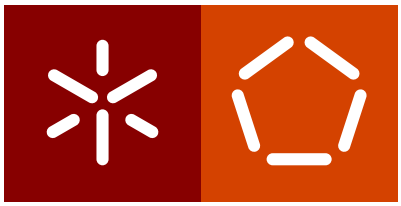


**Universidade do Minho**

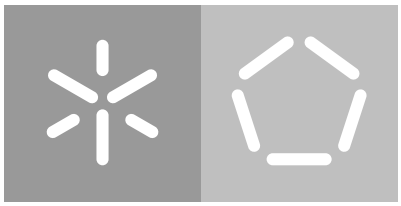
Escola de Engenharia

Departamento de Informática

Miguel Angelo Ferreira Dias Ribeiro

# Real-time MIDI device data analysis and context aware music generation

December 2017



**Universidade do Minho**
Escola de Engenharia
Departamento de Informática

Miguel Angelo Ferreira Dias Ribeiro

# Real-time MIDI device data analysis and context aware music generation

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
**Professor Cesar Analide**
**Professor Paulo Novais**

December 2017

# ACKNOWLEDGEMENTS

I would like to express my deep gratitude to Professor Cesar Analide and Professor Paulo Novais, my supervisors, for their guidance and enthusiastic encouragement during this dissertation and for giving me the freedom to explore a subject that I wanted very much to work on. I would also like to thank my friends for the relentless support and encouragement throughout this journey. My grateful thanks are also extended to Ms Helena Dias for her support and patience with my inexperience with bureaucracy. I would also like to extend my thanks to the technicians of the department of informatics for their help in offering resources and finding a place to quietly work. Finally, I wish to thank my parents and family for their support and specially to my nephew for his important contribution.

## ABSTRACT

Computer music generation is an active research field encompassing a wide range of approaches. As we involve more and more technology in our creative endeavors, it becomes vital that we provide our systems with the capability to understand and represent art concepts internally, as well as understand and predict artistic intent and emotion. In respect to music, being able to work with this information opens up fantastic possibilities for artist-machine synergetic collaborations as well as machine creativity endeavors. In this spirit, this dissertation explores a system capable of analyzing in real-time a performance piece played on a MIDI (Musical Instrument Digital Interface) capable instrument and produce a musically coherent piece of accompaniment music. This system comprises of two major subsystems: one responsible for analyzing and extracting features from the live performance and one that takes these features and generates MIDI tracks to be played in conjunction with the live audio, in a way that blends in with the performance.

# RESUMO

A geração algorítmica de música é um campo de investigação vasto e com múltiplas abordagens. À medida que incorporamos mais tecnologia nos nossos processos criativos, torna-se fundamental equipar os nossos sistemas com as capacidades para entender e representar arte, bem como analisar e prever intenção artística e emoções. No que diz respeito a música, ter este tipo de informação disponível abre possibilidades para colaborações entre artista e máquina, bem como sistemas capazes de exibir capacidades criativas. Esta dissertação propõe um sistema capaz de analisar uma performance musical num dispositivo controlador MIDI (Musical Instrument Digital Interface) em tempo real e produzir acompanhamento adequado e coerente. Este sistema é composto por dois subsistemas: um responsável por analisar e extrair características musicais de uma performance, e outro que a partir destas características seja capaz de gerar trechos MIDI de forma a se complementarem musicalmente.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ACRONYMS

**A**

**ANN** Artificial Neural Network.

**D**

**DAF** Delayed Auditory Feedback.

**G**

**GUI** Graphical User Interface.

**I**

**I/O** Input/Output.

**M**

**MIDI** Musical Instrument Digital Interface.

**MLP** Multi-Layer Perceptron.

**P**

**PI** Projecto Integrado.

# TERMS

**B**

**BUNDLE** The Bundle class serves as a wrapper for a list of Packet instances with time informaton. A bundle usually processes either one, five or thirty Packet instances, corresponding to the three time windows explored in the system.

**C**

**CHORD** In music, a chord is any harmonic set of pitchs consisting of two or more notes that are sounding simultaneously. In tonal Western music, a chord is most frequently a triad, consisting of three distinct notes: the root note, and the intervals of a third and a fifth above the root note.

**COMMONS MATH** The Apache Commons Mathematics Library is a library of lightweight, self-contained mathematics and statistics components which was used in this project to efficiently calculate statistics on observed data points. Version used: 3.6.1. Official website with further information can be reached at: http://commons.apache.org/.

**D**

**DYNAMICS** In music, the dynamics of a piece is the variation in loudness between notes or phrases. Dynamics can be indicated by specific musical notation in music representation mediums as a way for the author to describe how to imprint the intended expression to a motif or phrase. Piece dynamics are a major component of musical affective expression and help musicians sustain variety and interest in a musical performance and to communicate a particular emotional state or feeling.

**E**

**EVENT** In the *Musical Instrument Digital Interface (MIDI)* protocol, an event is a MIDI message that controls some aspect of the receiving device. The most common events are those that flag the beginning or ending of a single note. These events further carry information such as velocity, pitch and a timestamp. Other control events carry other type of information.

**K**

KEY In music theory, the key of a piece is the group of pitches or scale that form the basis of a music composition. The group features a tonic note and its corresponding chords which provides a subjective sense of rest and resolution, and also has a unique relationship to the other pitches of the same group, their corresponding chords, and pitches and chords outside the group.

N

NEUROPH Neuroph is a lightweight Java neural network framework to develop common neural network architectures. Included neural network implementations were used in multiple modules of the system. Version used: 2.94. Official website with further information can be reached at: http://neuroph.sourceforge.net/.

P

PACKET The Packet class is a wrapper for a list of events that also pocesses time information. In general, a single Packet instance corresponds to a single cycle of gathered event information.

PITCH A note pitch describes a certain frequency relation in music, usually represented in alphabetic characters from A to G repeating over several octaves. Pitch is the musical property of a note that allows a listener to describe a note as higher or lower than another. In the MIDI protocol, pitch value ranges from 0 (lowest note: C in octave -1) to 127 (highest note: G in octave 9), although a common vertical or grand piano only has 88 keys ranging from the MIDI pitch value of 21 (an A in octave 0) up to 108 (a C in octave 8) plural.

V

VELOCITY Note velocity expresses how hard a key has been pressed. It can be also thought as the loudness of a note on default behaviour. Depending on the software playing back the audio, actual perceived volume will depend on both note velocity and software loudness settings. In the MIDI protocol, velocity values range from 0 (inaudible key press and often used to represent note-off for device performance reasons) up to 127 (hardest key press).

# 1

# INTRODUCTION

Computer music generation is an active research field encompassing a wide range of approaches. There are many uses for such a system, such as context aware ambience music generator for a game or other highly interactive medium, research on creativity (be it human or computational), generating mood fitting music to influence the emotional experience of a target audience, enhancement of human musical composition process, inspiring an artist in a human-machine collaboration, live performance improvisation or show, the list is endless. The topic is also highly interesting and has contributions from various fields of research from Psychology to Artificial Intelligence. The purpose of this dissertation is to explore various methods of music analysis and generation to build a system capable of being a partner in a solo performance. The system should be able to detect things such as artistic intent, emotional state and which compositional components the performance is currently lacking (such as melody, harmonization, rhythm, among others) and to complement it, hopefully being a good musical partner capable of assisting in a performance and inspire. Learning methods will be explored so the system can improve if it is performing badly and adapt to his partner characteristics.

