A Methodology for the Computational Evaluation of Style Imitation Algorithms

by

Nicolas Gonzalez Thomas

B.Comp.Sc., CAECE, 2005

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Interactive Arts and Technology
Faculty of Communication, Art and Technology

SIMON FRASER UNIVERSITY
Fall 2016

APPROVAL

Name: **Nicolas Gonzalez Thomas**

Degree: **Master of Science**

Title of Thesis: **A Methodology for the Computational Evaluation of Style Imitation Algorithms**

Examining Committee

Chair: **Dr. Robert Woodbury**
University Professor

Dr. Philippe Pasquier
Senior Supervisor
Associate Professor

Dr. Arne Eigenfeldt
Supervisor
Professor

Dr. George Tzanetakis
External Examiner
Associate Professor
Department of Computer Science
University of Victoria

Date Defended/Approved: December 5, 2016

Abstract

We investigate Musical Metacreation algorithms by applying Music Information Retrieval techniques for comparing the output of three off-line, corpus-based style imitation algorithms. The first is *Variable Order Markov Chains*, a statistical model; second is the *Factor Oracle*, a pattern matcher; and third, *MusiCOG*, a novel graphical model based on perceptual processes. Our focus is on discovering which musical biases are introduced by the algorithms, that is, the characteristics of the output which are shaped directly by the formalism of the algorithms and not by the corpus itself. We describe META-MELO, a system that implements the three algorithms, along with a methodology for the quantitative analysis of algorithm output, when trained on a corpus of melodies in symbolic form. Results show that the algorithms' output are indeed different, although none of them encompass completely the full feature-set belonging to the style of the corpus. We conclude that this methodology is promising for aiding in the informed application and development of generative algorithms for music composition problems.

Keywords: Music; Methodology; Computational Evaluation; Style Imitation Algorithms; Melodic Generation; Computational Creativity

Dedication

I dedicate this thesis to my parents, my father, for the passion for science and technology, my mother for the gift in the arts.

Acknowledgements

I want to thank my lab members for their support and insight, specially Matthieu Macret and Dr. James Maxwell. My supervisors Dr. Philippe Pasquier and Dr. Arne Eigenfeldt for their insights, direction and patience. Dr. Klaus Frieler for providing me his software for melodic analysis and troubleshooting correspondence. Also, Dr. Tom Loughin for his assistance in designing the statistical methods presented here. Finally, this research was made possible by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computational Musicology has generally focused on studying human composed music; however, algorithms for music generation within the growing field of computational creativity [15] provide a rich, relatively unexplored and new area of study: the *musicology of computer-generated music*. Algorithmic and generative music systems [24, 44, 56, 70, 74] are becoming more sophisticated, diverse and complex as their designers explore models of computational creativity [11]. When designing generative algorithms for accomplishing musical tasks - applied musical AI - it is imperative to be able to effectively investigate which techniques are most suitable for specific problems. It is evident that the applications of these experimental and novel techniques into commercial tools for musicians will be accelerated by the means to efficiently evaluate the outcome of these systems.

This thesis is a step towards the computational evaluation of generative algorithms in the context of *Style Imitation*. *Style Imitation*, a particular Musical Metacreative task, focuses on producing new works that fit a given musical style [4]. We define *Style* as: "A group of rules and characteristics that uniquely and consistently describe a set of creative artifacts". We define *Stylistic Imitation* as: "The production of novel creative artifacts that would be consistently classified as belonging to a particular style by unbiased observers".

The techniques applied for this task can be broadly categorized into two methodological groups: corpus based and non-corpus based methods. In the former, musical knowledge of the style is obtained through empirical induction from existing music compositions, using machine learning techniques. Whereas in the latter, this knowledge is provided by designers in the form of theoretical and/or rule-based representations.

It is common practice to evaluate the output of metacreative systems that follow a stylistic imitation approach by means of listener studies [5, 30, 42, 63]. Agres *et al.* [1] describe several evaluation approaches for creative systems and categorize them as *Internal* vs. *External*: the *Internal* evaluation approach involves mechanisms with which the systems self-evaluate output and incorporate a feed-back into the system to inform further creative generation; the *External* approach involves diverse ways of testing the response of audience/jury members. For example, the output of the system can be compared to the original music (utilized for training the algorithms) by expert judges - this is the *Consensual Assessment Technique* [3]. Another example of external evaluation is a Turing test approach; in a Turing test an uninformed listener attempts to distinguish the original artifacts from the machine generated versions.

As useful as these external evaluation approaches have been, there are three basic drawbacks:

1. they are relatively expensive, time-wise and monetarily. Since studies with human subjects require an extensive process of preparing surveys, finding participants and analyzing the resulting data in ways that are not easily automated;

2. As noted by Wiggins *et al*. [75], in evaluations performed by human listener's the processes and criteria by which they evaluate the artifacts is not evident. Therefore the insight obtained provides little or no accurate empirical knowledge of what exactly are the differences that distinguish these pieces; and

3. although not empirically proven, listeners *unbiased judgment* of computer generated music is in question [51, 60].

We present and describe a methodology for determining:

1. Whether a group of generated artifacts fit the style of a corpus and to what degree that style is imitated, using statistical and purely computational means.;

2. Which musical features serve to more evidently distinguish the group of artifacts from the style;

3. Which characteristics of the output are determined by the algorithms rather than the training material used? That is, we aim to discover the *musical biases* which arise from the formalism of the models, by determining what are the differences that more clearly differentiate them.

This thesis continues the work by Gonzalez Thomas *et al*. [35]. Also, this work has been described in the session report by Wiering and Benetos [73] as a contribution to musicologists and as a useful method for automating the evaluation of generative music.

Finally, an applied example of this work is the doctoral dissertation by Maxwell [47] where the author primarily utilizes the *permutation testing* approach presented here (Section 4.2.7) for an in-depth evaluation of *MusiCOG*.

1.1 Motivation

The main motivation is the empirically informed advancement of intelligent-collaborative music composition environments. Focusing on off-line, corpus-based stylistic imitation algorithms, there appears to be a gap where a computationally automated process could inform the development of these techniques. This could possibly even be part of the algorithm itself, where a real-time feedback-loop informing the algorithm design could be followed by an analysis of the output. This reflection or evaluation would inform the system on the most relevant tweaks required to further improve the output. It has been argued that the use of an internal evaluation feature for learning would distinguish a creative model from a purely generative one [1].

Human composers first learn to create music by imitation. It seems that machines which master style imitation techniques before incorporating methods that seek novelty would be following a time

tested approach. As interactive music technology evolves, some tasks which were deemed possible only to trained musicians will become accessible to a broader audience, much like cell phone cameras with simple/automated but effective image filtering and video editing has allowed a broad crowd of users to venture into the creative role of amateur photographers or videographers.

We envision a toolset able to capture and virtualize most of the techniques which in the past have been hard-learned compositional practices: for example harmonization, orchestration, counterpoint. This will enable an acceleration of higher level meta-choices: those where composing will be not only be a matter of note-against-note (counterpoint) but also one of process-against-process. We expect that there will be a growth in demand for music composition environments for this level of thinking, for experts and novices alike.

For sophisticated or expert musicians, a higher level of complexity and customization could be offered, with several interaction modes: composition environment, intelligent instrument or improvising partner. These systems will also be able to span the continuum of autonomous generation of music vs. interaction in generation.

Also, the mastering of style imitation techniques by metacreative algorithms will potentially allow a fast exploration of unknown musical spaces. For example, the question of what music is obtained by crossing Mozart with Metallica, or by applying Philip Glass orchestration and arrangement style to Miles Davis harmonic progressions. What music is in the space between Bach and the Beatles?

1.2 Methodology Overview

The initial challenge was determining the appropriate methodology for this research problem. A subtask of music style imitation was chosen in order to simplify the task and evaluate the process and methodology. Also, we selected a constrained task in music generation: melodies.

We distinguish the tasks of composition from interpretation and are concerned here only with the former. Therefore, the music representation and analysis does not include performance information (Section 3.3.6).

The first step requires defining the formal space which the artifacts in the style-defining corpus occupy. We use Gärdenfors' theory of conceptual spaces [33] and find it well suited for this domain, as Agres *et al.*. describe [1]: concepts (or features of a style) can be represented as a set of dimensions with geometric features. Whether this space is conceptual, Euclidean or multidimensional, similarity between stimuli in this set is implicitly defined by proximity or distance between points in the space. For example, we can define the space by choosing a sub-set of features which describe rhythmic aspects of a melody: tempo, syncopation, and density. Once the feature-set of the space is defined, the output of the trained algorithms is placed in the space occupied by the corpus. Finally, a method for comparing and evaluating the overlap of these sub-spaces is needed.

We use Music Information Retrieval (MIR) techniques to extract the features which define the space; an inter-algorithm analysis compares features from the melodic output of each algorithm with the corpus and with the output of all other algorithms. A method using similarity measurements is a core factor in how these comparisons are done. The problem of music similarity is an open one and heavily researched in the MIR community; we discuss our approach in Section 4.2.4.

As mentioned, the features used for defining the space and the algorithms used for measuring similarity will provide a completely different result in the analysis. Rendering this work as non-trivial and simultaneously powerful, since it can be potentially applied to analyze and examine several musical dimensions. We are able to focus the analysis on any musical feature or combination of features (hybrid) for which we have a similarity measure. Thus, the results of diverse approaches will be useful in their own way. For example, pitch features can be selected to define the space, melodic interval, contour, etc. ignoring rhythmic information. Then we can use similarity measurements that emphasize melodic complexity, thus isolating a particular dimension in the evaluation. In contrast, we perform an analysis on a weighted hybrid feature that aggregates several dimensions altogether. Since a multitude of analysis perspectives are possible, only some examples used are selected in this thesis, presented and discussed in Chapter 4.

1.3 Contributions

Our contributions are:

1. a corpus of classical, popular, and jazz melodies [34],
2. *META-MELO*: a MAX/MSP implementation of the three algorithms, used for melodic generation from a corpus (Section 3.4),
3. a framework which applies Machine Learning and MIR techniques for algorithm output comparison (Section 4.1).
4. a new algorithm for evaluating corpus similarity with statistical significance, using permutation testing (Section 4.2.7), and
5. the results of a study where we apply this methodology to three methodologically distinct music generation algorithms: statistical, pattern-matching and cognitive (Section 4.2).

1.4 Thesis Structure

This thesis is structured in 4 chapters: Chapter 1 introduces the subject, research problem, methodology and basic concepts. Chapter 2 introduces the relevant background and previous work. Chapter 3 describes the algorithms used in the study, together with META-MELO, the system developed for the study. Chapter 4 presents the results of the study and finally, Chapter 5 present a conclusion with future work.

Chapter 2

Background and Related Works

In this section we present basic concepts in musical computational creativity and human-machine interactions. Broadly speaking, as mentioned in the introduction, two methodologies for musical computational creativity can be distinguished: corpus based and non-corpus based. The first is a process where the knowledge is learned from music composed by humans, the second is a rather large group of diverse techniques based on theoretical representations (rule-based systems, formal grammars, mathematical models, etc.).

2.1 Background to Computational Creativity

If we are to develop systems for aiding musical composition, a series of questions must be answered, but primarily: to what extent is the agent/system open to feedback from the user or external input from the environment? Even though we can disagree on what is good or bad taste, an artist/composer always develops a notion of taste as a key factor in their creative process. This aspect of creativity is reflected in the small decisions practiced through a lifetime that become 'meta' and goes almost unnoticed as a process in itself while composing.

The challenge of *internal evaluation* [1] needs to be addressed. Computational creativity is a young field and we perceive a series of misunderstandings regarding what is implied in music composition. It seems there has been an unbalanced focus on *external evaluation*. A composer rarely arrives at a finished composition without exhaustive iteration, listening and evaluation. This happens regardless of whether we are addressing composition or improvisation, as these two are inextricably linked [9], we can see them as different aspects of the same process.

In the composition process there exists ongoing active listening that is informing the decisions being made by the composer, many of which are not occurring consciously [9]. Finke describes two distinct modules involved in the creative act from a cognitive perspective [31], generation and evaluation. These have been computationally approximated in what are the generation and evaluation functions.

These two stages involve listening orientated processes that parallel the percepts that creative agents would utilize to learn from a corpus and evaluate the music generated (we can call this evaluation 'taste'). If the evaluation function is neither implemented by the system nor performed by the composer, there exists only a generation phase that is an incomplete automated creative process [1]. It is merely a con-

straint or rule based free association of musical notes generated according to statistical procedures, much like the wandering output of a Markov Chain. In this sense, the process of good machine listening may be the hardest aspect of musical computational creativity [14], this would involve following an external stream of audio as in improvisation or 'contemplating' and learning from its own output.

Improvisation follows a similar process to composition except that the frequency of the generation-evaluation loop is much higher. From a cognitive perspective an improviser generates an internal representation of immediate possible notes (generation) and selects (evaluation) which ones will fit more satisfactorily accordingly to the musical context. In this case, all the decisions are made without the opportunity for either trial and error or revisions [9].

These concepts are relevant to this discussion in that, for corpus-based music generation systems (i. e. *style imitation*), great emphasis has been made developing systems purely for generation without an *internal evaluation*. The goal to generate music is not complete without a process of iteration, editing, listening, and evaluating the diverse artifacts produced by the system or creator. According to Ariza [5], in the history of computational creativity there has been a lack of empirical evaluation, due to this task being seen as more relevant to aesthetic criticism rather than an experimental creative methodology. That is, there is a need for the experimental development of the evaluation methodology as part of the creative process.

The challenge within *style imitation* can be represented more formally by specifying how to evaluate music within this problem space. First of all, knowing when a system is generating 'in-style' is of utmost importance. Consequently, once this problem is addressed, any intentional deviations from this style in the creative process can be managed and controlled more precisely. This is addressed by Agres *et at*. [1] and referred to as a method of *internal evaluation* through *conceptual representations*. The task of completely automating a successful *internal evaluation* function is beyond the scope of this thesis, but the work is a step further in this direction.

2.1.1 Forms of Creativity

Margaret Boden [10] broadly defines types of creativity by classifying novel, positively-valued artifacts, as those that at a previous moment in time appeared to be either *improbable* or *impossible*. In Boden's definition, the 'improbabilist' artifacts are generated by a *combinatorial* process where new ones are produced from known ones. These known ideas are obtained from the conceptual creative spaces or *structured styles of thought*, a familiar set of highly structured elements. On the other hand, 'impossibilist' artifacts are those that could not have been conceived within the known conceptual space, therefore the space itself is required to be either further *explored* beyond the known boundaries or in some way *transformed*. The first type of artifacts, 'improbabilist', are the ones obtained through *style imitation*.

Determining what creative type a particular artifact belongs to is not always evident in the framework described by Boden. Instead, it is greatly determined by how the style or initial conceptual space is defined and constrained, as well as the degree of novelty of the new idea within that context. In our work, the conceptual space is implicitly constrained by the corpus used to trained the algorithms.

As a musical example we can look at Beethoven's music and the style of Classical Piano Concertos (within the broad genre of western classical music). His Piano Concerto Number 4 in G Major Op. 58

is an addition to this style, therefore it can be considered as exploratory creativity within the conceptual space of 'western classical music piano concerti'. But if we focus on the concerto *form* itself as it existed in 1805, this opus is the first where the opening bars consist of a piano solo rather than the traditional orchestral introduction of the theme. In this respect the creative idea slightly transformed the conceptual space of the *stylistic form*.

A more significant case is John Cage's 4'33", a musical work where, in an example interpretation, a performer sits quietly at the instrument and opens the lid for four minutes and thirty-three seconds. This work produced a clear transformation of what was previously considered 'music': silence as a complete musical artifact was an apparently impossible and astonishing idea that transformed the conceptual space of music itself. Thus, the definition of music and music listening was shifted to include any sounds produced by the audience and the environment, including coughing, breathing, laughing, etc.

We argue that creativity cannot be measured or evaluated outside of a context, since there is no objective ground truth or optimal solution to the task. Formalizations and methodologies are required in order to standardize and possibly automate the internal or external evaluation of computationally creative systems. The constrained musical task of *Style Imitation* is valuable as an end in itself, and our methodology is a contribution to the evaluation of this process. But within the larger context of musical computational creativity we see *style imitation* as a foundational requirement, as is the study of harmony, counterpoint, orchestration, etc. for a human composer.

2.1.2 Style Imitation

In the first chapter we defined *Style* as: "A group of rules and characteristics that uniquely and consistently describe a set of creative artifacts", and *Style Imitation* as: "The production of novel creative artifacts that would be consistently classified as belonging to a particular style by unbiased observers".

Artificial Intelligence enthusiasts have, from the very beginning, attempted to develop autonomous creative agents. As early as 1834, (in reaction to the Analytic Engine by Charles Babbage) Ada Lovelace, considered one of the founders of scientific computing, had anticipated in her notes the development of computer generated music [43].

Style Imitation can be described more precisely as creativity arising from within a pre-established conceptual space. As mentioned, this is referred to as Exploratory Creativity [11]. In practical musical terms the task is concerned with generating new and original compositions that roughly cover the same space as the corpus, thus fitting the given musical style [4, 28]. The conceptual space of a style can be defined by observing the musical features that remain invariant across the corpus if, indeed, there are any such features. Therefore an artifact can be considered pertaining to a specific style if it is classified as such by an unbiased judge or process. If invariant features are not observed within the corpus we may have a collection of artifacts without an given style.

For simplicity and clarity, we adopt the general notion of two types of creativity, whether a new work fits a previously existing style versus when it transforms or transcends the style itself. In the first case there are certain features that remain invariant across all the artifacts within that space, in the second case the creative idea is acting on the constraints and rules of the space itself. The former is what we refer to as *Style Imitation*. We take Boden's notion of style as a conceptual space, less as an abstraction and more

as a geometric object of study, as described by Gärdenfors [33] : in (Section 4.2.3) Multi-Dimensional Scaling is used for visualizing it. This technique offers a map of the topology based on similarity where the corpora and generated artifacts lie. It is practical for studying, at least informally, the effects of different generative algorithms.

Within the historical arc of Computational Creativity and specifically *Style Imitation* it is worth to mention the first example of computer generated music: the Illiac Suite, produced by the Illinois Automatic Computer in 1956, programmed by Lejaren Hiller and Leonard Isaacson. The four movements were generated through a combination of rule-based and statistical approaches. David Cope [20–22] is considered one of the pioneers in Musical Style Imitation, his Experiments in Musical Composition (Emmy or previously EMI) was a system coded in Lisp that is regarded as the first successful at generating imitations of various classical composers, including J. S. Bach, Chopin and Rachmaninoff. This system is based on recombinancy of structural, melodic and harmonic aspects of music (SPEAC) [21]. It involves a significant amount of expert knowledge and coding of rules. Generally the first step is the deconstruction of musical parts, then the detection and retention of commonality or *signatures* (the features that signify style in a composer), finally the recombinance of these into new works. In more recent work [22], the systems named *Emily Howell* is based on *Associative nets* and is not a style imitation system. Rather, it has an input corpus of works created by Emmy and composes through an interaction with Cope in the form of question and answer through a console.

2.1.3 Background on Parsing and The Problem of Melody and Harmony Dependency

Music is a multidimensional language. A monophonic melody can be broadly separated into three dimensions: pitch, rhythm and harmony. Harmony within melodies is an emergent property that can be implicit as in monophonic melodies, or explicit, for example in melodic jazz improvisations following a standard progression. The machine parsing of polyphonic or monophonic musical data, static or real-time, is an open problem, we describe some basic approaches and their relationship to our task of melodic generation.

When learning from a MIDI input stream, polyphony is parsed by either following and synchronizing to a beat in performance or by "beat tracking" techniques that can infer the beat within the music. This second option requires that the input be sliced according to several predefined rules that are not always style independent. For example Assayag *et al*. [7] propose using every note onset as a time event for obtaining a snapshot of all note-on's at that time interval. This naturally extracts a low level grain of the vertical dependencies: the harmonic and polyphonic dimensions.

Although, the complexity and amount of data generated requires complex techniques for handling, Pachet has noted that simpler approaches are possible and proposes a variation [57]: By aggregating all notes that arrive segregated by empty spaces in time ("negative time"), harmonic information can also be obtained. Nevertheless problems do arise in generation that are related to note legatos and rhythm. As pointed out by Triviño-Rodrigues, Morales-Bueno [72], and Pachet [57], parsing and learning all the attributes present in a stream of music input involves a cartesian product of the attribute domains. Therefore the growth of dimensional complexity requires special attention.

A compensation methodology is developed to cope with these issues that require subtracting and memorizing a level of complexity in parsing (inter-note delays), and later re-introducing them in generation. In this way polyphony and harmony can be parsed by detecting clusters of notes, inter-onset intervals, "negative time" and applying quantization [57].

In the case of *Virtuoso* [58], since the model is style specific, an approach that incorporates expert *bebop* improvisation knowledge is used. The databases music used are chord specific and all melodies are individual phrases transposed to C by scale and mode. Chord substitutions typical to that style are encoded from jazz theory books and a series of Markov models are used to train separate harmonic sets. The system then generates a melodic improvisation according to the present harmony using the melodic patterns learned for that case (chord and scale). There are no note durations learned, just rhythmic patterns at the beat level. In generation a series of beats is selected and pitches are added based on the present harmonic constraint. Therefore the harmonic, melodic and rhythmic information is separated in parsing and recombined in generation, thus inevitably losing musical information.

Pachet coins the expression "Markovian boredom" referring to the lack of long term coherence in Markov generation and escapes it by expert modelling of musical techniques (ex. sidestepping in jazz improvisation) and meta-level interactive control. He introduces style specific musical techniques that a bebop performer would use and encodes them in the model. In this example, Melodic continuity is also achieved by "carefully choosing the training corpus", the importance of which has also been noted by David Cope [22]. This is one way that the problem of melodic generation, which is consistent with a particular harmonic progression can be solved: by system design and corpus selection, rather than by learning from the corpus. The later implies learning the relationship between harmony and melody from complete pieces, including how to commence and end a piece. Whereas the former approach greatly simplifies the challenge, since the system designer breaks down the problem through expert knowledge. For example, a database of all the endings of Mozart sonatas can be created and learned from. Then a rule-based technique on how to transition to a learned ending can be added to the system.

Several models have been proposed for addressing this, for example by training separate instances of an algorithm with a selection of several musical attributes and recombining in generation as is the case for Multiple Viewpoint System [17], the factor oracle multi-channel approach proposed by Assayag and Dubnov [6] (Parallel Multi-attribute Algorithms), or the MPSG [72] (Multi-attribute Prediction Suffix Graph).

In the case of OMax [19] or the IPGenerate [23], a cross alphabet representation of multi-voice MIDI is used to build and generate from the input stream. The incremental parser uses Factor Oracles trained with attributes parsed at the beat level.

Other example approaches implement statistical context models [61], Incremental Parsing (IP) [39], connectionist approaches such as Recurrent Neural Networks (RNNs), or Evolutionary Algorithms [29]. In general, all are to some degree probabilistic learning models that generate musical segments or note sequences according to the context and are stationary after training [44]. Another notable framework that has influenced the advancement of music computational creativity () is the Generative Theory of Tonal Music (GTTM) by Lerdahl and Jackendoff [38, 41]. It is a foundational musical grammar theory based on music perception and cognition, with an approach to constructing a hierarchy based on music form.

This study focuses on three algorithms that are fundamentally different in their approach. All of which are corpus based algorithms: the statistical *Variable Order Markov Model* (VOMM) used by the Continuator [57], the *Factor Oracle* (FO) pattern matcher used by Omax [19], and MusiCog [49], a novel, cognitively inspired approach used for the suggestion and contextual continuation (reflexive interaction) of musical ideas in the notation-based CAC system "Manuscore" [48].

The above discussion highlights the complex problem and limitations of learning and generating melodies without an explicit representation of harmony. None of the algorithms used in this thesis contains an explicit model of tonality nor for learning harmony, therefore explicitly addressing harmony is outside the scope of this work. This problem is nevertheless relevant to our work in that the learning and generation of melodies cannot be completely isolated from the underlying harmony. Even if not explicit, as could be said of some folk songs, there is an implicit underlying harmony in melodic construction. Moreover, these harmonic progressions often significantly guide melodic development, and as Schoenberg said "a composer invents melody and harmony simultaneously".

In this regard, the corpus of folk songs is arguably the most loosely associated to an explicit underlying harmonic progression, as this harmony is relatively simpler and plays a smaller role in comparison to the other corpora. Since the algorithms used here lack an explicit model for learning harmony, we focus on this folk song corpus in our study as a way to reduce the possible adverse effects of generating melodies without this explicit model.

An in-depth description of these algorithms will be presented in the next chapter (Section 3).

