also stored.

3.4.2.4 Pitch Viewpoints

The *pitch viewpoints* depict the fundamental characteristics inherent to a note. Pitch is often described as the perceptual property of sound that makes it possible to qualify the sounds in a perceptual scale according to their physical property of frequency. It can be viewed in a numeric scale, which we call the chromatic scale, stored as the *cpitch* viewpoint, or as a combination of properties that we know as diatonic note name (*dnote*), *accidental* and *octave* number. In contemporary Western music practice and non-Western musical traditions, the pitch is often divided into smaller intervals which are called microtones. They are measured in cents, and its use is named as microtonality. Because we aim to provide the means for potentiating the creativity of living contemporary composers that usually incorporate these techniques in their works, we expand existing viewpoint models by storing this information as the *microtone* viewpoint. Quarter-tones are also expressed in the *accidental* viewpoint, as they are frequently adopted. We also store the value of the pitch in the twelve-tone scale as the *pitch class*.

For events that are chords in the parts (for example, chords on a piano part), we store the root as the pitch and store the other pitches that make part of the chord as *chordPitches*, in their chromatic pitch value.

3.4.2.5 Expression Viewpoints

This category of viewpoints portray the symbolic notations that alter the meaning of the musical events, described by their basic viewpoints. In this category, we included *articulations*, *text expressions*, *ornamentation*, existence of *fermatas*, *breath marks* or *rehearsal marks*, *type* and *fill* of noteheads, if there is *parenthesis* surrounding an event, *volume*, *dynamic* marks and spanner-like information such as if the event is at the *beginning*, *middle* or *end* of *slurs* or *crescendo* and *diminuendo* lines. Although not really an expression, we also include the *clef* of the staff at that point in time as an expression viewpoint. Some of these viewpoints are usually very important in extracting temporal meaning of musical excerpts, such as *fermata* and *breath mark* and *slurs* that often delimit phrases.

3.4.2.6 Time Viewpoints

The temporal organization of the musical events is one of the most important things when we talk about music. For that motive, it is essential that the information on time be stored in a way that facilitates the learning of how the organization of music takes place. To this effect, we store the *time signature* information in its conventional form but also the *pulses* that form the length of a bar and that same *length* as numeric values. We also extract metronome marks as a tuple of *text*, to identify movements, *value* and *sounding* value, in order to describe tempo and changes in tempo and the referent *type* and *value*. We store all this information as it happens at the time of the part event.

Other temporal organization of music related to measures and phrasing is the existence of repetitions or explicit separations. In order to absorb these characteristics, we save the existence of *double barlines* after an event and the existence of *repeating barlines* before an event, as they can appear at the start and end of a sequence and the respective *direction*. We also note as a single viewpoint if the *last barline* is a repetition barline, as this one is not included in the others because it happens after the end of the score.

3.4.2.7 Key Viewpoints

The *key viewpoints* convey the relationships between musical events as a whole. The *key signature*, expressed in number of sharps (positive) or flats (negative) alludes to the musical convention of reducing the number of accidentals used in the staff to make reading of it easier.

We used the music21 analysis module for deducing the keys, a concept that translates possible perceptual relations between notes and are the basis of classical, Western music. Acquiring this information by using the key signature is not always reliable as, sometimes, the music implies relations that are too fast for changing the key signature and it is more efficient to use simple accidentals in that passage or simply because the key signature does not discriminate between modes. Because the music21 analysis module works on the basis of key areas, which is very complicated to calculate automatically, we store the keys discovered in *measure* areas (the analysis is done every measure) and at every change in key *signature*. We know that both of these area measurements are not ideal as they can lead to too small and too large windows of analysis but we thought that combined, they could lead to a better perception than if we used only one of them.

Then, we store the *degree* of the event (in case it is a note) in relation to the scale inherent to both areas' keys.

3.4.2.8 Phrase Viewpoints

The parsing of a musical score doesn't explicitly express phrasing information, such as start and end of phrases, but we can predict it by analysing the values of the other viewpoints. This information, although it could be interpreted as derived, is important enough to be stored on its own and it is only calculated after choosing the importance of the other viewpoints for the model. The viewpoints stored are the *boundary* value of the event (that identifies if the event is in the beginning,

ending or middle of a phrase) and *length* of the phrase in which the event is inserted. The methodology used for calculating this information is detailed in the next Chapter, in the Section 4.2.2.

3.4.2.9 Derived Viewpoints

Derived viewpoints, as observed before, represent features of the score that cannot be directly extracted from its parsing and need to be calculated from other viewpoints' information and, in some cases, using knowledge of other events' viewpoints.

Most of the derivations that we chose relate to Pitch viewpoints. These are:

- *Sequential Interval* between event and last note event
- *Contour* indication from last note event to current event, in which it can be a descendent movement (-1), an ascendant movement (1) or no movement (0)
- *HD Contour* which indicates a quantified sequential interval value in a step of -4 to 4:

$$HDContour(int) = \begin{cases} 0, & \text{if } 0 \leq int < 1 \\ \text{sgn}(int), & \text{if } 1 \leq int < 3 \\ 2 * \text{sgn}(int), & \text{if } 3 \leq int < 5 \\ 3 * \text{sgn}(int), & \text{if } 5 \leq int < 8 \\ 4 * \text{sgn}(int), & \text{if } int \geq 8 \end{cases}$$

- The last three events are doing an *upwards* movement?
- The last three events are doing a *downwards* movement?
- The last three events have *no movement*?
- The last three movements are in a *closure* shape? This viewpoint is a score viewpoint that can have as values 0, 1 and 2. A point is attributed to the event if there is a change of direction and another point is attributed for a tone smaller than the preceding one.
- *Registral Direction*: Is the event in a large jump (\geq perfect fifth) followed by a direction change or a small (\leq perfect fourth) jump followed by a move in the same direction?
- *Intervalic Difference*: Is the event in a large jump followed by a smaller (3 semitones smaller if in the same direction or 2 semitones if reversing the direction) jump? Is a small jump followed by a similar interval?

Related to *bioi* and *duration* information we have:

- *Bioi Contour* indication from last event to current event, in which it can be less space between events (-1), more space between events (1) or the same space between events (0)

- *Bioi Ratio* indication of ratio from last event to current event, calculated by the division of the respective *bioi* values
- *Duration Contour* indication from last event to current event, in which it can be a faster duration (-1), an slower duration (1) or the exact same duration (0)
- *Duration Ratio* indication of ratio from last event to current event, calculated by the division of the respective *duration* values

The other ones relate mostly to time information. These are:

- Event is the first element in a bar? (*fib*)
- Position of the Event in the bar in which it belongs. (*posinbar*)
- *Strength* of the Event in relation to bar to which it belongs. (For example, in a three beats measure, the first beat has a higher strength than the other two and the third beat has a smaller strength than the second.
- Is the event in a *tactus* beat?
- Is the event an *anacrusis*?

The next two viewpoints have a relationship with more than one viewpoint. The *intfib* represents the sequential interval of the event to the *fib* of the same measure, while the *thrbar* represents the sequential interval of a *fib* event to the last *fib* before it, thus joining an information derived from time and information related to pitch.

At last, after the segmentation process, the *intphrase* viewpoint connects each event to the beginning of the phrase to which it belongs, by storing the sequential interval between these two events.

3.5 Abstracting multiple viewpoints from inter-part dependencies

The information related to inter-part dependencies is extracted from a music by "chordifying" the originally parsed musical score. The word "chordifying" was invented by the music21 developer with the meaning of "reducing a complex score with multiple parts to a succession of chords in one part that represent everything that is happening in the score" [24]. This process can also be called "salami slicing", because it cuts the score to completely represent every single moment that happens in it. Every one of these slices will be regarded as a Vertical event.

An example of chordifying a musical piece can be seen in Figure 3.7, where the first represents an excerpt of the original score and the second, the reduced result of the chordifying process.

The viewpoints that characterize the inter-part dependencies of musical events in a musical score in a similar manner to those that represent musical events that come from parts. As we can observe in the next sections, some of them can even be applied equally. However, these events



Figure 3.7: Original and Chordified Versions of the first five measures of J.S. Bach's Cantata, "Schwingt freudig euch empr", BWV 36

embed different perspectives than the ones above in the sense that they must concentrate joined information of simultaneous part events belonging to different parts.

A visual graph that explains the relations between viewpoints belonging to vertical events and their general organization can be observed in Figure 3.8 and every category of vertical viewpoints is detailed in the next sections.

3.5.1 Metadata and Duration Viewpoints

The metadata viewpoints function similarly to the ones that are present in the part events. However, only the ones related to common score attributes (title of the work and composer) make sense in this context and thus, they are the only ones extracted.

The duration information relative to vertical events is stored exactly in the same way as the one for the part events.

3.5.2 Key Viewpoints

The representation of the key information of a vertical event works similarly to the manner in which it is represented in the part events. Instead of storing the scale degree, we store the function of the chord in relation to the key analysed at both measurements (measure and signature) and the certainty of the analysis.

3.5.3 Chord Viewpoints

As the result of a chordifying process, the inter-part events are essentially chord-like events. Therefore, the most important information that we must take from this type of events is chord basic characteristics. A chord is firstly defined by its *itches*. From those, we can identify the *cardinality* of the chord, as the number of pitches that exist in the chord, the *root* of the chord (as its most important note), and its *inversion*, the *prime form*, which is described as the most compact form the chord can take (i.e., leftwards packed or smallest in lexicographic order), either in the normal form or as one of its inversions and its *quality*. In the sense of simplifying the quality of a chord, we implemented test viewpoints (called quality viewpoints) that evaluate the chord to most of the known chord qualities: is the chord *consonant*? a *major* or *minor* (complete or *incomplete*) triad?

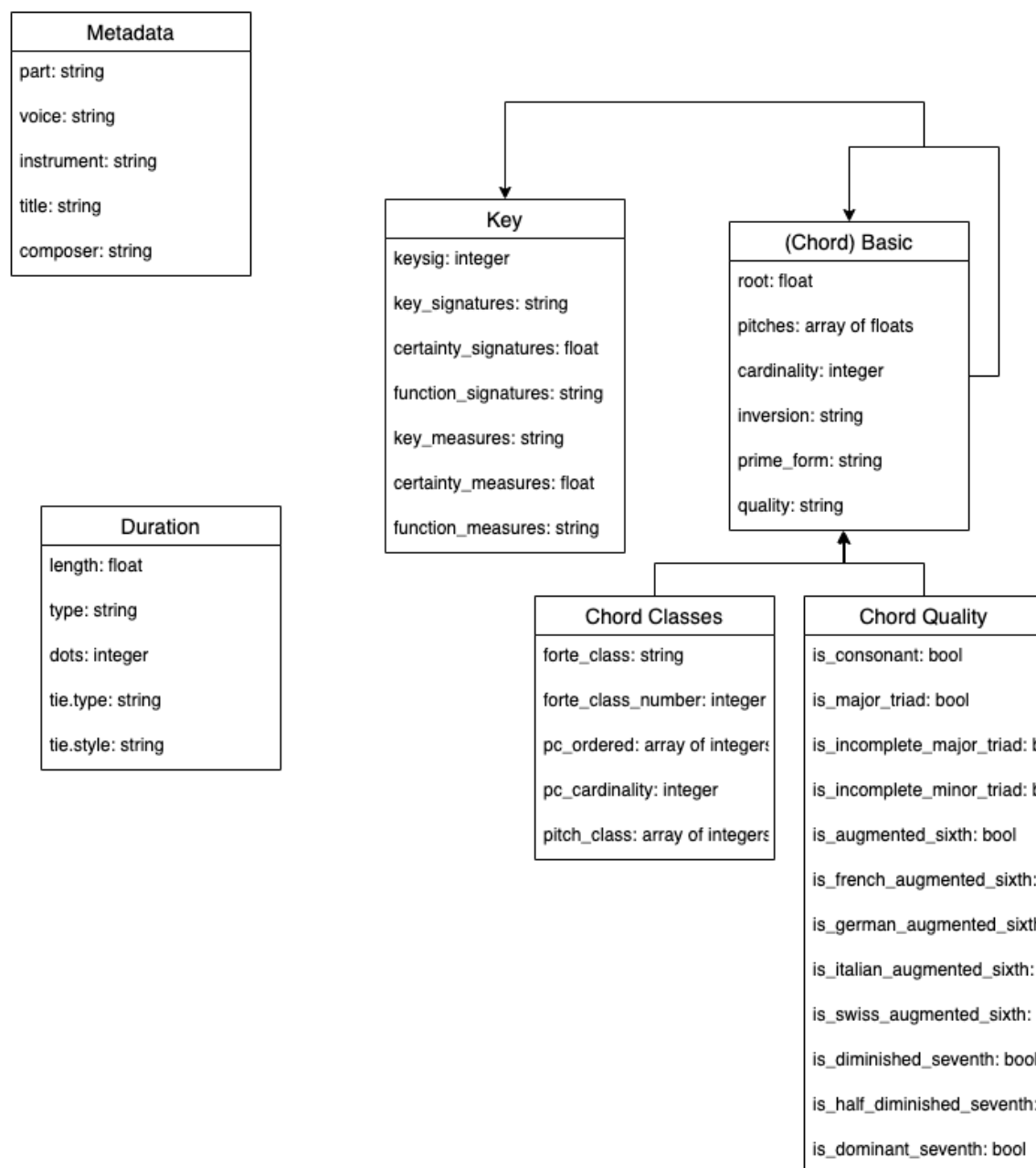


Figure 3.8: Organization of Inter-Part Viewpoints.

an *augmented triad*? an *augmented sixth*? Of which type: *german*, *swiss*, *italian* or *french*? Does it have a seventh? Is it *half-diminished*, *diminished* or *dominant*?

Other characteristics can be extracted from chords, including pitch class information. This category deals with this knowledge, and stores the *pitch classes* that compose the chord (*ordered* and *unordered*), the *cardinality* of the unique pitch classes and the *Forte Class Number*, that represents the pair of Allen Forte numbers assigned to the prime form of each pitch class set, with and without inversion distinction.

3.6 From part events to a new Musical Score

At the moment, our application only allows the creation of sequences for parts, as it is what a composer usually needs in their composition process. As the result depends on what viewpoints where used, the conversion to a musical score can be very different every usage.

If the user selected the generation of multiple part sequences, the names of the parts are analysed to verify if they should be written in the same staff or in separate staves. Voices that were originally in the same staff and chords that were interpreted as voices (belonging to monophonic instruments) will be written in the same staff, as opposed to different instruments or separated instrumental parts, that will be written to different staves. Each part will be written using the process explained bellow and at the end, the first event of every part will be analysed to understand if the note is an anacrusis. In that case, every part will be moved the amount of the respective position in bar, if existent.

Converting each sequence of events related to a single part presents some problems, regarding the multiplicity of possible combinations of used viewpoints.

On the beginning, if the part is being joined with an already written staff, it will acquire the last's starting characteristics, such as its key and time signatures, instrument name and metronome mark. Otherwise, it will start as an empty staff. A new voice will be created in order to include the new notes, chords and rests that are represented by the events, in both situations.

At the processing of each event, the viewpoints regarding general components of a score, in particular the ones mentioned above, will be analysed to determine if a change occurred. In such a case, the respective music21's object will be inserted into the staff, with the new information. In the same sense, if new dynamics exist, they will also be inserted.

Then, the duration of the event is evaluated. If existent, length, type and dots information of duration are used to create the Tuple. If not, only length will be processed. Whereas some duration type values were not being correctly read by the music21 *Duration* object, in particular, complex duration values and the ones minor to a *demisemihemidemisemiquaver* (type signaled as 2048th), we choose to bypass type and dots when this situation occurs, also. In the case that the user doesn't pretend to learn the rhythm, considering that the default value of duration length is one (equivalent to a quarter note), all notes, chords and rests generated will be thus interpreted as quarter notes.

The processing for rest events is very simple: a music21 rest object with the duration calculated previously is inserted in the voice at its' highest offset. If articulations, expression marks or ornaments were learnt for that event, they will be appended to the rest prior to insertion.