## 1.1 MOTIVATION

Although vast work has been done, true understanding (or even formalization) of the creative process is still not a reality. To integrate technology in an art form such as music, it is required that a broad and accurate understanding of music and the psychological effects of music be present in our software. From that basis, we can build solutions that understand artists and the art itself and act on that information, be it to help the a human-machine collaborative process or to achieve machine creativity by understanding how humans do it. Either way, the possibilities are interesting and they all start with a need for the machine to be able to understand artistic intent. Such capabilities can be used as a tool to feed new information into a compositional system that can generate adequate music, hopefully complementing the artist original intent. The real-time immediacy allows for very fast interaction between artist and machine where they can influence each other. This specific

approach is not explored and can be very interesting to see how different techniques can be adapted to provide real-time results.

## 1.2 OBJECTIVES

This project has as a main goal the implementation of a computational system capable of analyzing music being played in a connected MIDI device in real-time, calculating features such as dynamics, pitch and rhythmic statistics and be able to infer musical evaluations such as piece emotional connotation and artistic intent. Another objective is to extend the previous system so it can, based on the retrieved information, generate in real-time a musical composition to accompany the original performance that fits the artist intent and is structurally and musically coherent.

## 1.3 DOCUMENT STRUCTURE

This dissertation is structured into six chapters, with the following content:

1. Introduction

   This first chapter lays down the context for this thesis, an introduction to musical analysis and generation. It also explains the motivations behind this project and exposes the main objectives and ends with a brief explanation of the dissertation structure.

2. State of the art

   The second chapter presents commonly used methodologies for musical analysis and algorithmic music generation, presenting examples of previous work in the area which used those techniques and how are they relevant to this project. It also contains a description and results of previous work done on this subject.

3. The Problem

   In this chapter, the problem is detailed. Expected challenges are discussed and some possible solutions are presented. A proposed methodological approach is presented with explanations for the chosen course of action.

4. Development

   The fourth chapter presents the development of the system, from taken decisions when considering the problem to the system implementation. Preliminary results of the built system are presented in the ending section.

5. Case Study

   In this chapter, a case study involving the prediction module is presented with the experimental setup and the corresponding result. The chapter ends with a discussion on the observed results.

6. Conclusion

   The last chapter presents the final results and considerations on the whole project and then possible improvements to the system as well as new approaches that might yield good results.

# 2

# STATE OF THE ART

## 2.1 MUSICAL ANALYSIS

Be it with symbolic [1], audio signal input [2] or both [3], musical analysis is commonly achieved by using feature extraction techniques, which reduce large volumes of data to, hopefully, the most important information contained in them. These features can be anything deemed important for the specific application and should, if broad and extensive enough, provide a complete description of the main characteristics of the input and usually include statistical analysis on information such as note pitch, duration, velocity and higher level musical abstractions such as piece key and mode.

A piece musical features need to be sampled multiple times and with different data windows to capture music evolution, especially on more complex pieces where key and rhythmic changes occur often. While the features by themselves provide some important information about the input piece, they can be fed to structures such as classifiers to infer subtler information, structure and relationship between the features. Many abstract musical and psychological concepts can only be captured in a piece by relating multiple distinct features, such as player mood, emotional state, music rhythmic information and genre.

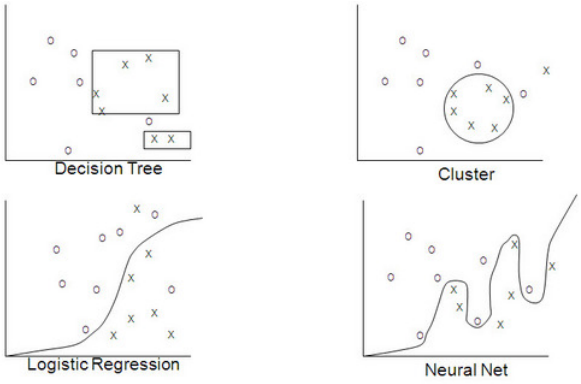



Figure 1.: Different methods of classification

Extracted features by themselves are rarely useful, so most examples use extracted features as a step to achieve higher abstraction analysis, such as classification of music genre [4, 5] or affective content [6, 7]. These types of classifications can be useful for automatically organizing a music library by detecting music genre or assessing the emotional charge of a piece or emotional state of an artist.

## 2.2 ALGORITHMIC COMPOSITION

Over the years, multiple different techniques have been used to build music generation systems. However, the following five categories of techniques will be reviewed as they are the most relevant to this thesis [8]:

- Grammars;
- Knowledge-based Systems;
- Markov Chains;
- Artificial Neural Networks;
- Evolutionary Methods;

The most successful system will likely combine several of these techniques. Different methodologies and hybrid combinations will suit better for different aspects of musical generation such as melody, harmony, rhythm and bass sequence generation. It is, therefore, important to be able to analyze the input performance to determine in which area it is better for the system to focus and which techniques to use.

### 2.2.1 *Formal Grammar*

A formal grammar can be defined as a set of rules to expand high-level symbols into more detailed sequences of symbols, known as words. Words are generated by repeatedly applying these rules which need to be valid according to the language syntax. A grammar does not describe the meaning of these sequences, only their form. Due to their behavior, grammars are suited to represent systems with hierarchical structure due to the recursive nature of rule application. As most styles of music show hierarchical structures, formal grammar theory has been frequently applied to analyze and compose music.

To drive the generative process, a set of rules for the grammar must be defined. These rules are usually multi-layered for different phases of the compositional process: from the general themes of the composition to the arrangement of individual notes. Rules can be defined by hand from principles of music theory or can be automatically inferred by

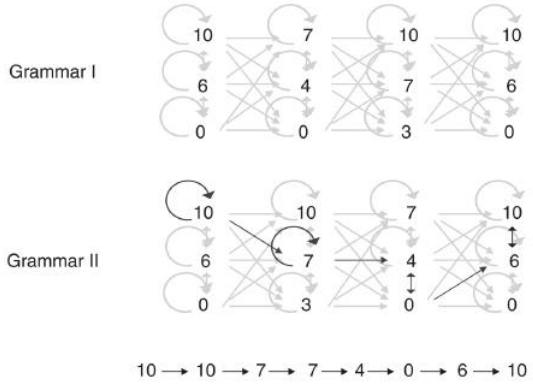



Figure 2.: Visualization of two different grammars for melody generation

analyzing existing compositions, which make them fit to generate music in that specific style. It is common to use activation probabilities for the rules to induce variety and thus improve the musical quality.

Examples of grammar usage in algorithmic composition include the synthetization of jazz solos [9, 10], production of real-time improvised drum rhythms accompaniment with grammatical inference [11] or even generation of classical voices [12].