2.2 Synthetic Evaluation of Style Imitation

Pearce et. al. [62] state that one of the main challenges to progress in the development of generative music systems is a lack of clear goals and empirical evaluation methodologies. They criticize the use of Turing Tests for evaluating style imitation systems due to their lack of precision; we do not know why an artifact passes or fails the test. Ariza [5] argues that the Turing Test can only be appropriately applied to language, since language discourse allows for comparisons between different statements with similar meanings. An interrogator can verify a statement independently of the language used to express it, but this is not the case with music. A music listener thus cannot be an interrogator, only a critic.

In previous work on computational evaluation, Manaris *et al*. [45] use artificial "music critics" based on power-law metrics trained on a corpus as a fitness function for an evolutionary generative algorithm (GA). Pearce and Wiggins [63] describe a framework for generating music that seeks to satisfy algorithmic 'critics' trained on a corpus. In both cases there is a computational evaluation of the similarity of generated melodies to the corpus. These examples, as well as others, generally do not consider the comparative analysis of different music generation formalisms. On the other hand, Begleiter *et al*. [8] compare the output of different VOMM algorithms in the music domain with the goal of measuring performance in terms of prediction; i.e., how closely an algorithm predicts a particular style. In this case the distinction with our work is that the task of prediction is not the same as ours in comparing algorithms that generate *novel pieces* in the same style.

The above approaches use computational systems to evaluate the output of algorithmic music programs, but they do not consider the comparative analysis of different music generation formalisms.

The evaluation provided by Pearce and Wiggins [65] is very relevant to this work; it empirically compares the output of three variants of Markov models trained on chorale melodies using musical judges. The stepwise regression also described provides directions for improving the algorithms by indicating the quantifiable musical features that are most predictive in their failure.

As mentioned, Agres *et al*. [1] categorize the evaluation methods for creative systems as *internal* or *external*, the external requires the testing of responses in the audience, whereas the internal requires the *modeling* of these responses within the system. Our approach is one of the later type, the modeling of *conceptual representations* and the measurement of similarity within these spaces. We present this methodology in Chapter 4.

2.3 Research Problem

We address the problem of evaluating the models of musical metacreation by using computational techniques to investigate not only algorithmic output in comparison to human-composed corpora, but also in terms of algorithm self-consistency. This analysis is useful both for determining which algorithm is truer to a corpus, and also for discovering more precisely how the algorithms differ. We test the hypothesis that the algorithms differ to a degree that is statistically significant, and if so, this difference has an effect that is perceptible and can be described as a musical bias.

2.3.1 Study

Meta-Melo (Chapter 3) was designed for the task of understanding the differences between the selected algorithms. One concern was how to compare the algorithms in their most 'puristic' form: by implementing them in the simplest way possible. If more sophisticated implementations with musical heuristics are used for improving musical output, we would likely obtain results that are of poor generalization power with regards to describing the inherent characteristics (and biases) of the underlying algorithms. No attempt is made to evaluate the creativity or aesthetic value of any algorithm as this is an entirely different problem.

The approach presented here empirically compares the output of methodologically distinct corpus-based music generation algorithms (statistical, pattern-matching and cognitive), without the intervention of human listeners. One advantage of this is that, by avoiding listening studies, methodologies may be developed that algorithms can incorporate for introspection in real-time generation (*internal evaluation*).

We also provide a simple and alternative technique for aiding the development of the models using decision trees, where specific features are demonstrated to be most significant in seperating the output of the algorithms from the space of the corpus. Music Information Retrieval (MIR) tools are used in a controlled setting for the purpose of understanding more completely how these methods behave in real world applications. The motivation for this work is the empirically informed advancement of intelligent-collaborative music composition environments. The focus being off-line, corpus-based style imitation algorithms.

2.3.2 Algorithm Bias

The main problem addressed is: given three corpus-based style-imitative algorithms, which characteristics or tendencies of the output are shaped by the underlying algorithms themselves and not by the corpus?

That is, the aim is to discover the *musical biases* that arise and are dependent on the formalism of the algorithms and not the corpus. To answer this, the differences of algorithm output is analyzed using computational techniques based on Music Information Retrieval: the search for the musical features that show the most variance between the corpus and algorithm output. These sets of features are the musical bias.

This bias can also be described as the "set difference" between the generated style space and the original before it was transformed or expanded. This calculation is possible by using similarity metrics to map out an abstract, and relative, creative space. In this context, the 'task' of *style imitation* is not designed to achieve identity, the complete recreation of the corpus, but rather to have the greatest overlap in this abstract conceptual space.

Chapter 3

META-MELO: Model Implementations

In this chapter we begin by presenting the three algorithms chosen for the study, Variable Markov, Factor Oracle and MusiCog. Then describing in further detail the implementation of these within the system we call META-MELO, including the music representation, parsing method and corpora used.

3.1 Markov Models

Markov models were introduced by the mathematician Andrey Markov in the beginning of the 20th century as a class of stochastic process. This body of work originated from the observation of one-dimensional time-dependant random variables and continues the work of Laplace and Bernoulli. The original subject of study was the relationship between the sequences of letters within Russian poetic texts (natural language processing), this work would eventually transition to the music field by the middle of the century.

Markov Chains, a simple Markov model, are extensively used in computational creativity. The theoretical basis lies in the field of *stochastics* that studies the description of random sequences dependent on a time parameter t. In their most basic form Markov Chains describe processes where the *Markov property* holds, that is, a process where the conditional probability of a future event X_{t+1} depends on the current state X_t alone and not on previous events, that is, a finite horizon. In this way a sequence of musical notes can be analyzed to obtain a set of probabilities that describe the transitions between states: in our case, transitions between musical events. Two well known real-time systems implementing Markov Chains are Zicarelli's "M" [76] and Pachet's "Continuator" [57]. These models allow not only the classification of sequences but also the generation of sequences with similarities to the original by traversing the model with a *random walk* weighted by the probabilities.

As described by Conklin [16], perhaps the most common from of generation from Markov models is the so-called "random walk," where an event from the distribution is sampled at each time step, given the current state of the model. After each selection the state is updated to reflect the new selection. A Markov Chain may consider a finite memory of length m, that is, the number of previous states that is considered when constructing the probability distribution of the chain, therefore the future state depends on the past m states. The length of the *memory* defines the *order* of this non-stationary model. A model may have

a *fixed* or *variable* order, we implement a Variable Order Markov Model (VOMM) with a variability of 1-4 events.

3.1.1 Definitions

There exists an independent random variable X_t associated to every discrete time *t* thus: $X(t) = s$, where the process is said to be in state *s* at time *t*. More precisely, the countable set of states is called the *state space* $S = [s_1, s_2, ...s_n]$. The Markov Chain adheres to the Markov property which states that:

$$P(s_t|s_{t-1}, s_{t-2}, ..., s_1) = P(s_t|s_{t-1}) \tag{3.1}$$

The first order chain can be represented as an *N X N* state-transition matrix (transition table):

$$P_{i,j} = P(X_{t+1} = s_j|X_t = s_i) \tag{3.2}$$

Here $P_{i,j}$ denotes the elements of the matrix at indexes *i* and *j*, the right hand side expression represents the conditional *transition probability* distribution between states. The probabilities are obtained by empirical counts of the occurrences in the observed sequences within the training set. This transition table holds the set of all possible continuations for the sub-sequence of s considered.

The *memory* of the model is the number of previous states that is considered and thus defines the order of the Markov Chain. A *zeroth* order model depicts one where the probabilities only indicate the ratio of occurrence of events without a notion of correlation between the states. A *first* order model has a memory of one previous state.

A higher-order model of order *n* holds the assumption that:

$$P(s_t|s_{t-1}, s_{t-2}, ..., s_1) = P(s_t|s_{t-1}, ..., s_{max(t-n,1)}) \tag{3.3}$$

Thus giving multiple probabilities between states, an equivalent first order chain can be created by replacing the *state space* S with $\bigcup_{i=1}^{n} S^i$, that is, the union of unique state sequences of length i.

In Variable-Order Markov Models (VOMM), the chain allows transitions of various memory lengths. When generating sequences from the VOMM, the longest subsequence for which there is a continuation is usually considered. Another approach is to aggregate the probabilities of sequence continuations of mixed length, this is known as *smoothing* [64].

We implement a Variable Order Markov Model (VOMM) with variability of 1-4 events.

3.1.2 Training and Generation

Figure 3.1 shows an example representation when parsing a melody from a Bach Chorale. The sequence of pitches are *EFDGGGE | FGEE* and the sequence of note durations is (1/8)*2 - 1/4 - (1/8)*6 - 1/4 - 1/2.

Table 3.1 shows the eleven notes of the chorale and how they are parsed by observing the Inter-Onset-Interval and pitch interval in semitones (instead of pitch class), this is the number of semitone leap leading to the next note.

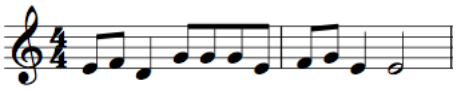

Figure 3.1: A simple melodic example, the sequence of pitches are: *EFDGGGE | FGEE* and the sequence of note durations is (1/8)*2 - 1/4 - (1/8)*6 - 1/4 - 1/2.

	IOI	MI
1	0.25	2
2	0.25	0
3	0.25	-2
4	0.25	-3
5	0.25	-2
6	0.125	2
7	0.125	2
8	0.25	1
9	0.25	2
10	0.25	-2

Table 3.1: The result of parsing the first eleven notes of the Bach Chorale. Pitches are parsed as melodic intervals (MI) in semitones and duration is the Inter-Onset Interval (IOI)

As an example for training and generation we will use the opening motif from Mozart's 40th symphony, shown in Figure 3.2. The pitches in the melody can be represented by a *zeroth* order Markov model as a vector of the ratio or probability of occurrences of the pitch classes. Thus in ratios of the 20 occurring notes (silences are omitted): *A*=1/20, *B*=1/10, *G*=1/20, *D*=7/20, *E*=1/4, *F*=1/20, *G*=1/20. Markov Models of any order can be utilized for generation by a random walk through the distributions described in the transition table, in this manner the output melody of a *zeroth* order will approximate the ratios of pitch occurrences of the melody used for training, with the order randomized.

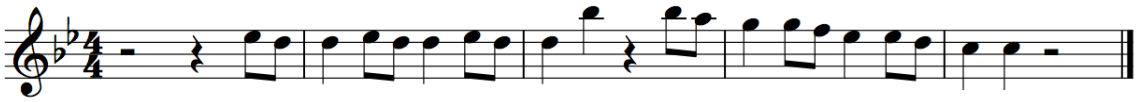

Figure 3.2: Opening motif from Mozart's 40th symphony.

In the case of a *first* order model, an example melody can be represented in a Transition Matrix, see Table 3.2, or Figure 3.3 for graph form. As the order of the Markov Chain increases more past events are considered for the construction of the transition matrix, thus increasing the memory or context of the model at each state.

In Table 3.3 we have a transition table for a first order model of the note durations of the melody.

Continuing with the above example, Table 3.4 shows the transition matrix of the melody pitches in a *higher-order* model with *memory = 2* states.

In Table 3.5 we show transitions of the Mozart theme parsed by observing combined attributes of note duration and pitch class. The *e* and *q* along the pitch class stand for eighth and quarter notes respectively.

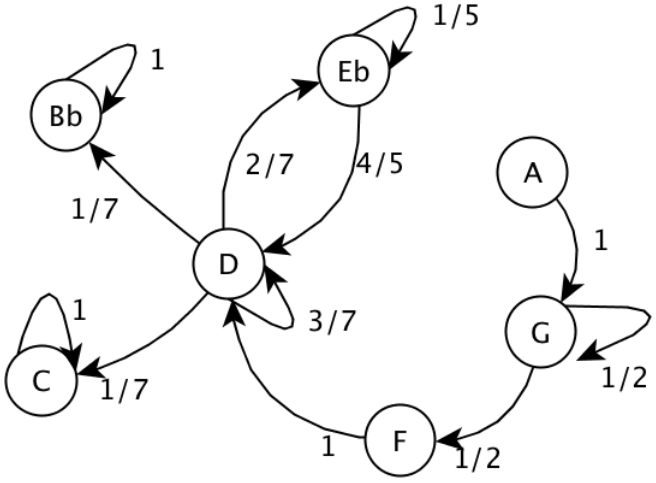

Figure 3.3: Representation of the pitches in Mozart's 40th theme in a graph form for a Markov Chain of first order. The nodes represent the seven pitch classes in the melody and the labels are the ratio probabilities of transition between the states.

	$A_t{+}1$	$B\flat_t{+}1$	$C_t{+}1$	$D_t{+}1$	$E\flat_t{+}1$	$F_t{+}1$	$G_t{+}1$
A_t	0	0	0	0	0	0	1
$B\flat_t$	0	1	0	0	0	0	0
C_t	0	0	1	0	0	0	0
D_t	0	1/7	1/7	3/7	2/7	0	0
$E\flat_t$	0	0	0	4/5	1/5	0	0
F_t	0	0	0	1	0	0	0
G_t	0	0	0	0	0	1/2	1/2

Table 3.2: A transition matrix for the Markov Chain of First Order, also shown in graph form in Figure 3.3. The left column indicates the state in the current time, the right columns represent the probabilities of occurrence for the states indicated as *t+1*. Note that all rows must add to 1.

This process can naturally be extended to *N-th order* models, in the case of our system we have implemented a Variable Order Markov Model (VOMM) with maximum order of 4. Variable Order Markov has a flexible memory length in the moment of generation, the algorithm searches for a sequence to obtain the next possible transition and uses the maximum order for which it finds a value in the transition table. This is achieved through the greedy process of first constructing a transition table for all orders (Algorithm 3.1.1), then in generation the Algorithm 3.1.2 searches for the sequence in the

	$\♪_t+_1$	$\♩_t+_1$
$\♪_t$	1/2	1/2
$\♩_t$	1/3	2/3

Table 3.3: A transition matrix for the note durations of the above melody, first order (silences are omitted).

	A	$B\flat$	C	D	$E\flat$	F	G
$E\flat.D$	0	0	1/4	3/4	0	0	0
$D.D$	0	0	0	0	3/4	0	1/4
$D.E$	0	0	0	1	0	0	0
$D.B\flat$	0	1	0	0	0	0	0
$B\flat.B\flat$	1	0	0	0	0	0	0
$B\flat.A$	0	0	0	0	0	0	1
$A.G$	0	0	0	0	0	0	1
$G.G$	0	0	0	0	0	1	0
$G.F$	0	0	0	0	1	0	0
$F.E\flat$	0	0	0	0	1	0	0
$E\flat.E\flat$	0	0	0	1	0	0	0
$D.C$	0	0	1	0	0	0	0

Table 3.4: A second order transition matrix for the pitches in the example melody. The left column indicates the sequences of pitches that occur consecutively. First row: when we have an *E* followed by a *D* there is a 25% chance of a *C* and 75% for a *D* occurring next.

highest order. If it does not find an occurrence of that sequence it reduces the memory by -1 (Function *ReduceOrder*) and searches for the sequence of one order lower in the transition table.

Algorithm 3.1.1: UPDATEVOMM(TT, Seq, m)

comment: TT : transition table. Seq : input sequence. m : order.

$L \leftarrow$ LENGTH(Seq)
for $i \leftarrow 1$ **to** $L - 1$
do {
  $j \leftarrow i$
  while $j < i + m$ **and** $j < L$
  do {
    $Ind \leftarrow$ MEMORY(Seq, i, j)
    if EXISTS(TT, Ind)
    then { TT($Ind, j + 1$) $\leftarrow$ TT($Ind, j + 1$) $+ 1$
    else { NEWTRANSITION(TT, Ind)
             { TT($Ind, j + 1$) $\leftarrow 1$
return (TT)

In detail: *UpdateVOMM* 3.1.1 takes the sequence *Seq* as input and the order *m* of the desired model and returns the transition table *TT* for that sequence and order. The transition table is a table of size *R* x

	A_e	$B\flat_e$	$B\flat_q$	C_q	D_e	D_q	$E\flat_e$	$E\flat_q$	F_e	G_e	G_q
$E\flat_e$	0	0	0	0	1	0	0	0	0	0	0
$E\flat_q$	0	0	0	0	0	0	1	0	0	0	0
D_e	0	0	0	1/4	0	3/4	0	0	0	0	0
D_q	0	0	1/3	0	0	0	2/3	0	0	0	0
$B\flat_e$	1	0	0	0	0	0	0	0	0	0	0
$B\flat_q$	0	1	0	0	0	0	0	0	0	0	0
A_e	0	0	0	0	0	0	0	0	0	0	1
G_e	0	0	0	0	0	0	0	0	1	0	0
G_q	0	0	0	0	0	0	0	0	0	1	0
F_e	0	0	0	0	0	0	1	0	0	0	0
C_q	0	0	0	1	0	0	0	0	0	0	0

Table 3.5: A first order transition matrix for the combined pitches and note durations from the example melody.

$|A|$ where the $|A|$ is the size of the alphabet and rows *R* is the number of all possible ordered subsets of length up to *m* (order) elements from the alphabet A.

Algorithm 3.1.2: VOMMGENERATE(TT, N, S, M)

comment: N : length of sequence to generate. S : starting state. M : max order to use.

$Seq \leftarrow 0$
$Row \leftarrow 0$
$i \leftarrow 1$
while $i < N$ **and** Row **not** $Null$
do
- **comment:** Keep a sequence of M length as memory.
- $Mem \leftarrow$ BUFFER(M, Seq, S)
- $Ord \leftarrow Mem$
- **comment:** Search for the Row corresponding to Mem.
- $Row \leftarrow$ SEARCHTT(Mem)
- **while** $Row == Null$ **and** LENGTH(Ord)
  - **do**
    - REDUCEORDER(Ord)
    - $Row \leftarrow$ SEARCHTT(Mem)
- **if** Row **not** $Null$
  - **then**
    - **comment:** Obtain the element with greater probability from the found Row,
    - $E \leftarrow$ CALCULATENEXT(Row)
    - ADDE(Seq, E)
    - $Mem \leftarrow$ BUFFER(M, Seq, E)
- $i \leftarrow i + 1$

return (Seq)

The algorithm uses a sliding window of length *m* and increments a count for the occurrence of that sequence in the transition table. Function *Buffer(M, Seq, S)* returns a concatenated string of characters from array Seq of maximum length M.

As a simple implementation the table can have numbered columns and numbered rows as index, or use a matrix. The alphabet can be mapped in real time: as a new symbol arrives, a new index is created with that symbol.

Here we can see that for the use of VOMM to be *tractable* either the alphabet size or the order must be low, otherwise both the complexity in time and space will grow exponentially.

Based on low *N* number of previous events Markov Chains generate strings that do not recognizably resemble the strings input from the corpus, although at very high orders, the model will replicate exactly the sequence of attributes learned from the corpus. As we describe below, this characteristic similarly resembles a property that can be achieved by generating from the factor oracle, therefore an interesting correlation and comparison can be made between the order of the Markov Model and the probability of congruence of the FO generation. The generation is seeded with the first note of the corpus used in training, in order to parallel the generation mechanism used in the Factor Oracle.

Our implementation has parameters to the algorithm that enabled generation from a Variable Order (max 4), fixed order Markov (with options 1-4), and also a *nudge factor* that steers the order with a weighted probability. This latter parameter is a value from 1...100 that restricts the use of higher orders, for example a value of 100 would enable the use of order 4 whenever possible, whereas a value of 50 only allows a 50% chance of using higher orders (when available).

Figure 3.4 shows an example melody generated by the VOMM algorithm using an alphabet of pitch intervals and note duration events combined. We added a parameters to the algorithm that enabled generation from a fixed order Markov (1-4), a Variable Order (max 4) and also a *nudge factor* that constrained the order with a weighted probability.

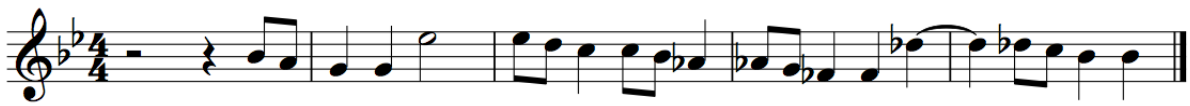

Figure 3.4: Generation from VOMM using relative pitches (intervals) and combined attributes. 75% chance of maximum possible order.

3.2 Factor Oracle

One of the most fundamental tasks in pattern matching is finding the occurrences of one string within a larger string or word. Amongst the techniques for addressing this are factor-based matching. A *factor* is a substring of a word and is at times called a *sub-word*. Efforts are focused on the efficient construction of a structure that represents all the sub-words or *factors* of a word *P*, at times allowing the indication of locations within *P* where this substring occurs.

Since music can be represented in a symbolic and sequential manner, pattern-matching techniques can be useful for the learning and generation of pattern redundancy and variations, respectively. The

Factor Oracle [2] is one example of a text and/or biological sequence search algorithm that has been applied to music. It is an acyclic automaton with a linear growth in number of transitions with regards to the input pattern, which has been utilized in string searches [6]. There exists a construction mechanism [2] allowing the alphabet to be grown sequentially and the complete structure incremented online, thus allowing for the search of string factors in real-time.

Variations of these representation structures used are *suffix tries*, *suffix automaton* (Directed Acyclic Word-Graph, or DAWG), and *Factor Oracles* [2]. As an example we use a subsegment (Figure 3.5) from the same melody shown in Figure 3.2, particularly the four note transition from bar two to bar three. A diagram of each model built on the word *E♭.D.D.B♭* is presented in Figure 3.6.

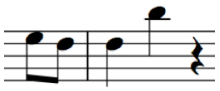

Figure 3.5: Musical motif example *E♭.D.D.B♭*, (silence omitted).

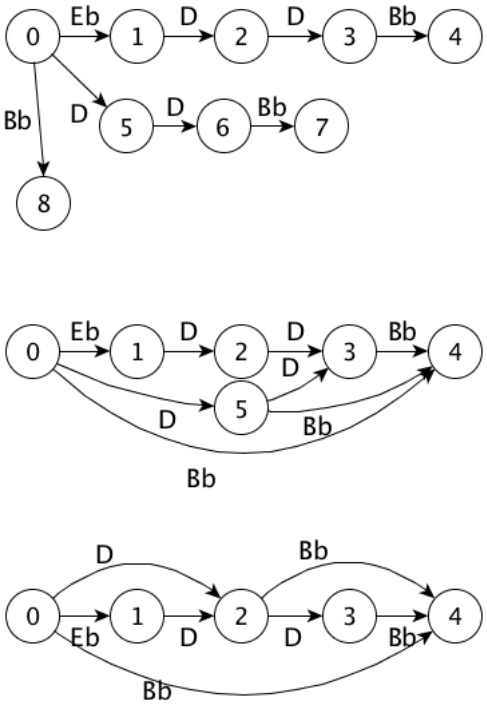

Figure 3.6: *Top:* Suffix Trie, *Center:* Suffix Automaton (DAWG), *Bottom:* Factor Oracle. For the word *E♭.D.D.B♭*.

3.2.1 Formal Definitions: Alphabet, Strings and Languages

Let the alphabet A be a finite set of symbols p_i, we define a *word* as a string $w = [p_1, p_2, ...p_m]$ with length $|w| = m$ of elements from the alphabet A. The empty word is denoted by ϵ, and the language L as a subset of words $w \in A^*$, the set of all finite strings over A.

Prefix, *Suffix* and *Factor* strings: We denote the word *w* composed as the concatenation of the words *u*, *s* and *v* where: $w = usv$ thus we define *u* as a *preffix*, *s* as a *factor* and *v* as a *suffix* of the word *w*. When considering the factor *s*, both *u* and *v* are possibly the empty word ϵ, with all *u*, *s* and $v \in A^*$.

3.2.2 Automaton

We define an *automaton M* on the alphabet A as a 5-tuple (Q, q_0, T, A, F) where Q is a finite set of states, one initial state denoted by q_0, there exists a set $T \subseteq Q$ of *terminal* or final states, and the set $F \subseteq Q \times A \times Q$ of *transitions* between these states. The arcs are individually denoted by (q_s, a, q_t), with q_s being the *source* state, *a* the *label* and q_t the target state.

3.2.3 Links

The Factor Oracle is constructed with two types of links, the *suffix* link and the *forward* link. Suffix links point backwards and connect a state q_t to q_k with $t > k$, they lack labels and can be denoted as $sfx[t] = k$. The suffix link aids in the the discovery of the longest repeated suffix at each state q_t, the length of the longest repeated suffix at q_t is calculated and denoted by the algorithm $lrs[t]$. Forward links are labeled and used to obtain the factors from the input string. Every state contains a forward link pointing to the consecutive next state, i. e. from state $q_t - 1$ to q_t and labeled by the symbol a_t, this is denoted as $F(t-1, a_t) = t$. The other type of forward link jumps forward from state q_t to q_{t+k} and is labeled by a_{t+k} with $k > 1$.