On the other hand, the processing of note or chord events is much more complex, particularly if pitch components were not selected. If neither pitch components nor sequential intervals were learnt, the pitch chosen will be the last pitch of the original part. In case sequential intervals exist but not pitch information, the first pitch of the original score is used as reference for constructing the sequence from the intervals between pitches. Otherwise, if diatonic name notes, accidentals and octave exist, it will try to construct from that information, primarily using name notes. Provided that chromatic pitch values exist, the note achieved using the last viewpoints is compared and corrected, if necessary. Chords will be written if the chord pitches were learnt. Otherwise, only the root of the chord will be processed as a single note. After, volume, notehead fill and type information and existence of parentheses surrounding the note are assimilated, if existent. At the end, if the note was annotated as a grace note, the respective conversion will be done and the current event will be regarded as an *appoggiatura*. Articulation, expression marks and ornaments are appended to the note prior to insertion in the voice.

When all events were processed, the existence of slurs, crescendos and diminuendos will be evaluated and written. These musical components are particularly difficult to process as they denote a relationship among elements of the original part that are now in different orders. Nonetheless, they are very important to understand the expressiveness of the music and creating new sequences without them, if selected, would result in poorer musical suggestions. The implemented processing is similar for the three spanner-like components: from the events, we extract the ones that begin the specified spanner; for every one, we select the corresponding note, chord or rest already inserted in the staff, create a new spanner of that type and connect it to that basic component. Then, for every note, chord or rest following that one, the note is inserted in that spanner until one whose event's viewpoint is at the ending of a spanner of that specific type or it is the last note in the sequence. That will be the last note of that spanner. One problem with this implementation is that it often leads to too big spanners. A possible future improvement, could pass for annotating the length of each spanner for all events that share a relationship with it, as is done with phrase length, and implement a threshold system that allows to divide too big spanners.

The measures are calculated from the notes and time signatures, automatically, using a function called `makeNotation` that is provided by the music21 toolkit.

The conversion to a MusicXML score from a single-part generation will proceed similarly.

Chapter 4

My Musical Suggester

This chapter details the implementation of **My Musical Suggester**, a software prototype which explores a model for assisting the creative process of a composer. Briefly, the software learns stylistic features from a symbolic music corpus and suggests novel musical content to the user (i.e., composer). Multiple ranked sequences for varying a given musical phrase or creating continuations of a given context are proposed. The ranking order of the musical sequences follows a proximity-based metric, which should ultimately guide the user in the selecting process.

In Section 4.1, we present the architecture of the software. In Section 4.2, we briefly address the representation of symbolic musical structures and the implementation a Multiple Viewpoint Model. In Section 4.3, the VMO implementation and the generation of new sequences from the temporal models is detailed. Finally, in Section 4.4, we detail the interface of **My Musical Suggester**, as well as its integration with the notation software, Musescore.

4.1 Architecture

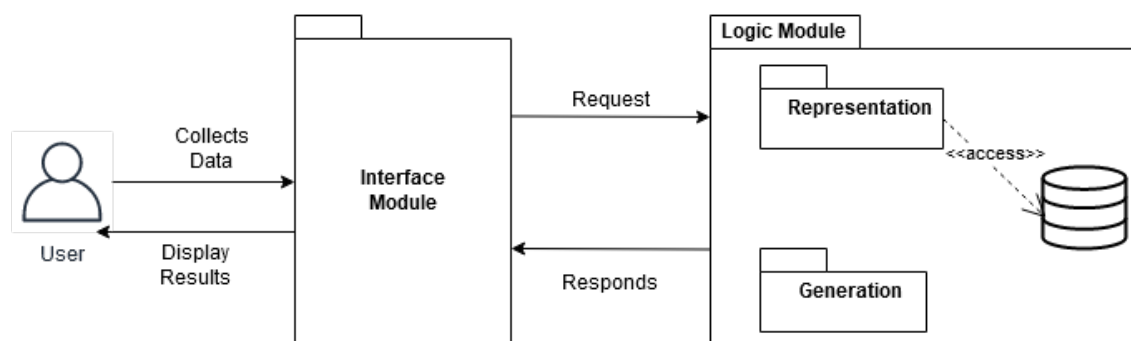


Figure 4.1: General Architecture of **My Musical Suggester**.

My Musical Suggester was developed in Python and QML. The first was used because of the already existing implementation of VMO and greatly-used libraries for dealing with musical

scores in MusicXML, as well as its suitability for this kind of tasks. The second was chosen as it enables an easy integration with Python and is the language in which the plugins of Musescore must be implemented. Figure 4.1 shows the architecture of . It features two core modules, the logic and interface modules.

The first module addresses the logic behind **My Musical Suggester** and communicates with the interface module that makes them visible to the user. It is divided in two sub-modules: The representation sub-module and the sub-generation module, which will be the ones tackled in Sections 4.2 and 4.3. This module also deals with retrieving files from an existing database of already parsed musical information.

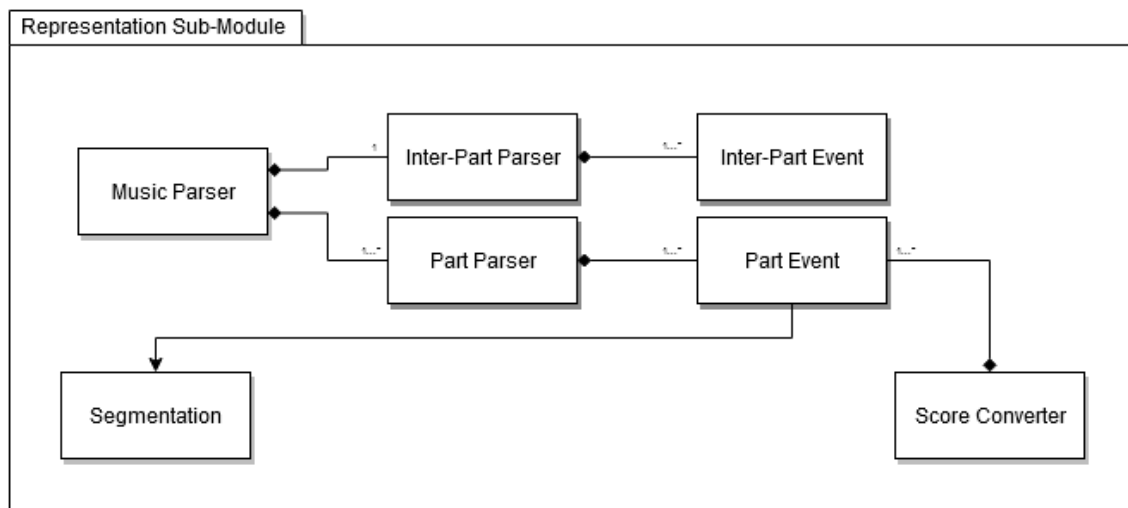
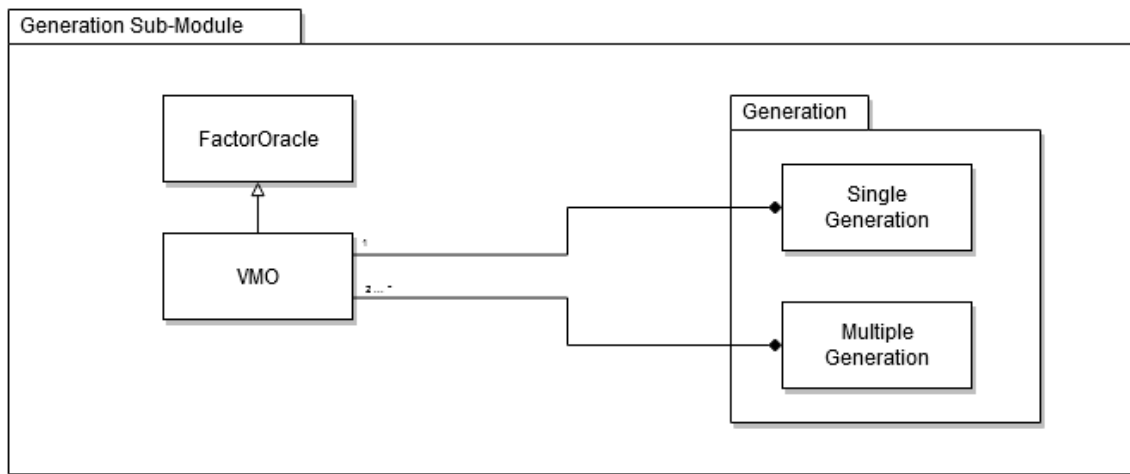
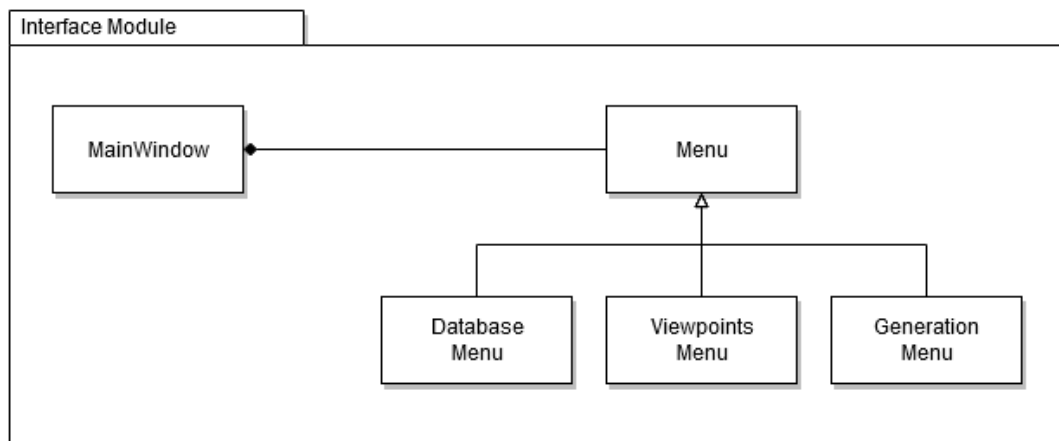


Figure 4.2: Structure of **My Musical Suggester**'s Representation Sub-Module.

The representation sub-module is structured as shown in Figure 4.2 to address the representation of symbolic musical surfaces, by implementing a Multiple Viewpoint Model. It consists on multiple parsers for extracting and storing symbolic musical information from MusicXML files, notably temporal structure, such as phrasing and segmentation and reverse-parsing of these structures for creating new, standard-compatible musical files.

The generation sub-module addresses the generation of stylistic-driven music structures. In modeling existing musical structures, the component algorithms of this module learn characteristics enforced by the representations to promote the generation of new sequences with structural resemblances. This sub-module is structured as shown in Figure 4.3.

The second module concerns the interface of **My Musical Suggester**. It allows composers to interact with the processes that are incorporated in the first module, either in the form of a single cross-platform application, or as a plugin integrated with the music notation editor, Musescore. This module is structured as shown in Figure 4.4.

Figure 4.3: Structure of **My Musical Suggester**'s Generation Sub-Module.Figure 4.4: Structure of **My Musical Suggester**'s Interface Module.

4.2 Representation Sub-Module

This section details the implementation of the Multiple Viewpoint Model as discussed in Chapter 3, as well as pre-processing functions for formatting the resulting data according to the requirements of the temporal model, VMO.

4.2.1 Multiple Viewpoint Models

A musical score incoming in the MusicXML format is parsed by a *MusicParser* class that divides the musical information in Parts by the process described in 3.4.1. It retains every part information as an array of part events, whose name is the respective instrument (if existent, otherwise as a generic Piano) and voice number (if more than one exists for the same instrument). This events are obtained by a *PartParser*, that traverses the original part score and retains the information of every note, rest or chord into an event containing viewpoints. The *MusicParser* also creates the artificial 'inter-part' by 'chordifying' the score and retaining the information that pertains to the

relationship between the multiple-parts using a *InterPartParser*. The *MusicParser* exports and imports already parsed music to an existing database, in either JSON or pickle format, retaining its parts and inter-part and respective events.

The part and inter-part events are encoded as classes *PartEvent* and *InterPartEvent* that descend from a new class *Event*. The latter represents an *Event* in a musical score. An *Event* is identified by its offset and a dictionary of the respective viewpoints, described in 3.4 and 3.5, nested by categories. For compatibility with the distance functions and the VMO creation algorithm used, an *Event* can easily be converted in a feature list or a dictionary of flattened features, in which features that can have more than one value are subdivided in the necessary boolean features. To non-existent values in features, a large value of 10000 is attributed. Before being used, all events as lists of features are normalized using a normalization function that sums all normalized values to one with media to zero. This process is showed in Figure 4.5.

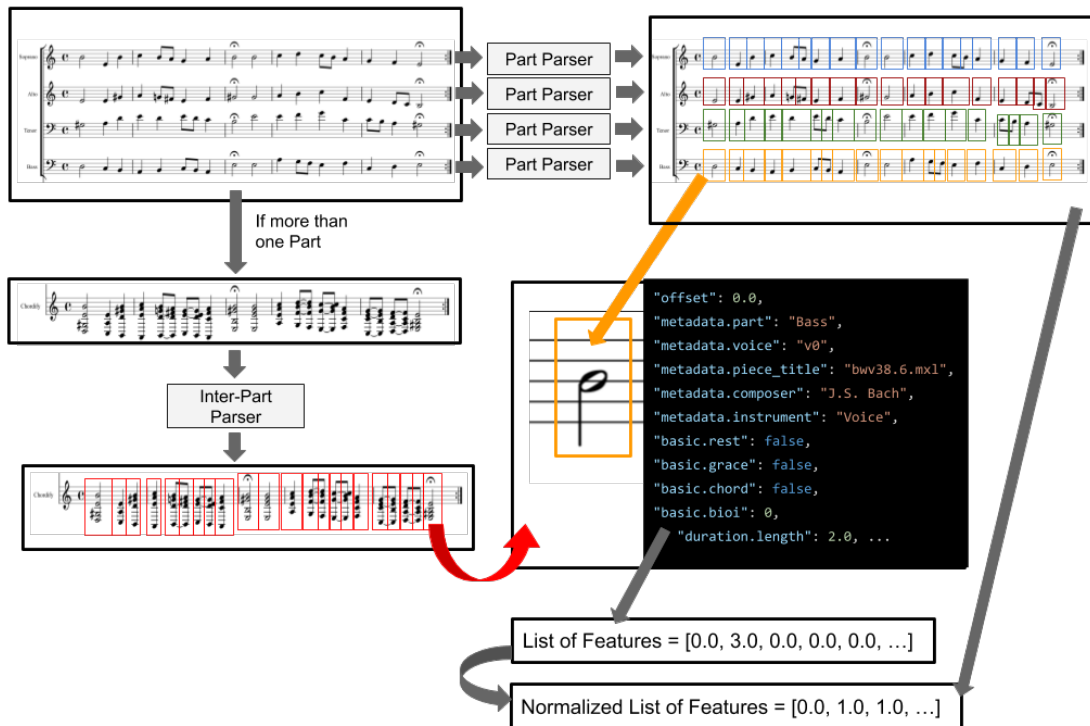


Figure 4.5: Process of Extracting the Multiple Viewpoints for both Part and Inter-Part Events, starting from Part extraction to normalization of features.

The part events as features can be reversed into *PartEvents* and from there a *ScoreConverter* translates the information back into a musical score in MusicXML format, using the process described in Section 3.6.

4.2.2 Segmentation of events

Temporal music structure, such phrases, sections (e.g., chorus) or any above hierarchy is not explicitly encoded in a musical score, or its digital MusicXML counterpart. Due to the relevancy of this information in modelling music structure, it was vital to implement an automatic segmentation method that could identify in the musical time these structural boundaries, so that a much deeper understating of the surface could be captured by the model..

Particular relevancy was given to phrase segmentation. Several algorithmic method based on music-theoretical and computational approaches have been proposed for the task at hand, such as The Local Boundary Detection Model [17], Grouper [81], ATTA [35], the multi-strategy system proposed in [73] and the state-of-the-art Peak Picking Boundary Location [67].

We adopted the Peak Picking Boundary Location algorithm by Pearce and Wiggins, that is based on statistical learning. It identifies phrase boundaries from musical sequences as peaks in a function capturing the information content and entropy from multiple attributes of the musical surface. This algorithm can be summarized in the following three steps:

1. The event following a boundary should have a greater or equal boundary strength than the one following it;
2. The event following a boundary should have greater or equal boundary strength than the one preceding it;
3. The event following a boundary should be higher than a threshold based on the linearly weighted mean and standard deviation of all events preceding it, which translates in the formula:

$$S_n > k \sqrt{\frac{\sum_{i=1}^{n-1} (w_i S_i - \bar{S}_{w,i...n-1})^2}{\sum_{i=1}^{n-1} w_i}} + \frac{\sum_{i=1}^{n-1} w_i S_i}{\sum_{i=1}^{n-1} w_i} \quad (4.1)$$

The parameter k determines how many standard deviations higher than the mean of the preceding values a peak must be to be selected.