### 2.2.2 *Knowledge-based Systems*

A knowledge-based system is any program that attempts to represent knowledge explicitly using tools such as rules and ontologies with structured symbols. Since music composition has been frequently described as formalized rules to manipulate music symbols, knowledge-based systems come up often as a way to implement algorithmic composition. These systems have strong foundations in AI as expert systems, and often require an expert musician to hand-code the rules into the system. Although the implemented knowledge is often static, parts can be dynamically changed. For example, Widmer [13] harmonizes melodies, extracting harmonization rules from analyzing a training set and Spangler [14] generates rule systems for harmonizing chorales in Bach style.

Knowledge-based systems can also be used with evolutionary algorithms by using a fitness function where compliance with the rules can be measured in a graduated scale instead of a binary response. As example, McIntyre [15] codified a set of functions based on harmonization rules extracted from classical pieces. The fitness function was built as a weighted sum of those function results.

### 2.2.3 *Markov Chains*

A Markov chain is stochastic a process that is characterized by having a set of discrete states and a set of probabilities to jump between states. Each possible state depends only on the current state (known as the Markov property), which means the process has no memory and does not depend on previous states. Markov chains are usually represented as weighted directed graphs, where the nodes represent the possible states, the edges represent possible transitions, and their weights represent the probability of that transition happening.

1st-order matrix

| Note | A | C♯ | E♭ |
|---|---|---|---|
| A | 0.1 | 0.6 | 0.3 |
| C♯ | 0.25 | 0.05 | 0.7 |
| E♭ | 0.7 | 0.3 | 0 |

2nd-order matrix

| Note | A | D | G |
|---|---|---|---|
| AA | 0.18 | 0.6 | 0.22 |
| AD | 0.5 | 0.5 | 0 |
| AG | 0.15 | 0.75 | 0.1 |
| DD | 0 | 0 | 1 |
| DA | 0.25 | 0 | 0.75 |
| DG | 0.9 | 0.1 | 0 |
| GG | 0.4 | 0.4 | 0.2 |
| GA | 0.5 | 0.25 | 0.25 |
| GD | 1 | 0 | 0 |

Figure 3.: First and second order probability matrix

In algorithmic music composition, the simplest symbolic model is to interpret each node as a possible note, and an algorithm produces output note values by consulting the crafted probability matrix. As per the Markov property, future states do not depend on past states, which make simple Markov chains generate results that tend to wander aimlessly. Therefore, higher order chains can be crafted, by making each node represent not just a single note, but a sequence, grouping particular notes together. These higher-order chains tend to generate results with more structure, sounding better and more planned. However, a balance is necessary as very high-order chains much more computationally expensive and tend to generate very predictable music similar to the segments used to craft the probability matrix [8].

Markov chains are often used to generate melodies, although most practical usages pair Markov chains with other methods, such as using Markov chains to generate patterns to be fed into a *Artificial Neural Network (ANN)* [16] or using them as a fitness function to evaluate melodies in an evolutionary algorithm [17]. More sophisticated Markovian models can be used, such as variable-order chains and multiple parallel chains which can be used to get the best results from lower and higher order chains [18]. Combination of Markovian models with constraint-based reasoning has also been proposed to generate better results [19].

### 2.2.4 *Artificial Neural Networks*

ANN are computational models consisting of large numbers of very simple computing units called neurons, linked to each other in layers. Neurons compute one output value with a simple function from a numeric input aggregated, usually, from the output of other neurons. These networks of neurons typically have one input layer, connected to the exterior to take in data, and one output layer with the results of the computation.

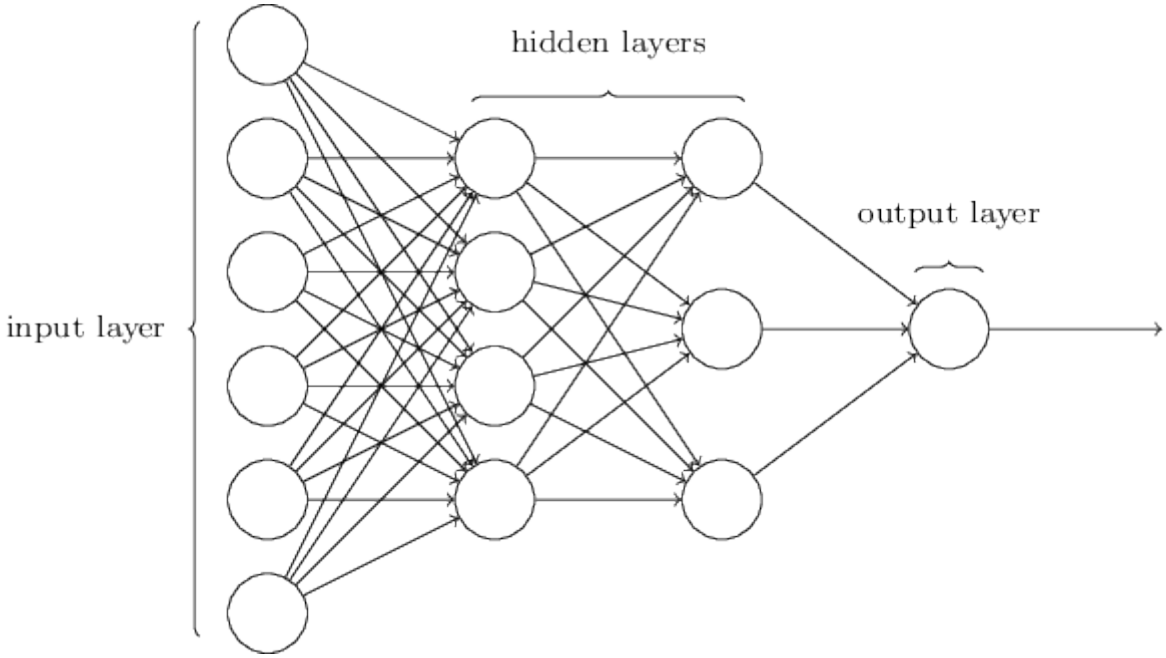



Figure 4.: Example of a simple ANN

An important step of using ANN with music, is how to feed the desired data into the network. There are multiple ways to do so, such as feeding temporal sequences, the full music or to extract features from the music and feed them to the network.

Most common use of neural network models requires previous training with a pre-existing compilation of music with something in common, such as being from the same genre or author. Such training limits their use to a specific style recognition, generation of similar sequences and harmonies, or to predict sequences of notes likely to follow a given piece of music of that genre or style. For example, ANN has been used to improvise jazz solos [20], melodies [21] and harmonies [22].

### 2.2.5 *Evolutionary Methods*

Evolutionary algorithms (often called genetic algorithms, although they are a subset of evolutionary algorithms) are a methodology inspired by biological evolution mechanisms to produce results to optimization problems. An evolutionary process generally goes as follows:

1. An initial population of valid solutions to the optimization problem is generated from random or crafted examples.

2. Candidate solutions are evaluated by a fitness function that describes qualitatively how good they are as a solution to the problem.

3. A set of parents is chosen, usually either the best fit individuals in the population or a weighted probabilistic choice favoring better scoring individuals.

4. A new population is generated by replacing badly fit individuals with new ones by performing crossover and mutation operations on the previous generation (using the set of parents and other individuals). It is a common practice to also maintain some of the best fit individuals from the previous generation unchanged.

5. Repeat these steps until a stop condition is reached, such as getting an individual with high enough fitness score or achieving a limit time or iteration steps.