3.2.4 Construction

An *on-line construction algorithm* is an algorithm type classification based on efficiency, measuring time in computations taken for the construction of the automaton. As opposed to off-line algorithms, an on-line algorithm arrives at an intermediary result in time *T(n)*, that is, a finished automaton is produced at every state of the string for the input number of elements *n* received. In the case of an off-line algorithm, the complete input string must be read first for the automaton to be constructed, thus the advantage of on-line processing is that the input can be treated as an infinite stream and processed in real-time.

Assayag et. al. [6] have described that, when traversing a Factor Oracle with any given procedure, the lesser the ratio of suffix links chosen in generation with regards to forward links, the greater the correlation between the output string and the input string. This ratio can be construed as the *probability of congruence* [23] between the input and the output strings. That is, while generating we can control the distance from the input sequence and produce less direct 'quotes' from the training material. We do this by following more than one suffix link before a direct forward jump, or by decrementing the number of successive forward link steps, thus the generated output string is less similar to the learned input.

Since the Factor Oracle was not created for generation, a method for navigating the structure is required. The simplest approach suggested by Dubnov [23] is to use a parameter to determine the degree of replication vs. recombination. Since the entire sequence is encoded in the structure, if one navigates and generates from it from left to right in a step-wise fashion, the exact input sequence is reproduced. To the degree that we take alternative routes, variation sequences are created.

Cont [18] suggests that there could be benefit in using an intelligent approach for navigation of the Factor Oracle consisting of intentionally selecting the combination of factor and suffix links with a weigthed random parameter. We have expanded this simple algorithm by including extra parameters that add some intelligence in the selection of links: for example, by avoiding or escaping certain loops found within the structure. It is important to note that neither the Markov Model (MM) nor the Factor Oracle (FO) will generate a transition that is not in the corpus.

Algorithm 3.2.1: NEWLETTER($Oracle(p_1, p_2...p_m), \sigma$)

CREATENEWSTATE($m+1$)
CREATENEWTRANSITION($m, m+1, \sigma$)
$k \leftarrow S_p(m)$
while $k > -1$ **and** TRANSITION(k, σ) $== Null$
  do $\begin{cases} \text{CREATENEWTRANSITION}(k, m+1, \sigma) \\ k \leftarrow S_p(k) \end{cases}$
if $k = -1$
  then $\{ s \leftarrow 0$
  else $s \leftarrow$ TRANSITION(k, σ)
$S_p\epsilon(m+1) \leftarrow s$
return ($Oracle(p_1, p_2...p_m\sigma)$

Algorithm description:

Input: Oracle for string $p_1, p_2...p_m$ plus new symbol σ.

1. Create a new state numbered $m + 1$.
2. Assign a new transition labelled σ from state m to the new state $m + 1$.
3. Define the index k as the position of the node reached by the suffix link of m given by the suffix function S_p, return -1 if no suffix exists.
4. Iteratively backtrack suffix links from state m using k. While k exists and there exist no transitions from state k by σ, then do: Create new transition from state k to $m + 1$ by σ.
5. If no suffix exists then create a suffix link from the new state $m + 1$ to state 0, else: create a suffix link from the new state $m + 1$ to the state reached by σ form state k.

The transition matrix for the Mozart theme is represented in Figure 3.6. The Factor Oracle suffix vector for the complete Mozart theme example melody is displayed below, each position of the array contains the number of the state to where the suffix link points to. That is, first state contains no link, second state countains a link to the first state, etc.:

	$E\flat$	D	$B\flat$	A	G	F	C
1	2	3	11	13	14	16	20
2	18	3	0	0	0	0	0
3	5	4	11	0	0	0	20
4	5	0	11	0	0	0	0
5	0	6	0	0	0	0	0
6	0	7	0	0	0	0	0
7	8	11	0	0	0	0	0
8	0	9	0	0	0	0	0
9	0	10	0	0	0	0	0
10	0	0	11	0	0	0	0
11	0	0	12	13	0	0	0
12	0	0	0	13	0	0	0
13	0	0	0	0	14	0	0
14	0	0	0	0	15	16	0
15	0	0	0	0	0	16	0
16	17	0	0	0	0	0	0
17	18	0	0	0	0	0	0
18	0	19	0	0	0	0	0
19	0	0	0	0	0	0	20
20	0	0	0	0	0	0	21

Table 3.6: The transition matrix for the example factor oracle built on the Mozart excerpt.

Suffix vector : (0, 0, 2, 1, 2, 3, 4, 6, 7, 1, 11, 1, 1, 14, 1, 2, 2, 3, 0, 20)

Figures 3.7 to Figure 3.11 show the first five steps in the construction of the FO for the Mozart theme. Figure 3.12 shows the complete structure. For simplicity in the diagrams, forward links are in brown, suffix links are in red. Note that thedirection of the links is not relevant in generation.

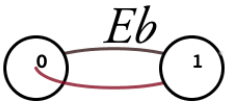

Figure 3.7: Step one: Input (E♭). Add a new transition from state 0 to 1 by (E♭) and suffix link from state 1 to 0.

Figure 3.8: Step two: Input (D). Add new transitions from state 0 and 1 to state 2 by (D) and suffix link to 0.

3.2.5 Generation

We developed an algorithm for generating from the FO based on the basic replication vs. recombination approach that consists in using a parameter for steering the degree of consecutive single-step forward

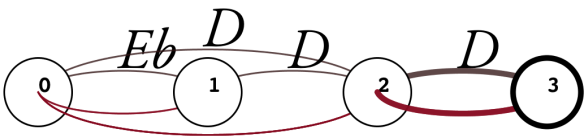

Figure 3.9: Step three: Input (D). Add a new transition from state 2 to state 3 by (D) and suffix link to 2.

Figure 3.10: Step four: Input (E♭). Add new transitions from state 2 and 3 to state 4 by (D) and (E♭) respectively, suffix link to 1.

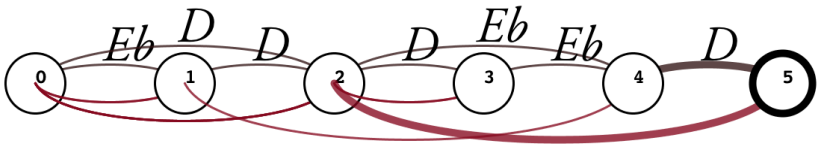

Figure 3.11: Step five: New transitions are highlighted.

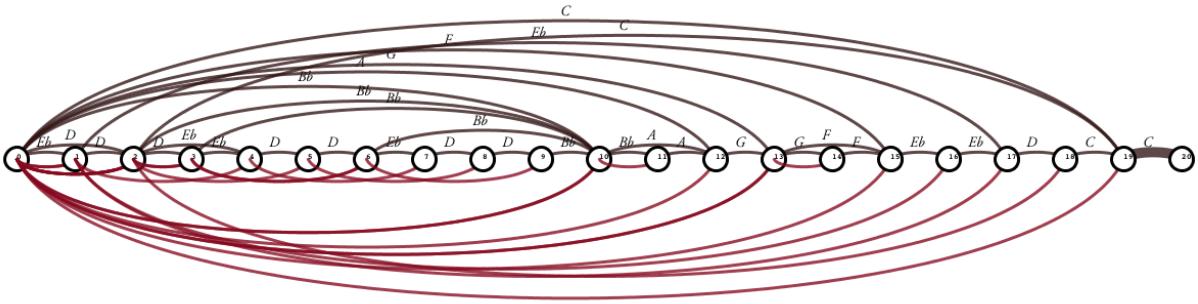

Figure 3.12: Completed FO Structure.

links used versus jumps (through forward or suffix links). The algorithm is displayed completely below in MATLAB as this was the environment we developed and tested before porting to MAX. It consists of three parameters, p_1, p_2 and p_3.

- p_1: probability of change (replication vs. recombination). The chance of using a directly forward link vs jumps through forward of suffix links.
- p_2: probability of Forward vs Suffix recombination.
- p_3: probability of long jump recombination. If there are several options for jumping, use steer the length of the jump.

Throughout the study we set p_1 to 75% and both p_2 and p_3 to 50%

```
function [s,kend,ktrace] = FO_gen(trn,sfx,n,p1,p2,p3,k)
% Generate new sequence using a Factor Oracle
% input:
% trn - transition table
% sfx - suffix vector
% n - length of the string to be generated
% p1 - probability of change (replication vs. recombination)
% p2 - probability of Fwd vs Sfx recombination. non-zero
% p3 - probability of long jump recombination. non-zero
% k - starting point
% output:
% s - new sequence
% kend - end point (index of the original sequence)

a = size(trn,2); %size of the alphabet
ktrace = 1;

special = 1 & 0;
k = 1;
for i = 1:n,

    PStep = (rand < p1);
    PFwd = (rand < p2);
    N = size(trn,1);
    Lsfx = (length(sfx{k})==1);
    sfx1 = sfx{k}(1);
    Lfwd = (length(find(trn(k,:)))==1);

    if ¬PStep && k == N,
        special = 1 & 1;
    end

    if (PStep) || (Lsfx && Lfwd && sfx1 < 2) || (Lsfx && sfx1 > 1 && ¬PFwd),
        if (k == N) || (¬PStep && sfx1 > 1),
            k = sfx{k}(1); % back to initial state
        end
        %copy next symbol
        s(i) = find(trn(k,:)==k+1);
        k = k+1;
        ktrace = [ktrace k];
    elseif (¬Lfwd && PFwd) || (¬Lfwd && ¬PFwd && Lsfx),
            if special, k = sfx{k}(1);
            end
            %copy forward any symbol
            I = find(trn(k,:));
            O = (sort(trn(k,I)));
            M = ceil(p3*(length(O)-1))+1;
            sym = find(trn(k,:)==O(M));
            s(i) = sym;
            k = trn(k,sym);
```

```
51              ktrace = [ktrace k];
52          else
53              if sfx1 > 1,
54                  k = sfx{k}(ceil(p3*length(sfx{k})));
55              elseif ¬Lsfx %for debugging
56                  k = sfx{k}(ceil(p3*(length(sfx{k})-1))+1);
57                  else
58                      return;
59                  end
60              end
61              %use sfx and copy any of the next symbols
62              ktrace = [ktrace k];
63              I = find(trn(k,:));
64              O = (sort(trn(k,I)));
65              M = ceil(p3*(length(O)));
66              sym = find(trn(k,:)==O(M));
67              s(i) = sym;
68              k = trn(k,sym);
69              ktrace = [ktrace k];
70          end
71      end
72  end
73
74  kend = k;
```

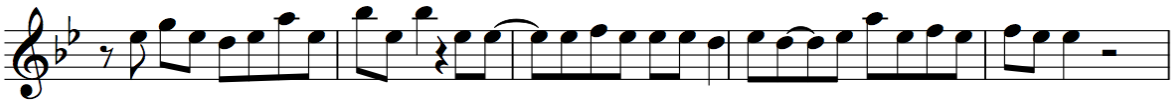

Figure 3.13: Generation from FO using absolute pitches (not intervals) and IOI, with combined attributes. 85% probability of replication (p_1), p_2 and p_2 = 50.

3.3 MusiCog

MusiCog (MC), developed at the Metacreation Lab by Dr. J. B. Maxwell *et al*. [49], is an approach based on modelling perceptual and cognitive processes, with a special focus on the formation of structural representations of music in memory. The architecture is designed for music generation through the learning of hierarchical structures in melodies, with an emphasis on the interaction between short-term and long-term memory during listening and composition. As a cognitive model, with a complex hierarchical memory structure, there are many possible ways to generate output from MC. For this study, in order to reflect a similar systematic approach to the FO and MM, and to avoid music theoretical or compositional heuristics, we selected a relatively simple stochastic approach, which attempts to balance the application of both top-down (i.e., structural) and bottom-up (i.e., event transition) information.

MC is a feedback system in the sense that it parses its own output to learn during generation. An option to disable this behaviour has been added and applied to half of the generation examples used

for all tests in the current study. This was done in order to bring MC closer in functionality to the MM and FO for a 'puristic' comparison of the algorithms, without entirely negating important aspects of its design. We will describe some of the fundamental aspects of the algorithm, for a more in-depth description see [49].

3.3.1 Modules

MusiCOG consists of four modules, the Perception Module (PE) that handles the parsing and segmentation of input, the Working Memory (WM) which is basically a buffer for temporary storage and responsible for the process of chunking, the Long-Term Memory or Cueing Model (CM), which constructs the hierarchical graph from the input provided by the perception module, and the Production Module (PM) that is in charge with music generation from the CM and WM.

3.3.2 Musical representation

The hierarchy in the CM consists of three degrees, the first one contains pitch contour (+ -), the second contains interval size and the third contains the actual MIDI note. Time is represented as the note Inter-Onset Interval (IOI) in milliseconds for the higher level and a novel representation *Beat Entry-Delay* (BED) for the lower level. This aspect of MusiCOG presents a discrepancy between the representation used by the other models, since in the other cases the the rhythmic dimension is represented with IOI alone. The *Beat Entry-Delay* was developed by Dr. Maxwell and is exclusive to MusiCOG, the main difference with IOI is that there is an aspect of learning rhythm at a slightly more structural level, since IOI is beat agnostic, whereas BED considers the position of the beat at which the event occurs. We can assume that this will give MusiCOG a rhythmic advantage.

3.3.3 Working Memory

One of the main functions of the WM is the process of chunking, this process is defined in music perception and memory and consists of the function of grouping salient aspects of the acoustic environment into motifs and phrases. The feature that drives this function is *parallelism* [13], i. e. the degree of cognitive salience that an element shares with other content in Working Memory, this emerges from musical self-similarity within motivic ideas. Therefore, when Parallelism is detected there is a form of compression:

elements form chunks but occupy the same memory space as a single concept. In this way parallelism maintains cognitive salience by preventing the decay of similar elements from memory.

Algorithm 3.3.1: PARALLELCHUNKS($C(v_j^L, S)$)

comment: Algorithm credit: Dr. J. B. Maxwell

$i \leftarrow$ Index of $C(v_j^L)$
$n \leftarrow |S| - 1$
$B \leftarrow$ **false**
$C' \leftarrow (c_0^{L+1})$
comment: C' is an empty, higher-level Chunk

for $k \leftarrow n$ **to** $i + 1$
do
- $\epsilon \leftarrow S_k$
- **if** ISCHUNK(ϵ) **and** $L =$ GETLEVEL(ϵ)
  - **then**
    - **if** $B =$ **true**
      - **then** ADDELEMENT(C', ϵ)
    - **if** $\epsilon^{\eta^0}\theta = C(v_k^L)^{\eta^0}\theta$ **and** $k - j > 1$ **and** $B =$ **false**
      - **then**
        - ADDELEMENT(C', ϵ)
        - $B \leftarrow$ **true**
  - **else if** $B =$ **true**
    - **then break**
- $n \leftarrow n - 1$

return (C')

The algorithm (3.3.1) reflects the details of the chunking process in MusiCOG. By taking a chunk and a stream from WM as input, it traverses the WM beginning at the element with the last time stamp and first searches for parallel chunks. The main IF statement ensures that the level we are comparing is the same and that we are looking at chunks and not events. Sequentially the elements with contours (lowest degree) matching the input chunk are added to the new higher-level chunk.

3.3.4 Cueing Model

The main function of the CM is learning the input and building of the hierarchical structure of nodes, the Long Term Memory (LTM). This process involves the addition of nodes to specific trees of a level in a way such that a particular pattern in a sequence found in WM is permanently encoded in the CM. The nodes added at the highest degree possible starting at the contour tier (+ -) called *Schema*, then the *Invariance* tier which records interval distance and finally at the actual MIDI notes at the *Identity* tier.

The algorithm (3.3.2) shows this process. Initialization begins with a single root node 0 at level one, there is an iteration through the entire input segment doing the following. First, *SearchTransition* attempts to match a transition at the highest degree possible and returns the node with a match or null.

The function *ExtendLevel* creates a node and a transition to this node with level k and degree 0. A new node is added by *AddNode* at degree i and depth k. *UpdateRecognition* computes the node recognition rate, which is a function of its degree and represents to what extent the node has been recognized: degree 0 = .3, degree 1 = .8, degree 2 = 1. Finally, *SetTerminalNode* determines the last node of the segment. The algorithm returns the new state of the model after processing the segment and modifying the structure accordingly.

Figure 3.14 presents a diagram of the structure built by the CM after processing the first two phrases of Mozart's theme. All three tiers; Schema, Invariance and Identity are shown.

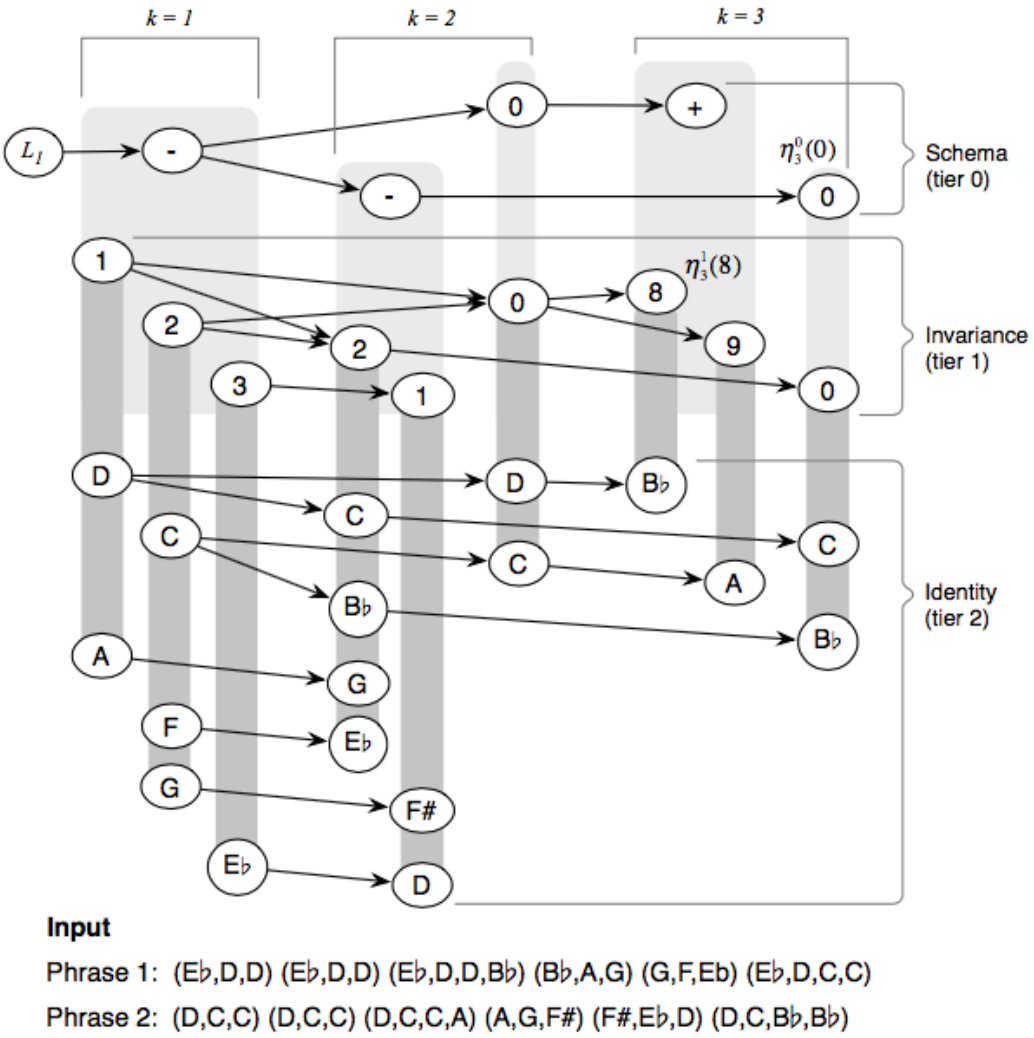

Figure 3.14: CM L_1 trained on the first two phrases of Mozart's 40th Symphony. The three tiers; Schema, Invariance and Identity are shown.

Credit: *Dr. J. B. Maxwell*

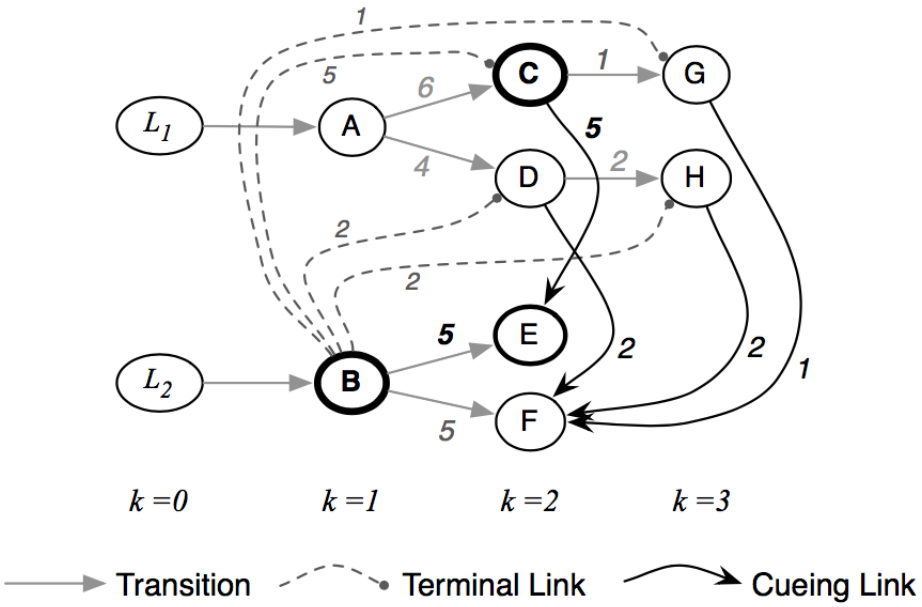

Figure 3.15: A detailed Invariance tier view of a hypothetical CM for pitch information showing weights for transitions, terminal links, and cueing links.

Credit: *Dr. J. B. Maxwell*

Algorithm 3.3.2: PROCESSEVENTSEGMENT((v_k^1))

---

comment: Algorithm credit: Dr. J. B. Maxwell

$i \leftarrow 0$
$\delta \leftarrow L_1 \eta_0^0$
for $k \leftarrow 1$ **to** $|(v_k^1)| - 1$
do {
  $\delta' \leftarrow$ SEARCHTRANSITION($\tau(\delta, \epsilon_k)$)
  if $\delta' = \perp$
    then $\delta' \leftarrow$ EXTENDLEVEL(δ, ϵ_k)
    else {
      if $i < 2$
        then {
          $i \leftarrow i + 1$
          $\delta' \leftarrow$ ADDNODE(δ, i, ϵ_k)
        else {
          $\delta' \leftarrow \delta$
          UPDATERECOGNITION((v_k^1))
          SETTERMINALNODE($(v_k^1), \delta'$)
return (δ')

3.3.5 Production Module

To summarize the learning of melodies from MusiCOG involves the following steps:

1. The inputting of events into the Production Module
2. Chunking of the events in Working Memory
3. Learning and inference of Chunks in Long Term Memory

As the LTM grows it creates higher levels in a bottom-up approach, generation is done by traversing or 'unwinding' the structure backwards (top-down). We can first infer a high level position in the structure by 'listening' to a current segment in WM (in the case of producing continuations), or randomly select a high-level-node to begin with.

Figure 3.15 shows a hypothetical CM with two Levels. Each L_i node links to the end of a segment at L_{i-1}. 'Unwinding' involves crossing from level L_i downwards to the leaf of L_{i-1} and traversing up to the root, the sequence extracted in this path is an output segment. We continue from the higher level node towards a leaf, at each step unwinding the lower level segment as described above. When arriving at a leaf in the high-level node we either output the phrase or jump to a higher level if available, by making a stochastic selection from the leaf's cueing links to L_{i+1}.

Steps for Generation

1. Stochastic selection of a high-level-node L_i (Ex. L_2 node, $i = 2$).
2. From the set of terminal links arriving to that node from L_{i-1}, make a stochastic selection and step one level down.
3. From the chosen L_{i-1} node, extract a segment by traversing the branch up to the root and output the resulting L_{i-1} segment produced.
4. Continue from the initial L_i node: if not a leaf, make a stochastic selection of a transition from the set of children towards a leaf in the L_i node sequence.
5. Repeat step 4.
6. If L_i is a leaf, make a stochastic selection from the cueing links to L_{i+1} if available. Otherwise output the segment.