To calculate the boundary strength of an event, its normalized feature list representation, and the ones of the previous and posterior events are used. It is estimated by multiplying itself to the change degree to both previous and following events, as can be observed in the Algorithm 4.1.

Not all features achieve good results in segmenting a musical phrase. Those that do are bioi, being a rest event, chromatic pitch, having a fermata, being in a closure sequence, the degree of the event in the scale, both considering the key analysis at the measure level and change in key signature and existence of double barlines or repeats, as they have direct influence in the perception of musical phrases.

The features used for segmenting the part scores are dependent on the viewpoints selected by the user of the application and those mentioned above. From the viewpoints selected, those that do not belong in the group of 'better' segmentation viewpoints are filtered and not used in the segmentation process.

Algorithm 4.1 Boundary Strength Calculation

```

1: function CHANGEDEGREE( $Event1 = (v1_0v1_1...v1_n)$ ,  $Event2 = (v2_0v2_1...v2_n)$ )
2:    $D \leftarrow$  Array of length  $n + 1$ 
3:   for  $i = 0 \rightarrow n$  do
4:     if  $v1_i \neq v2_i$  then
5:        $D[i] \leftarrow |v1_i - v2_i| / (v1_i + v2_i)$ 
6:     else
7:        $D[i] \leftarrow 0$ 
8:   return  $D$ 

9: function BOUNDARYSTRENGTH( $CurrentEvent = (v1_0v1_1...v1_n)$ ,  $PreviousEvent$ ,
    $NextEvent$ )
10:   $D1 \leftarrow$  CHANGEDEGREE( $PreviousEvent$ ,  $CurrentEvent$ )
11:   $D2 \leftarrow$  CHANGEDEGREE( $CurrentEvent$ ,  $NextEvent$ )
12:   $S \leftarrow$  Array of length  $n + 1$ 
13:  for  $i = 0 \rightarrow n$  do
14:     $S[i] \leftarrow D1[i] + D2[i]$ 
15:     $S[i] \leftarrow S[i] * v1_i$ 
16:  return  $S$ 

```

The first and last events are always considered boundaries (beginning and ending, respectively).

After the calculation of the boundaries, the following viewpoints are added to the event and attributed the highest existent weight value to the respective weights:

- 'phrase.boundary', with possible values minus one, zero and one, corresponding to beginning, middle and ending of a phrase.
- 'phrase.length', matching the length of the phrase in which the event is located.
- 'derived.intphrase', that corresponds to the numeric interval to the last phrase beginning.

4.2.3 Multiple Viewpoint Weights

One of the core features of our system is that we allow for choosing the weights to assign to each viewpoint that will be used to differentiate the importance of each viewpoint while calculating the level of similarity of the events. In order to assign weights to the viewpoints, including weight zero that means that the viewpoint will not be used at all, we explore two different solutions: manual assignment and automatic computation.

We always wanted to give the possibility of complete decision to the user, so that they can decide which information to learn. For example, a composer that has a pre-defined melodic sequence in mind, but is uncertain what rhythm to use, he/she could probably be more interested in learning rhythmic-only features. Furthermore, not every feature is either considered in the database works or equally important. Each user is then able to subjectively adapt their generation by defining different sets of weights. In this sense, manual assignment was always the preferred solution.

However, given the elevated number of different possible viewpoints to choose from, it is confusing and time-consuming to assign the weight of each viewpoint, one to one. To overcome this difficulty, we implemented an automatic calculation of the viewpoints' weights. It is a good initial strategy for the user, as it makes an initial guess of the importance of each feature so the end combination better captures style. The more prominent the weight, the more relevant the viewpoint is for the model.

To automatically compute viewpoints' weights we consider the standard deviation of a viewpoint as a good indicator of its relevancy within the musical structure under analysis, as it measures the amount of variation present in its value. If a feature has a standard deviation of zero, it means that it does not vary at all during the piece. Then, it is not a good feature to learn, as it means that in the final model, every event will be considered equal, using that viewpoint. On the other hand, a high standard deviation indicates that the values of the viewpoints can spread far from the mean and thus, the values will be important to learn as they will lead to less equal events and more realistic connections. All the viewpoint's values variances are normalized to the $[0, 100]$ range, in order to facilitate its perception.

While experimenting with different features and weights, we found that some features required a stronger weight calculation in order to achieve better results in view of metric and modulations. Those are, for example, the position of an event in a bar or the keys at an event, which if not all equal, can lead to a lot of jumps between events in different keys, that musically does not make much sense. Therefore, we implemented a flag system to identify these viewpoints, which we called '*fixed*' weight. A '*fixed*' weight forces the evaluation of the similarity of events to give even more importance to the features marked, in the sense that they must be exactly equal for the events to be considered similar. Otherwise, the events will be considered not-similar, even if the distance between events calculated with the other normally-weighted viewpoints would not be so great.

4.3 Generation Sub-Module

The Generation Sub-Module includes the algorithm for modeling the temporal structures derived from the multiple viewpoints model in the as an VMO automata, which can then be adopted to generate new musical sequences that retain stylistic traits from the original modelled music. Ultimately, we aim to promote the generation of novel, diverse musical sequences that can unlock the creative process of a composer.

In the next subsections, we will introduce the mechanics of the VMO automata (Section 4.3.1) and the generation of new sequences from these (Section 4.3.2). To generate new sequences, we explore two strategies: a Single-Part generation (Section 4.3.2.1), that goes through a single VMO in order to generate a musical sequence of just a part, and a Multiple-Part generation mechanism (Section 4.3.2.2) that synchronizes multiple VMOs, corresponding to different parts of a musical score, including an artificial line extracted from inter-parts information, in order to generate a sequence comprised of multiple parts.

4.3.1 Variable Order Markov Oracle

The algorithm for the Variable Order Markov Oracle (VMO) was first proposed by Wang and Dubnov [85] in 2014 for clustering multivariate time series of discrete data without *a priori* specification of the number of clusters. The algorithm is based on the following two automata models: FO [1] and FO [30].

Departing from the Python implementation of the VMO made available by Wang on his Github repository¹, we modified the code in order to make possible the use of different weights, including a distance measure that allows for a mix of fixed and non-fixed weights.

The VMO is, essentially, a finite state automaton that is constructed online from incoming discrete symbols. Contrarily to the finite nature of the alphabet in the FO, the VMO (and its predecessor, FO) can process multi-variate feature in infinite range values. Each symbol is associated with a new state and the VMO states can be connected by two types of links: forward and suffix links. For computing the links, a threshold θ is used as the criterion to determine the similarity between symbols, represented by feature vectors. The online construction of a VMO is documented in the Algorithms 4.2 and 4.3.

Algorithm 4.2 Online Construction of VMO

Require: Time Series as $O = O_1 O_2 \dots O_T$

- 1: Create an Oracle P with initial state p_0
 - 2: $sfx_P[0] \leftarrow -1$, $B \leftarrow \emptyset$, $N \leftarrow 1$
 - 3: **for** $t = 1 \rightarrow T$ **do**
 - 4: $Oracle(P = p_1 \dots p_t) \leftarrow AddSymbol(Oracle(P = p_1 \dots p_{t-1}), O_t)$
 - 5: **return** $Oracle(P = p_1 \dots p_T)$
-

¹<https://github.com/wangsix/vmo>

Algorithm 4.3 Adding Symbol to VMO

Require: Oracle $P = p_1 \dots p_t$, time series instance O_{t+1}

- 1: Create a new state $t + 1$
- 2: $q_{t+1} \leftarrow 0$, $sfx_p[t + 1] \leftarrow 0$
- 3: Create a new transition from t to $t + 1$, $\delta(t, q_{t+1}) = t + 1$
- 4: $k \leftarrow sfx_p[t]$
- 5: **while** $k > -1$ **do**
- 6: $D \leftarrow$ distances between O_{t+1} and $O[\delta(k, :)]$
- 7: **if** all distances in D is greater than θ **then**
- 8: $\delta(k, q_{t+1}) \leftarrow t + 1$
- 9: $k \leftarrow sfx_p[k]$
- 10: **else**
- 11: Find the forward link from k that minimizes D , $k' \leftarrow \delta(k, :)[\text{argmin}(D)]$
- 12: $sfx_p[k] \leftarrow k'$
- 13: **break**
- 14: **if** $k = -1$ **then**
- 15: $sfx_p[t + 1] = 0$
- 16: Initialize a new cluster with current frame index, $b_{N+1} \leftarrow t + 1$
- 17: $B \leftarrow [B; b_{N+1}]$
- 18: Assign a label to the new cluster, $q_{t+1} \leftarrow N + 1$
- 19: Update number of clusters, $N \leftarrow N + 1$
- 20: **else**
- 21: Assign cluster label based on assigned suffix link, $q_{t+1} \leftarrow q_{k'}$
- 22: $b_{q_{k'}} \leftarrow [b_{q_{k'}}; t + 1]$
- 23: **return** Oracle($P = p_1 \dots p_{t+1}$)

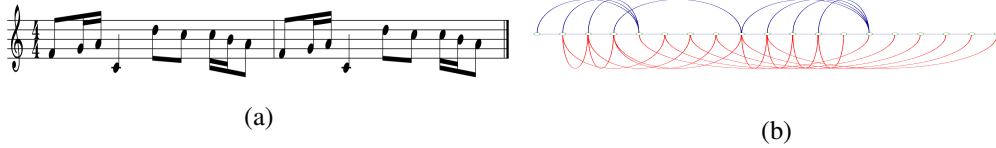


Figure 4.6: VMO in 4.6b constructed with the duration length information of the notes in the score in 4.6a.

Figure 4.6b presents a VMO constructed from the duration length information of the score in Figure 4.6a. The blue lines represent forward links, while the red lines correspond to suffix links. The first state is an artificial state that happens before the start of the music. As we can observe, the forward links capture continuations in the temporal structure, by storing the symbol associated, providing an efficient way to retrieve the factors of a sequence from its beginning and following a unique path. There are two types of these connections: the internal forward links, that connect two consecutive states; and the external forward links, that point to non-consecutive states. From these links, we can get sequences similar to the original one.

The suffix links point backwards and connect states to ones that share the longest repeating suffix of the online sequence up to the current state. It identifies repeated patterns in the sequence.

The VMO implementation adopted equally promote a greater number of paths across the states by storing reverse suffix links (i.e., pointers of the states that jump to this one), without losing the characteristics that distinguish the suffix links. In the case of Figure 4.6, the first pattern to appear is that of two repeated semiquavers. Then, when a quaver and two semiquavers appear, even if not in the exact positions in the measure, the suffix links recognize that this is the new longest repeated pattern in those states. In the second measure, as the duration lengths are exactly equal to the ones of the first measure, the suffix links go back to the exact state in which the same pattern, starting at the current length appeared the first time. The set of the first measure's duration lengths becomes the pattern.

Tracking the suffix links across the states of the oracle structure and grouping the states gives rise to the formation of clusters where: 1) a minor distance than θ is ensured between states connected by suffix links; 2) their relation to other clusters is sequential, by virtue of their first state being dependent on its previous one, which belongs to the last cluster; and 3) the unique suffix link at each state guarantees the every state belongs exclusively to a single cluster. The clusters are maintained by introducing a list of pointers that relate the states to each cluster.

The threshold θ is calculated by analysing the Information Rate (IR) measure for various candidate thresholds and selecting the one with the highest value, similarly to the threshold estimation created for AO [28]. This is made using a cumulative IR calculation, specified in Algorithm 4.4, upon oracles constructed with every candidate threshold. It is a time-expensive process if the number of events to be accumulated is very high.

Algorithm 4.4 IR using VMO introduced during the development of AO [28].

Require: Array K containing a list of VMO encoding event occurrences, unconditional complexity $C = \log_2(|\delta(0, :)|)$, $M = \max(LRS)$, and sequence length N

- 1: **for** $i = 1 \rightarrow |K| - 1$ **do**
- 2: $L = K(i + 1) - K(i)$
- 3: $IR[K(i) : K(i + 1)] = \max(C - \log_2(N) + \log_2(M)L, 0)$
- 4: **return** Array IR

However, this value is extremely important, as a too small IR value can lead to the formation of a very high number of clusters. As every symbol is typically unique, no “real” pattern or repeated motifs could be extracted otherwise. A too large IR threshold identifies all symbols as equals, grouping them in the same cluster which will mean that the whole sequence will be identified as a single pattern and no stylistic attributes from the original sequence will be learned, as is possible to observe in Figure 4.7. For the information that was used to construct the VMOs presented in the figure, the one represented by Figure 4.7d ($\theta = 0.3$) would be the selected one, as it is the one that achieves the best IR result. We can see that the VMO constructed with this threshold is more balanced than the others: does not have too many forward nor suffix links, which indicates that neither the whole sentence will be identified as a pattern (as clearly happens in Figure 4.7a), or no pattern will be identified (Figure 4.7f). Using the IR metric, we guarantee that the best possible amount of patterns from the original sequence will be recognized [85].

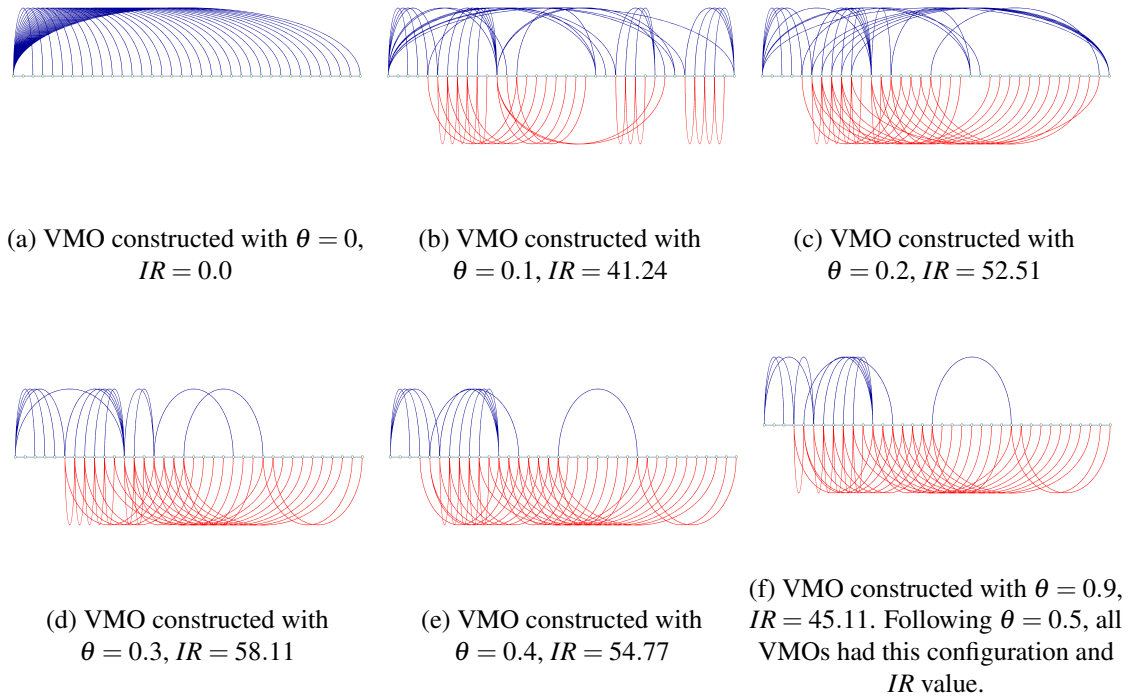


Figure 4.7: VMOs constructed with different threshold values from the same sequence and same feature weights.

Our alteration to the original implementation allows for different feature weights when calculating the distance between symbols (represented as feature lists). This similarity is calculated using weighted cosine distances conjugated with fixed weights. First, the features selected as fixed are compared. If they are not exactly equal, the distance is discarded as too long, using a fixed value of 100000. Otherwise, the cosine distance will be calculated using the weights coming from the user. This increment is important because we want the possibility of getting stronger links between states in accordance with different criteria and different features.

4.3.2 Generation of sequences

Generating new musical sequences from the resulting VMO is a fundamental problem of current research. There are two different modalities that the user can choose for the sequences to be generated, either musical sequences for extending an existing part of a score or to address the entire texture and instrumentation, thus tanking into account all the parts and their synchronization. What follows details these two modalities. Note that, at the moment, **MyMusicalSuggester** only permits continuations of the musical events modeled and, as such, all the generations should start in the last state of the oracle. In future versions of the software, we hope to facilitate the choice to the user of whether the sequences created should be continuations or variations (starting on the first state of the model).