6. Use the best fit individual as a solution to the problem.

As this iterative process tends to choose and manipulate the best scoring individuals, the overall population fitness tends to increase. Mutation operators are usually context dependent functions that take an individual and randomly change some aspect of it (while maintaining the validity of the individual as a possible solution). Since simply combining and maintaining best fit individuals can easily lead to a local maximum in the fitness function and stagnate the population, operators such as mutation allow for more broad exploration of the solution space. In music, the two most common approaches have either a musical piece itself or an artist agent capable of producing music as the individual in a population.

Evaluating music qualitatively is a very difficult problem, as music aesthetic is considered to be subjective. Approaches therefore either try to define a fitness function in some way (such as a weighted sum of some musical features) or just use the evolutionary process in conjunction with other methods to solve optimization problems in some part of the algorithmic music generation process. In many cases, a fitness function proves impractical, so manual human evaluation is used as a fitness function.

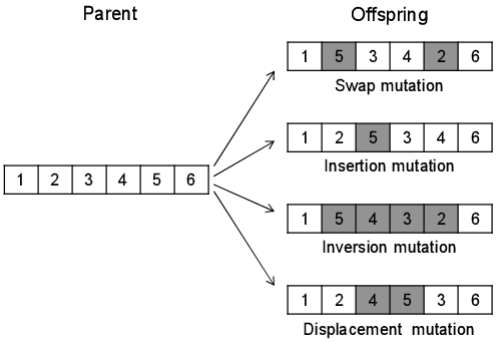



Figure 5.: Examples of operations performed on a integer array

Examples of evolutionary music composers include MetaCompose in Scirea et al. [23], which combines multi-objective optimization with *FI-2POP* to create pleasant and interesting compositions according to a participant-based evaluation. Other attempts try to build simple fitness functions, based on hand-crafted evaluation functions of harmony and melodic value to evolve a population of melodies [24]. Some approaches use a measurement of distance, usually to a chosen piece of music, as a fitness function[25, 26].

## 2.3 PREVIOUS WORK

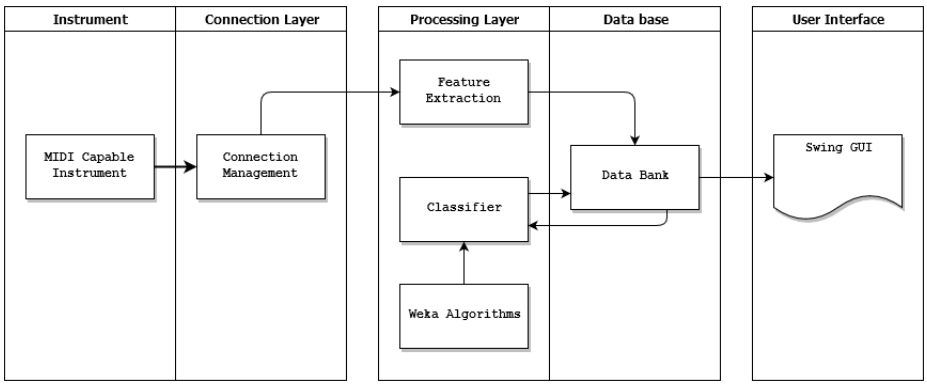



Figure 6.: Original proposed architecture for the project

Under the *Projecto Integrado (PI)* course, previous work was done on musical analysis. A prototype system was built that is able to analyze a real-time performance in a MIDI compliant device, and extract 24 features and derive the artist emotional state from them.

The system architecture was based on segmenting incoming note information into one second blocks, and building three block containers: 1, 5 and 30 block containers. The reasoning behind it was to perform feature extraction on all containers so that both long term

musical structure and subtleties, and immediate musical changes could be captured. Different weights of importance can be given to features from each container depending on what their purpose is (for example, rhythmic analysis only makes sense on longer containers).

The emotional state was measured in a musical arousal and valence bi-dimensional plane. These two axes resulted in four possible measured states: excited, angry, content and sad. The arousal metric refers to the piece perceived arousal or energy values, how energetic the music is and valence refers to the positivity value of the piece, how positive (or negative) emotional connotation the piece has. Both metrics are calculated as a result of an ANN using J48 decision tree method. This algorithm is fast and efficient, and provides good results where discovering which metrics are more influential to the overall value is important. The ANN was trained with 20 different performances of about one minute each.

Features harvested from the performance included piece key and mode discovery and statistical analysis on music dynamics statistics, note duration and rhythmic metrics. Music key discovery was implemented using the Krumhansl-Schmuckler [27] and Temperley [28] key-finding algorithms which works by calculating the Pearson correlation coefficient between the total duration of each note pitch in the music piece and a pre-set of profiles for each key and mode.

Overall the results were satisfactory and the architecture was proven to be suitable for real-time symbolic music analysis.

# 3

# THE PROBLEM

## 3.1 CHALLENGES

Music generation presets some challenges and thus decisions to be taken when tackling the problem. The main one is how to actually generate the audio or note information in the first place. As exposed in the previous chapter, some techniques and methodologies have been used with various degrees of success, with solutions ranging from using a probabilistic approach where the chance of each action for the system to take, that is, each musical note or set of notes is what is being optimized by the system, to collage of hard-coded or learned pieces of music.

Generating context adequate music for a given musical piece, either recorded or simply the musical information about it, brings forth a few considerations and challenges:

1. Contextuality

   One of the hardest behaviours to express, both in a music generation system as in a composer as well, is the one of musical coherence both structurally and aesthetically. There are various concepts involved when one wishes to build upon an existing piece of music, from matching tempo, key and overall mood to musical saturation. The input piece might, and often is, a well-rounded stand-alone musical creation. Building layers carelessly on top of such piece might result in a displeasing, noisy piece with too many elements competing for prominence and listener attention. Such piece requires very considerate and subtle composition, maybe just accents on certain elements of the music is enough. On the opposite spectrum, one might find a piece that is just a melody or a simple a chord progression, which needs to be identified and built upon, composing the elements that are missing.

2. Musicality

   Another important aspect is to ensure the generated result is interesting and pleasing. In musical terms, both "interesting" and "pleasing" might be hard to evaluate, but let us consider the interest of a piece to be defined by how unpredictable and refreshing

are the musical elements and the pleasure of such piece to be an evaluation of musical structural integrity and resolution from dissonant sounds to consonant sounds. In fact, these two elements of music can very broadly be the descriptors of the musicality of a piece and they are, as their definitions suggest, opposite forces. Some modern musical genres, such as modern classical music, show great amounts of musical tension in their pieces by exploring dissonant tones liberally, which makes for very interesting music, despite many listeners finding it grating and unsettling, preferring older classical music mentioning it as much more pleasing.

Building a real-time system reveals three new problems that need to be addressed:

1. Processing Time

   Firstly and foremost, in real-time, there is no processing time available unless we build the result in the background to be appreciated later. Musicians are very sensitive to having their output delayed, with research showing 20 to 25 milliseconds as the maximum delay before it becomes noticeable and starts to disturb the performer ability to play [29]. This effect makes online real-time collaboration at best impractical, at worst impossible. A common analogy is to experiment talking or singing into a microphone while listening back with a slight delay. Most people are unable to keep talking in a coherent and stable manner if listening to themselves with a slight delay, an effect known as *Delayed Auditory Feedback (DAF)*. With the hardware and techniques we have available today, 25 ms is an unreasonably low amount of time to perform any kind of complex analysis and music generation.