3.3.6 Music Representation and MIDI Parsing

The system uses a simple but flexible representation for learning melodic (monophonic) music inspired by the multi-viewpoint approach proposed by Conklin and Witten [17]. The symbols used for the Markov model and Factor Oracle systems are derived from the concatenation of melodic interval and onset interval information (i. e. pitch and rhythm). A viewpoint is an abstraction that is derived from the musical surface. The interval viewpoint is derived from the sequence of MIDI note numbers (MNN's), likewise Inter-Onset Intervals (IOI's) are derived from note onset events. Several attributes can be used to train the algorithms, that bring immediate challenges for the evaluation and comparison methodology. We use a multiple-viewpoint consisting of interval size and direction, with inter-onset-intervals, resulting in a

cartesian product as a balance between pitch and rhythmic information. Velocity information is implemented but not used in the study as it corresponds to performance/interpretation, aspects that we do not address in this work. As mentioned in Section 3.3, there is a discrepancy between MusiCOG and both VOMM and FO. Since MusiCOG's lower level rhythmic representation is *Beat Entry Delay* versus IOI in the other two models. This is arguably an advantage of the model, in future work we propose to add the BED representation to META-MELO in order to optionally use it for all models.

For the Markov and Factor Oracle implementations, the set of attributes is indexed and a dictionary is built for the set of symbols corresponding to that attribute. This way, when an event is observed that has not appeared before, a new entry is created in the dictionary. An important function in this initial stage is the *quantization* of temporal values to a reasonable minimum resolution of sixteenth notes. This allows the parsing function to: (1) group and classify similar events as the same element in the dictionary and (2) avoid creating entries with negligible differences since notation uses quantized representations.

We parse the melody by individual note events rather than grouping by beats or bars for the purpose of obtaining an event-level granularity. Therefore there is no preservation of beat or measure information. For example, if there are two eighth-notes in a beat, we create an event for each note (note level) rather than one event with two notes (beat level). This will make evident certain biases that may otherwise be masked by parsing musical segments, and this encoding a higher level structure.

Another important point is that since the corpus is a mix of tonalities and modes, the parsing in pitch classes or pitch intervals will have diverse issues, for example the switching or wandering of tonalities/-modes.

Since MusiCOG is a cognitive model [49], it handles the parsing of MIDI input using principles of music perception and cognition that are not included in the other two algorithms, therefore not requiring some of the preliminary parsing described above.

3.3.7 Corpora

The corpora consist of monophonic MIDI files from Classical, Jazz, Popular, and Folk songs. These classes were selected for the purpose of investigating the algorithm behaviour in contrasting musical settings. We use a Finnish folk song collection that is available for research purposes [27], and manually created the other corpora by extracting melodies from MIDI files freely available on the web.

A subset of 100 pieces of 16 bars in length was selected from the folk songs, totaling ~ 6400 notes. The other corpora are around 30 pieces each, adding up to around 7000 notes per corpus.

An example melody (BACH Chorale BWV 385) is presented in Figure 3.19, the midi file is presented in Figure 3.20 as a contour diagram with time resolution in beats. The results from parsing up to the first $fermata$ (bar 3) are presented in Table 3.1, note that the bar is equivalent to 1, therefore a $\text{♩} = 0.25$ and an $\text{♪} = 0.125$. In this case the pitches are parsed as melodic intervals (MI) in semitones and time is the Inter-Onset Interval (IOI).

The parser separates tracks and loads them as individual lines as for example in the case of Bach Chorales, each voice being loaded as a single independent melody. We have given preference to non-performed MIDI files since these include different forms of timing variations. In the case of popular and

	IOI	MI
1	0.25	2
2	0.25	0
3	0.25	-2
4	0.25	-3
5	0.25	-2
6	0.125	2
7	0.125	2
8	0.25	1
9	0.25	2
10	0.25	-2

Table 3.7: The result of parsing the first eleven notes of the Bach Chorale. Pitches are parsed as melodic intervals (MI) in semitones and time is the Inter-Onset Interval (IOI)

jazz music this has not always been possible, therefore we have also included a quantizer in the parser that will reduce the grain of the variables to specific discrete values (sixteenth note as the shortest IOI).

A subset of 100 pieces of 16 bars in length was selected from this corpus, totaling ~ 6400 notes. It consists of 94 combined pitch-time interval types, therefore a total of ~ 70 samples of transitions, assuming a uniform distribution. Figures 3.16- 3.18 show the pitch class distribution of the different corpora with melodies transposed to C.

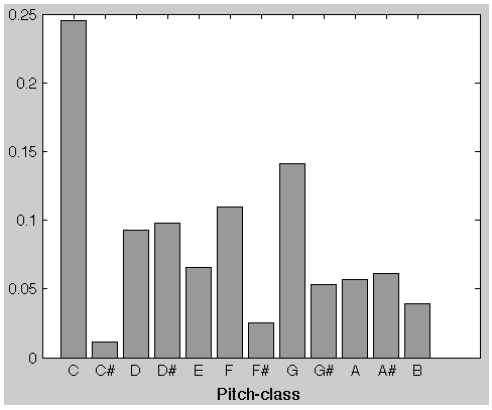

Figure 3.16: Pitch Class Distribution of the jazz corpus, transposed to C/c.

3.4 Implementation of the Models

The system, implemented in MAX and developed by the author, is available for download together with the training corpus [34]. MusiCOG is integrated to the system as an external developed by Dr. Maxwell. Figure 3.21 shows the interface of the system that enables the user to customize the training and generation of the different algorithms. The parsing panel sets the corpus (Section 3.3.7) that will be used, accepting individual files or folders containing large melodic data-sets. Also, the mode at which each corpus will be parsed is set, *relative* or *absolute* pitch, and rhythmic parsing mode. The details of

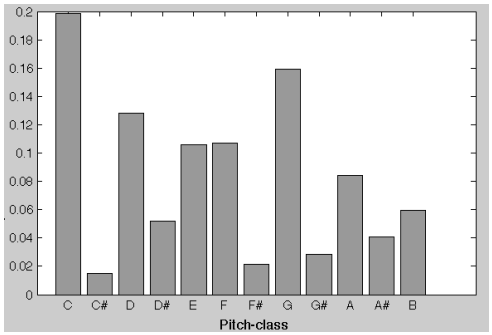

Figure 3.17: Pitch Class Distribution of the classical corpus, transposed to C/c.

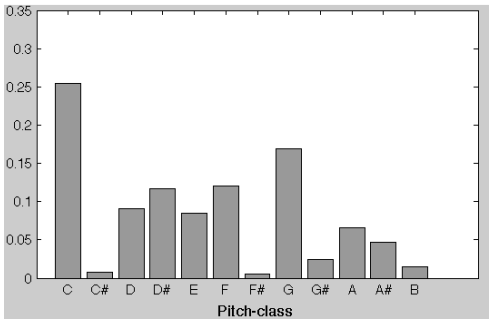

Figure 3.18: Pitch Class Distribution of the pop corpus, transposed to C/c.

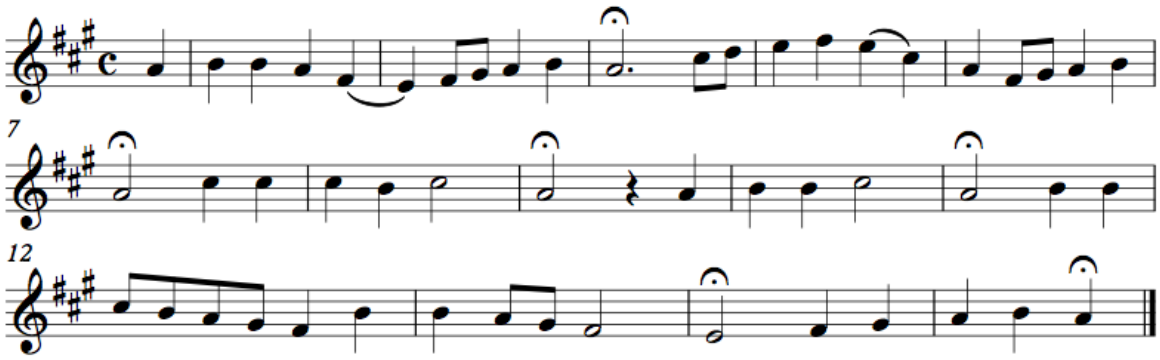

Figure 3.19: Bach Chorale: Nun bitten wir den Heiligen Geist BWV 385.

this process are described in Section 3.3.6. The generation panel sets the algorithm to be used for each attribute as also the mode of attribute parsing to use, *separate* or *combined*, for details see Section 3.3.6. This design enables testing of melodic generation with a diverse range of configurations. We are able to, for example, generate a melody based on the rhythm of a specific jazz piece with the pitch information from a corpus of Bach chorales. In another example we can have the rhythmic output generated from the Factor Oracle and the pitches generated from the VOMM (Section 3.1), or vice versa. None of the algorithms in the system check for corpus plagiarism, that is, there is no control for avoiding the output of sequences that exactly replicate the corpus.

Figure 3.22 shows the basic modules of the systems and their interaction.

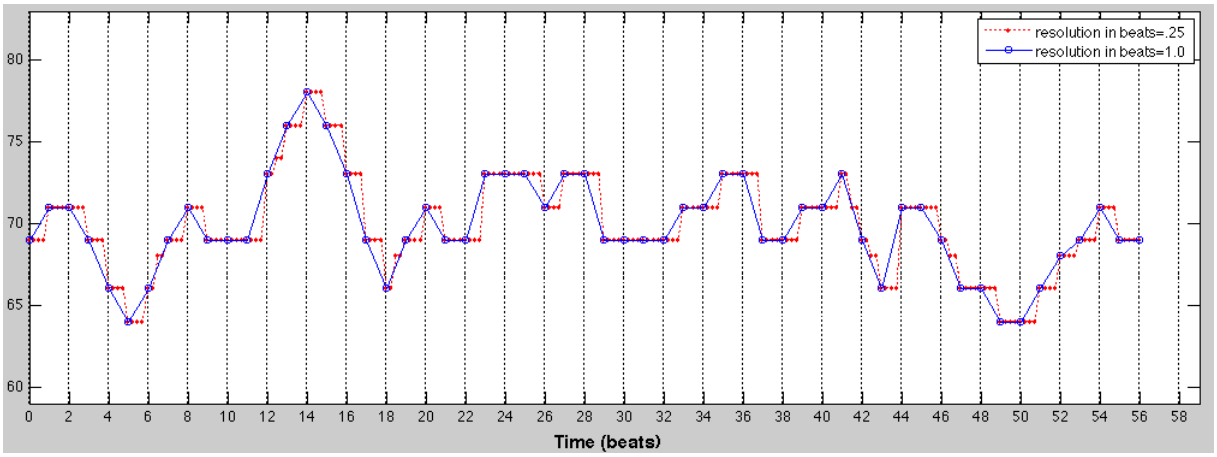

Figure 3.20: Bach Chorale contour diagram visualizing the MIDI representation. Pitch in *MIDI notes* on the vertical axis, time in *Beats* on the horizontal axis.

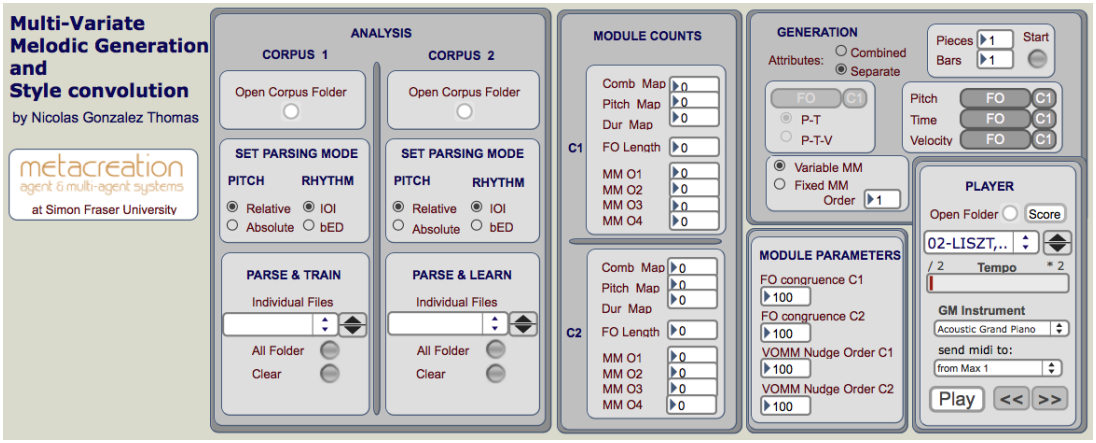

Figure 3.21: *Meta-Melo* interface in MAX/MSP.

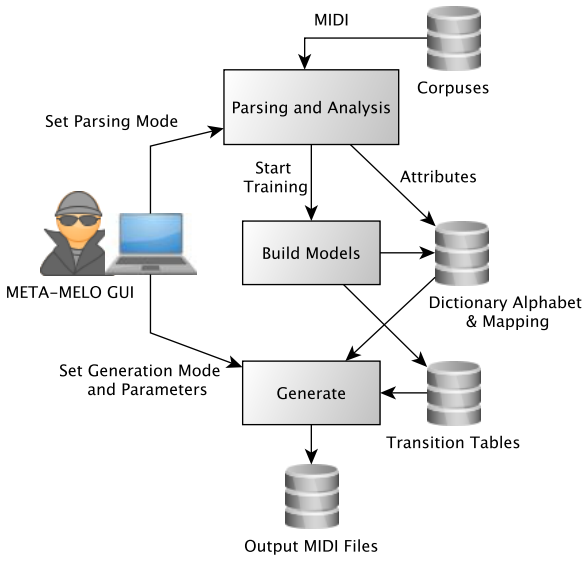

Figure 3.22: Meta-Melo system diagram.

Chapter 4

Evaluation Methodology and Experiments

In this chapter we present the methodology used for analyzing and evaluating the output of style imitation models. Section 4.1 gives an overview of the steps required and the types of analysis performed. Section 4.2 presents a selection of results from the analysis.

4.1 Methodology

In order to perform an in-depth analysis and evaluation of the output of the models, the methodology first requires the following pre-processing steps:

1. Converting melodies from MIDI to diverse custom representations.
2. Calculating similarity measures.
3. Extracting features through MIR tools.

Analysis and Evaluation Overview

The corpus and algorithm output are analyzed to establish the differences between them (inter-corpus analysis), as well as the similarity between the individual melodies within the output of a single algorithm (intra-corpus analysis). Similarity Analysis (Section 4.2.4), k-Means clustering (Section 4.2.5) and Classic Multidimensional Scaling (CMDS) (Section 4.2.3) are used for evaluating the closeness of the groups of melodies through cluster analysis, the calculation of confusion matrices and visualizations.

Since there are a multitude of features and similarity measurements available for studying melodies, we use Machine Learning for selecting the most relevant ones for differentiation with Decision Trees (Section 4.2.1). Then we show a plot using originality and complexity measures that shows a useful way to visualize the space of the corpus and the algorithm output (Section 4.2.2). In Section 4.2.7, we present a methodology for establishing differences with statistical significance using *permutation testing*. Finally in Section 4.2.8 we present a test that informs the behavior of the algorithms, through the process of iterating training and generation, and measuring their paths over time related to the original corpus.

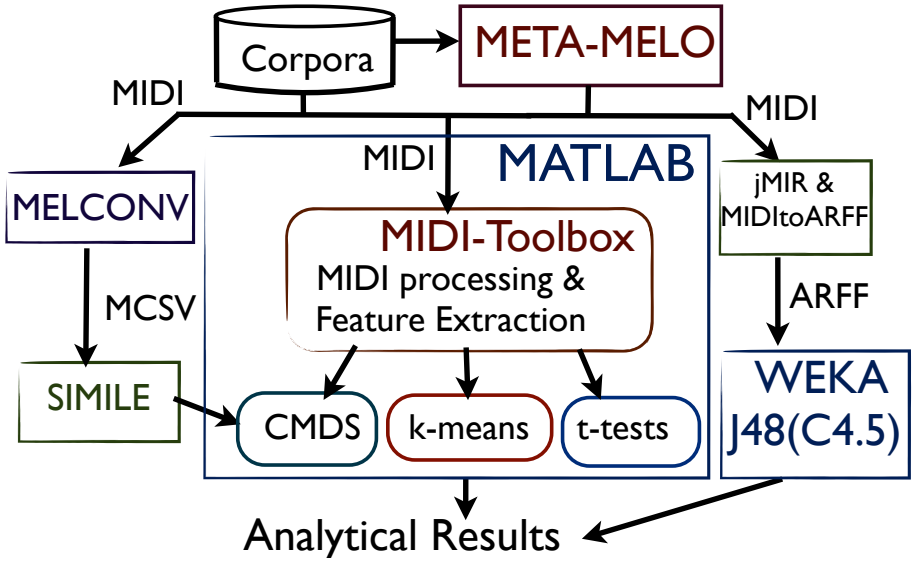

Figure 4.1: The implementation of the proposed methodology.

Implementation

Figure 4.1 depicts a diagram outlining the implementation of the proposed methodology. The evaluation requires an extraction of features from the corpora and the output. The following toolboxes and libraries were used:

- *MATLAB* Used for most processing of data, K-Means clustering, Multidimensional Scaling and permutation tests.
- *jSymbolic* [50] and *MIDItoARFF* [66]. MIDI feature extracting and conversion software in Java developed by Cory McKay at McGill (part of the jMIR toolbox). Over 180 features can be extracted and converted into ACE XML and WEKA ARFF formats.
- *FANTASTIC* [52]. Feature ANalysis Technology Accessing STatistics (In a Corpus) is a R platform software developed at Goldsmiths University of London for feature extraction and analysis of melodies in a corpus. Input is Music Comma Separated Value (MCSV) files that can be converted from MIDI by MELCONV (see below).
- *SIMILE and MELCONV* [32]. Are Windows command line programs developed by Dr Klaus Frieler for the conversion and comparison of MIDI monophonic files. MELCONV converts MIDI files to MCSV format for input to FANTASTIC and SIMILE. SIMILE is a melody similarity calculator, it computes a distance metric from two MCSV files using 80 optional feature analysis algorithms.
- *WEKA* [37]. The Waikato Environment for Knowledge Analysis is a machine learning Java software developed at the University of Waikato. It is used here for selecting the most significant features (C4.5 Java clone: J48) and for further data analysis and exploration.

- *Matlab MIDI Toolbox* [26]. Used for importing MIDI files directly into MATLAB and also for extracting melodic features.

4.1.1 Feature Extraction and Similarity Measures

There exists an abundance of algorithms for extracting features and similarity measurements from melodies [26, 32, 50, 52]. A thorough initial investigation was carried out for the purpose of evaluating and selecting the feature extraction algorithms and similarity measures that were most useful for our study. Below is a list and description of the features used, through a manual and also computational analysis we were able to select the most relevant ones based on their power to highlight the greater dissimilarity between melodies, combined with their correlation with the perception of human listeners. For example, Decision Trees (Section 4.2.1) were useful in determining which features were most powerful in distinguishing the groups of melodies output form different generative algorithms.

Feature extraction functions

The following is a list of some of the features used for analyzing the corpora and algorithm output:

- *Pcdist1*: Calculates the pitch-class frequency distribution of the melody weighted by note durations. An example distribution of the 498 Bach Chorales in the Classical corpus section can be seen in Figure 4.2. The note durations are based on duration in seconds that are modified according to Parncutt's durational accent model [59]. PCD = 12-component vector listing the probabilities of pitch-classes (C, C sharp, D, D sharp, E, F, F sharp, G, G sharp, A, A sharp, B).

- *Pcdist2*: Calculates the 2nd order pitch-class frequency distribution of the melody weighted by note durations. The note durations are based on duration in seconds that are modified according to Parncutt's durational accent model [59]. PCD = 2nd order pitch-class distribution of the melody is a 12 * 12 matrix. Ex. PCD(1,2) = probability of transitions from C to C sharp

- *Ivdist1*: Returns the frequency distribution of intervals in the melody as a 25-component vector. The components are spaced at semitone distances with the first component representing the downward octave and the last component the upward octave. The distribution is weighted by note durations. The note durations are based on duration in seconds that are modified according to Parncutt's durational accent model [59].

- *Ivdist2*: Returns the frequency distribution of interval dyads in the melody. The distribution is weighted by note durations. The note durations are based on duration in seconds that are modified according to Parncutt's durational accent model [59].

- *Contour*: Distribution of contour vectors are calculated using a relative sampling interval of the melody (instead of an absolute interval in bars or beats) We normalize the contours of each melody to a fixed length vector and obtain a distribution of the collection on each point. The distance of the contours in a similarity measurement is taken by calculating the distance of the vectors after normalization by subtracting the means of the vector. This is required in order to discard pitch heights.

- *Combcontour*: For a melody with n notes, we build an n x n matrix of ones and zeros. A one is inserted in the i, jth entry if the pitch of note i is higher than the pitch of note j. A zero is inserted otherwise. This matrix is a representation of melodic contour by Marvin *et al*. [46], preserving relative rather than specific pitch height information.
- *Durdist1*: $\Delta t_i = t(i+1) - t_i$. Returns the distribution of note durations in the melody as a 9-component vector.
- *Durdist2*: As durdist1, this function returns the distribution of pairs of note durations of successive notes as a 9 * 9 matrix.
- *Entropy*: Returns the of distribution [71].
- *Pitch Range*: The distance in semitones between the lowest and highest notes of the melody.
- *Pitch Centre*: The mean pitch of a melody.
- *Pitch SD*: The standard deviation of the pitch within the melody.
- *Interval Range*: The distance in semitones between the smallest and largest intervals. as well as the distribution of this mean across collection.
- *Interval Size*: Mean interval size within a piece as well as a collection.
- *Interval Dissonance Ratio*: Number of dissonant intervals within a piece or collection.
- *Chromaticism*: Number of distinct pitch classes. Through observing the pitch class distribution we determine a ratio of chromaticism given by the the use of pitches outside the scale tones.
- *Repetition and motif*: Self similarity and auto-correlation is measured according to Box *et al*. [12].
- *Rhythmic density*: The mean number of note events per beat.
- *Rhythmic Variability*: The degree of change in note duration.
- *Syncopation*: The deviation from the regular beat pattern. [25].

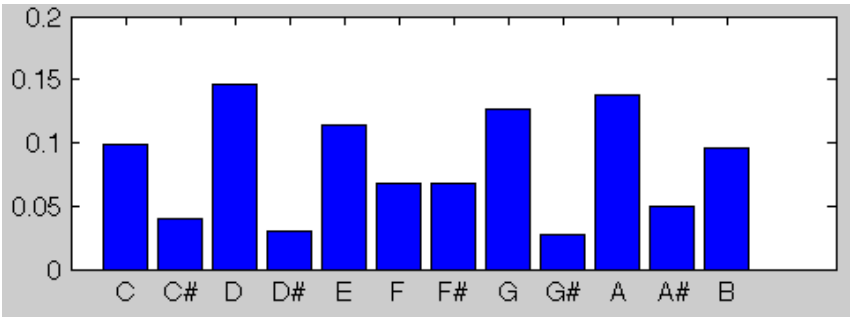

Figure 4.2: Pitch-Class Distribution of the 498 Bach Chorales, weighted by note duration.

Note that for most features used we also calculate the similarity of the melodies by obtaining the Euclidean or Taxi distance and re-scaling to a value between $0-1$ where 1 indicates highest similarity. As it has been noted by Müllensiefen and Frieler [32, 53, 54], hybrid measures have a greater predictive power than single-attribute measures for estimating melodic similarity, these have proved useful in our study as we will see in the results section. They provide a series of optimized metrics that have been shown to be comparable to human similarity judgments [55]. For large melody collections 'Opti3' is recommended, it is based on a weighted linear combination of interval-based pitch, harmonic content and rhythmic similarity estimates, normalized between 0-1, where a value of 1 indicates that melodies are identical.

4.2 Results

4.2.1 Decision Trees

After a thorough exploration of techniques available in WEKA for feature reduction [37], we present here the results which we found most relevant, obtained with Decision Trees. In this example we generated C4.5 decision trees and used the results as guidance for reducing our feature space. This method was useful for arriving at an initial result in the search for musical biases by detecting which features, amongst the many extracted with *jSymbolic* (~ 180), distinguish the output of different algorithms from one another and from the corpus. We are interested in investigating precisely what aspects of the output of the algorithms are different from the training corpus. And also for example, how is the output of Markov different if at all, from the Factor Oracle? The classifier takes as input the melodies with category and values for all the features selected in *jSymbolic*. By selecting a high level of pruning we constrain the algorithm to select the features that have the greatest accuracy in determining the correct category in the least amount of steps (levels of the tree). This is extremely useful since the MIR field has provided a countless supply of algorithms and similarity metrics for multiple dimensions of music and specifically melodies, only a few of which have great differentiation power in a sense that is useful to our study. We repeatedly found results where the features selected were the same and so these were used in other parts of the study.

Figure 4.3 shows a tree learned on the output from the algorithms (32 pieces generated from Factor oracle: FO, Markov Models: MM, MusiCOG: MC) trained on the 100 piece Folk corpus collection (CC). We see that the tree is based on, first, the Syncopation in the track (the ratio of notes *on* versus *off* the beat) and then principally the longest duration (in beats). Particularly we can observe that a low TrackSyncopation and high LongestDuration classify most MC instances. This example was useful for discovering a programming error in MC that was duplicating note durations in certain parts of the generation. Another aspect to note is that FO and MM output are in the extremes of syncopation.