4.3.2.1 Single-Part Generation

The first modality was already implemented in the original version of the VMO and was ported by its developers from the earlier FO algorithm.

The process of a single oracle generation, starts in the last state of the oracle, as we want to generate continuations sequences, and at each new iteration of the algorithm, a new state will be calculated and the respective symbol will be saved to the new sequence.

Algorithm 4.5 Single Part Generation

Require: Oracle P , length to generate N , probability of forward links p , minimum LRS θ , algorithm type *choice*

```

1:  $k \leftarrow$  number of states in  $P - 1$ 
2: for  $i = 0 \rightarrow N - 1$  do
3:   if  $sfx_P[k] \neq 0$  then
4:     if  $\text{RAND}(0.0, 1.0) < p$  then Copy forward according to transitions
5:        $I \leftarrow \delta_P(k, :)$ 
6:       if  $I = \emptyset$  then
7:          $k \leftarrow sfx_P[k]$ 
8:          $ktrace \leftarrow [ktrace; k]$ 
9:          $I \leftarrow \delta_P(k, :)$ 
10:       $sym \leftarrow I[\text{RANDINT}(0, \text{length}(I) - 1)]$ 
11:       $Seq \leftarrow [Seq; sym]$ 
12:    else find backward links and copy any of the next symbols
13:       $ktrace \leftarrow [ktrace; k]$ 
14:       $kvec \leftarrow \text{FINDLINKS}(\emptyset, sfx_P, rsfx_P, k)$ 
15:       $kvec \leftarrow [x \mid x \in kvec, lrs[x] \geq \theta]$ 
16:       $lrs\_vec \leftarrow [lrs[x] \mid x \in kvec]$ 
17:      if  $kvec \neq \emptyset$  then
18:        if  $choice = \text{"max"}$  then
19:           $sym \leftarrow kvec[\text{argmax}(lrs\_vec)]$ 
20:        else if  $choice = \text{"weight"}$  then
21:           $sym \leftarrow \text{BALANCEDSYM}(kvec, lrs\_vec)$ 
22:        else
23:           $sym \leftarrow kvec[\text{RANDINT}(0, \text{length}(kvec) - 1)]$ 
24:        if  $sym = \text{length}(sfx_P) - 1$  then
25:           $sym = sfx_P[sym] + 1$ 
26:        else
27:           $Seq \leftarrow [Seq; sym + 1]$ 
28:      else
29:         $sym \leftarrow k + 1$ 
30:        if  $k \geq \text{length}(sfx_P) - 1$  then

```

```

31:           $sym \leftarrow sfx_P[k] + 1$ 
32:           $Seq \leftarrow [Seq; sym]$ 
33:      else
34:           $sym \leftarrow k + 1$ 
35:          if  $k \geq \text{length}(sfx_P) - 1$  then
36:               $sym \leftarrow sfx_P[k] + 1$ 
37:           $Seq \leftarrow [Seq; sym]$ 
38:       $k \leftarrow sym$ 
39:       $ktrace \leftarrow [ktrace; k]$ 
40:      if  $k \geq \text{length}(sfx_P) - 1$  then
41:           $k = 0$ 
42:      Return  $Seq, ktrace$ 

```

In Algorithms 4.5 the algorithm for generating a new sequence is provided. At every step, if no suffix links exist in the current state of the VMO, it goes forward to the next state, adopting the symbol that corresponds to this transition. In the specific case of reaching the last state of the VMO if no suffix links exist, the generative algorithm moves to the first state of the automaton (state zero) which directs itself to the next state, i.e., the first symbol of the original sequence.

Algorithm 4.6 Single Part Generation Auxiliary Calculations

```

1: function FINDLINKS( $kvec, sfx, rsfx, k$ )
2:    $kvec \leftarrow \text{SORT}(kvec)$ 
3:   if  $0 \notin kvec$  then
4:       if  $sfx[k] \notin kvec$  then
5:            $kvec \leftarrow [kvec; sfx[k]]$ 
6:       for  $link \in rsfx[k]$  do
7:           if  $link \notin kvec$  then
8:                $kvec \leftarrow [kvec; link]$ 
9:       for  $link \in kvec$  do
10:           $kvec \leftarrow \text{FINDLINKS}(kvec, sfx, rsfx, link)$ 
11:          if  $0 \in kvec$  then break
12:   return  $kvec$ 

13: function BALANCEDSYM( $kvec, lrs\_vec$ )
14:    $max\_lrs \leftarrow \max(lrs\_vec)$ 
15:    $query\_lrs \leftarrow max\_lrs - \text{floor}(\text{RAND}(0, \text{expovariate}(1)))$ 
16:   if  $query\_lrs \in lrs\_vec$  then
17:        $indexes \leftarrow [x \mid x \in lrs\_vec, lrs\_vec[x] = query\_lrs]$ 
18:        $tmp \leftarrow indexes[\text{RANDINT}(0, \text{length}(indexes) - 1)]$ 
19:   else

```

```

20:       $tmp \leftarrow \operatorname{argmin}([|x - \text{query\_vec}| \mid x \in \text{lrs\_vec}])$ 
      return  $kvec[tmp]$ 

```

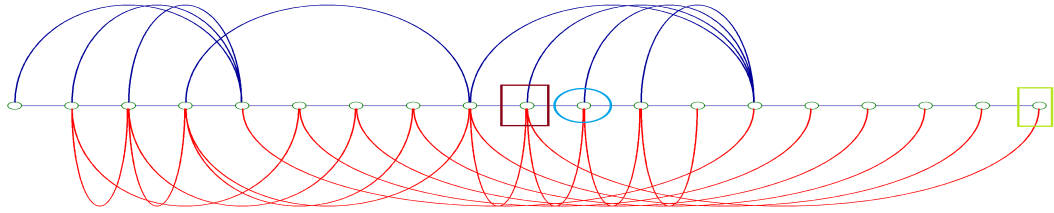
If the state has a connection to another state by a suffix link, a random probability dictates whether it will follow one of the possible transitions, randomly selected and learn its respective symbol or try to find the possible sfx and rsfx recursively (by using the `FindLinks` function in Algorithm 4.6 and pick one to jump to. This choice is done depending on the algorithm selected when calling the generator, comprised by the three options: 1) uniformly among all the possible sfx and rsfx given the current state; 2) the one having the longest repeated suffix (LRS); or one backward jump that favors longer LRSs than shorter ones, done using the `BalancedSym` function in Algorithm 4.6. After, if the state to which the jump occurred is not the last one, it follows the forward transition to the next state and stores the symbol in the sequence. At the end of each iteration of the algorithm, it is analysed if the current state is the last one of the VMO, and in that case, it returns to the state zero.

In Figure 4.8, we can observe three of the four different situations that can happen during the generation process. In this case, as the last state has a suffix link, we cannot show the case in which we reach the last state and it does not have a suffix link.

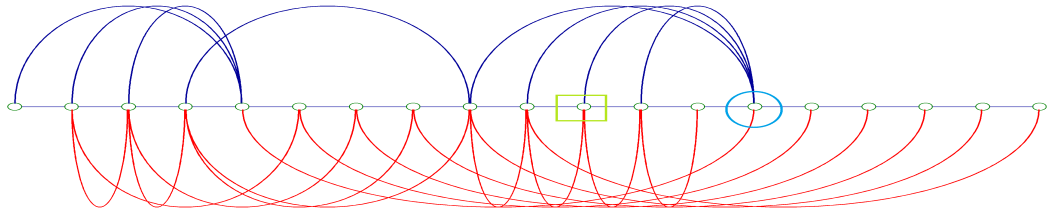
4.3.2.2 Generation of synchronized Multiple Parts

The multiple-part generation presents an additional synchronization challenge in comparison with the single part generation. An existing solution to this problem has not been addressed in the literature using the FO or any of its subordinate models (such as the FO or VMO). Yet, the task at hand is fundamental in composition, thus we advance with a solution which aims to capture the stylistic and relational attributes of the inter-part texture (i.e., the vertical relations between the parts of a musical score). To this end, we propose a parallel VMO algorithm, which includes as many VMO as the number of individual parts of a composition to be modelled plus one artificial line resultant from the extracted inter-part events. Figure 4.9 represents four part oracles and an artificial line, surrounded by a green box, corresponding to a Bach choral.

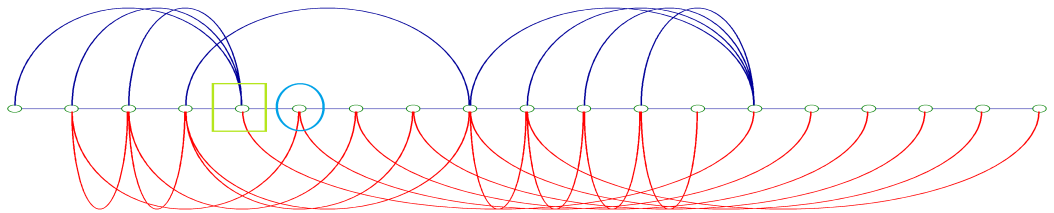
The parallel VMO algorithm starts by identifying a leading oracle as the one featuring the larger number of states. It is important that we make this choice, in order to have a reference during the generation process, because not every oracle will have a state at a certain point in time. Choosing the one with the larger number of states (and probably the one with most movement) will assure that the generation can reach all points in time. In most cases, as we are using the 'inter-part' line, it will be the one selected, because it samples the original parts in chords of smaller duration comprising at least the same amount of states as the original part with the highest number of states, as we can see in Figure 4.9, where the grey lines denote the synchronization of the states at a point in time. If another part has an event in an offset where that part doesn't, this 'inter-part' line will still have that as another event. This happens, for example, at the seventh line, in which only the first two parts have a state and still, the artificial line includes that state.



(a) We start at the last state of the oracle. From this state, we do not have a forward transition, so we will choose one of the suffix links. As there are various possible suffix links that have LRS greater than 2, we randomly jump to state 9 and go to state 10.



(b) Now we are at state 10. As the probability that was randomly selected is lower than 0.26, we can choose from the forward links. We randomly choose to take the one that is not going to the next state, but to state 13.



(c) After some iterations, we are now on state 4. In this state, no suffix links go backwards. Then, we go forwards, to state 5.

Figure 4.8: Three Conditions in a generation from a single oracle, that started with a threshold LRS of 2 and probability of taking forward links of 26%. The algorithm used to choose sfx is the one that chooses uniformly among all the possible suffix (or reverse suffix) links given the current state. The current state is annotated by a green box and the next real state by a blue circle. In the jump by suffix links, a red box is used to annotate the state to which the jump occurs.

The multiple-part generation follows similar principles as the single-part generation. At each iteration of the algorithm, we follow a similar approach to the one used in the single-line generation, except that for the symbol stored, the whole synchronized block will be copied, instead of just the one symbol for each part. As described by Algorithm 4.7, this is done by finding the beginning offset of all parts at the offset of the next state and the correspondent part and the highest offset of all parts at the beginning of the state right after. If there is no event in a part at the beginning offset, the difference to the first event offset after is annotated as a "no state". The same occurs when the last event of a part finishes before the max offset for all parts. That way we can guarantee that the whole block is synchronous and there won't be complete destruction of the vertical relations

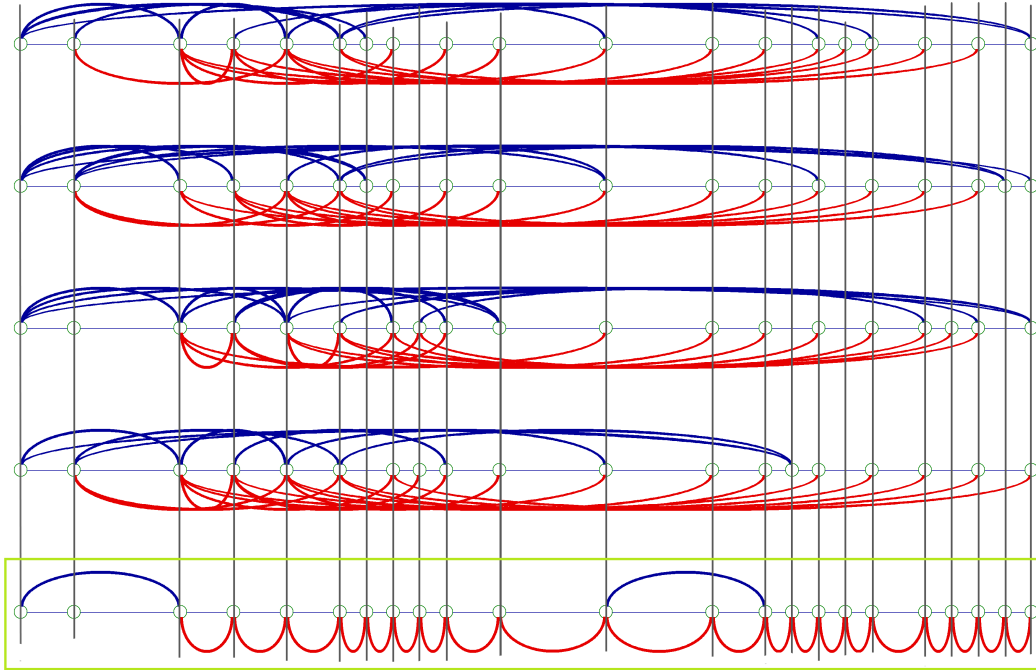


Figure 4.9: A multiple part model, in which the last VMO, surrounded by a green box, matches the artificial line extracted from inter-part information.

between parts of the generated score, although some new pauses that were not in the original score will be written. The process of establishing a block for a transition is illustrated in Figure 4.10.

To start the generation, if the last synchronized state does not correspond to the last state of every oracle, we depart from the state before the last state in which all the oracles are synchronized and then make a straight-forward transition to that completely synchronized state, without storing any symbol. This allows for synchronization of the parts, as every oracle will start its generation from the initial state of the last block of states.