2. Reactivity

   Another challenge when dealing with a real-time system is the lack of information about what is coming up. Unless a perfect predictive module is available, the system will always have a delayed response when adapting to changes in the input audio, such as when the human player decides to change substantially what he is playing. If the new piece is different or not related, a real-time system can not deal with the sudden change in context and will not produce adequate accompaniment while it updates the internal state. When processing a recorded piece, all the context and variations are known beforehand so the result would be able to be produced taking the piece as a whole into account.

3. Synchronization

   The last big problem is that of synchronization. Imagining two musicians playing different parts of a piece, it is clear that if there is no previous agreement on rhythmic and timing elements, even though they might be playing at the same time, their actual music might not be synchronized with one player lagging behind the other

contextually. This is further exacerbated if the players are not able to maintain a steady tempo, which means that a real-time system must either have insight about the tempo of the piece being played or the human player needs to perform somewhat consistently so the system can adapt to him. The first option requires information that both is a chore to do before every single part and which data might not even be known to the human controller. The second option assumes some skill and planning for the human, which might be incorrect assumptions especially on improvisational music where constant tempo and structure changes are common.

## 3.2 PROPOSED APPROACH

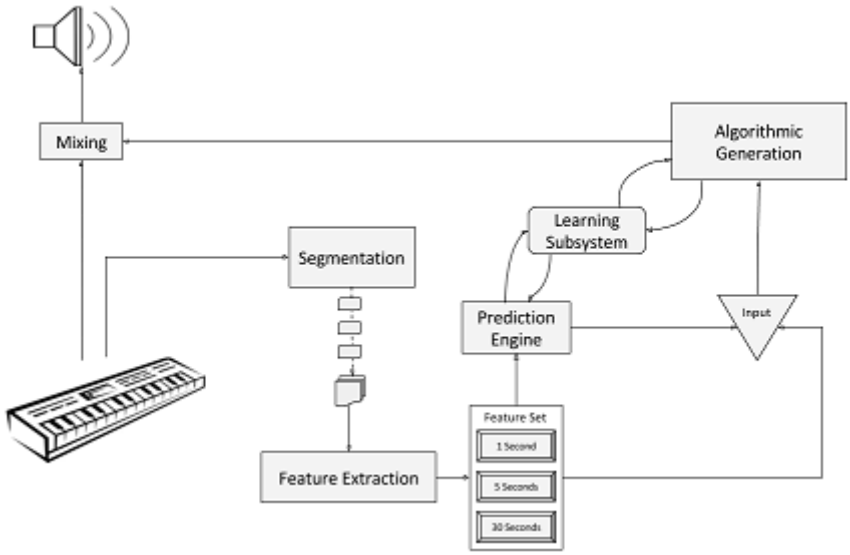



Figure 7.: Proposed system architecture

1. Input

   Current testing and developing setup is made up of a Yamaha PSR-e413 61-key keyboard as a MIDI controller and a computer with this project running and a software synthesizer. This setup means that development of this project is focused towards a single user interaction with the system via a MIDI controller. The MIDI communication protocol functions as an event driven state machine, where messages containing information are sent as events from an input port to an output port, a device can simultaneously have input and output ports. Most common event messages include system handshake, note-on (and the corresponding note-off) and control events to manipulate sounds, such as pedals and effects.

In particular to this application, note events are the most important ones, since they possess information such as which note has been triggered (note pitch), how hard was it pressed (velocity of the note), special command designations and which channel it originates from (for multi-device purposes or multi-channel polyphonic music).

2. Segmentation and Buffering

   As each note played triggers at least two events (Note-ON and the corresponding Note-OFF) and music can contain multiple notes being played at the same time and in a short amount of time, performing full calculations and updates every time a new event is fired is unfeasible with current hardware. Also, many statistics and analysis (such as detecting chords and key played) require information from multiple notes so even if possible, updating on each new event is not practical either. Therefore, some systematic buffering is required to bundle up some events and trigger analysis on a constant timing. As there are multiple types of statistics and information we want to extract that require different time scale samples (such as analyzing overall piece key information but also wanting to detect sudden changes), multiple sized buffers will be used. As a result of PI, three different time scales of a single, five and thirty units will be used as were found to be suitable for both capturing longer spaced behavioural and musical structure information, and to detect sudden changes and new tendencies so that proper reaction can take place.

   Each time unit cycle (the buffer time before a unit is packed and sent to processing) has the duration of one second, which was found to be a good middle point between reaction delay and time to process each cycle. At any time during system operation, all three buffers containing the last single, five and thirty event buffer units are available for analysis and processing.

3. Feature Extraction

   Each time a new event buffer unit is built, feature extraction is to occur on every newly updated buffer. This feature extraction contains any intended analysis to be performed on each buffer separately such as statistical analysis on note pitch, velocity and duration (mean, median, sum, variance, among others), musical structure evaluation such as key and beat finding and emotional evaluation, among others.

   Inherently to the process, some information will be lost during the conversion from one musical representation (raw data) to another (musical features), but these features should represent a succinct and easy to work with description of the input music, to be fed as needed to the music generation systems (or to be displayed to the user as input analysis).

4. Prediction Engine

   Two of the new problems that surge when approaching a real-time system of this nature, namely processing time limitations and delayed reaction time as described in the previous section, can be mitigated if a prediction engine can anticipate musical changes and user intention. A prediction module is proposed that should be able to predict, from continuous analysis of each new processed feature set, changes to these features in the near future. Such module would enable the system to work not with current information about the music, but with near future information. Available processing time is thus extended by how far we can accurately predict user input without any delay in reaction time from the system. This also allows for generated music to be available in advance to synchronize with the user.

   A system based on ANN is proposed to predict feature values, with past observed features as input. Such methodology is suitable if we consider feature prediction as a regression analysis problem on a N-dimensional feature space.

5. Learning Module

   Since minimal user interaction on the system process is a desirable property (which would be impractical in a real-time scenario), user supervised learning is not desirable. However, considering the goal is to predict values that we expect to observe in the near future, we can use those actual values as a real-time network training set. This allows the prediction system to self-correct over time, adapting to both user and current music characteristics. A trained network for a specific user can be saved to a user profile, allowing for better prediction performance in the future. Therefore, on each processing cycle, current observed features are compared against previous cycle predictions to evaluate prediction accuracy and correct the system.

6. Algorithmic Music Generation

   Feature extraction from music is not a bijective function. Information is lost during the compression into statistics and observations. A result of that property is that there is no inverse function to transform observed features back into a set of events or audio. Having that in mind, a heuristic based on modified Markov chains is proposed to generate simple melodies and accompaniment that are musically coherent.

   Although music generation does not satisfy the Markov property (that past or future states depends only upon the present state), the process produces good results when modifications such as adding event sequences as new states (known as higher order Markov chain) and discretizing velocity and duration values. Caution is required, however, as adding too long sequences will, *reductio ad absurdum*, exponentially increase memory requirements and over-fit the result to the input resulting in a program that just mimics what it is fed.