Once the note duration problem in MC was addressed, the output of the tree was as shown in Figure 4.4. The three features arrived at in this example by using the C4.5 decision tree are:

1. *Compltrans* [68, 69], a melodic originality measure that scores melodies based on a 2nd order pitch-class distribution (transitions between pitch classes) obtained from a set of ~ 15 thousand

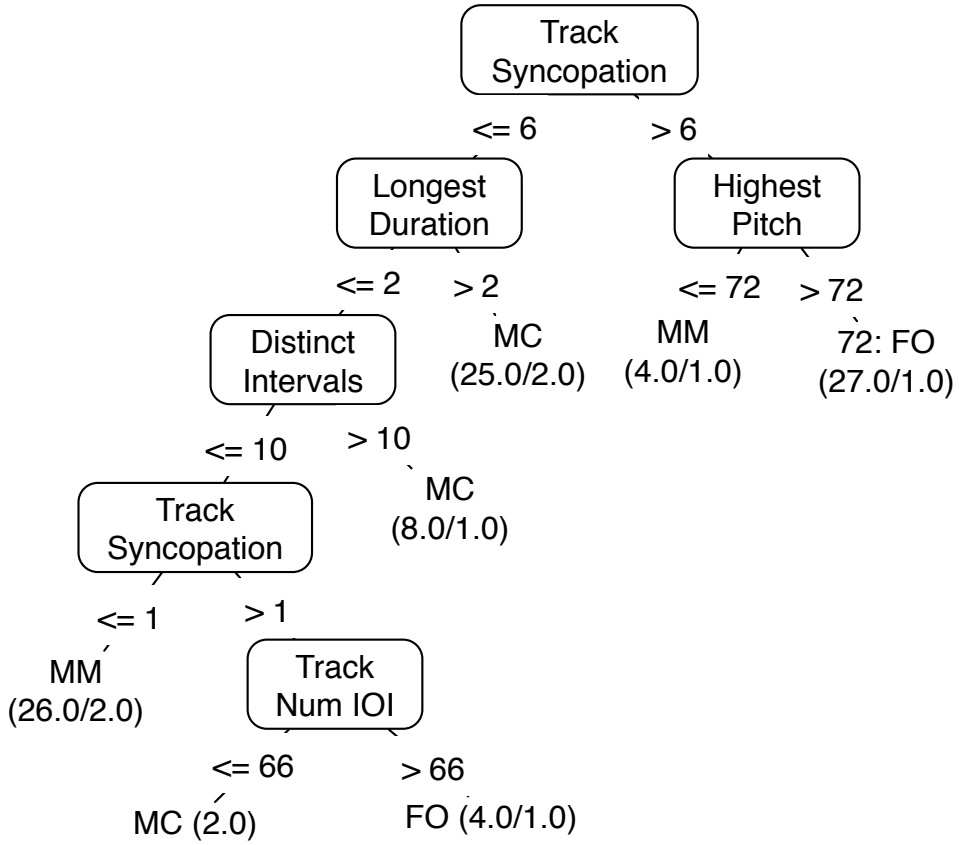

Figure 4.3: C4.5 decision tree (J48). The features closest to the root of the tree are the ones with strongest prediction power. The first number in the leaf is the count of instances that reach that leaf, the number after the dash, if present, indicates the count of those instances that are miss-classified along with the correct class.

classical themes. The value is scaled between 0 and 10 where a higher number indicates greater originality.

2. *Complebm* [25], an expectancy-based model of melodic complexity that measures the complexity of melodies based on pitch and rhythm components calibrated with the Essen collection [67]. The mean value is 5 and the standard deviation is 1, the higher the value the greater the complexity of the melody.

3. *Notedensity*, the number of notes per beat. Details of these features can be found in the *MIDI Toolbox* documentation [26].

Three following aspects of the tree stand out:

1. In the right-most leaf we see that a value of Complebm > 4.5 (with Compltrans > 7.48) classifies 58 melodies as FO, but of those 3 are of the corpus and 25 are MM. Therefore MM and FO and are not easy to distinguish.

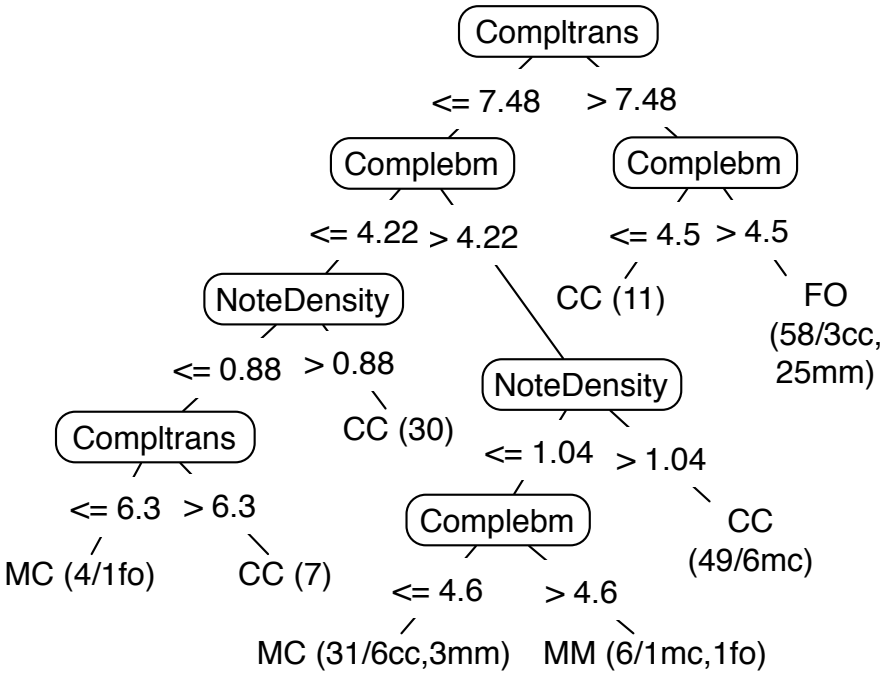

Figure 4.4: Another C4.5 decision tree (J48). TIn the right-most leaf we see that a value of Complebm > 4.5 (with Compltrans > 7.48) classifies 58 melodies as FO, but of those 3 are of the corpus and 25 are MM. Therefore MM and FO and are not easy to distinguish.

2. The root (*Compltrans*) successfully separates 94% of FO and 78% of MM instances from 86% of CC and 100% of MC (all of the MC pieces are to the left of the root), an indication of greater similarity between CC and MC since these two groups are not distinguished so clearly.

3. The *Notedensity* feature seems to greatly aid in classifying and distinguishing MC from CC where other features are less successful. This type of analysis provides valuable diagnostic insight on the MC algorithm since we can deduce that an increase in note density on the output would potentially improve the imitation capabilities of the algorithm.

4.2.2 Originality and Complexity

In Figure 4.5 the corpus and the output is plotted using the two features from the *MIDI Toolbox* [26] that were the result of running the decision trees in the previous section:

As we were able to see in the results of the decision trees, the plot shows a clear overlap between the corpus and MC, whereas MM and FO output cluster together with higher values on both dimensions. The regression lines (in black) are calculated one per group and graphically display the position, angle and length that best separates the group in half (i. e. the line is in the median position). Observing these lines we see how close the MM and FO output are, in length angle and position. Similarly with CC and MC, the middle point of the lines are close but the angle and length are not. The the CC collection having three times the size, naturally occupies a wider range.

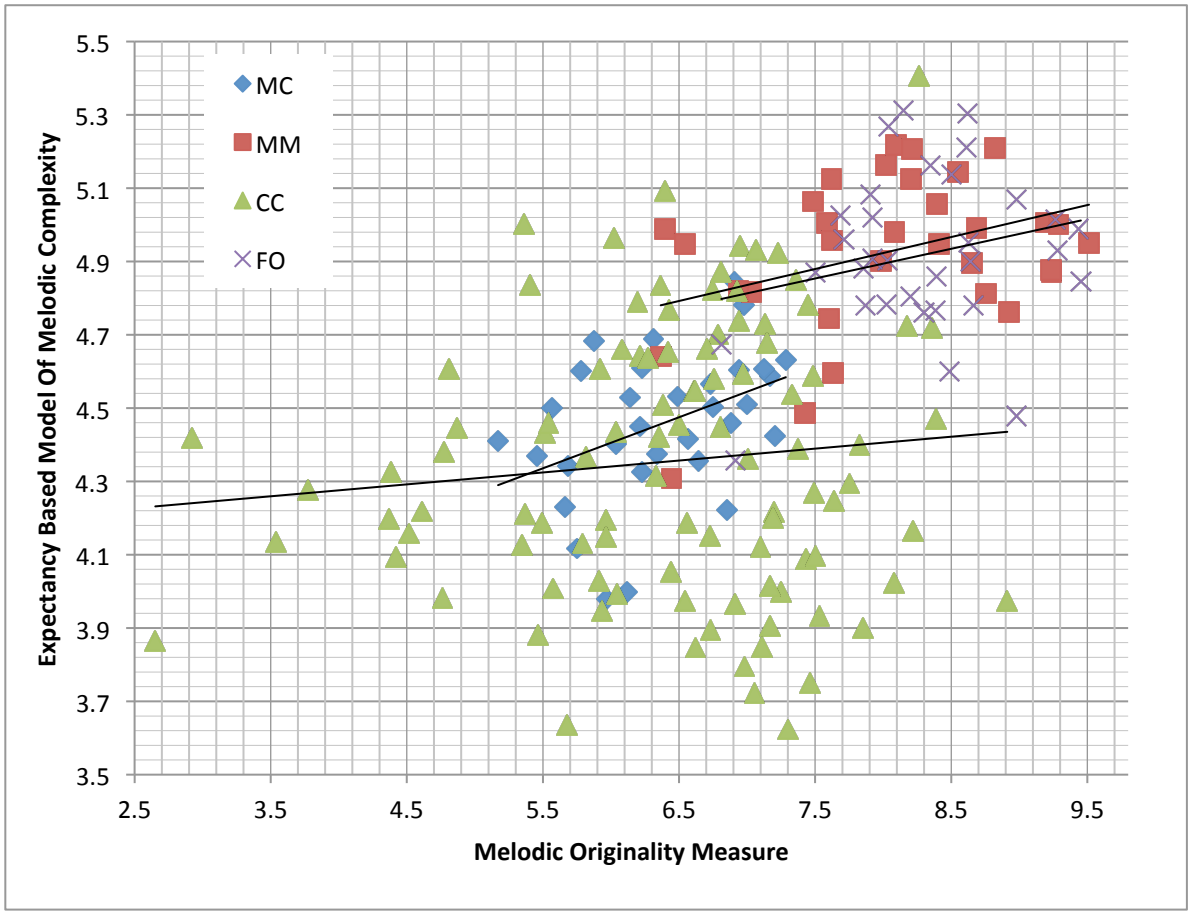

Figure 4.5: Expectancy Based Complexity and Originality Measure. Folk corpus (CC) and algorithm output. We can see that FO and MM output cluster and, separately there is an overlap between MC and corpus. The black regression lines help associate closeness-distance in the groups.

4.2.3 Classic Multi-dimensional Scaling

Classic Multi-dimensional Scaling (CMDS) is a multivariate analysis process similar to Principal Component Analysis (PCA), used for visualizing the clustering of N-dimensional data. The plots created with this process display elements in the positions that most clearly visualizes the distances between them. We first calculated dissimilarity matrices from pitch-class, interval distribution, contour vectors and note durations (*pcdist1*, *pcdist2*, *ivdist1*, *ivdist2*, *contour*, *combcontour*, *durdist2* and *durdist2*). Figure 4.6 shows the CMDS diagram for FO and MC alone (for clarity). We can see that pitch class and interval distributions distinguish the two groups quite clearly, whereas note duration is the least helpful. The axes in a CMDS diagram have no particular meaning but sometimes they can be inferred through studying the particular points. For example we notices that in the *pcdist1* based diagram (top left) the x axis correlates strongly with chromaticity: right = chromatic or highly modulating, left = tonal. On the other hand, in the *ivdist1* based diagram, the x axis correlates with interval size.

Figure 4.7 shows the CMDS diagram for all the algorithms output and folk corpus. In this example, only pitch-class provides a clear distinction, but evidently shows how significantly distant the algorithm's

output is from the space occupied by the corpus. Although in these examples the number of melodies in the corpus is greater, we can notice how there is a large space occupied by the corpus melodies with which no generated melody has significant similarity.

Figure 4.8 shows a larger version of the CMDS diagram in the top right corner of figure 4.7 created from *pcdist1* measurements.

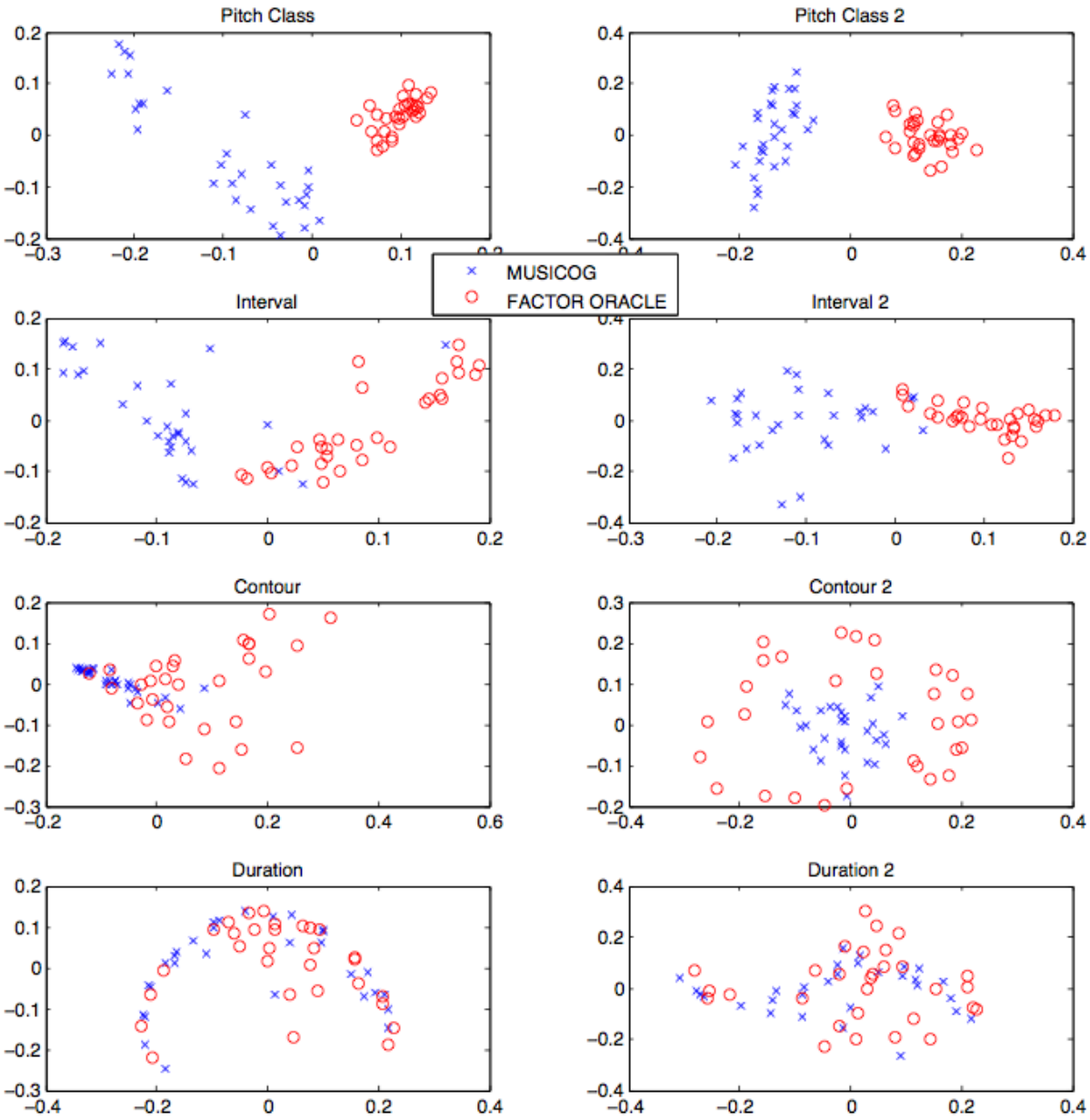

Figure 4.6: Multidimensional scaling for MusiCOG and Factor Oracle, for the following features: *pcdist1*, *pcdist2*, *ivdist1*, *ivdist2*, *contour*, *combcontour*, *durdist2* and *durdist2*

We continued by calculating dissimilarity matrices from the similarity measures obtained with 'Opti3'. In Figure 4.10 we can see that the horizontal axis separates quite generally the corpus from the algorithm output. Although as mentioned the dimensions do not have an explicit meaning, it is evident again that

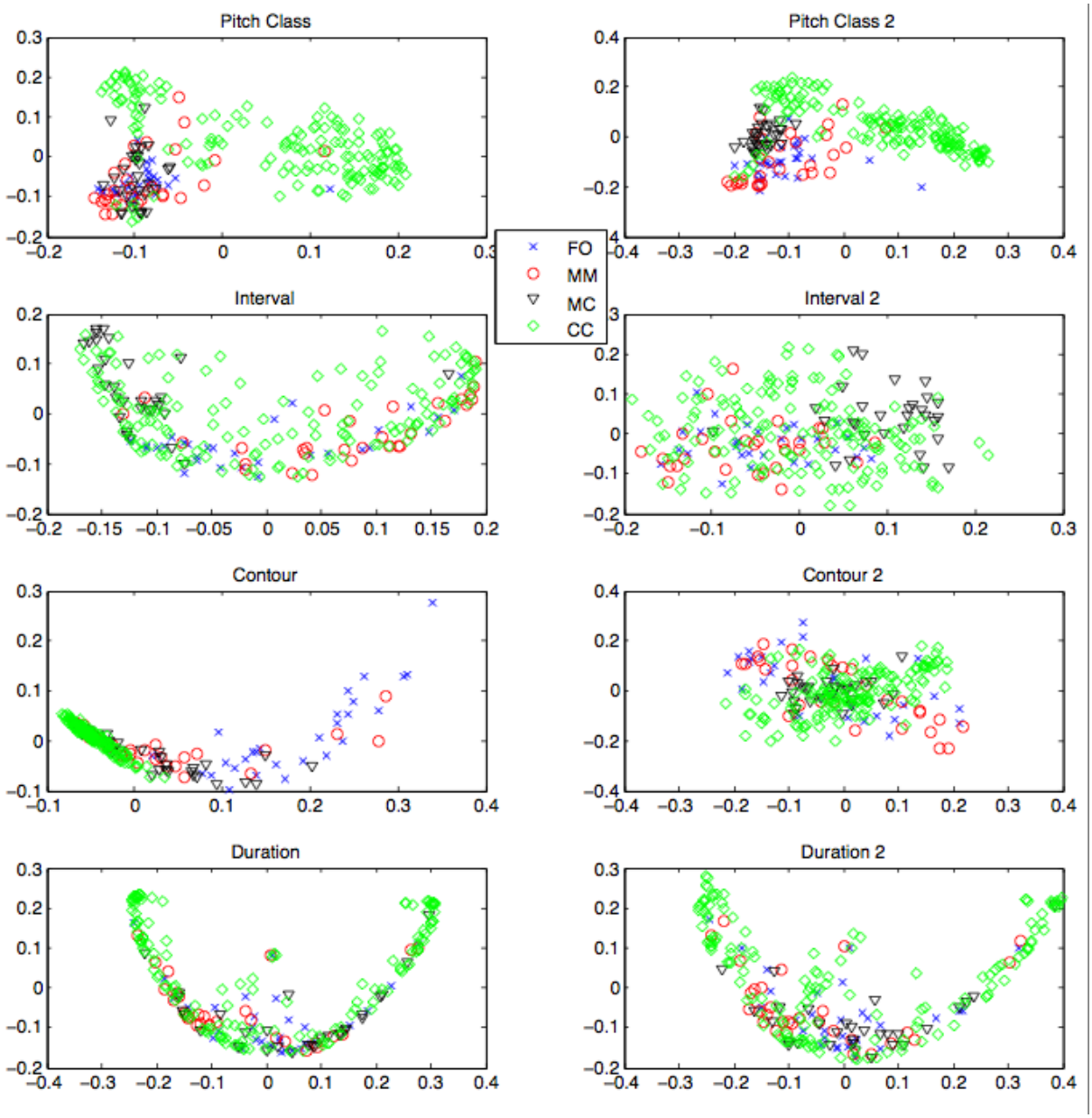

Figure 4.7: Multidimensional scaling for all models and corpus, for the following features: *pcdist1*, *pcdist2*, *ivdist1*, *ivdist2*, *contour*, *combcontour*, *durdist2* and *durdist2*.

the algorithms do not explore the full 'creative' range of the corpus. We can also see that in contrast to MM and MC output, FO is not as clustered, occupying a broader range in the space.

In Figure 4.11 we show a CMDS diagram using the 'Opti3' again but for 100 melodies per algorithm. CMDS calculations output several dimensions, generally the first two are the most relevant to plot. In this example we have taken the first four dimensions and show the data as two dimensional plots, each pair of dimensions indicated on the top label. Both diagrams with dimension 2 (left side) display the corpus cluster distinctly from the three groups of generated melodies. Figure 4.12 shows an amplified diagram of dimensions 1 and 2, where we can see all the algorithms output clustering in a section, most of the corpus on a separate space and the FO having a section of melodies entirely separate.

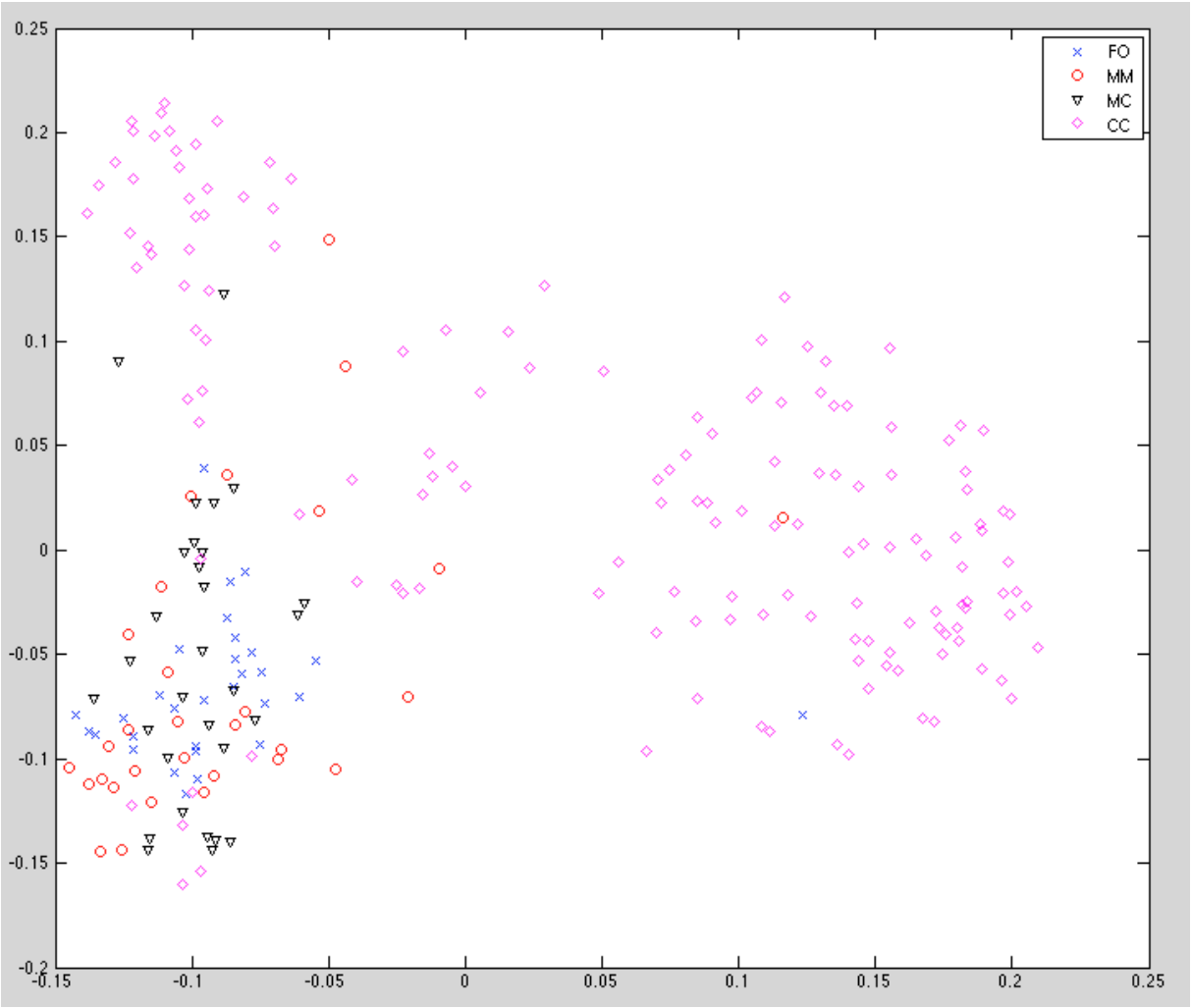

Figure 4.8: Multidimensional scaling for the corpus and all models output observing Pitch-Class alone (*pcdist1*).