Because the description of the multi-line generation algorithm is very complex and cannot be fully represented in a single figure, a video demo is provided in ² to facilitate the understanding of the process. There, we can observe that if the current state is not the last and there are no suffix links from the current states, every oracle will transition to their next state. In the new generated sequence, a vertical block of symbols that corresponds to the transition will be retained. This block comprises the symbols of all oracles until reaching the end duration of the largest retrieved symbol. If any of the oracles is in the last state, a suffix link is chosen from those of all parts, preferably one that leads to a state which features synchronized events in all oracles. Then, we compute the block of events for that suffix link (taking into account the oracle to which it belongs) and transition to the state after the end of the block. If all oracles have suffix links departing from the current states, a randomly generated number will indicate the probability of transitioning to another state by following a forward-link to a symbol-connected state or a suffix link. In both

²<https://github.com/NadiaCarvalho/Dissertation/wiki>

Algorithm 4.7 Block Recognition and Storing

Require: $Oracles = P_1 P_2 \dots P_N$, states at resulting 'symbol' for every oracle $next_states$, $offsets$ at every state of each oracle

```

1:  $min\_offset \leftarrow \min([offsets[i][next\_states[i]] \mid i \in Oracles])$ 
2:  $max\_offset \leftarrow \max([offsets[i][next\_states[i] + 1] \mid i \in Oracles])$ 
3: for  $i = 1 \rightarrow N$  do
4:   if  $offsets_i[next\_states[i]] \neq min\_offset$  then If there is no state starting at that time for the oracle, add an artificial state
5:      $Sequences[i] \leftarrow [Sequences[i]; 'None_' + string(offsets_i[next\_states[i]] - min\_offset)]$ 
6:   for  $j = next\_states[i] \rightarrow length(offsets_i)$  do
7:     if  $offsets_i[j + 1] \leq max\_offset$  then
8:        $Sequences[i] \leftarrow [Sequences[i]; j]$ 
9:        $ktraces[i] \leftarrow [ktraces[i]; j]$ 
10:    else Add an artificial state to rectify lack at end of block
11:       $Sequences[i] \leftarrow [Sequences[i]; 'None_' + string(max\_offset - offsets_i[j])]$ 
12:    Continue

```

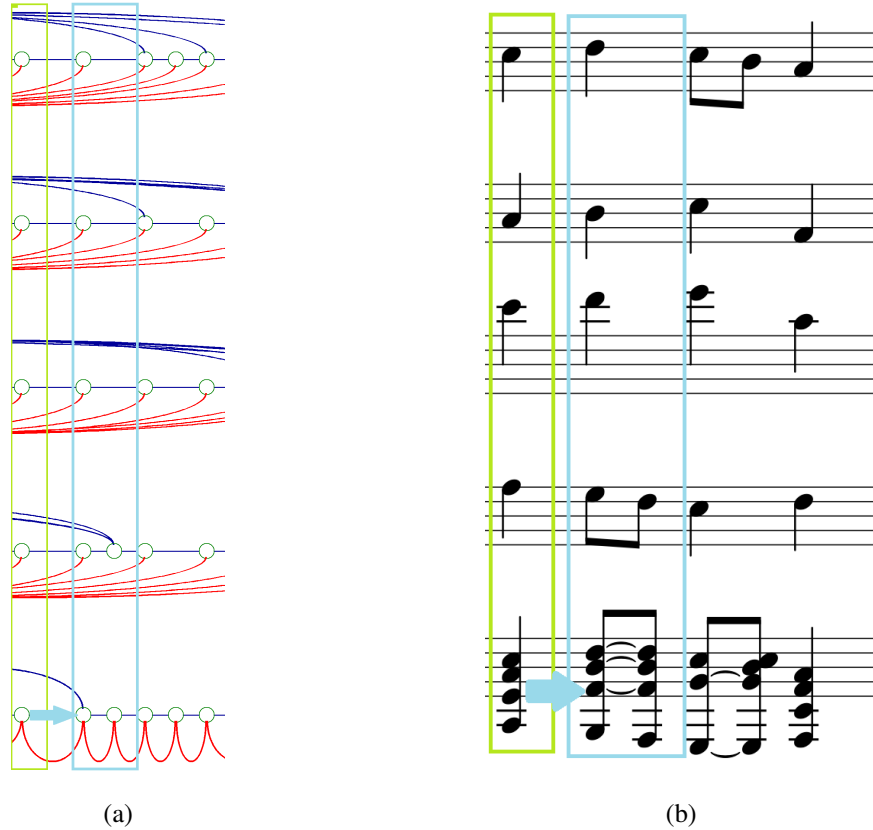


Figure 4.10: In Figure 4.10a, we can observe a block identification for a forward transition, where the current state is the one in the green box and the transition chosen was the one made by the blue arrow in the artificial line. The block stored is the one in the blue box. In Figure 4.10b, we can see the respective musical score.

cases, the preference will be given to connections that lead to states where all the parts have a state at that offset. The choice from the suffix links for this type of generation always maximizes the largest longest repeated suffix, as they are the ones that capture the most important patterns in the music.

4.3.2.3 Ordering the Generated Sequences

We established from the beginning that the sequences should be provided to the user in an ordered way, by proximity to the original sequence. To calculate this distance, we use a similar measure for both single-part and multiple-part generations, as the use of blocks of states in the multiple-part generation guarantees the synchronous characteristic of the original sequences.

We measure the ratio of jumps per number of transitions, as that number indicates the number of times that the resulting sequence does not lead to original sequence continuations. For multiple-part sequences, we make this calculation for every part and use its average as the distance value.

4.4 Interface Module

This section details the interface module which aims to provide a high-level access to all aforementioned functions via a GUI. The interface was implemented in the framework Qt, and has two different components: 1) a cross-platform application, that can run independently by simply calling the executable; and 2) a plugin for the notation software Musescore that merely calls the application with the score currently open in that software and being modified.

The **MyMusicalSuggesterPlugin** serves as a connection for the application to the notation software Musescore, so that a composer working on that software can call the application from the notation software. By simply clicking on the plugin the software loads the currently open score as the principal music from which it will create the VMO.

In what follows, we will explain the functioning of the standalone application, its interfaces and flow of utilization.

4.4.1 Standalone Application

The application module was developed in PyQt (version 5)³, a popular Python binding for the Qt cross-platform C++ framework.

³<https://wiki.python.org/moin/PyQt>

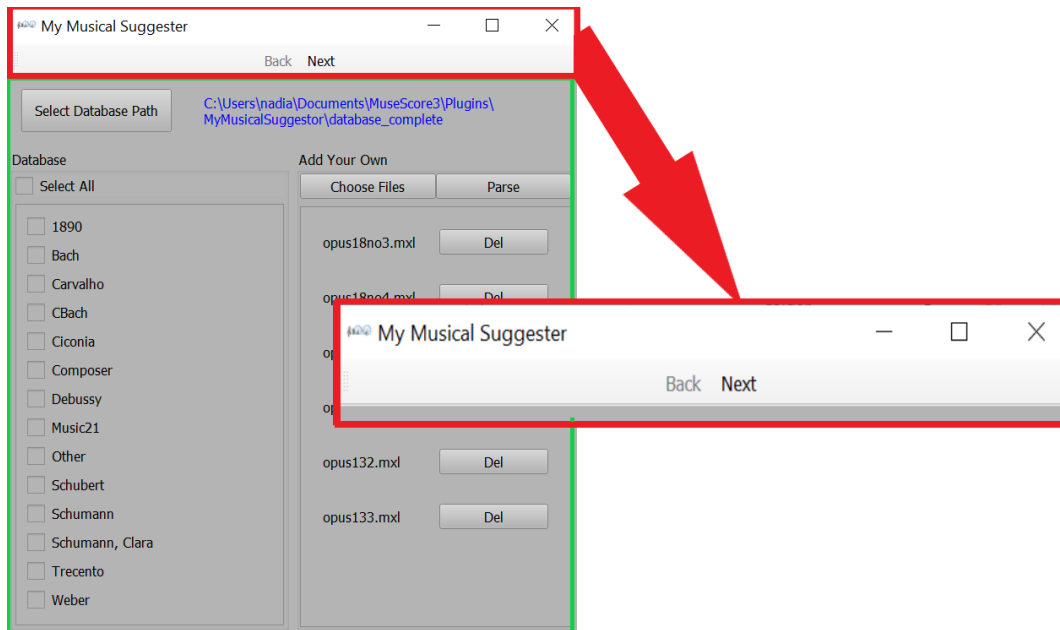


Figure 4.11: General Features of the GUI.

The graphical user interface (GUI) features a main window, with a toolbar for the two main buttons that allow for going back and forward between menus, which are presented in the central, top part of the window, as seen in Figure 4.11. The user can go back on their decisions at any time, after the processes are concluded, signaled by the stopping of a waiting spinner, that appears at the beginning of processing or an alert message. The three following menus are the primary GUI elements in **My Musical Suggester** and appear in the space fixed in Figure 4.11 by a green box. They allow the user to control: 1) the database – a menu to select information to be learned by the model; 2) the viewpoints – a menu for visualizing and adapting the viewpoints weights; and 3) the generation – a menu for selecting the generative modality between single or multiple. Each of these menus will be detailed next.

4.4.2 The Database Menu

The database menu is the first menu that appears and is when the application loads. It includes three main panels, illustrated in Figure 4.12.

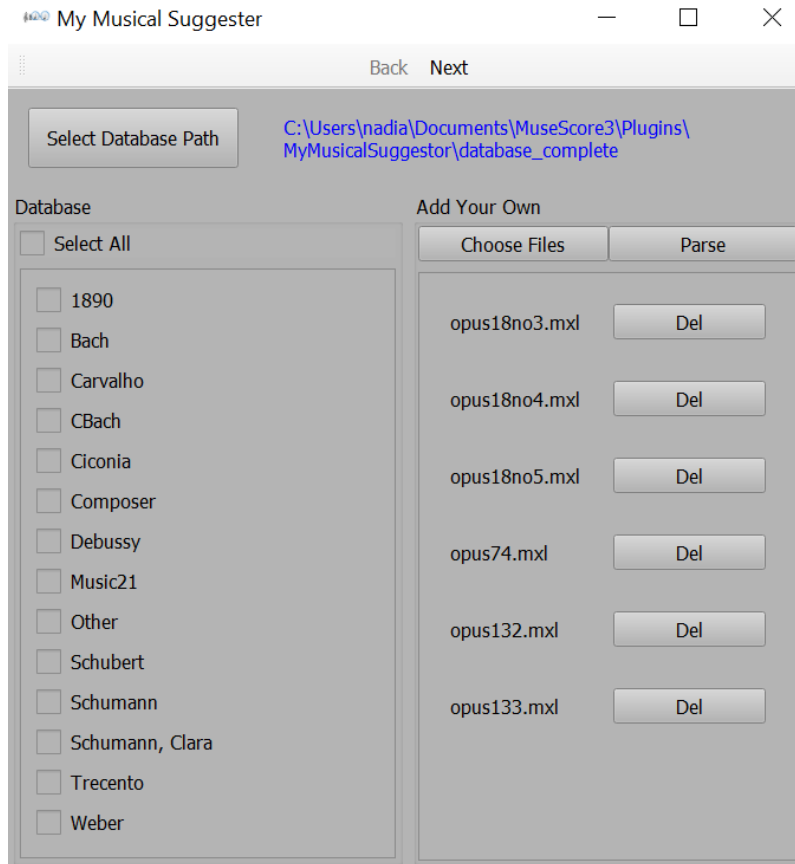


Figure 4.12: The Database Menu with music on the database path, and a few music selected to parse.

In the left panel, the user is able to select folders from an existing database the user wants to include in the model. If necessary, they can choose the path to the database, by clicking on the button on the top panel and traverse the file selector dialog. The current path will appear next to the button.

In the right panel, the user can import their own musical collection in MusicXML (i.e., .xml or .mxl). Then, the contents are parsed and saved to the database in the syntax adopted by the format used by **My Musical Suggester**. A progress bar indicates the state of completion of the parsing. Once the parsing is completed, the folders with the composers' last name will appear in the left panel featuring the *internal* database folders.

4.4.3 The Viewpoints Menu

The viewpoints menu is one of the core GUI elements in **My Musical Suggester**. It allows the definition of the weights to be adopted in the model construction.

The user definition of the weights was implemented by a panel for presenting the computed statistics of each viewpoint, by type of event, and a weight decider as showed in Figure 4.13.

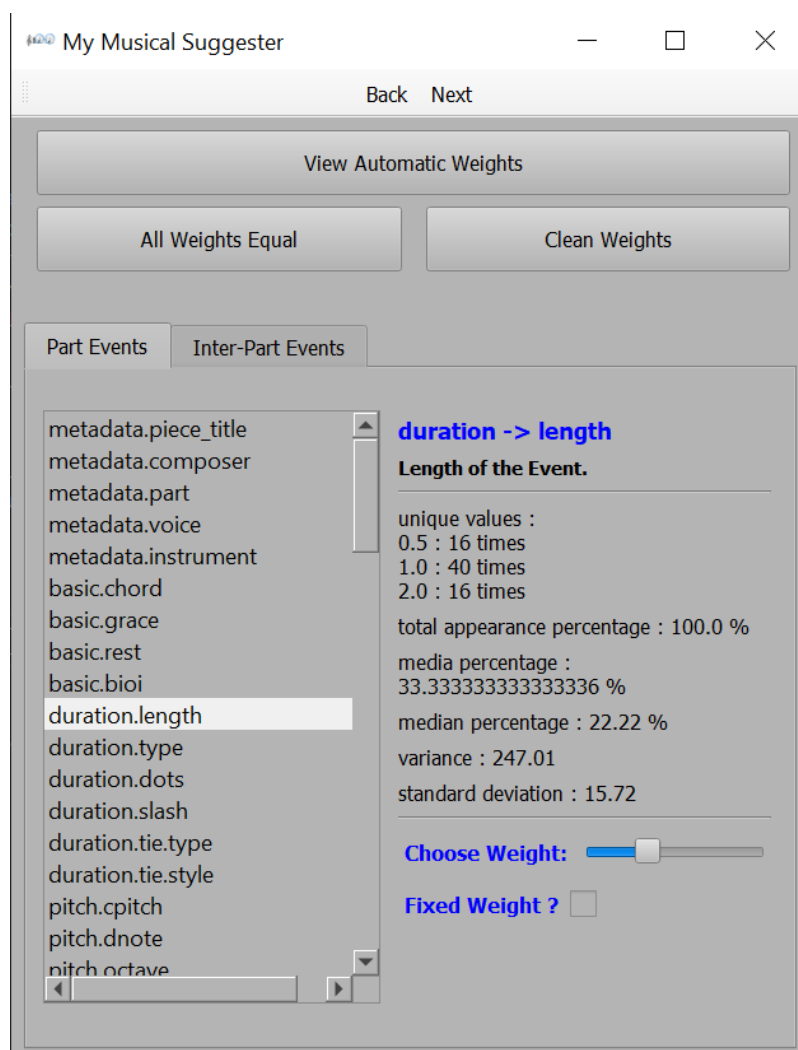


Figure 4.13: The Viewpoints Menu after clicking on *View Automatic Weights*.

In the top part of the menu we encounter three primary buttons: 'Show Automatic Viewpoints', 'All Weights Equal' and 'Clear Weights'. The first button must be used before the other two. It triggers the calculation of the viewpoints' statistics, as well as computing the best combination of weights for the viewpoints using the method described in 4.2.3. This automatic weights assignment streamlines the process, as the user has about 80 and 40 viewpoints to specify for the single part and multiple-part generation methods, respectively.

In the main part of the menu, we encounter two tabs for part and inter-part viewpoint settings. The latter only appears if there are inter-part events (multiple parts) in the original scores. Each tab is composed by a list of viewpoints and a panel in which the information regarding its statistics are displayed. This information consists of a brief description of the viewpoint and their occurrence in the music processed. It consists in unique values of viewpoint (or number if higher than three), the percentage of total appearance of the viewpoint, the media and median of the percentages of each unique value and its variance and standard deviation, as shown in the right side of Figure 4.13.

In the lower part of the panel, the weights for the viewpoints can be manually set by the user

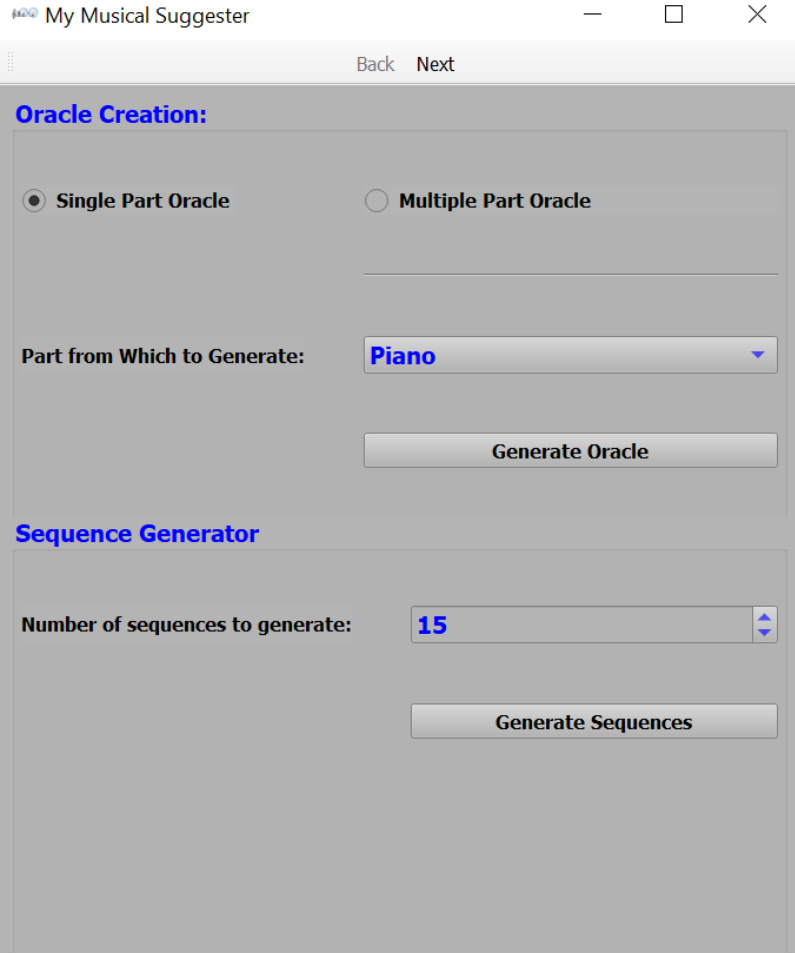
by manipulating the slider values (0-100 scale range). Furthermore, a checkbox, named 'Fixed Weight', forces the viewpoints condition to be met in the model. If selected, the weights slider is disabled and the specified value ignored.