# 4

# DEVELOPMENT

## 4.1 DECISIONS

Considering the architecture proposed in the previous chapter, some decisions as for the exact approach for each module implementation must be taken. In the following sections, the rationale for each system component implementation will be explained, as well as general considerations such as system platform of choice and implementation language.

### 4.1.1 *Software and Libraries*

Choosing JAVA as the implementation language was rather straightforward as it has excellent support and native MIDI handling capabilities. Java also boasts great native parallelization which is desirable to achieve the performance required for real-time processing. Another advantage for JAVA is being multi-platform since, while not a main concern for this project, the possibility of integrating a project of this nature directly in a MIDI device through a small-form computer such as a Raspberry PI is very enticing. In addition, high performing libraries for statistical analysis and neural network frameworks are widely available which was also a requirement. Last but not least, previous work was done in JAVA which speeds up development as some code might be reusable.

For auxiliary libraries Commons Math and Neuroph were chosen as a statistics and neural network libraries respectively. Both libraries offer efficient and modular implementations which are easy to integrate on the system. They also have fairly good documentation which helps with troubleshooting. As such, the final software set-up is a system built in java with a SWING *Graphical User Interface (GUI)*, with the statistical features calculated with modules from Commons Math and machine learning modules from Neuroph.

### 4.1.2 *Feature Extraction*

There is a pertinent decision that affects every system part, starting with feature extraction: how long should a single cycle be? A trade-off arises when considering buffering length: on one side there is the capability to timely react to musical changes that might occur mid-generation process if the cycle is too long, on the other there will not be enough time for processing if the cycle is too small. From previous work on PI, having three differently sized blocks of data with the cycle duration of one second was found to be well suited for musical analysis, and the availability of a training set for affective analysis with that one second window speeds up development significantly enough to accept that result and use it in this system. However, it might be unreasonable to assume that the same aspects that work for a musical analysis operation should also be suitable for algorithmic composition.

Having a variable cycle window would be another option but brings some problems to feature extraction. Most features are hard to compare directly if the time unit they were extracted with was different, such as simple counts of notes or sums of values, and would be needed to be scaled to the same time measurements by being dividing all values by the cycle length for example. Some musical features lose significance when going through scaling operations, such as key finding which might detect chords instead of the piece key if the interval is too small, and others are outright different due to analyzing different parts of a song. Another problem with a variable cycle window is the incompatibility of the current datasets for ANN training, as they would need to be fed the window of time the features were extracted with, which would require a new and larger data set to be properly trained.

Below shows the types of features extracted or inferred and the reasoning behind them:

1. Descriptive statistics

   Descriptive statistics form the broader set of features. They provide a computational cheap way to calculate dozens of results independently, that describe the musical piece in a format very suitable for feeding the different ANN calculating higher abstraction concepts. Statistics are built on various distinct datasets, the lists of all notes duration, velocity, pitch and cardinality. As such, for each set, the following values are calculated:

   - Arithmetic, geometric, quadratic and harmonic mean;
   - Maximum, minimum and sum values;
   - Kurtosis and skewness moments;
   - Variance and standard deviation dispersions.

These statistics total up to 44 computed metrics per bundle of events (132 values when considering the three bundles of data analyzed separately) to give a precise evaluation of an input musical piece during the captured time windows.

2. Probability tables

   Probability tables (or matrices) are a simple way to represent state transition graphs. Markov chains are characterized by an initial state, a state space and one such state transition graph, so these tables will be needed for the music generation step. Transition matrix are build for notes duration, velocity and pitch. As there is as need to build melodies and musical progressions independent of the current key (so such information is not lost when a shift in music occurs), there are also an extra table for pitch-agnostic scale information, which maps every note played to a twelve states table where the states are not represented by the note pitch but by their position on the scale at the recorded moment. More on the uses for these tables will be discussed during the music generation topic.

   As a discrete state space is needed for the compositional process and possible value interval is large for some of these tables, measures with large value possibilities need to be segmented into a more manageable number of states. If not, a rather useless sparse matrix would be created as, for example, 20 recorded notes were added to a table with 128 possible states as is the case with velocity or pitch. With that in mind, the duration values were segmented into 200 milliseconds wide blocks which, unless the piece presents some unusual long notes, reduces the state space to a more reasonable average 15 states. Such segments provide a middle-ground between a large sparse matrix and large bins that would bundle very different notes in the same category. Following a similar logic, the velocity transition matrix was segmented in 8 integer wide states (recall that velocity value ranges from 0 to 127) but since velocity value 0 is meaningless on this context, only 15 states are needed. Since there is a static 12 states progression table to help the generative process, making bins out of pitch values makes no musical sense and a common MIDI controller such as the one used for development only has 61 to 88 keys, the pitch transition matrix was left complete with all possible states. Also to note that a piece of music generally only uses a small subset of these states. In sum, the following probability tables are considered adequate:

   - Duration transition matrix: 200 milliseconds wide bins for an average total of 15 states;
   - Velocity transition matrix: 8 integers wide bins for a total of 15 states;
   - Pitch transition matrix: 61 to 88 states although on average only a small subset is used;

- Progression transition matrix: 12 states.

While it does not make sense for velocity and duration transition tables, both the progression and pitch tables are able to include second and third order states. Since these exponentially increase the state space, only actual occurring states are added to the table, reducing the space requirements greatly.

3. Key-finding algorithms

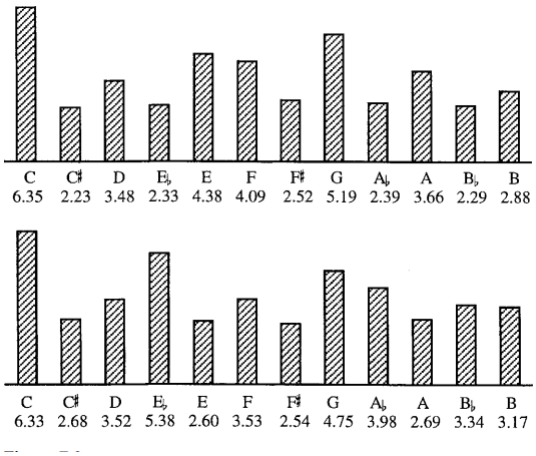



Figure 8.: C major and C minor key profiles.

A piece key is one of its major characteristics and imposes some subtle structure to the music. It dictates which chords should be played and sound better together and exposes some relations between the notes, as well as revealing which notes and chords provide a subjective sense of tension, resolution or rest. Knowing the piece key and tonic note reveals information about context, mood and melodic behaviors. Naturally, it plays an important role in the system for both affective evaluation and music generation. However, finding a piece key is not trivial as two different keys and modes can use the exact same pitchs, rendering impossible to find out the key and mode of a piece just by knowing which notes were played.

As also found in previous results on PI, the *Krumhansl-Schmuckler* key-finding algorithm produces good results [27]. However, the number of notes in a one second pieces is, in many slower musics, not very representative and so the algorithm result might fluctuate a lot for small time windows. It functions by calculating the Pearson correlation coefficient between the duration sum of each of twelve pitchs in an octave from the music piece and a set of profiles of each key and mode. These profiles were produced by empirical testing in the case of Krumhansl [27] or influenced by music theory such as Temperley [28].