Figure 4.13 shows a diagram that includes a group of randomly generated melodies. As a way to gain insight on the 'scale' of difference between the melodies trained on the corpus and the corpus itself. Finally, Figure 4.9 shows a 2 dimensional MDS plot for dimensions 1 and 2 using 'Opti3' distance measure. This time we selected only 30 melodies from the corpus and generated the same number. It appears that MC is 'under-trained' in this example, as the melodies generated are more distant to the corpus than the example with 100 pieces.

4.2.4 Similarity Analysis

We have seen in the previous sections that the melodic originality and complexity measures have been successful with decision trees, and the 'opti3' similarity measure with CMDS diagrams. We will continue exploring other forms of analysis using these measures.

In this section the corpus and algorithm output are analyzed to establish the similarity between them (inter-corpus analysis), as well as the diversity within the sets (intra-corpus analysis). Table 4.1 shows

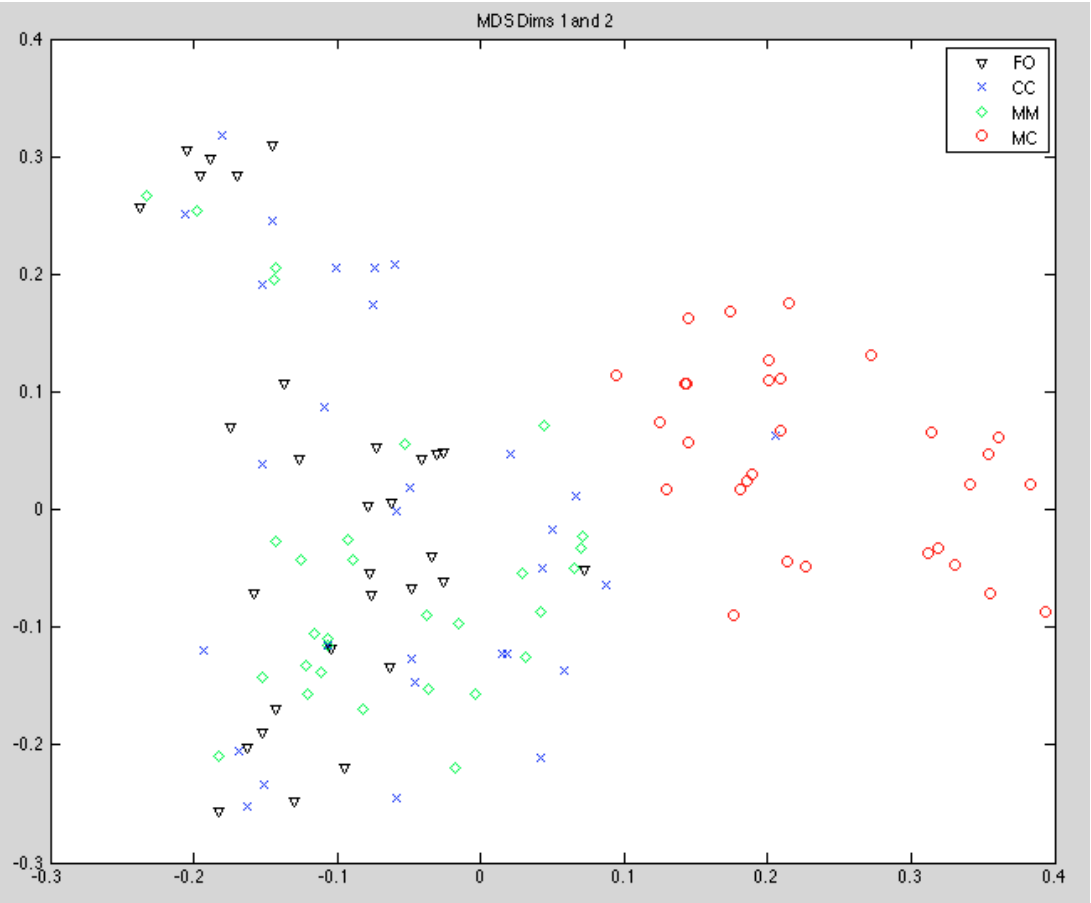

Figure 4.9: The 2 dimensional MDS plot for dimensions 1 and 2 using Opti3 distance measure. It appears that MC is 'under-trained' in this example, as the melodies generated are more distant to the corpus than the example with 100 pieces.

results from using the 'Opti3' measure to calculate the mean of the distances between each melody from the set (row) against all melodies in the column set (cartesian product). We can interpret the diagonal as intra-algorithm dissimilarity, i.e. the diversity or uniformity of each set, for which a low value indicates higher diversity.

The top section of Table 4.1, presents the values for the Folk corpus, where we can interpret some the following insights:

1. Observing the diagonal: MC (.208) is marginally the least and FO (.198) the most diverse of all sets, including the corpus (.201).

2. Against the corpus: MM at .178 produces the output that is most similar to the corpus, FO the least.

The rest of Table 4.1, presents the values for the the other corpora (Jazz, Pop, Classical). We can then interpret some the following insights:

1. MC is closest to the Pop and Jazz corpora.

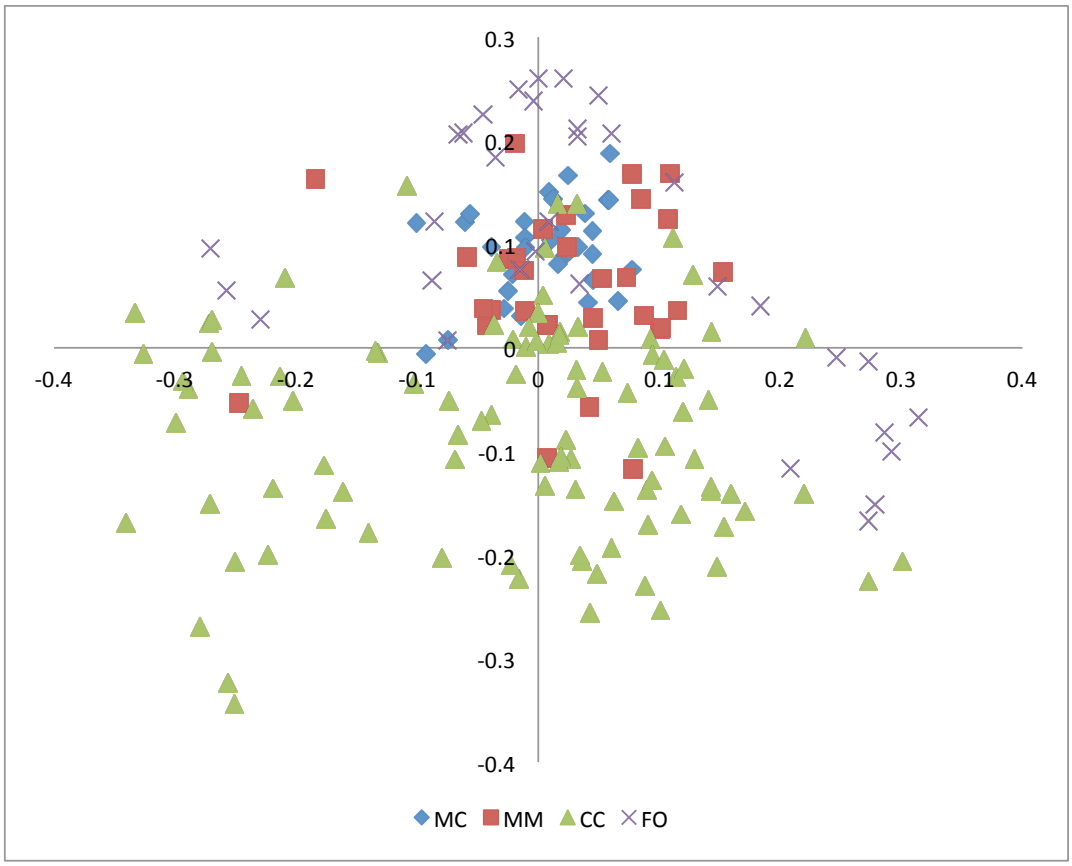

Figure 4.10: Multi-dimensional scaling for all algorithms and corpus, using the optimized distance metric 'Opti3'.

2. FO is furthest from the corpora in every case.

3. The FO Pop output is tightly clustered but relatively far from the corpus. Listening to samples from the MM and FO that indicate good performance, it was empirically noticed that there is a tendency toward direct 'quotations' of the original melodies. This is likely due to the order of the MM, and the probability of congruence parameter in the FO (Section 3.2), respectively.

Table 4.2, presents the Melodic similarity of the corpora alone. We can then interpret some the following insights:

1. With an intra-corpus distance of .201, we see that Pop melodies have the lowest diversity and the Jazz has the highest with .062.

2. The Classical and Folk corpora are the furthest apart with an inter-corpus distance of .012, the Pop and Classical being the most similar.

Intra and Inter-algorithm Melodic Similarity

Algorithm	Markov	MC	F.Oracle	Corpus
Markov (MM)	.206			
MusiCOG (MC)	.183	.208		
Factor Oracle (FO)	.166	.165	.198	
Folk Corpus (CC)	.178	.174	.154	.201
Markov	.101			
MusiCOG	.076	.146		
Factor Oracle	.081	.053	.141	
Jazz Corpus	.078	.080	.055	.062
Markov	.177			
MusiCOG	.158	.207		
Factor Oracle	.124	.121	.264	
Pop Corpus	.076	.092	.058	.139
Markov	.143			
MusiCOG	.081	.147		
Factor Oracle	.082	.055	.156	
Classic Corpus	.080	.075	.062	.079

Table 4.1: Mean melodic similarity of algorithm output and corpora using the “Opti3" similarity measure (1.0 = identity). Intra-algorithm similarity is represented in the diagonal, higher value indicates higher similarity.

Intra and Inter-Corpus Melodic Similarity

Corpus	Classical	Folk	Jazz	Pop
Classical	.079			
Folk	.012	.201		
Jazz	.043	.041	.062	
Pop	.085	.014	.063	.139

Table 4.2: Melodic similarity of the corpora using the “Opti3" similarity measure (1.0 = identity).

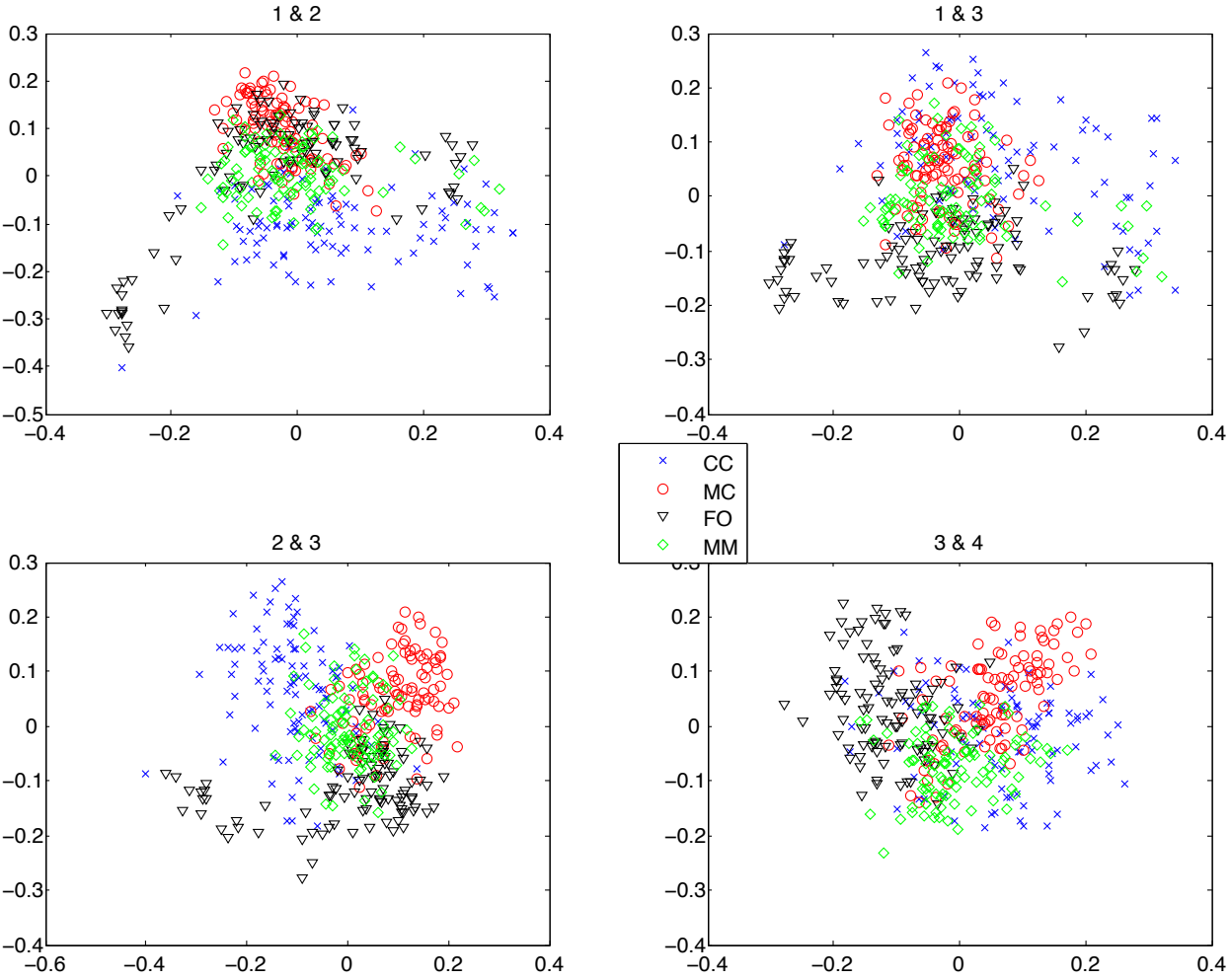

Figure 4.11: An MDS plot using the *Opti3* distance measure. Four dimensional output of the data shown as two dimensional plots, each pair of dimensions indicated on the top label. Inner label indicates the Folk corpus collection (CC) and algorithm output symbols.

4.2.5 k-means Clustering Analysis

We ran a clustering algorithm (k-means) as another method for investigating differences and similarities between the corpus and the generated output. We focused on the Folk corpus and first generated 10 melodies with each algorithm, looking at clustering with values from selected features.

In Table 4.3 interval distances were used for the clustering algorithm. Using a K of 3 we tested whether the algorithm was capable of separating the individual melodies, that were mixed together, by grouping them correctly. We see that with interval distances the algorithm performs rather poorly in separating Markov output from Factor Oracle Output, whereas again, MusiCOG stands aside. This is in accordance to our system, since we train and generate both models (FO and MM) at the interval level, whereas MusiCOG initially parses the melodies by phrases.

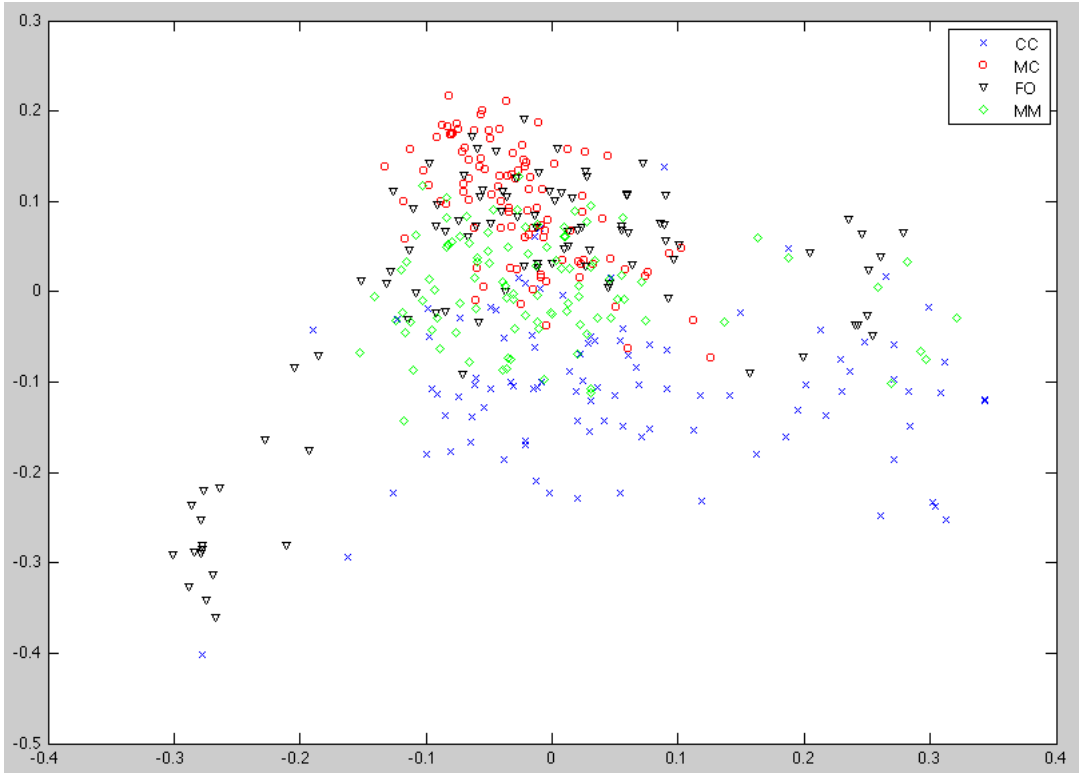

Figure 4.12: The MDS plot for dimensions 1 and 2.

K-Means Interval

Model	Cluster1	Cluster2	Cluster3
Markov	6	4	
Factor Oracle	7	3	
MusiCOG			10

Table 4.3: The K-Means Clustering of the models using Interval features (K=3).

In Table 4.4 we observe that the clustering is much more accurate using a Pitch-Class distribution, assigning only one Markov melody erroneously in the Factor Oracle cluster. In Table 4.5 we observe the same results when using pitch-class and interval distributions combined.

Thus, the pitch-class feature alone is sufficient for clustering with the greatest accuracy in this example.

In the following K-Means tables (4.6 to 4.11) we present the clustering in pairs of the training collection and the model output using either interval or pitch class distributions. Again MusiCOG is clustered with high accuracy with either distribution, showing a greater degree of difference between corpus and output.

Markov and Factor Oracle output are poorly clustered with intervalic distance but highly with the pitch-class distribution.

Then we used the set of 32 melodies per algorithm and a selection of over 100 features were extracted using WEKAtoARFF, the MIDI Toolbox, and jSymbolic for the clustering algorithm. The confusion

K-Means Pitch-Class

Model	Cluster1	Cluster2	Cluster3
Markov	9	1	
Factor Oracle		10	
MusiCOG			10

Table 4.4: The K-Means Clustering of the models using Pitch-Class features (K=3).

K-Means Interval and Pitch-Class

Model	Cluster 1	Cluster 2	Cluster 3
Markov	9	1	
Factor Oracle		10	
MusiCOG			10

Table 4.5: The K-Means Clustering of the models using Pitch-Class and Interval features (K=3).

K-Means Interval

Markov	Cluster 1	Cluster 2
Folk Corpus	8	2
Model Output	5	5

Table 4.6: The K-Means Clustering of Markov output and Folk Corpus using Interval features (K=2).

K-Means Interval

Factor Oracle	Cluster 1	Cluster 2
Folk Corpus	6	4
Model Output	3	7

Table 4.7: The K-Means Clustering of Factor Oracle output and Folk Corpus using Interval features (K=2).

K-Means Interval

MusiCOG	Cluster 1	Cluster 2
Folk Corpus	8	2
Model Output		10

Table 4.8: The K-Means Clustering of MusiCOG output and Folk Corpus using Interval features (K=2).

K-Means Pitch-Class

Markov	Cluster 1	Cluster 2
Folk Corpus	5	5
Model Output	8	2

Table 4.9: The K-Means Clustering of Markov output and Folk Corpus using Pitch-Class features (K=2).

K-Means Pitch-Class

Factor Oracle	Cluster 1	Cluster 2
Folk Corpus	10	
Model Output		10

Table 4.10: The K-Means Clustering of Factor Oracle output and Folk Corpus using Pitch-Class features (K=2).

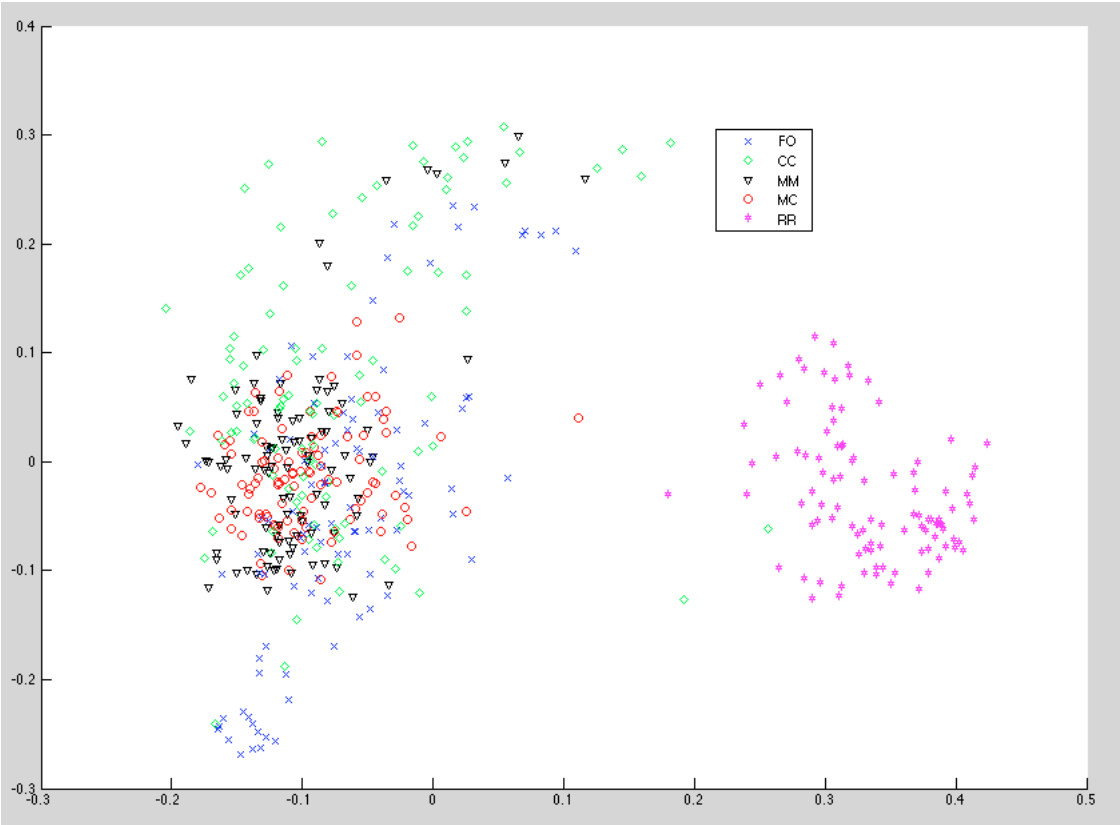

Figure 4.13: Multidimensional scaling for the corpus and all algorithms output plus a random generation of melodies.

matrices are presented in Tables 4.12 and 4.13, which also indicate the clustering of algorithm output and corpus.

We see in the first example on Table 4.12 that the MM and FO are mostly clustered together whereas MusiCOG stands on it's own in cluster3 (overlapping partly with Factor Oracle in cluster 2). In Table 4.13 we notice that the corpus stands mostly on it's own cluster, there is no particular distinction between the algorithms except for MusiCOG that is partly clustered alone.