Once the weights are selected, should advance to a new menu by clicking the 'next'. If the all weights equals zero, the last menu becomes inaccessible.

As the names clearly indicate, the 'All Weights Equal' and 'Clear Weights' buttons set all weights to equal values (arbitrarily set to half of the slider) or zero, respectively.

4.4.4 The Generation Menu

In the Generation Menu, the user specifies the generative strategy as either single, selecting 'Single Part Oracle' and the part from which to generate in the selection box bellow, or 'Multiple Part Oracle' and click on the 'Generate Oracle' button. Once the oracle is constructed, a 'number box' requires the user to specify the number of sequences to be generated. Figure 4.14 shows the interface of the menu after selecting the generative strategy.



The screenshot shows a window titled 'My Musical Suggester' with standard window controls (minimize, maximize, close). Inside the window, there are two main sections: 'Oracle Creation' and 'Sequence Generator'. The 'Oracle Creation' section has two radio buttons: 'Single Part Oracle' (selected) and 'Multiple Part Oracle'. Below these is a dropdown menu labeled 'Part from Which to Generate:' with 'Piano' selected. A 'Generate Oracle' button is positioned below the dropdown. The 'Sequence Generator' section has a text label 'Number of sequences to generate:' followed by a numeric input field containing '15'. A 'Generate Sequences' button is located below the input field. At the top of the window content area, there are 'Back' and 'Next' navigation buttons.

Figure 4.14: The Generation Menu after clicking on *Create Oracle*.

After the generation is finished, a message will guide the user to the folder (one per generation) where the generated sequences are ordered by similarity to the original data. The distance measure can be observed in the name of the file following a 'distC'. The user can, then, open the sequences in the notation software they wish, as they are saved in the MusicXML format.

Chapter 5

Evaluation and Results

This chapter details the evaluation of **MyMusicalSuggester**. To this end, we collected expert-based feedback across composers. A twofold experiment protocol was adopted, aiming to evaluate complementary aspects of **MyMusicalSuggester**. Next, we start by detailing the tests protocol and then the collected results.

5.1 Evaluation protocols

The twofold experiment protocol was adopted to evaluate both the utility of **MyMusicalSuggester** and its potential for an expert-based community of composers. As such, the participants of this test were selected from a pool of post-graduate composition students.

The evaluation was conducted online and consisted of two phases. The first task was a task-oriented test, conducted in an informal setting, and designed to assess most of the features of **MyMusicalSuggester** and evaluate its interface. The second followed was the Creativity Support Index questionnaire, in order to evaluate the creative potential of **MyMusicalSuggester**.

5.1.1 Task-Oriented Test

We defined three tasks aiming to cover the multiple modelling and generation modes of **MyMusicalSuggester**. An interview with open-questions followed the tasks, which were designed to not only allow the participants to learn the tool but also experiment with their own musical material. In order to cover the widest possible number of features, we developed three different usage tests within various contexts of the application.

The first test aimed at evaluating the capacity of the application to learn the style of a Bach chorals corpus. The first task was equally designed for the participants to get acquainted with the interface of the **MyMusicalSuggester** and the backend techniques that are used to generate novel musical sequences. The task could be performed either on **MyMusicalSuggesterPlugin** in Muscore or in the standalone application. The details list of steps are shown in Figure 5.1.

1. Open **MyMusicalSuggester**.
2. Observe the initial menu and choose the Bach Chorals folder provided in the compressed folder that came with the application or 'select all'.
3. Click 'Next' in the toolbar.
4. Explore the viewpoints menu, by clicking on the 'View Automatic Weights' button and reviewing the existent viewpoints and their statistics.
5. Observe the weights automatically attributed to the viewpoints and relate them to their statistics.
6. (Optional) Choose which viewpoints you want to model and their relevance by moving the weight slider from left to right, increasing their importance. If you want a specific viewpoint to be strict in calculating similarity of musical events, click on its 'fixed' checkbox. You can clean the viewpoints or set their importance to the same, central level by using the 'Clear Weights' and 'All Weights Equal' buttons.
7. Click 'Next' in the toolbar.
8. Select 'Multiple Part Oracle'.
9. Click on 'Generate Oracle'.
10. Wait for oracle to be generated.
11. Input the number of suggestions that should be generated.
12. Click on 'Generate Sequences'.
13. Wait for sequences to be generated.
14. Click on the link in the dialog box and go to the folder where the sequences were stored.
15. Open the sequences in your notation software, taking into account the order in which they are present in the folder.
16. Comment on the style-coherency of the results with the original sequence, also present in the generated folder, including the general synchronization of the multiple parts generated.
17. Review the viewpoints used and their weights by going back to the viewpoints menu and analyse the generated sequences in face of the viewpoints chosen.

Figure 5.1: Instructions of First Task.

The second and third tasks were used to evaluate the application's ability to process the composers' own work, either by learning characteristics and generating sequences from a single part of the original musical score, as in the case of the second test or from its multiple parts. Both of the tests were done from a single score and no database, for the participant better understand the resulting output. The participant composers chose viewpoints that they know to be relevant for their compositions or specific viewpoint combinations that lead to different results. At the end, the participants should comment on the utility of the tool in the flow of their composition process and elaborate on the similarity and usefulness of the resulting sequences, in the context of their own works. The steps they followed are detailed in Figure 5.2. The major differences in the two tests are in the item 11 whose selection in the first test corresponds to 'Single Part Oracle' and in the second to 'Multiple Part Oracle' and in the second test, the step 12 should be ignored.

At the end of the tasks, participants were asked to detail the 1) suitability of the interface and its potential utility in real life situations, as well as 2) suggestions for the interface and new features to develop.

5.1.2 Creative Support Index

Based on concepts and theories from creativity research, the Creativity Support Index (CSI) was proposed by Carroll and Latulipe [18] as a standard for evaluating creative tools, such as the ability of a tool to support, promote, or enable creativity.

The CSI stems from the structure and easy usability of the NASA Task Load Index, a standard in human-computer interaction. However, the original factors like work performance, mental, physical and temporal demand and effort and frustration were replaced by dimensions better adapted to the creative flow, namely:

- **exploration**: capacity of the tool for exploring different options and ideas without too much repetition;
- **collaboration**: capacity of working with others during the use of the tool;
- **engagement**: level of involvement while using the tool and possibility of frequent usage
- **effort/reward trade-off**: whether the results are worth the effort of using the tool;
- **transparency**: ability to concentrate in the activity instead of simply in the tool that supports the activity; and
- **expressiveness**: creative potential of the composer while using the tool for support.

The test is divided in two parts. In the first part, the participant must fill a short survey with six statements, one for each factor of the ones listed above, to indicate their level of agreement with each statement on a twenty point scale, where zero stands for total disagreement and 20 for total agreement. Then, the participant has to rate the relative importance of each factor against all the others.

1. Open **MyMusicalSuggester**.
2. Observe the initial menu and select 'Choose Files' on the right panel.
3. Choose a file of your own, in MusicXML format.
4. Click on 'Parse Files' and wait for file to be parsed.
5. Make sure no database is selected on the left panel.
6. Click 'Next' in the toolbar.
7. Explore the viewpoints menu, by clicking on the 'View Automatic Weights' button and reviewing the existent viewpoints and their statistics.
8. Observe the weights automatically attributed to the viewpoints and relate them to their statistics.
9. (Optional) Choose which viewpoints you want to model and their relevance by moving the weight slider from left to right, increasing their importance. If you want a specific viewpoint to be strict in calculating similarity of musical events, click on its 'fixed' checkbox. You can clean the viewpoints or set their importance to the same, central level by using the 'Clear Weights' and 'All Weights Equal' buttons.
10. Click 'Next' in the toolbar.
11. Select 'Single Part Oracle'.
12. Select the Part to generate the oracle from. (If applicable.)
13. Click on 'Generate Oracle'.
14. Wait for oracle to be generated.
15. Input the number of suggestions that should be generated.
16. Click on 'Generate Sequences'.
17. Wait for sequences to be generated.
18. Click on the link in the dialog box and go to the folder where the sequences were stored.
19. Open the sequences in your notation software, taking into account the order in which they are present in the folder.
20. Comment on the style-coherency of the results with the original sequence, also present in the generated folder, and if they would make good continuations for that specific musical work.
21. Review the viewpoints used and their weights by going back to the viewpoints menu and analyse the generated sequences in face of the viewpoints chosen.

Figure 5.2: Instructions of Second Task.

The value of the creativity support index is computed by multiplying each factor ranking in the first part of CSI by its importance, counted as number of times it is more important than another factor. Then, we sum these values and divide the sum by 3 to normalize the value to a scale of zero to one-hundred.

5.2 Results

We add a pool of four composer participants (alumni) that are in the beginning of their careers. Half of them are currently following different artistic directions, namely by purposing their careers abroad. This trait was meant to promote a large array of diversity in the tested ideas. In the next sections, we detail the feedback that was provided in the interview phase and the results of the creative support index.

5.2.1 Task-Oriented Test

For the first task, the opinions were that the harmonic successions seem to work predominantly well but sometimes the melodies had not always very Bach-like jumps. Nevertheless, depending on the viewpoints chosen, the result can variate greatly in the capture of some stylistic features of the original. For example, a general opinion seemed to be that the end of a phrase (given by suspended chords, placed on fermata points), seem to appear at indiscriminate times, not very similarly to the original.

When using **MyMusicalSuggester** with original, contemporary music, either monophonic or polyphonic, the repetition of ideas is quickly learned and simple or containing the same motive repeated a lot of times generates sequences very similar to the original. It was discussed that it could be a good thing, in the sense that it would work as an advice to the composer that the material used is in a static state and it should be worked on, in order to achieve greater variety in the composition. At the same time, if there was a better variety of material, the results would be equally more variate and, sometimes, completely different than expected. These were the traits highlighted by the composers, as they thought immediately of combinations that they could use in the tool. It was unanimous that the tool achieved its purpose and the material generated could give a composer a new way of bypassing potential block.

All the tasks were easily done, without too much hassle and the participants were very impressed with the results and the possibility of using **MyMusicalSuggester** to help support their composition process.

5.2.2 Creative Support Index

The results of the Creative Support Index support the interview results. In the first part of the test, where the participants had to assign a score of 0 to 20 to each factor, according to their views on the software and its capacities, there was no negative score attributed to any factor, which we thought that transmitted the positive impact of **MyMusicalSuggester**. At the same time, all the

factors have an average value rounding from 15.5 to 19, as seen in Figure 5.3. This leads us to the same conclusion.

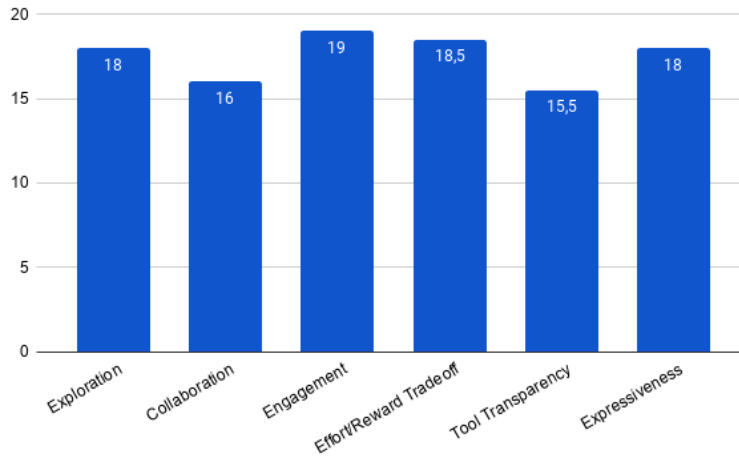


Figure 5.3: CSI average score by factor, measured from 0 (less relevance of the software in this factor) to 20 (higher relevance of the software in this factor).

From the factors' average result, a clear distinction can be made from the ones that scored higher to the other ones. Although not the worst, we start by evaluating the *collaboration* factor, that had an average score of 16. This score was relatively higher than we were expecting as **MyMusicalSuggester** did not have a *collaboration* component, per se. We think that it was interpreted as the possibility of composers sharing their pieces and learning from those, or even the possibility of collaborating on viewpoint choice, not on the application itself, but during the composition process. At the same time, the interest in this factor is clearly low, as may be observed in the graphic in Figure 5.4, which perhaps explains the not so low score in the part one results. The low interest was expected, in the sense that the composition process is mostly an individual process and only in singular situations, such as improvisation, *collaboration* between composers or musicians is really expected, and in the case of **MyMusicalSuggester**, it was not factored.

On the other hand, the *transparency* of the tool, was the factor that rated the most negative. This can be derived from some bugs that occurred during the tests, as well as the time that it took to generate the oracles and sequences from a larger number of events. Additionally, as the process behind the sequence generation is transparent, it can lead to frustration, due to the resulting lack of understanding. More integration with the notation software could help in increasing the score of this factor, as well as increasing the efficiency of the generation processes. The interest of the users in this factor is, however, less relevant to their composition activity.

Exploration and *expressiveness* both rate high in the interest of the participants. They are the two most important factors for composers while creating their works. The *exploration* part is easily understood in **MyMusicalSuggester**: from a single model generated, it is possible to generate countless sequences, without making a lot of repetitive steps: after the VMO(s) are generated, simply clicking on the generate sequences button can be done multiple times. Even if a user

wants to modify the viewpoints for the same music, they only have to go back a menu, modify the ones they want and generate the VMO(s) once more, to remember new information. At the same time, the high score attributed to *expressiveness* is equally explained by the potential of using the composer's own musical pieces and not only style-limited music, as in most tools. This increases the perception of possible expression of the composer's own ideas, which is most relevant to all the participants.

Effort/Reward Tradeoff and *engagement* had the highest factor scores, as all participants agreed that the resulting sequences, specially the ones that were generated from models of their own works were worth the time they took to generate. However, we must note that this factor was ranked lower in terms of interests. The participants all concurred that they would use **MyMusical-Suggester** with frequency in their composition process, particularly in the beginning phase, were generating a lot of different suggestions, even if they will not be all used, can trigger new ideas for the musical compositions.

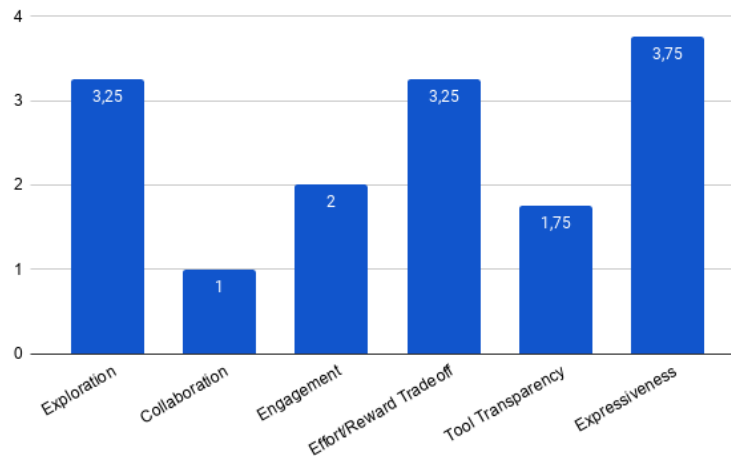


Figure 5.4: Average values of factors importance, measured from 0 (less interest) to 5 (higher interest).

At the end, the final score of the CSI test (89.1 out of 100, with the participants scores ranging from 81 to 98). The final score suggests that **MyMusicalSuggester** shows great potential to support creative music tasks, namely in assisting the composition process.

Chapter 6

Conclusions

6.1 Summary

In this dissertation, we presented **MyMusicalSuggester**, a software prototype which explores a model for assisting the creative process of a composer by learning stylistic features from a symbolic music corpus and suggesting novel musical content to the user. Generated sequences are presented to the user ranked by proximity to the original. The model behind **MyMusicalSuggester** expands on the combination of Multiple Viewpoint Models for representing musical information from variable length contexts (e.g., single works or large corpus). VMO was adopted to capture temporal relations between musical events and generate new sequences from the user-input material. The information captured by the models is always dependent on the user, that has total freedom to choose the viewpoints adopted, as well as their relative weight.

The main challenge encountered in the development of this tool concerned the synchronization of multiple parts, either in how to deal with their extraction from the musical scores or how to learn and generate music synchronized sequences.

The results of the evaluation process, both the task-oriented test commentaries and the creativity support index result of 89.1 in 100, support the conclusion that **MyMusicalSuggester** has the potential to be a great tool for composers in their creative activity, to help unlock the process in blocking situations but also to use generically, during the composition process.