As seen in figure 8, each set of profiles is comprised of an array of weights given to each pitch for both minor and major key modes. Since neither set of profiles is objectively better, a simple ANN was trained to calculate the key from the current and previous results of using both profile sets. Using this simple network, there is no need to choose and use just one of the profile sets, which should give a better result.

4. Rhythmic evaluation

   One important part of the generative process is to properly identify the musical space where the machine can explore. In simple terms, it is needed to evaluate what part the system can take in the music, one of lead melodic generation or a more rhythmic accompaniment role in the overall composition. In a spectrum of pure rhythmic to pure melody, it is proposed a small subsystem based on a previously trained *Multi-Layer Perceptron (MLP)* ANN to perform classification on the human musical part, so the generation can explore a complementary role. This network receives as input various descriptive statistics related to general pitch areas and variation as well as cardinality, to output a value in the rhythmic spectrum describing the musical piece. Training was done by playing multiple accompaniment pieces (such as simple chord progressions), multiple simple melodies and various mixed performances while recording extracted features for each one and building a data-set. The data-set used for training has about 2200 entries which correspond to about 35 minutes of music.

5. Affective classification

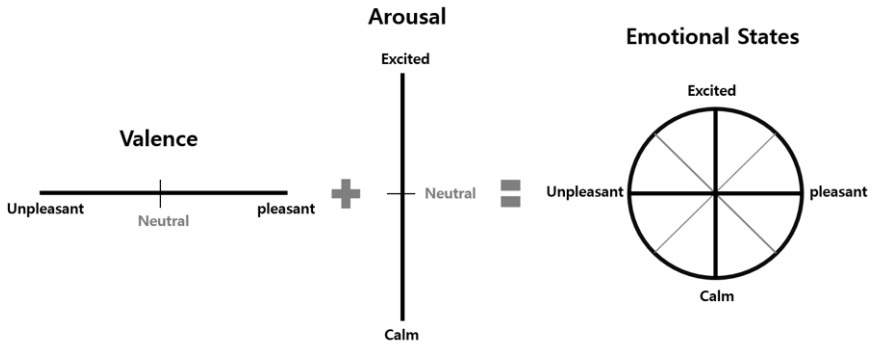



Figure 9.: Mood diagram using arousal and valence measurements.

Discovering a musical piece affective value is of importance not only for the generative process but by itself, giving insight into the author intentions and emotional expression he intends to imprint on a music. Music expressivity and feeling are subjective metrics, and finding out a musician emotional state by its performance is a clear classification problem. ANN not only are suitable for classification problems, they do

so by outputting values instead of a names, which means that if a multidimensional framework is used to classify the author emotional state, one can calculate how each dimensions contribute to the result. One such framework is shown in figure 9, which evaluates a emotional state based on two metrics: arousal and valence. The arousal metric measures how energetic the music feels while valence measures how positive (or negative) emotional connotation the piece exhibits to the listeners. Using valence and arousal as metrics we can classify the artist emotional state as follows:

- High valence and arousal: Excited;
- High valence and low arousal: Content;
- Low valence and high arousal: Angry;
- Low valence and low arousal: Sad.

The network was trained by playing music pieces while recording extracted statistical features and music key. Each music piece was roughly given a value for overall arousal and valence metrics. A data-set was built using those values and the extracted features. Ideally, one would manually classify small music sequences in a more granular manner but building such data-set would be a large endeavour. A possible solution would be to build many music sequences, make a questionnaire for a large audience to evaluate and then aggregate the survey results into a data-set. Building such a data-set would likely improve the affective classification accuracy, allowing for additional more minute intermediate emotional states to be inferred from the classification output.

### 4.1.3 *Features Prediction*

As previously discussed, feature prediction can improve the reaction time of the system drastically up to instant reaction if the prediction is very accurate. Not all features are needed for music generation so there is not a need to predict every single one which should make training more effective on the wanted results with use further on. A very wide ANN will be used, due to the large size of the input and output layers, with a MLP architecture which provides both good computational and accuracy performance.

As per Hornik et al. [30] and Bandyopadhyay and Chattopadhyay [31], no more than a single hidden layers is needed for good results, which is helpful since using fewer layers will increase usage and training computational performance. The input layer consists of previously extracted features and the output layer consists of the required values for algorithmic composition, which is a small subset of the input nodes. Following Blum [32] rule of thumb, the initial layer size was set as the average of the input layer size and the output layer size and slightly tuned while testing. As to vastly reduce converging speed, the initial

state of the prediction network has been already trained with multiple musical pieces during development.The feature prediction subsystem will also perform profiling duties as the constant learning and adaptation will infer some quirks and tendencies from the human partner. In fact, the initial network state is no other than the developer personal profile. Higher weight will be given to new training cases to offset bias present due to pre-training.

### 4.1.4 *Algorithmic Composition*

Algorithmic composition is approached through multiple higher order Markov chains combined with hand-crafted rules using extracted features. This approach provides a simple but powerful way to generate notes that fit the original music, especially as specific rules are implemented to guide the generation process. Manipulation of the probability matrices is a tool that can be used to fine tune how similar to the original music generation should be, and can be done by using the extracted features to guide the decision process. Poisoning the probability matrices with a small bias to different musical modes depending on the detected affective state can be a powerful tool to generate interesting melodies. Using shifted pitchs generated from the progression table merged with previous generated music gives good cyclic progressions for accompaniment.

The generation system saves a personal state of features, altered tables and generated music. When new information is generated, this data needs to be updated. The system state is then further modified depending on observed affective information, music key and mode. Then a fork is evaluated depending on the rhythmic feature so the system can properly generate a melody or a rhythm which have different characteristics. Depending on how binary the rhythmic evaluation turns out to be, different probabilities are affected: if a pure melody is detected, then a pure rhythm comprised of mostly chords is generated. If the input is more balanced, organic music will be generated as the system will mix up both styles. To provide a good, cyclic rhythm, the progression table is consulted. On the other side, to provide a varied and interesting melody, the pitch table is modified with the probabilities of the musical mode profile, giving the possibility to generate notes that were not observed but still fall inside the detected key and mode.

The actual note generation process is iterative until the music sequence meets the time requirement. Note also that there needs to exist a pitch value denoting a pause or rest, which is an important part of the music expressivity and cadence. Finalized the generation, the result is copied to the state records and returned for reproduction.

## 4.2 IMPLEMENTATION

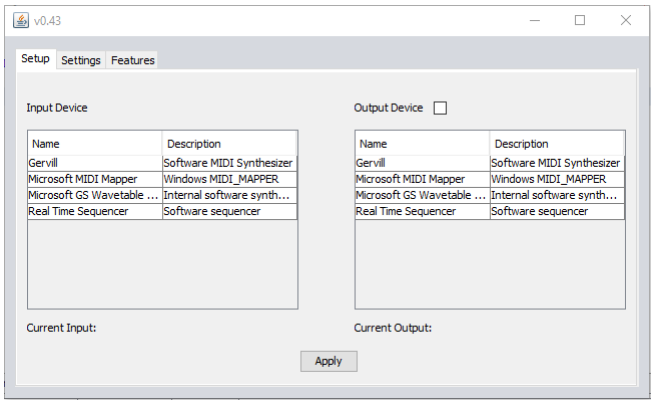



Figure 10.: Graphical user interface of the main menu.