4.2.6 Similarity Contrast Diagrams

In this section we show diagrams that are often used to visualize similarity matrices. These are created by assigning a contrast value to each similarity value on the matrix. Each value is a measure between two

K-Means Pitch-Class

MusiCOG	Cluster 1	Cluster 2
Folk Corpus	10	
Model Output		10

Table 4.11: The K-Means Clustering of MusiCOG output and Folk Corpus using Pitch-Class features (K=2).

k-means algorithm Output

Algorithm	Cluster1	Cluster2	Cluster3
Markov	32		
Factor Oracle	21	11	
MusiCOG	2	11	19

Table 4.12: The confusion matrix with k-means clustering of the algorithms output (K=3).

k-means algorithms and Corpus

Algorithm	Clust1	Clust2	Clust3	Clust4
Markov	22	3	7	
Factor Oracle	7	16	9	
MusiCOG	11	3	3	15
Corpus	4		96	

Table 4.13: The confusion matrix with k-means Clustering of the algorithms and Corpus (K=4).

melodies. In this analysis we used a sliding value for generating from the FO (congruence), therefore the melodies were diverse in similarity to the corpus, since a high value would guarantee replication.

Figure 4.14 shows an example of a $400X400$ matrix with corpus and algorithm output 400 melodies from all the algorithms and corpus compared. Figure 4.15 shows an example of a $500X500$ matrix, this time including a group of randomly generated melodies. A few aspects to notice are:

- As the FO algorithm is set to vary the replication parameter as generates, from a low to a high value, the melodies generated become more deterministic by following the same path of the Oracle, and therefore more similar to themselves. This explains the brightest section of the diagram on the top left.
- MC shares a similar result as it is set to learn from it's output in the second half of the melodies generated. Therefore the brighter section seen in the melodies compared to themselves.
- The evident contrast on how dissimilar randomly generated melodies are compared to not only the other groups but also with the melodies in the same group.

4.2.7 Significance: Computational Discrimination Test Through Permutation Testing

In this section we test the difference between groups and determine statistically whether they can be distinguished as separate using similarity measurements, informally this is similar to a Discrimination Test but executed computationally. We arrive at a results where we can test for statistical significance using Permutation testing [40].

The problem with t-tests or ANOVA

In our methodology these tests cannot be used as the assumption of independence of all observations is violated. The issue is not the use of similarity, for if the observations were sets of separate pairwise similarity measurements (exclusive one to one) there would be no issue (since these observations would

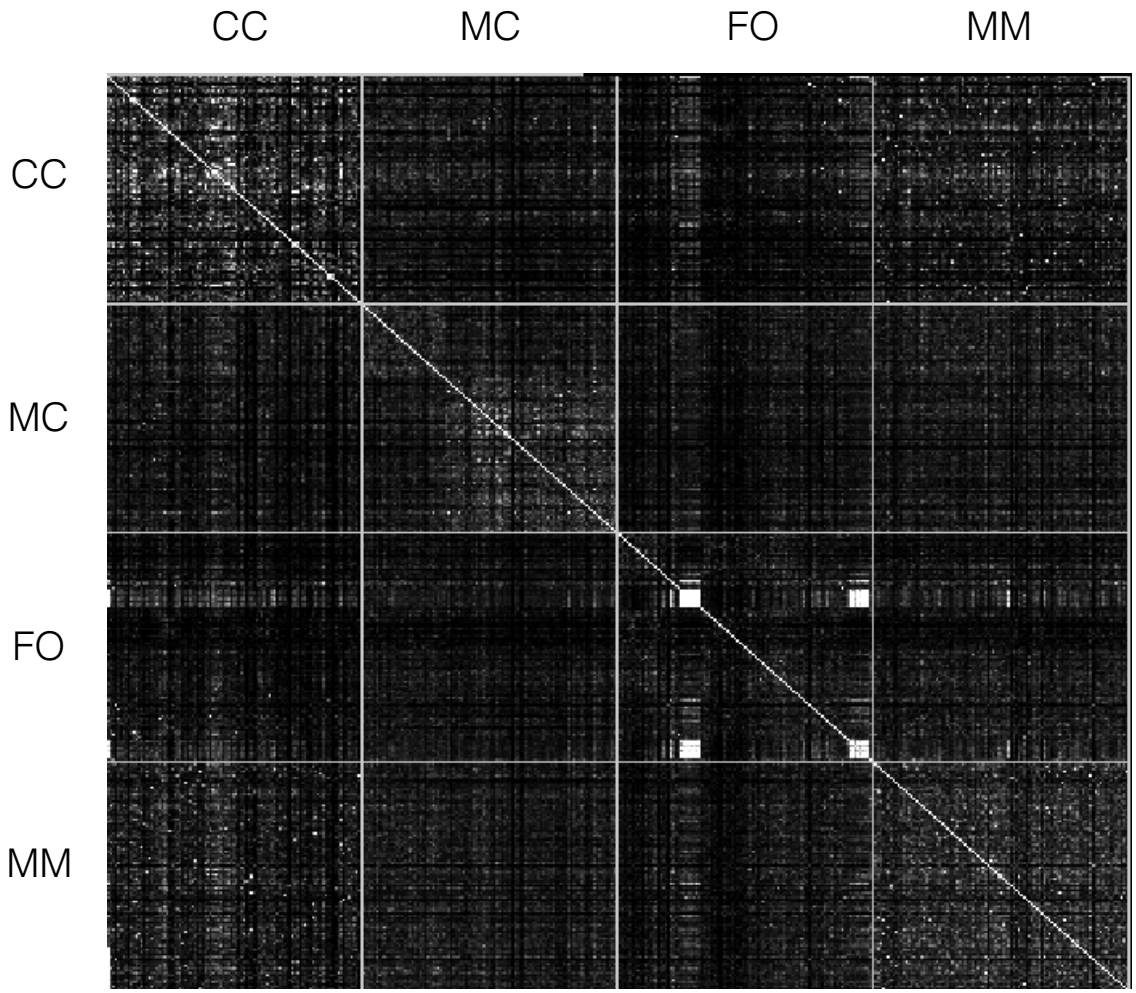

Figure 4.14: Similarity Contrast Diagram.

be independent). The problem arises with the *one to many* relationship in the measurements: as similarity measurements are calculated to obtain a *similarity matrix*, every melody is compared with every other in both sets, the corpus and the output. Therefore all of the pairwise similarity observations related to each single melody are correlated, violating the independence of all observations required for utilizing a t-test or ANOVA.

Permutation testing

Permutation tests [40], a type of re-sampling method also known as *Randomization testing* [36], do not have the requirement of independence of observations and are therefore applicable for the task of testing similarities between groups with correlated observations.

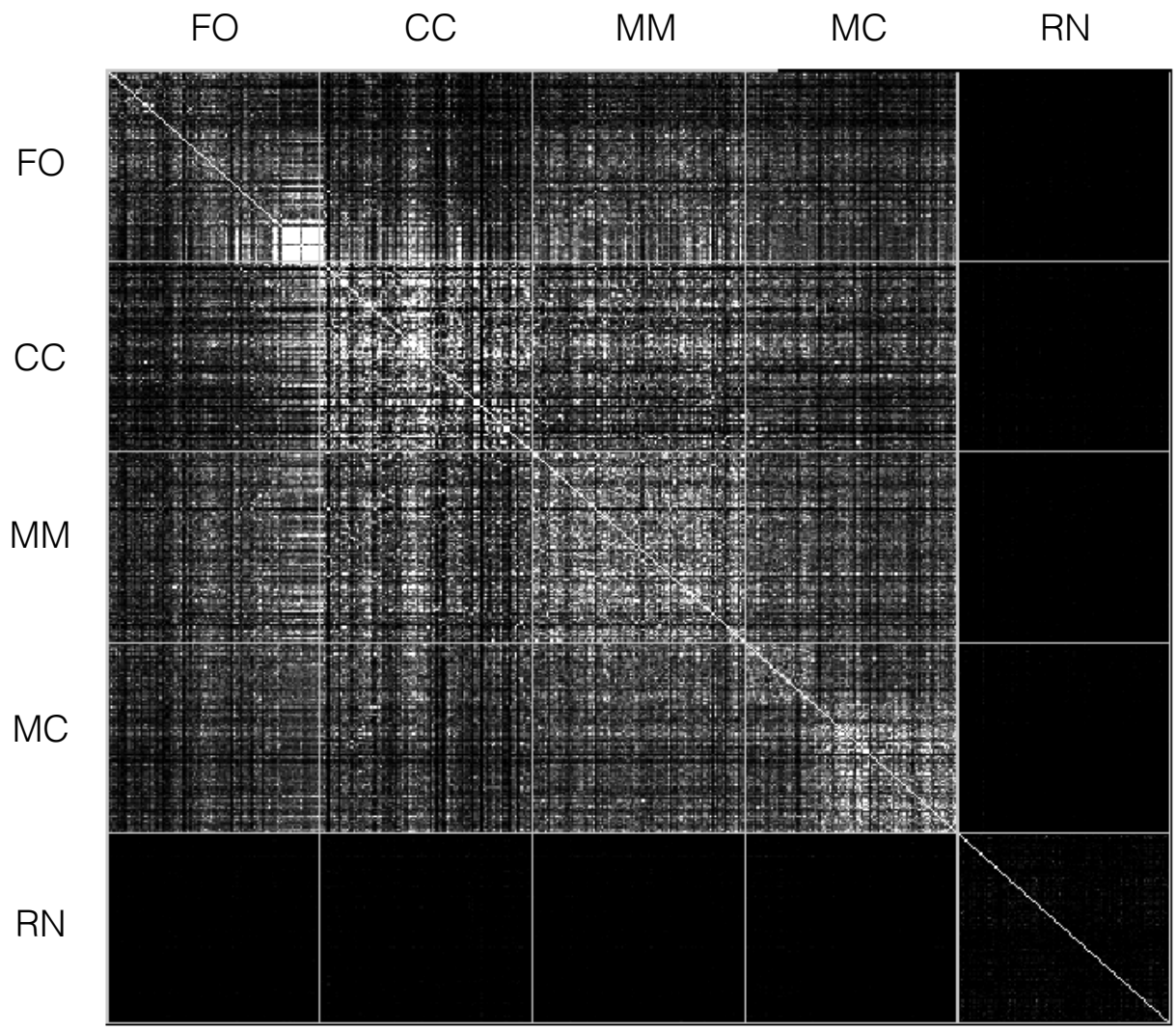

Figure 4.15: Similarity Contrast Diagram, including a random group of melodies.

Our null hypothesis H_0 is that there is *(*no-difference) $A \approx B$ between the two groups being tested, corpus training set (group A) and model output (group B), the alternative is *difference* $A \neq B$. The test requires the repeated comparison of an initial distribution calculated from an ordered set of observations (A next to B), this is called the *test statistic* T_0. This measure is then compared multiple times to a randomized order of measures (the order of values of A and B are randomized). Therefore the approach is to discover or test whether the similarity measurements between the melodies of two groups is meaningful when correctly ordered. We assume that if H_0 is true, the order of the observations is exchangeable, the test statistic will be located in the central part of the permuted distribution. If the order of the groups/observations is meaningful and not a random selection, then the test statistic will be located in a non-central part of the permuted distribution. In this latter case H_0 must be rejected.

	A_1	A_2	A_3	B_1	B_2	B_3
A_1	1	X	X	X	X	X
A_2	X	1	X	X	X	X
A_3	X	X	1	X	X	X
B_1	X	X	X	1	X	X
B_2	X	X	X	X	1	X
B_3	X	X	X	X	X	1

Table 4.14: Similarity Matrix for Permutation Testing, X's are similarity measurements between melodies.

The input data to the test is the similarity matrix in the form shown in Table 4.14. The similarity measure used is not relevant, we could test any aspect of music for which we have a reliable similarity measure. We calculate the difference in the means with equation 4.1, where $Mean(AB)$ calculates the mean of the similarity measures between all the melodies in group A against all the melodies in group B.

We first calculate the *initial distribution T0* using the above formula for calculating means (with the correctly ordered labels), this sets the constant *distance baseline* between the two groups.

Now, if the null hypothesis is true, then the labeling of the groups as A or B is irrelevant and switching around the labels (the group classification of the melodies) will produce the same results. This is the principle behind the permutation test: *permuting the labels that assign the group membership of the individual melodies*. If the groups are not-different (a significantly successful model output) we would have the same randomly varying results no matter which melodies we call A and which we call B, and would not be able to tell the difference. Therefore, we are interested in randomly permuting the linkage between the melodies and the groups.

We can achieve this by either permuting the labels of the melodies or those of the group, doing both is redundant. This provides us with the *reference distribution*. By iteratively permuting the labels, comparing the new mean distance with the *T0* baseline, and counting the occurrences of difference, we arrive at a p value with function 4.2. The *alpha* or significance level is set to 0.05, this is the probability of making a type I error (false positive).

To clarify, if we consistently arrive at the result that they are different, we can imply that the initial ordering of these melodies has a certain characteristic other than random. Said differently, the hypothesis that is being tested is whether this labeling of melodies in the two groups is *relevant*. Therefore if the labeling is irrelevant, we will arrive at similar, randomly varying results no matter what labeling is used. But, if this original labeling is not random and actually distinguishes characteristics from the groups, we establish this difference with statistical significance, by measuring the likeliness of this happening due to group assignment.

$$S = (Mean(AA) + Mean(BB))/2 - Mean(AB) \tag{4.1}$$

The test statistic is the value of S in Equation 4.1 when the groups are ordered $S(AB)$, randomly permuted groups are denoted as $S(\widetilde{AB})$. We obtain a *p* value by calculating the fraction of permuted test

statistics that lie beyond the initial ordered statistic, the Test statistic T_0. N is the number of permutations and C is a function that counts the positive instances of $S(\widetilde{AB}) \geq T_0$:

$$p = C[S(\widetilde{AB}) \geq T_0]/N \tag{4.2}$$

By running the re-sampling of permutations a large number of times and arrive at a more confident p value. Also to note: the larger the matrix is used in the test, the subtler the differences that are required to distinguish the two groups.

Results

As a baseline, we have run the test using 'opti3' on two subgroups from the group of 100 melodies from the folk corpus (two groups from the corpus and not from generated melodies). Organized in two ways

1. First, we repeatedly tested (with 100,000 iterations each) a random selection of the melodies in two subgroups. In this way we have consistently obtained values of $p > 0.05$ (~ 0.95). Therefore the two randomly selected groups are not significantly different.

2. We then ran the test with the corpus separated in half, but preserved the order in which they are organized (the melodies are numerically indexed). This way we have obtained a value of $p = 0.0201$: indicating that the two groups selected are significantly different.

These tests clearly indicate that the ordering of the melodies in the folk corpus is not random. Effectively, we see in the description by the corpus creators [26] that they are ordered and classified by, among other things, sub-collections related to melody origin location and year. Lastly as a proof of concept, and as a way to test the consistency of style within the corpus, we ran the test with all the 100 melodies against a second identical group of the same 100 melodies. This time obtaining $p = 0.473$. This shows us that a larger selection of the corpus encompasses a wider space of the style, since the p value is lower than the one obtained by comparing 50 melodies against 50.

We have seen that this test can be used in two ways:

1. Binary fashion: if $p < 0.05 \rightarrow$ *fail*, if $p < 0.05 \rightarrow$ *pass*. Example: With 100,000 permutations, a 3475 count means $100,000/3475 : p = .035$, therefore the dissimilarity of the groups is great enough for it to be statistically significant.

2. The p value is determined by a ratio that can be interpreted as a similarity measure (the higher the number, the higher the similarity). Therefore, the p value, in a range $0 < p < 1$ indicates the similarity of the groups in a continuum. with 1 being very similar and 0 being very different.

Using similarity measure *opti3*, no model that we have tested has passed this test, that is, obtaining a value of $p > 0.05$. In fact, every algorithm output received a value of $p = 0.000$ when performing 100,00 iterations in the permutation test. As a reminder, we define *passing* the test as not being able to reject the null hypothesis H_0, i. e. obtaining values of $p < 0.05$.

In this section we have seen that permutation testing is a powerful way to evaluate the success of style imitation algorithms when trained on a corpus.

4.2.8 Fixed Point

In this final test we consecutively train the algorithms on their own output, generation by generation, without preserving the history of previous learning. This way we intend to observe how the algorithms diverge by testing the 'distance' or similarity to the original corpus.

We assume that models that incorporate greater biases in their output will become evident as they more quickly separate from the training set. Thus, we expect to see the most successful models, i. e. the ones that introduce less bias, to remain closer to the corpus as generations pass.

Hypothetically, it is possible that models that are inherently creative, rather than biased, will diverge more quickly as novelty is introduced after each generation, thus rating a positive outcome in a negative way. So we see this test as an evaluation of a specific point of view: how consistent is a model in replicating the style without introducing any novelty or bias that is not inherent to that style.

Figure 4.16 shows the result of a Markov model (blue), MusiCOG (orange) and Factor Oracle (grey) after 20 generations. The corpus, as seen in the intra-group similarity analysis, has a value of .201. The aggregated similarity measure is calculated as a mean of similarities between all melodies of the output and the corpus using *Opti3*. We clearly see that the Markov Model preserves the highest similarity mean across generations (according to this measure). The investigation which would explain these results is outside of the scope of this work and is left for future work. Nevertheless, we present the hypothesis that Markov, being a Statistical model, represents more closely the mechanism through which the similarity measure Opti3 operates. Thus the distribution observed by Opti3 is more accurately represented by a statistical approach such as Markov.

In Figures 4.17- 4.19 we see 3D plots with Originality, Complexity and Entropy measures for the three algorithms. The red group is the corpus, the green group is the 1st iteration of generation and the blue group is the 20th. We can observe that both MM and FO have an overall increase in complexity and originality, whereas MC relatively preserves a more stable overlap in the area with the corpus (with a shift in the same direction). We also noticed that for all algorithms, entropy increased slightly through generations (not as noticeable from the plot POV).

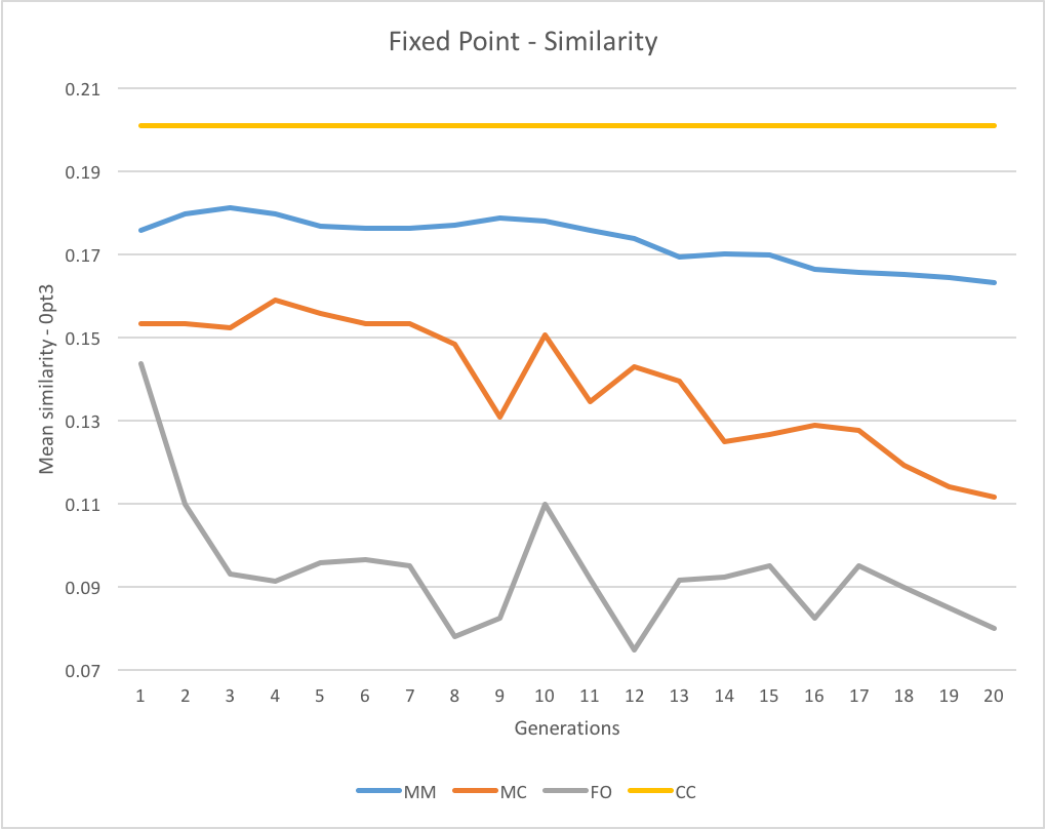

Figure 4.16: Fixed point with mean similarity using Opt3. Blue is the Markov Model, Orange is MusiCOG and Grey is the Factor Oracle. The corpus, as seen in the intra-group similarity analysis, has a value of 0.201.

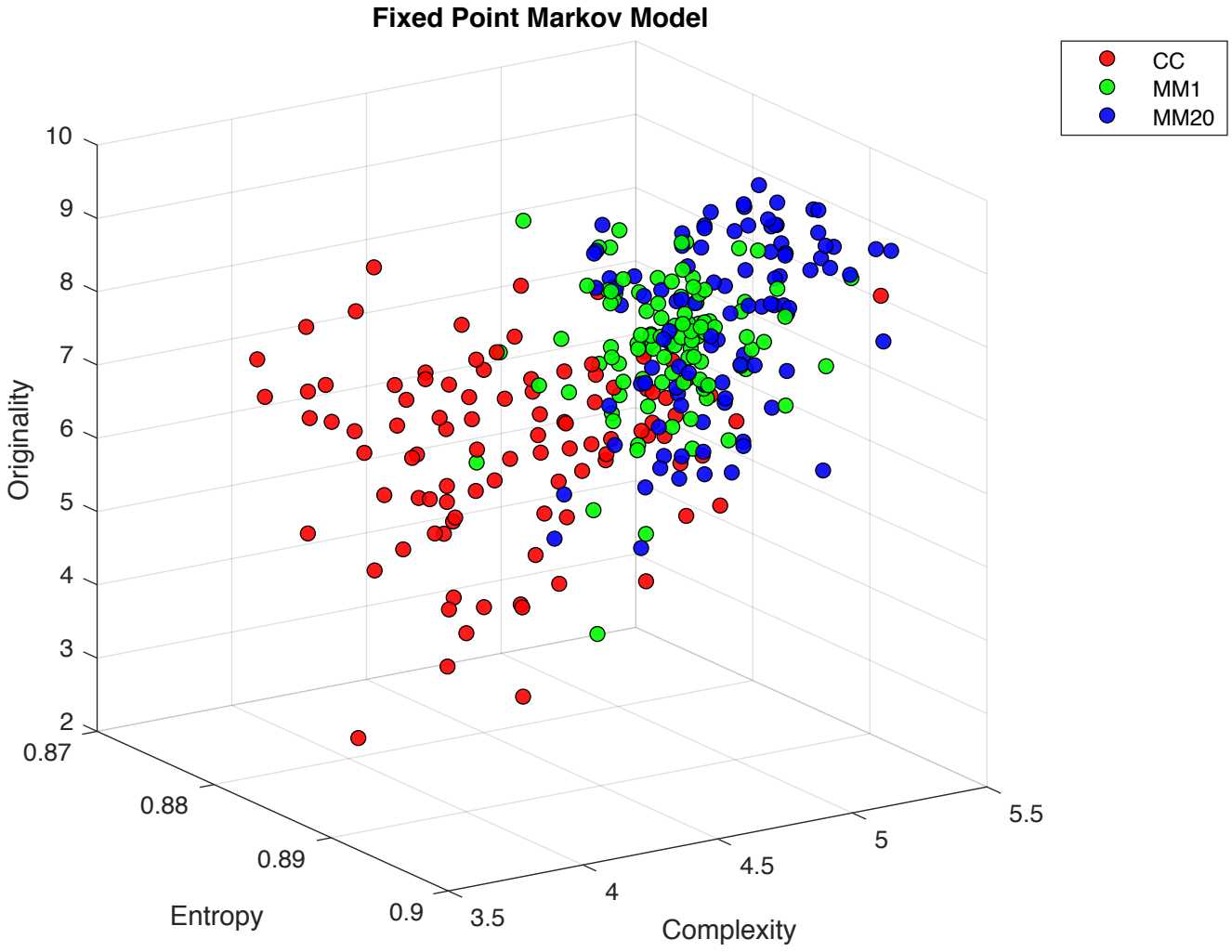

Figure 4.17: Fixed point 3D plot for Markov Model showing dimensions for Originality, Complexity and Entropy measures. The red group is the corpus, the green group is the 1st iteration of generation and the blue group is the 20th.