6.2 Contribution

With this research, we contributed to the artistic and scientific community with the development of a new support tool for the compositional creative process that suggests new musical sequences, ordered by level of proximity to the original. The created sequences aim to unlock the composition process and give the total freedom to the composers to manipulate what the system learns from a musical corpus, as well as the type of sequences we want to generate.

We implemented a generation system for simultaneous musical lines, expanding of the VMO algorithm, which has not been widely explored in literature to process harmonic generation (from

multiple layers). In literature, only the generation from a single automaton has been explored. Our work advances with a solution to the problem with relative success. Furthermore, while undergoing the construction of the oracles, the application of weights to the viewpoints, including fixed weights was not at all envisioned in previous VMO systems, yet they showed great potential in constructing different degrees of dependency across the viewpoints.

The process of viewpoint extraction from musical scores in MusicXML, while not new, introduced a series of new possible features to learn from and their organization in part and inter-part events made it possible to increase the level of information that can be extracted and modeled from a musical score. At the level of part extraction, we feel that our method brings some improvement over the one already existent in music21 because it allows for taking into account the different characteristics of the various instruments while extracting every part.

6.3 Future Work

The interface was not our main priority at this stage of development of the work. The developed interface was designed to accommodate most functions and does not strive for a more simple or appealing visual. In order to reach a wider community of users, i.e., composers, we aim to improve the interface design, as well as allow further degrees of freedom to the user. Some features suggested include the possibility to request the threshold of similarity values of the final sequences, as well as the possibility to choose single works from the folders in the database. Remaining improvements should enhance the viewpoints organization in the interface per by category and integrate different controls for weight selection, such as a MIDI controller.

Concerning the backend processes, further optimization of the VMO should be considered in the future for greater efficiency. Notably, reducing the time needed for calculating the oracle when the size of the information to learn is very high ought to be considered. Parsing the information could not be much faster, unfortunately. We aim to introduce even more derived viewpoints in next versions of the **MyMusicalSuggester**, notably those related to the duration in the part events and with temporal cadences in the inter-part events.

References

- [1] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Factor Oracle: A New Structure for Pattern Matching. In Jan Pavelka, Gerard Tel, and Miroslav Bartošek, editors, *Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics on Theory and Practice of Informatics*, volume 1725 of *SOFSEM '99*, pages 295–310, Berlin, Heidelberg, 1999. Springer-Verlag.
- [2] Charles Ames. Automated Composition in Retrospect: 1956-1986. *Leonardo*, 20(2):169–186, 1987.
- [3] Christopher Ariza. Navigating the landscape of computer aided algorithmic composition systems: A definition, seven descriptors, and a lexicon of systems and research. In *Proceedings of International Computer Music Conference*, 2005.
- [4] Christopher Ariza. *An Open Design for Computer-Aided Algorithmic Music Composition: Athenacl*. PhD thesis, New York University, USA, 2005.
- [5] G. Assayag and S. Dubnov. Using factor oracles for machine improvisation, 2004.
- [6] Gérard Assayag, Georges Bloch, Marc Chemillier, Arshia Cont, and Shlomo Dubnov. OMax brothers: A dynamic topology of agents for improvisation learning. *Proceedings of the ACM International Multimedia Conference and Exhibition*, (October):125–132, 2006.
- [7] Gérard Assayag, Camilo Rueda, Mikael Laurson, Carlos Agon, and Olivier Delerue. Computer-assisted composition at ircam: From patchwork to openmusic. *Comput. Music J.*, 23(3):59–72, September 1999.
- [8] Ron Begleiter, Ran El-Yaniv, and Golan Yona. On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, 22(1):385–421, 2004.
- [9] D E Berlyne. Novelty, complexity, and hedonic value. *Perception & Psychophysics*, 8(5):279–286, 1970.
- [10] John Biles. Genjam: A genetic algorithm for generating jazz solos. 07 1994.
- [11] Rens Bod. A general parsing model for music and language. In Christina Anagnostopoulou, Miguel Ferrand, and Alan Smaill, editors, *Music and Artificial Intelligence*, pages 5–17, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.
- [12] Dimitri Bouche, Jérôme Nika, Alex Chechile, and Jean Bresson. Computer-aided Composition of Musical Processes. *Journal of New Music Research*, 46(1):3–14, jan 2017.
- [13] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. *Deep Learning Techniques for Music Generation*. Computational Synthesis and Creative Systems Series. Springer, November 2019.

- [14] F. P. Brooks, A. L. Hopkins, P. G. Neumann, and W. V. Wright. An experiment in musical composition. *IRE Transactions on Electronic Computers*, EC-6(3):175–182, Sep. 1957.
- [15] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, 1994.
- [16] Allen Clayton Cadwallader, David Gagne, and Frank Samarotto. *Analysis of tonal music*. Oxford University Press, 1998.
- [17] Emiliós Cambouropoulos. The local boundary detection model (lbdm) and its application in the study of expressive timing. 01 2001.
- [18] Erin Cherry and Celine Latulipe. The creativity support index. pages 4009–4014, 01 2009.
- [19] J. Cleary and I. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, April 1984.
- [20] Darrell Conklin and Christina Anagnostopoulou. Representation and discovery of multiple viewpoint patterns. In *International Computer Music Conference*, pages 479–485, 2001.
- [21] Darrell Conklin and Ian H. Witten. Multiple Viewpoint Systems for Music Prediction. *Journal of New Music Research*, 24(1):51–73, 1995.
- [22] Lelouda Custodero, Lori A. and Stamou. 9. International Conference on music perception and cognition : 6. Triennial Conference of the European society for the cognitive sciences of music : abstracts : Alma Mater Studiorum, University of Bologna Italy, August 22-26, 2006. In *Engaging classrooms: Flow Indicators as tools for pedagogical transformation*, pages 1666 – 1673. Bononia University Press, 2006.
- [23] Michael Cuthbert and Christopher Ariza. Music21: A toolkit for computer-aided musicology and symbolic music data. pages 637–642, 01 2010.
- [24] Michael Scott Cuthbert. User’s guide, chapter 9: Chordify, 2006-2017. https://web.mit.edu/music21/doc/usersGuide/usersGuide_09_chordify.html, Last accessed on 2020-06-18.
- [25] Daichi Ando and Hitoshi Iba. Interactive composition aid system by means of tree representation of musical phrase. In *2007 IEEE Congress on Evolutionary Computation*, pages 4258–4265, Sep. 2007.
- [26] Ken Déguernel, Emmanuel Vincent, and Gérard Assayag. Probabilistic Factor Oracles for Multidimensional Machine Improvisation. *Computer Music Journal*, 42(2):52–66, jun 2018.
- [27] Christos Dimitrakakis. Bayesian variable order Markov models. In *Journal of Machine Learning Research*, volume 9, pages 161–168, 2010.
- [28] S. Dubnov, G. Assayag, and A. Cont. Audio oracle analysis of musical information rate. In *2011 IEEE Fifth International Conference on Semantic Computing*, pages 567–571, 2011.
- [29] Shlomo Dubnov, Gérard Assayag, Gill Bejerano, and Olivier Lartillot. A System for Computer Music Generation by Learning and Improvisation in a Particular Style. *IEEE Computer*, 10(38):1–15, 2003.

- [30] Shlomo Dubnov, Gérard Assayag, and Arshia Cont. Audio oracle: A new algorithm for fast learning of audio structures. In *International Computer Music Conference, ICMC 2007*, pages 224–227, 2007.
- [31] B. Eno. *A Year with Swollen Appendices*. Faber & Faber, 1996.
- [32] Lee Frankel-Goldwater. Computers Composing Music: An Artistic Utilization of Hidden Markov Models for Music Composition. *Journal of Undergraduate Research*, 5(1 and 2):17–20, 2007.
- [33] Édouard Gilbert and Darrell Conklin. A probabilistic context-free grammar for melodic reduction. 01 2007.
- [34] Michael Good. Musicxml: An internet-friendly format for sheet music. 01 2001.
- [35] Masatoshi Hamanaka, Keiji Hirata, and Satoshi Tojo. Automatic time-span tree analyzer based on extended gttm. In *in Proceedings of the Sixth International Conference on Music Information Retrieval, ISMIR 2005*, pages 358–365, 2005.
- [36] Pierre Hanna, Matthias Robine, Pascal Ferraro, and Julien Allali. Improvements of alignment algorithms for polyphonic music retrieval. pages 244–251, 2008.
- [37] D. Herremans and E. Chew. Morpheus: automatic music generation with recurrent pattern constraints and tension profiles. In *2016 IEEE Region 10 Conference (TENCON)*, pages 282–285, Nov 2016.
- [38] Dorien Herremans, Ching-Hua Chuan, and Elaine Chew. A functional taxonomy of music generation systems. *ACM Computing Surveys*, 50(5):1–30, Sep 2017.
- [39] Andrew Horner and David Goldberg. Genetic algorithms and computer-assisted music composition. *Urbana*, 51:437–441, 01 1991.
- [40] Phillip B. Kirlin and Paul E. Utgoff. A framework for automated Schenkerian analysis. In *ISMIR 2008 - 9th International Conference on Music Information Retrieval*, pages 363–368, feb 2008.
- [41] C. S. Lee. The rhythmic interpretation of simple musical sequences : towards a perceptual model. *Musical Structure and Cognition*, 1985.
- [42] Kjell Lemström and Jorma Tarhio. Transposition invariant pattern matching for multi-track strings. *Nordic Journal of Computing*, 10(3):185–205, 2003.
- [43] Fred Lerdahl and Ray Jackendoff. *A generative theory of tonal music*. The MIT Press, 1983.
- [44] Ming Li and Ronan Sleep. Genre classification via an lz78-based string kernel. pages 252–259, 01 2005.
- [45] Feynman T. Liang, Mark Gotham, Matthew Johnson, and Jamie Shotton. Automatic stylistic composition of bach chorales with deep lstm. In *ISMIR*, 2017.
- [46] Christiane Linster. *On Analyzing and Representing Musical Rhythm*, page 414–427. MIT Press, Cambridge, MA, USA, 1992.
- [47] Corentin Louboutin and David Meredith. Using general-purpose compression algorithms for music analysis. *Journal of New Music Research*, 45:1–16, 02 2016.

- [48] Donncha O Maidín. A geometrical algorithm for melodic difference. In *Computing in Musicology*, number 11, page 65–72, 1998.
- [49] Alan Marsden. Representing melodic patterns as networks of elaborations. *Language Resources and Evaluation*, 35(1):37–54, 2001.
- [50] Alan Marsden. Generative structural representation of tonal music. *Journal of New Music Research*, 34(4):409–428, dec 2005.
- [51] Alan Marsden and Anthony Pople. *Computer representations and models in music*. Academic Press, 1992.
- [52] Dave Meredith, Geraint Wiggins, and Kjell Lemström. Pattern induction and matching in polyphonic music and other multidimensional datasets. *Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics*, 10, 08 2002.
- [53] David Meredith. Point-set algorithms for pattern discovery and pattern matching in music. In Tim Crawford and Remco C. Veltkamp, editors, *Content-Based Retrieval*, number 06171 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.
- [54] David Meredith. The ps13 pitch spelling algorithm. *Journal of New Music Research*, 35(2):121–159, 2006.
- [55] David Meredith. RecurSIA-RRT: Recursive translatable point-set pattern discovery with removal of redundant translators. jun 2019.
- [56] David Meredith, Geraint Wiggins, and Kjell Lemström. Pattern discovery and pattern matching in polyphonic music and other multidimensional datasets. 06 2001.
- [57] James Anderson Moorer. Music and computer composition. *Commun. ACM*, 15(2):104–113, February 1972.
- [58] María Navarro-Cáceres, Marcelo Caetano, Gilberto Bernardes, and Leandro De Castro. Chordais: An assistive system for the generation of chord progressions with an artificial immune system. *Swarm and Evolutionary Computation*, 06 2019.
- [59] Gerhard Nierhaus. *Algorithmic Composition: Paradigms of Automated Music Generation*. Springer Publishing Company, Incorporated, 1st edition, 2009.
- [60] Doug Van Nort, Pauline Oliveros, and Jonas Braasch. Electro/acoustic improvisation and deeply listening machines. *Journal of New Music Research*, 42(4):303–324, 2013.
- [61] Francois Pachet. The continuator: Musical interaction with style. *Journal of New Music Research*, 32:333–341, 08 2010.
- [62] Alexandre Papadopoulos, Pierre Roy, and François Pachet. Avoiding plagiarism in Markov sequence generation. In *Proceedings of the National Conference on Artificial Intelligence*, volume 4, pages 2731–2737, 2014.
- [63] Alexandre Papadopoulos, Pierre Roy, and Francois Pachet. Assisted lead sheet composition using flowcomposer. volume 9892, 09 2016.
- [64] George Papadopoulos and Geraint A. Wiggins. A genetic algorithm for the generation of jazz melodies. 2000.

- [65] M. Pearce. *The Construction and Evaluation of Statistical Models of Melodic Structure In Music Perception And Composition*. PhD thesis, City University, London, 2005.
- [66] Marcus Pearce. Idiom, 2005-2020. <https://github.com/mtpearce/idiom/wiki>, Last accessed on 2020-06-28.
- [67] Marcus Pearce, Daniel Müllensiefen, and Geraint Wiggins. A comparison of statistical and rule-based models of melodic segmentation. pages 89–94, 01 2008.
- [68] Anna Pienimäki. Indexing Music Databases Using Automatic Extraction of Frequent Phrases. *3rd Int. Conf. on Music Information Retrieval*, pages 25–30, 2002.
- [69] Richard C. Pinkerton. Information theory and melody. *Scientific American*, 194(2):77–87, 1956.
- [70] D. Rizo. *Symbolic Music Comparison with Tree Data Structures*. PhD thesis, Universidad de Alicante, 2010.
- [71] C. Roads and Paul Wieneke. *Grammars as Representations for Music*, 1979.
- [72] Curtis Roads. *Microsound*, volume 1. 2019.
- [73] Marcelo E. Rodríguez-López, Anja Volk, and Dimitrios Bountouridis. Multi-strategy segmentation of melodies. In *ISMIR*, 2014.
- [74] Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25(2-3):117–149, 12 1996.
- [75] W. Schulze and B. van der Merwe. Music generation with markov models. *IEEE MultiMedia*, 18(03):78–85, jul 2011.
- [76] Marco Scirea, Julian Togelius, Peter Eklund, and Sebastian Risi. Metacompose: A compositional evolutionary music composer. In Colin Johnson, Vic Ciesielski, João Correia, and Penousal Machado, editors, *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, pages 202–217, Cham, 2016. Springer International Publishing.
- [77] David R. W. Sears, Andreas Arzt, Harald Frostel, Reinhard Sonnleitner, and Gerhard Widmer. Modeling harmony with skip-grams. *ArXiv*, abs/1707.04457, 2017.
- [78] SMC. Smcroadmap, 2020. <http://www.smcnetwork.org/roadmap>, Last accessed on 2020-07-03.
- [79] Stephen Smoliar. Schenker: a computer aid for analysing tonal music. *ACM SIGLASH Newsletter*, 10:30–61, 12 1976.
- [80] Paolo Tagliolato. A Generalized Graph-Spectral Approach to Melodic Modeling and Retrieval Categories and Subject Descriptors. pages 89–96, 2008.
- [81] David Temperley. *The Cognition of Basic Musical Structures*. The MIT Press, 2004.
- [82] Mauricio Toro. Probabilistic extension to the concurrent constraint factor oracle model for music improvisation. *Inteligencia Artificial*, 19(57):37–73, 2016.
- [83] Michael Towsey, Andrew Brown, Susan Wright, and Joachim Diederich. Towards melodic extension using genetic algorithms. *Educational Technology and Society*, 4, 04 2001.