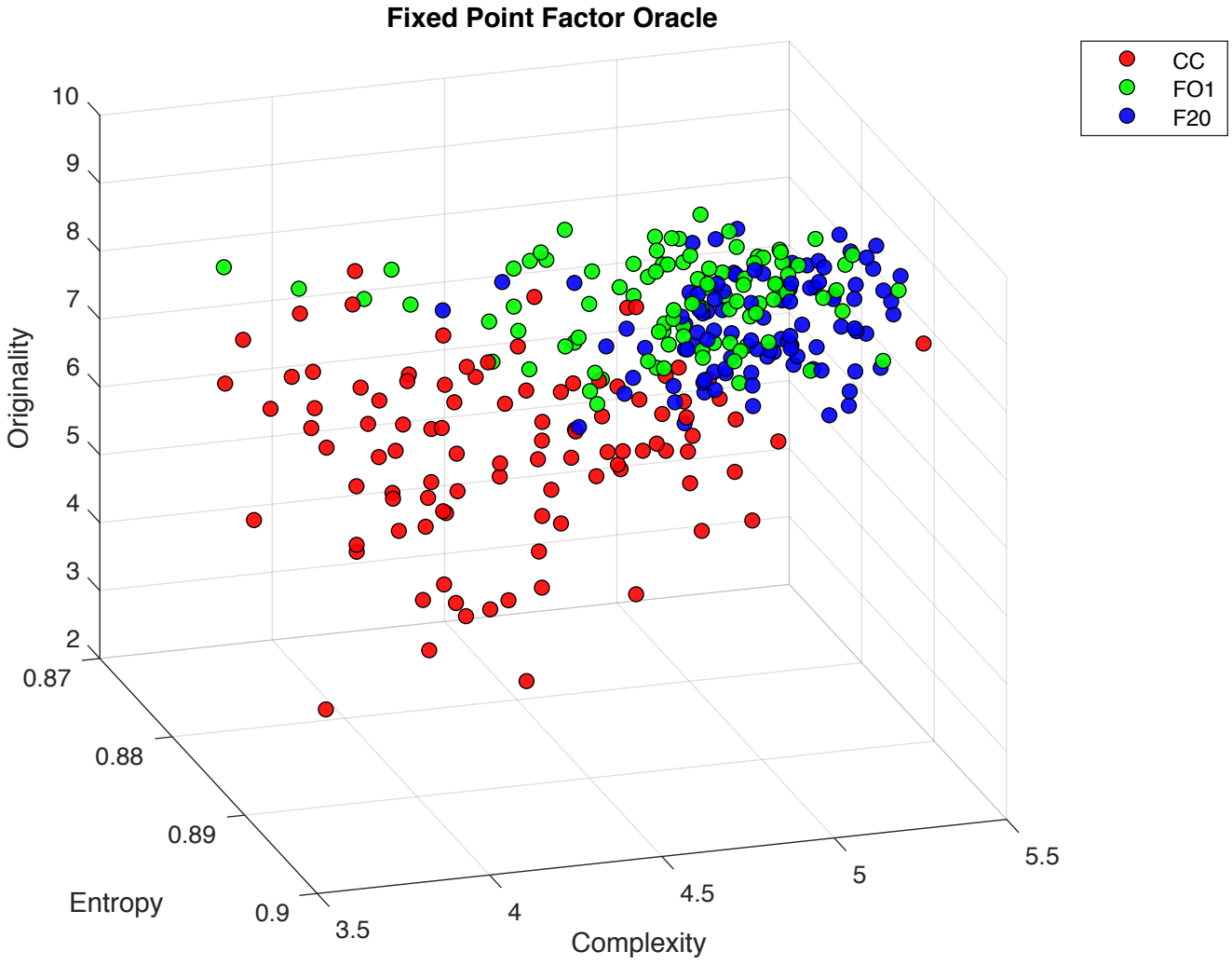

Figure 4.18: Fixed point 3D plot for Factor Oracle showing dimensions for Originality, Complexity and Entropy measures. The red group is the corpus, the green group is the 1st iteration of generation and the blue group is the 20th.

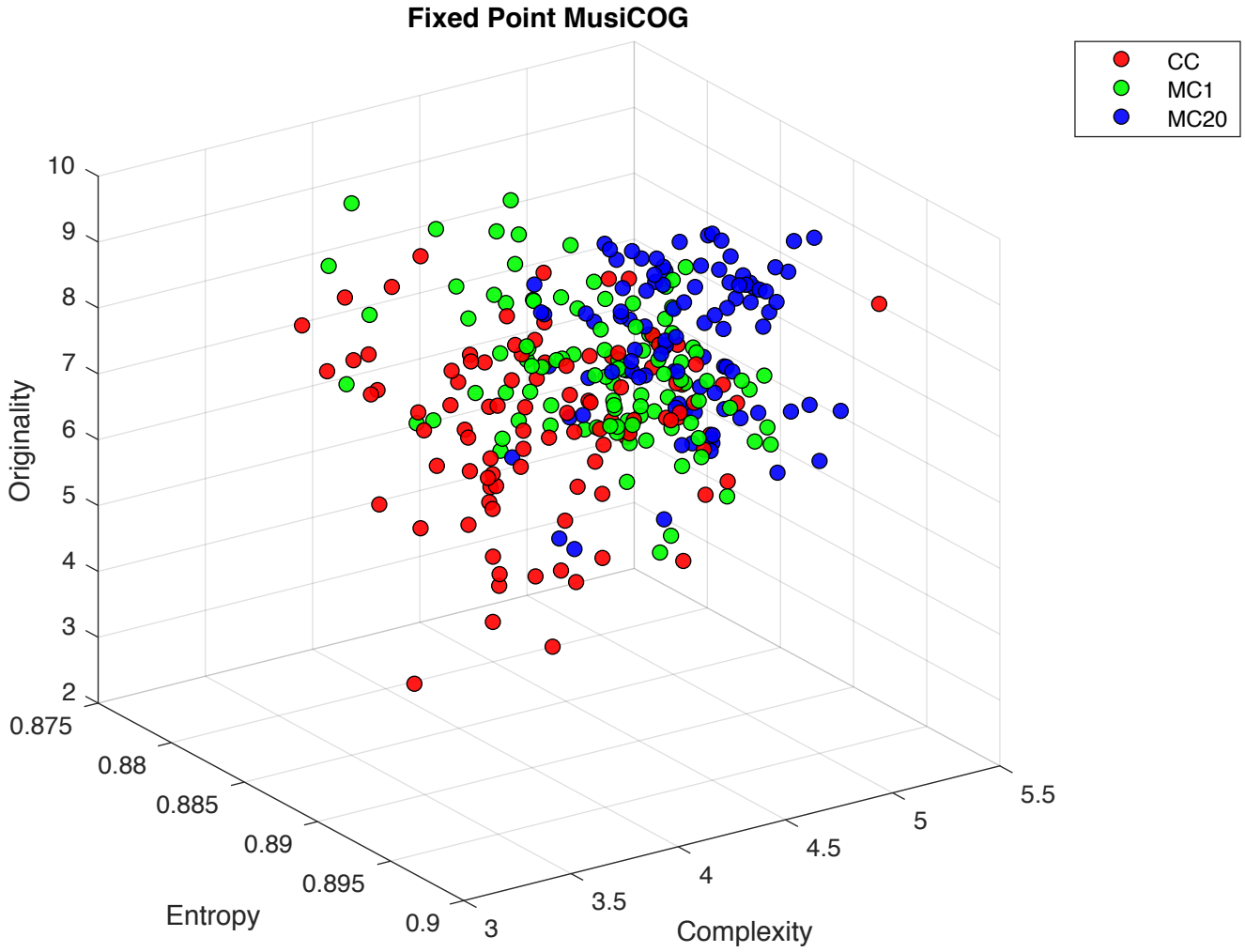

Figure 4.19: Fixed point 3D plot for MusiCOG showing dimensions for Originality, Complexity and Entropy measures. The red group is the corpus, the green group is the 1st iteration of generation and the blue group is the 20th.

Chapter 5

Conclusion

5.1 Conclusion

While this methodology is still in its infancy, it has demonstrated a capacity to perform reasonably reliable evaluations of *style imitation* algorithms when groups of melodies are generated from a corpus. To provide an exhaustive set of results for all tests applied to all algorithms and corpora would greatly exceed the scope of this thesis. Different problems would require tests to be tailored for these specific situations. We introduce a methodology that could be easily extended to other tasks in musical computational creativity and possibly beyond the task of music generation. We have shown a range of the most useful tests and for which problems they could be applied. The greatest contribution of this work is not the set of results to these particular tests, but rather the methodology of inquiry into the computational evaluation of generative algorithms.

Returning to our broad definition of stylistic imitation in the context of Exploratory Computational Creativity, we expect successful algorithms to roughly cover the same space as the corpus. One of the CMDS diagrams (Figure 4.7) shows graphically that this is not occurring in our study. All the results from the permutation tests show that the bar set by this methodology is high. In general, CMDS diagrams based on selected similarity measures could provide very valuable and intuitive insight into the output of models compared to a corpus. Also, the decision trees inform us, in more specific musical terms, which are the most important features that make up for these differences.

In the task of investigating for significance in the differences of the output and validating closeness to the corpus, we have shown the value the Permutation test. These are ideal for this type of study and also an extremely demanding benchmark to pass for an algorithm. The introductory study in fixed point testing shows that this method also provides a way to analyze how an algorithm learns and develops the representation of musical learning over time, thus providing an insight into algorithms over several interactions.

We have found several ways in which this research benefits the study of computation creativity and potentially systems for computer assisted creativity. For example this was the case where:

- Future systems could incorporate these methodologies to learn and improve their output without human intervention, thus incorporating a form of unsupervised learning;

- The development of new algorithms could incorporate some of the practices described in this work to understand more completely and precisely their performance. For example this was the case in the research on MusiCOG by Dr. James B. Maxwell, where a result in the decision tree analysis pointed to a problem in the function that determined note length;
- The use of computational methods for evaluation algorithm output could make this research less costly by reducing the time and expenses usually required by a user study involving human listeners;
- The permutation test could be used as a final pass or 'Discrimination Test' for *style imitation*

As a final note, this work clearly shows that the problem of style imitation warrants further research.

5.2 Future Work

We leave for future work the task of automating the methodology and the application to polyphonic music and algorithms that include harmonic knowledge. This most important dimension in learning and generation of melodies was not considered, therefore we expected the style imitation of melodies from training on isolated melodies alone to have a significant limitation. For example the wandering of tonality/modes in the output melodies. That said, since all algorithms used lack an explicit model for learning the harmonic dimension, it was fair to evaluate and compare them side by side in this study. Our interest and focus was primarily testing the validity of this methodology by evaluating these algorithms, and secondarily the overall quality of the algorithms used as example.

This work needs to be expanded to include algorithms with polyphonic learning, as well as an in-depth analysis of the output of the algorithms when trained on different corpora, and an evaluation of the behaviour of the algorithms when combining stylistically diverse corpora (combinatorial creativity).

Also, in the fixed point method we could test two variations:

1. Run a variation where we preserve the learned material from all generations in the model.
2. Run the same fixed point test but using the p value from permutation tests as the measure of similarity.

Other directions for exploration are:

1. More permutation tests can be carried out to investigate the effects of other similarity measures. Also validating whether these findings are audible/perceptible or just mathematical?
2. Eventually this work could lead to determining possible applications of the models based on the characteristics discovered. For example: which musical tasks are better or worst suited for each model?
3. This methodology could be used in an evolutionary system as a fitness function, where also negative generations can be rated positively as a way to explore the space.
4. In order to increase precision in the evaluation, it would be useful to run an outlier identification and removal process for the corpora, as a way to avoid extreme features in the learning material.

Bibliography

[1] K. Agres, J. Forth, and G. A. Wiggins. Evaluation of musical creativity and musical metacreation systems. *ACM Computers in Entertainment*, 2016/12 In Press.

[2] C. Allauzen, M. Crochemore, and M. Raffinot. Factor oracle: A new structure for pattern matching. *International Conference on Current Trends in Theory and Practice of Computer Science 1999: Theory and Practice of Informatics*, page 758, 2009.

[3] T. M. Amabile. *Creativity in Context*. Westview Press, Boulder, Colorado, 1996.

[4] S. Argamon, S. Dubnov, and K. Burns. *The Structure of Style: Algorithmic Approaches to Understanding Manner and Meaning*. Springer-Verlag, Berlin, Heidelberg, 2010.

[5] C. Ariza. The interrogator as critic: The turing test and the evaluation of generative music systems. *Computer Music Journal*, 33(2):48–70, June 2009.

[6] G. Assayag and S. Dubnov. Using factor oracles for machine improvisation. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 8(9):604–610, September 2004.

[7] G. Assayag, S. Dubnov, and O. Delerue. Guessing the composer's mind: Applying universal prediction to musical style. In *In Proceedings of the 1999 International Computer Music Conference, Beijing, China*, 1999.

[8] R. Begleiter, R. El-yaniv, and G. Yona. On prediction using variable order markov models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004.

[9] A. Berkowitz. *The Improvising Mind: Cognition and Creativity in the Musical Moment*. Oxford University Press, 2010.

[10] M. A. Boden. *The Creative Mind - Myths and Mechanisms (2. ed.)*. Routledge, 2003.

[11] M. A. Boden. Computer models of creativity. *AI Magazine*, 30(3):23, 2009.

[12] G. E. P. Box, G. Jenkins, and G. C. Reinsel. Time series analysis: Forecasting and control. *John Wiley and Sons, Hoboken, N. J., 4th edition.*, 2007.

[13] C. Brower. A cognitive theory of musical meaning. *Journal of Music Theory*, 44(2):323–379, 2000.

[14] N. Collins. *Towards Autonomous Agents for Live Computer Music: Realtime Machine Listening and Interactive Music Systems*. PhD thesis, Centre for Science and Music, Faculty of Music, University of Cambridge, 2007.

[15] S. Colton and G. A. Wiggins. Computational creativity: The final frontier? *In Proceedings of 20th European Conference on Artificial Intelligence (ECAI) (Frontiers in Artificial Intelligence and Applications). Montpellier, FR.*, 242:21–26, 2012.

[16] D. Conklin. Music generation from statistical models. In *Proceedings Of The Artificial Intelligence and Simulation of Intelligence (AISB) 2003. Symposium On Artificial Intelligence And Creativity In The Arts And Sciences*, pages 30–35, 2003.

[17] D. Conklin and I. H. Witten. Multiple viewpoint systems for music prediction. *Journal of New Music Research*, 24:51–73, 1995.

[18] A. Cont. *Modeling Musical Anticipation: From the time of music to the music of time*. Doctoral dissertation, University of Paris 6 (UPMC), and University of California San Diego (UCSD), Paris, 2008.

[19] A. Cont, S. Dubnov, and G. Assayag. A framework for anticipatory machine improvisation and style imitation. In *Anticipatory Behavior in Adaptive Learning Systems (ABiALS)*. Rome, Italy, September 2006.

[20] D. Cope. The composer's underscoring environment: Cue. *Computer Music Journal*, 21(3):20–37, 1997.

[21] D. Cope. *Virtual Music: Computer Synthesis of Musical Style*. The MIT Press, 2001.

[22] D. Cope. *Computer Models of Musical Creativity*. The MIT Press, 2005.

[23] S. Dubnov and G. Assayag. Universal prediction applied to stylistic music generation. In *Mathematics and Music, A Diderot Mathematical Forum*, pages 147–160, 2002.

[24] M. Edwards. Algorithmic composition: computational thinking in music. *Communnications of the Association for Computing Machinery (ACM)*, 54(7):58–67, July 2011.

[25] T. Eerola and A. C. North. Expectancy-based model of melodic complexity. *In Woods, C., Luck, G.B., Brochard, R., O'Neill, S. A., and Sloboda, J. A. (Eds.) Proceedings of the Sixth International Conference on Music Perception and Cognition. Keele, Staffordshire, UK: Department of Psychology*, (22):31–43, 2000.

[26] T. Eerola and P. Toiviainen. *MIDI Toolbox: MATLAB Tools for Music Research*. University of Jyväskylä, Jyväskylä, Finland, 2004.

[27] T. Eerola and P. Toiviainen. Suomen kansan esavelmat. suomalaisten kansansavelmien elektroninen tietovaranto [digital archive of finnish folk tunes]. 2004.

[28] A. Eigenfeldt. A composer's search for creativity within computational style modeling. *In Proceedings of the 21st International Symposium on Electronic Art*, 2015.

[29] A. Eigenfeldt. Generative music for live musicians: An unnatural selection. *Proceedings of the Sixth International Conference on Computational Creativity*, Jun 2015.

[30] A. Eigenfeldt, A. Burnett, and P. Pasquier. Evaluating musical metacreation in a live performance context. *Proceedings of the Third International Conference on Computational Creativity*, pages 140–144, may 2012.

[31] R. Finke, S. Smith, and T. Ward. *Creative Cognition Theory, Research, and Application*. The MIT Press, England, 1996.

[32] K. Frieler and D. Müllensiefen. The simile algorithm for melodic similarity. *Proceedings of the Annual Music Information Retrieval Evaluation exchange*, 2006.

[33] P. Gärdenfors. *Conceptual Spaces, The Geometry of Thought*. The MIT Press, 2000.

[34] N. Gonzalez Thomas. META-MELO. Online resource available at: http://metacreation.net/meta-melo/home.html.

[35] N. Gonzalez Thomas, P. Pasquier, A. Eigenfeldt, and J. B. Maxwell. A methodology for the comparison of melodic generation models using meta-melo. In *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR 2013)*, November 4-8 2013.

[36] P. I. Good. *Permutation, Parametric and Bootstrap Tests of Hypotheses*. Springer Series in Statistics, 2005.

[37] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *Association for Computing Machinery's (ACM) Special Interest Group Explorers Newsletter*, 11(1):10–18, November 2009.

[38] M. Hamanaka, K. Hirata, and S. Tojo. Implementing 'a generative theory of tonal music'. *Journal of New Music Research*, 35(4):249–277, 2006.

[39] O. Lartillot, S. Dubnov, G. Assayag, and G. Bejerano. Automatic modelling of musical style. In *Proceedings of the 2001 International Computer Music Conference*, pages 447–454. San Francisco: ICMA, 2001.

[40] P. Legendre and L. Legendre. *Statistical testing by permutation*. Elsevier Science BV, Amsterdam, 1998.

[41] F. Lerdahl and R. S. Jackendoff. *A Generative Theory of Tonal Music*. The MIT Press, 1996.

[42] A. Linson, C. Dobbyn, and R. Laney. Critical issues in evaluating freely improvising interactive music systems. *Proceedings of the Third International Conference on Computational Creativity (ICCC 2012)*, pages 145–149, May 2012.

[43] A. Lovelace and L. Menabrea. Sketch of the analytical engine invented by charles babbage esq. *Scientific Memoirs. Note A*, page 694, 1842.

[44] G. Loy. *Musimathics: the mathematical foundations of music. Vol. II.* Cambridge, MA, The MIT Press, 2007.

[45] B. Manaris, P. Roos, P. Machado, D. Krehbiel, L. Pellicoro, and J. Romero. A corpus-based hybrid approach to music analysis and composition. In *Proceedings of Twenty-Second Conference on Artificial Intelligence (AAAI 2007)*, volume 22, page 839, Jul 2007.

[46] E. W. Marvin and P. A. Laprade. Relating music contours: Extensions of a theory for contour. *Journal of Music Theory*, (22), 2000.

[47] J. B. Maxwell. *Generative Music, Cognitive Modelling, and Computer-Assisted Composition in MusiCOG and Manuscore*. PhD thesis, Simon Fraser University, 2014. Simon Fraser University, Vancouver, 2014.

[48] J. B. Maxwell, A. Eigenfeldt, and P. Pasquier. Manuscore: Music notation-based computer assisted composition. In *In proceeding of the 2012 International Computer Music Conference, At Ljubljana, Slovenia*, 2012.

[49] J. B. Maxwell, A. Eigenfeldt, P. Pasquier, and N. Gonzalez Thomas. Musicog: A cognitive architecture for music learning and generation. *Proceedings of the 9th Sound and Music Computing conference (SMC 2012)*, pages 521–528, jul 2012.

[50] C. Mckay and I. Fujinaga. jsymbolic: A feature extractor for midi files. In *International Computer Music Conference*, pages 302–305, 2006.

[51] D. Moffat and M. Kelly. An investigation into people's bias against computational creativity in music composition. *Proceedings of the third joint workshop on Computational Creativity (as part of ECAI 2006), Rivia del Garda, Italy*, 2006.

[52] D. Müllensiefen. Fantastic: Feature analysis technology accessing statistics (in a corpus) (technical report v. 1.5). *London: Goldsmiths, University of London. Retrieved from:*, pages 140–144, jun 2009.

[53] D. Müllensiefen and K. Frieler. Cognitive Adequacy in the Measurement of Melodic Similarity: Algorithmic vs. Human Judgments. *Computing in Musicology*, 13(2003):147–176, 2004.

[54] D. Müllensiefen and K. Frieler. Melodic similarity: Approaches and applications. In *Proceedings of the 8th International Conference on Music Perception and Cognition*, 2004.

[55] D. Müllensiefen and K. Frieler. Optimizing measures of melodic similarity for the exploration of a large folk-song database. *Proceedings of the 5th International Conference on Music Information Retrieval. Barcelona: Universitat Pompeu Fabra*, pages 274–280, 2004.

[56] G. Nierhaus. *Algorithmic Composition: Paradigms of Automated Music Generation*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[57] F. Pachet. The continuator: Musical interaction with style. *Journal of New Music Research*, 32(3):333–341, 2003.

[58] F. Pachet. Musical virtuosity and creativity. In *Computers and Creativity. J. McCormack and M. d'Inverno (Eds.)*, pages 115–146. Springer Berlin Heidelberg, 2012.

[59] R. Parncutt. A perceptual model of pulse and salience and metrical accent in musical rhythms. *Music Perception*, 11(4):409–464, 1994.

[60] P. Pasquier, A. Burnett, N. Gonzalez Thomas, J. B. Maxwell, A. Eigenfeldt, and T. Loughin. Investigating listener bias against musical metacreativity. *Proceedings of the Seventh International Conference on Computational Creativity*, jun 2016.

[61] M. Pearce, D. Conklin, and G. A. Wiggins. Methods for combining statistical models of music. In *Proceedings of the Second international conference on Computer Music Modeling and Retrieval*, Computer Music Modeling and Retrieval 2004, pages 295–312, Berlin, Heidelberg, 2005. Springer-Verlag.

[62] M. Pearce, D. Meredith, and G. A. Wiggins. Motivations and methodologies for automation of the compositional process. *Musicae Scientiae*, 6:200–2, 2002.

[63] M. Pearce and G. A. Wiggins. Towards a framework for the evaluation of machine compositions. In *Proceedings of the Artificial Intelligence and the Simulation of Behaviour 2001, Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*, pages 22–32, 2001.

[64] M. Pearce and G. A. Wiggins. Improved methods for statistical modeling of monophonic music. *Journal of New Music Research*, 33(4):367–385, 2004.

[65] M. Pearce and G. A. Wiggins. Evaluating cognitive models of musical composition. In *Proceedings of the 4th International Joint Workshop on Computational Creativity. A. Cardoso and G. A. Wiggins (Eds.), London: Goldsmiths, University of London*, pages 73–80. Citeseer, 2007.

[66] D. Rizo, P. J. Ponce de León, C. Pérez-Sancho, A. Pertusa, and J. M. Iñesta. A pattern recognition approach for melody track selection in midi files. In *Proceedings of the 7th International Symposium on Music Information Retrieval ISMIR 2006*, pages 61–66, Victoria, Canada, 2006.

[67] H. Schaffrath. The essen folksong collection in kern format. *Menlo Park, CA: Center for Computer Assisted Research in the Humanities*, 1995.

[68] D. K. Simonton. Melodic structure and note transition probabilities: A content analysis of 15,618 classical themes. *Psychology of Music*, (12):3–16, 1984.

[69] D. K. Simonton. Computer content analysis of melodic structure: Classical composers and their compositions. *Psychology of Music*, (22):31–43, 1994.

[70] H. Taube. *Notes From The Metalevel: An Introduction To Algorithmic Music Composition*. Studies on New Music Research Series. CRC PressINC, 2004.

[71] P. Toiviainen and T. Eerola. A method for comparative analysis of folk music based on musical feature extraction and neural networks. *Proceedings of the VII International Symposium on Systematic and Comparative Musicology and III International Conference on Cognitive Musicology, University of Jyväskylä, Finland.*, 2001.

[72] J. L. Triviño-Rodriguez and R. Morales-Bueno. Using multiattribute prediction suffix graphs to predict and generate music. *Computer Music Journal*, 25(3):62–79, September 2001.

[73] F. Wiering and E. Benetos. Digital musicology and mir: Papers, projects and challenges. In *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR)*, 2013.

[74] G. A. Wiggins. Models of musical similarity. *Musicae Scientiae*, 11:315–338, Mar 2007.

[75] G. A. Wiggins, M. T. Pearce, and D. Müllensiefen. Computational modelling of music cognition and musical creativity. In *The Oxford Handbook of Computer Music. Roger T. Dean (Ed.)*, pages 383–420. Oxford University Press, Oxford, 2009.

[76] D. Zicarelli. M and jam factory. *Computer Music Journal*, 11(4):13–29, 1987.