- [84] C. Wang and S. Dubnov. Variable markov oracle: A novel sequential data points clustering algorithm with application to 3d gesture query-matching. In *2014 IEEE International Symposium on Multimedia*, pages 215–222, Dec 2014.
- [85] Cheng I. Wang and Shlomo Dubnov. Guided music synthesis with variable Markov Oracle. In *AAAI Workshop - Technical Report*, volume WS-14-18, pages 55–62, 2014.
- [86] Cheng-i Wang and Shlomo Dubnov. Context-Aware Hidden Markov Models of Jazz Music with Variable Markov Oracle. In *Eighth International Conference on Computational Creativity (ICCC 2017)*, number October, 2017.
- [87] Cheng i. Wang, Jennifer Hsu, and Shlomo Dubnov. Music pattern discovery with variable Markov oracle: A unified approach to symbolic and audio representations. In *Proceedings of the 16th International Society for Music Information Retrieval Conference, ISMIR 2015*, pages 176–182, 2015.
- [88] Cheng I. Wang, Jennifer Hsu, and Shlomo Dubnov. Machine improvisation with Variable Markov Oracle: Toward guided and structured improvisation. *Computers in Entertainment*, 14(3):1–18, 2016.
- [89] Rodney Waschka II. *Composing with Genetic Algorithms: GenDash*, pages 117–136. Springer London, London, 2007.
- [90] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens. The context-tree weighting method: basic properties. *IEEE Transactions on Information Theory*, 41(3):653–664, May 1995.
- [91] R Wooller, AR Brown, E Miranda, R Berry, and D Joachim. A framework for comparison of processes in algorithmic music systems. Generative Arts Practice. *Creativity and Cognition Studios Press*, Generative:109–124., 2005.
- [92] Anna K. Yanchenko and Sayan Mukherjee. *Classical Music Composition Using State Space Models*. PhD thesis, 2017.
- [93] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977.
- [94] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, Sep. 1978.

Appendix A

Table of Viewpoints

Index	Name	Part/Inter-Part	Category	Derived From	Definition	Type
0.1 (Info)	part	Part	Metadata		Name of the Part in which the Event exists.	String
0.2 (Info)	voice	Part	Metadata		Voice related to Part in which the Event exists.	String
0.3 (Info)	instrument	Part	Metadata		Instrument name of the Part in which the Event exists.	String
0.4 (Info)	piece_title	Both	Metadata		Title of the musical Work in which the Event exists.	String
0.5 (Info)	composer	Both	Metadata		Composer of the musical Work in which the Event exists.	String
1.1	rest	Both	Basic		Is a rest Event?	Bool
1.2	chord	Part	Basic		Is a chord Event?	Bool
1.3	grace	Part	Basic		Is a grace note Event?	Bool

continued on next page

continued from previous page

1.4	bioi	Part	Basic		Basic Interval of Off-set of the Event to penultimate Event.	Float
2.1	length	Both	Duration		Length of the Event	Float
2.2	type	Both	Duration		Typical name of duration of Event (quarter/half note,...).	String
2.3	dots	Both	Duration		Number of dots added to type.	Integer
2.4	slash	Part	Duration		The rhythmic figure has a slash? (only applies to grace note).	Bool
2.5	type	Both	Duration (Tie)		Has a tie? Is it normal?.	String
2.6	style	Both	Duration (Tie)		Style of the Tie (Dotted, normal, etc. . .).	String
3.1	articulation	Part	Expressions		Articulations applied to Event.	List of Strings
3.2	breath_mark	Part	Expressions		Is in a breath mark?	Bool
3.3	dynamic	Part	Expressions		Dynamics applied to Event.	List of Strings
3.4	fermata	Part	Expressions		Has a fermata?	Bool
3.5	expression	Part	Expressions		Text Expressions applied to Event.	List of Strings
3.6	ornamentation	Part	Expressions		Special Expressions (ornaments) applied to Event.	List of Strings
3.7	rehearsal	Part	Expressions		Is in a rehearsal mark?	Bool

continued on next page

continued from previous page

3.8	volume	Part	Expressions		Volume of Event (Applies information from other expressions such as dynamics and articulation).	Integer
3.9	type	Part	Expressions (Notehead)		Type of Notehead.	String
3.10	fill	Part	Expressions (Notehead)		Notehead is filled?	Bool
3.11	parenthesis	Part	Expressions (Notehead)		Notehead has parenthesis?	Bool
3.12	slur.begin	Part	Expressions (Slur)		Is the Event beginning a slur?	Bool
3.13	slur.between	Part	Expressions (Slur)		Is the Event in a slur, but is not beginning or ending it?	Bool
3.13	slur.end	Part	Expressions (Slur)		Is the Event ending a slur?	Bool
3.14	crescendo.begin	Part	Expressions (Crescendo)		Is the Event beginning a crescendo?	Bool
3.15	crescendo.between	Part	Expressions (Crescendo)		Is the Event in a crescendo, but is not beginning or ending it?	Bool
3.16	crescendo.end	Part	Expressions (Crescendo)		Is the Event ending a slucrescendor?	Bool
3.17	decrescendo.begin	Part	Expressions (Decrescendo)		Is the Event beginning a decrescendo?	Bool

continued on next page

continued from previous page

3.18	decrescendo.betw	Part	Expressions (Decrescendo)		Is the Event in a decrescendo, but is not beginning or ending it?	Bool
3.19	decrescendo.end	Part	Expressions (Decrescendo)		Is the Event ending a decrescendo?	Bool
3.20	clef	Part	Expressions		Clef at the time of the event.	String
4.1	cpitch	Part	Pitch		Midi pitch (plus cents has decimals) of note Event.	Float
4.2	dnote	Part	Pitch		Name of note (C,D,E,F,G,A,B).	String
4.3	octave	Part	Pitch		Number of octave.	Integer
4.4	accidental	Part	Pitch		Accidental (including quarter tones).	String
4.5	microtonal	Part	Pitch		Cents applied to note.	Float
4.6	pitch_class	Part	Pitch		Pitch class of note.	Integer [0,11]
4.7	chordPitches	Part	Pitch		Pitches (cpithes) of Event if Event is chord.	Array of Floats
5.1	keysig	Both	Key		Number of sharps (+) or flats (-) in key signature.	Integer [-7,7]
5.2	key	Both	Key (at Key Signatures)		Name of key and mode.	String
5.3	Inter-Part	Inter-Part	Key (at Key Signatures)		Certainty in avaiating the key.	Float

continued on next page

continued from previous page

5.4	scale_degree	Part	Derived	Pitch, Key (at Key Signa- tures)	melodic degree of note relatively to key.	Integer
5.5	signatures.function	Inter-Part	Derived	Key (at Key Sig- natures)	Harmonic function of chord relatively to key.	String
5.6	key	Both	Key (at Mea- sures)		Name of key and mode.	String
5.7	certainty	Both	Key (at Mea- sures)		Certainty in avaiat- ing the key.	Float
5.8	scale_degree	Part	Derived	Pitch, Key (at Measures)	melodic degree of note relatively to key.	Integer
5.9	measures.function	Inter-Part	Derived	Key (at Measures)	Harmonic function of chord relatively to key.	String
6.1	timesig	Part	Time		Time signature (in string '4/4').	String
6.2	pulses	Part	Time		Nominator of time signature.	Integer
6.3	barlength	Part	Time		Denominator of time signature.	Integer
6.4	value	Part	Time - metro(nome)		Value of metronome if existent.	Integer
6.5	sound	Part	Time - metro(nome)		Value of audible metronome (may not be the same of value in specific situations).	Integer

continued on next page

continued from previous page

6.6	text	Part	Time - metro(nome)		Tempo as Text written on metronome, such as 'Andante', 'Adagio', etc.	String
6.7	value	Part	Time - reference		Value of reference duration.	Integer
6.8	type	Part	Time - reference		Type of reference duration (refer to duration).	String
6.9	double	Part	Time - barlines		Has a double barline after?	Bool
6.10	repeat(_exists_ before)	Part	Time - barlines - repeat		Has a repeat barline before?	Bool
6.11	direction	Part	Time - barlines - repeat		Direction of the repeat barline, if existent.	String
6.12	is_end	Part	Time - barlines - repeat		Is at the end of piece? (Does not appear in 6.10)	Bool
7.1	seq_int	Part	Derived	Pitch	Interval to last non-rest note.	Float
7.2	contour	Part	Derived	Pitch	Movement direction from last non-rest note.	Integer [-1,1]
7.3	contour_hd	Part	Derived	Pitch	Quantized Interval from last non-rest note.	Integer [-4,4]
7.4	closure	Part	Derived	Pitch	Shape defined by the last 3 notes: 1p for change of direction and 1p for a tone smaller than the preceding one.	Integer [0,2]

continued on next page

continued from previous page

7.5	regstral_ direction	Part	Derived	Pitch	Is a large (\geq perfect fifth) jump followed by a direction change or a small (\leq perfect fourth) jump followed by a move in the same direction?	Bool
7.6	intervalistic_ difference	Part	Derived	Pitch	Is a large jump followed by a smaller (3 semitones smaller if in the same direction or 2 semitones if reversing the direction) jump? Is a small jump followed by a similar interval?	Bool
7.7	upwards	Part	Derived	Pitch	The last 3 Events are an ascending sequence?	Bool
7.8	downwards	Part	Derived	Pitch	The last 3 Events are a descending sequence?	Bool
7.9	no_movement	Part	Derived	Pitch	The last 3 Events are the same pitch (there was no movement)?	Bool
7.10	bioi_ratio	Part	Derived	Basic (Bioi)	Bioi divided by the previous bioi.	Float
7.11	bioi_contour	Part	Derived	Basic (Bioi)	Contour (-1, 0, 1) from the bioi of the last Event.	Float
7.12	dur_ratio	Part	Derived	Duration	Current duration length divided by the previous duration.	Float

continued on next page

continued from previous page

7.13	dur_contour	Part	Derived	Duration	Contour (-1, 0, 1) from the duration length of the last Event.	Float
7.14	fib	Part	Derived	Time	Is the first element in a bar?	Bool
7.15	posinbar	Part	Derived	Time	Position of Event in bar.	Integer [0,bar-length]
7.16	beat_strength	Part	Derived	Time	Event Strength relating to position in bar and accentuations.	Float
7.17	tactus	Part	Derived	Time	Is on strong/tactus positions in a bar?	Bool
7.18	anacrusis	Part	Derived	Time	Event is in an Anacrusis Tempo.	Bool
7.19	intfib	Part	Derived	Pitch, Time	Interval from note to fib of respective bar.	Float
7.20	thrbar	Part	Derived	Pitch, Time	Interval from note (if fib) to last fib.	Float
7.21	intphrase	Part	Derived	Pitch, Time, Phrase	Interval from note to first note in respective phrase.	Float
8.1	boundary	Part	Phrase	All the others (or not, chosen)	Is a beginning (1), end (-1) or middle (0) of a phrase?	Integer [-1,1]
8.2	length	Part	Phrase		Number of events in the phrase to which the Event belongs	Integer
9.1	root	Inter-Part	Basic (Chord)	Pitch	Root of the chord.	Float (cpitch)

continued on next page

continued from previous page

9.2	pitches	Inter-Part	Basic (Chord)	Pitch	Pitches that make Part of the chord.	List of Float (cpitch)
9.2	cardinality	Inter-Part	Basic (Chord)	Pitch	Number of notes in chord.	Integer
9.3	inversion	Inter-Part	Basic (Chord)	Pitch	Inversion of the chord.	String
9.4	prime_form	Inter-Part	Basic (Chord)	Pitch	Chord pitches in 0-11 scale, sorted.	String
9.5	quality	Inter-Part	Basic (Chord)	Pitch	Quality of the underlying triad of a triad or sEventh, either major, minor, diminished, augmented, or other.	String
10.1	forte_class	Inter-Part	Classes (Chord)	Pitch	Forte set class name as a string.	String
10.2	forte_class_number	Inter-Part	Classes (Chord)	Pitch	Number of the Forte set class within the defined set group.	Integer
10.3	pc_ordered	Inter-Part	Classes (Chord)	Pitch	Pitch Classes of the chord, ordered.	Array of Integers
10.4	pc_cardinality	Inter-Part	Classes (Chord)	Pitch	Number of unique pitch classes.	Integer
10.5	pitch_class	Inter-Part	Classes (Chord)	Pitch	All pitch classes in the chord as integers. Not sorted.	Array of Integers
11.1	is_consonant	Inter-Part	Quality (Chord)	Basic (Chord)	is a consonant chord?	Bool
11.2	is_major_triad	Inter-Part	Quality (Chord)	Basic (Chord)	is a major triad chord?	Bool

continued on next page

continued from previous page

11.3	is_incomplete_ major_triad	Inter-Part	Quality (Chord)	Basic (Chord)	is an incomplete ma- jor triad chord?	Bool
11.4	is_minor_triad	Inter-Part	Quality (Chord)	Basic (Chord)	is a minor triad chord?	Bool
11.5	is_incomplete_ minor_triad	Inter-Part	Quality (Chord)	Basic (Chord)	is an incomplete mi- nor triad chord?	Bool
11.6	is_augmented_ sixth	Inter-Part	Quality (Chord)	Basic (Chord)	is an augmented sixth chord?	Bool
11.7	is_french_ augmented_ sixth	Inter-Part	Quality (Chord)	Basic (Chord)	is a french augmented sixth chord?	Bool
11.8	is_german_ augmented_ sixth	Inter-Part	Quality (Chord)	Basic (Chord)	is a german aug- mented sixth chord?	Bool
11.9	is_italian_ augmented_ sixth	Inter-Part	Quality (Chord)	Basic (Chord)	is an italian aug- mented sixth chord?	Bool
11.10	is_swiss_ augmented_ sixth	Inter-Part	Quality (Chord)	Basic (Chord)	is a swiss augmented sixth chord?	Bool
11.11	is_augmented_ triad	Inter-Part	Quality (Chord)	Basic (Chord)	is an augmented triad chord?	Bool
11.12	is_diminished_ seventh	Inter-Part	Quality (Chord)	Basic (Chord)	is a diminished sev- enth chord?	Bool
11.13	is_half_di- minished_ seventh	Inter-Part	Quality (Chord)	Basic (Chord)	is an half diminished seventh chord?	Bool
11.14	is_dominant_ seventh	Inter-Part	Quality (Chord)	Basic (Chord)	is a dominant seventh chord?	Bool

Appendix B

Creativity Support Index Form

B.1 Part I

Creativity Support Index

For each sentence below, please select your level of agreement, in relation to the tool.

Exploration

It was easy for me to explore many different options, ideas, designs or outcomes without a lot of tedious, repetitive interaction.

Agree ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ | ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Disagree

Collaboration

I was able to work together with others easily while doing this activity.

Agree ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ | ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Disagree

Engagement

I was very absorbed/engaged in this activity - I enjoyed it and would do it again.

Agree ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ | ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Disagree

Effort/Reward Tradeoff

What I was able to produce was worth the effort required to produce it.

Agree ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ | ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Disagree

Tool Transparency

While I was doing the activity, the tool/interface/system “disappeared” , and I was able to concentrate on the activity.

Agree ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ | ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Disagree

Expressiveness

I was able to be very expressive and creative while doing the activity.

Agree ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ | ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Disagree

Figure B.1: Part I of Creativity Support Index

B.2 Part II

Factor Rankings

For each pair below, please select which factor is most important to you when doing this activity.

- | | | |
|--------------------------------------------|----|-------------------------------------------------|
| <input type="checkbox"/> Exploration | OR | <input type="checkbox"/> Collaboration |
| <input type="checkbox"/> Exploration | OR | <input type="checkbox"/> Engagement |
| <input type="checkbox"/> Exploration | OR | <input type="checkbox"/> Effort/Reward Tradeoff |
| <input type="checkbox"/> Exploration | OR | <input type="checkbox"/> Tool Transparency |
| <input type="checkbox"/> Exploration | OR | <input type="checkbox"/> Expressiveness |
| <input type="checkbox"/> Collaboration | OR | <input type="checkbox"/> Engagement |
| <input type="checkbox"/> Collaboration | OR | <input type="checkbox"/> Effort/Reward Tradeoff |
| <input type="checkbox"/> Collaboration | OR | <input type="checkbox"/> Tool Transparency |
| <input type="checkbox"/> Collaboration | OR | <input type="checkbox"/> Expressiveness |
| <input type="checkbox"/> Expressiveness | OR | <input type="checkbox"/> Effort/Reward Tradeoff |
| <input type="checkbox"/> Expressiveness | OR | <input type="checkbox"/> Tool Transparency |
| <input type="checkbox"/> Expressiveness | OR | <input type="checkbox"/> Engagement |
| <input type="checkbox"/> Engagement | OR | <input type="checkbox"/> Tool Transparency |
| <input type="checkbox"/> Engagement | OR | <input type="checkbox"/> Effort/Reward Tradeoff |
| <input type="checkbox"/> Tool Transparency | OR | <input type="checkbox"/> Effort/Reward Tradeoff |

Figure B.2: Part II of Creativity Support Index