### 4.2.1 *Device Management*

At program start, the user is presented with a GUI containing list of MIDI devices (which can be either hardware or software) to choose from. The interface is built in *SWING* and MIDI device query and operations are native to *Java* through the *javax.sound.midi* package. When the user chooses the input and output devices, all the necessary connections using *listeners* and *receivers* is done, and a custom made listener is added to the input device to start capturing events. This listener, an instance of the class *BufferListener*, keeps a buffer of events which is retrieved by the system at every cycle. Listeners and receivers are managed by the class *DeviceManager* which takes care of all *Input/Output (I/O)* related to MIDI devices. This same class is responsible for passing along the generated music to the correct output device.

### 4.2.2 *Feature Extraction*

Feature extraction occurs, during standard operation, every second. It is treated as a separate module which is initialized during program load with a list of features to extract on each cycle. These features are implementations of the *Feature* interface. This approach allows for modular feature extraction, which can be easily added or removed during development. It also allows for complete parallelization as every implementation of the interface is a standalone group of operations, enabling the ability to increase the number of features extracted with a low performance impact. The results of each module are compiled in an array of *Double*s to easily feed ANN input layer and there is also a *FeaturesData* class to

wrap the most useful results (such as key analysis) for easy access and identification. Descriptive statistics are calculated with the help of the Commons Math library, most notably the *DescriptiveStatistics* class from the *org.apache.commons.math3.stat.descriptive* package.

In each cycle, the feature extraction subsystem receives as input three Bundle class instances containing a list of recorded events for each of the three different time windows of data of one, five and thirty seconds. All three bundles are processed every cycle as various features change even when considering a one-second data difference on a thirty seconds bundle.

Firstly, the bundles are passed to the *FeatureExtraction* class. This class has a pool of threads on standby to delegate work and perform the descriptive statistics extraction with the help of the Commons Math library, as well as building the set of probability matrices. After the results are retrieved, they can be used to infer the other higher abstraction features, such as key-finding, affective classification, and rhythmic evaluation. All three abstraction features use networks with the help of Neuroph ANN implementations, and all the input values were properly scaled to a range between 0 and 1 for better results. Built data-sets were trimmed as the initial 30 measures are done without the data bundles being full of data. All networks use the back-propagation training method. However, due to the modularity of the system, architecture and training methods can be easily swapped. These features have the following characteristics:

- Key-finding algorithm

  Key-finding is implemented using the *Krumhansl-Schmuckler* key-finding algorithm described before, resulting in three values per profile used: the piece detected key *pitch*, the mode (major or minor) and a correlation value. There are 6 result values with which to know the current key using two different profiles. Feeding these and previous results to a trained network results in a more accurate and stable evaluation of current key. This network has a MLP architecture with an input layer of size 9 per profile (plus a bias neuron), a middle hidden layer of 6 neurons and an output layer of 2 neurons (as there is no longer a need or meaning to the correlation value). The network was trained with a data-set comprised of 1400 entries, the recorded results of the algorithm running to 24 pieces of one-minute long music representative of all twelve possible keys in both modes.

- Affective classification

  Affective classification requires the results from the key-finding algorithm, so it can not be run in parallel. The MLP network has an input layer using all 44 statistical values, 2 key-finding results plus a bias node. Testing showed that a single hidden layer with 16 nodes was enough, and the results are two nodes representing arousal and valence. Training occurred using a data-set built by manually playing pre-categorized

pieces of music averaging 2 minutes each, 8 pieces in total, and recording extracted features for a data-set with a total number of 1100 entries. Upon inspection, many of the statistical values seem to be considered not relevant after the training process, so there is a possibility of performance gains by removing some statistics without accuracy loss.

- Rhythmic evaluation

  Rhythmic evaluation is performed in parallel with affective evaluation, although could be done in parallel with the key-finding algorithm. The MLP network has an input layer of 33 statistical values and a bias node as the statistics related to velocity are not significant for rhythmic evaluation. The single hidden layer is comprised of 20 nodes and the output layer has a single neuron. Network training was performed by recording features while playing 8 accompaniment pieces (such as simple chord progressions), 8 simple melodies and 4 mixed performances, building a data-set with 2200 entries which correspond to about 35 minutes of music. This network also shows some possibilities of pruning, both in the input layer and in the hidden layer.

After all the results are compiled in an array of data and on a *FeaturesData* wrapper class, they are sent to prediction analysis.

### 4.2.3 *Prediction*

At the start of the program, a pre-trained development MLP network is loaded. However, it is possible to load or create a custom profile through the GUI containing a different one. The network was trained without building a data-set, just by continuous use during development. Although there were some resets, the network has data from about 8 hours of playing. This large neural network receives extracted features in the input layer for a total of about 140 neurons: 132 statistical features, 6 from the key-finding algorithm, one from rhythmic evaluation and one bias term. These results include every result from the three data bundles. The network output layer size is 15, the number of features used by the composition process: 1 from rhythmic prediction, 2 from key and mode prediction and 12 statistical values relating to deviation and mean of all four measures. The single hidden layer has 40 neurons. The initial estimate was 80 neurons, but through testing a better result was found with half as many, most likely due to the small selective elements we wish to predict, making a significant portion of the input extraneous and leading the network to over-fitting.

After the network evaluation for the current input, a new data-set row is created by gathering the select 15 elements from the input layer, serving as expected output to the previous cycle input. That is, the future features predicted in the previous cycle are compared to the

current cycle input for an accuracy evaluation. The real observed values for the predictions of the previous cycle are stored, along with the previous input layer values, as a new data-set row. This new row is then both saved to file and used to further tweak the network using Neuroph capability to learn in a new thread in parallel to usage. The current input is saved temporarily to be used in creating another data-set row in the next cycle, and the results are returned for use in the algorithmic composition.

### 4.2.4 *Algorithmic Composition*

| Name | Mode | D'Arezzo | Fulda | Espinosa |
|---|---|---|---|---|
| **Dorian** | I | serious | any feeling | happy, taming the passions |
| **Hypodorian** | II | sad | sad | serious and tearful |
| **Phrygian** | III | mystic | vehement | inciting anger |
| **Hypophrygian** | IV | harmonious | tender | inciting delights, tempering fierceness |
| **Lydian** | V | happy | happy | happy |
| **Hypolydian** | VI | devout | pious | tearful and pious |
| **Mixolydian** | VII | angelical | of youth | uniting pleasure and sadness |
| **Hypomixolydian** | VIII | perfect | of knowledge | very happy |

Figure 11.: Artists interpretation of emotions imparted by different modes.

After all the data and analysis results are computed, the probability matrices, the results of the prediction and the affective evaluation results are sent to the *Generator* class module. New data is used to update the generator database, including probability matrices. The general process then occurs as following:

1. Merge new information in the system own state data; New information also includes timing data, required to properly encode duration information in the generated note events (recall that events do not carry a duration camp; a note on and a note off event must be sent at correct times).

2. Evaluate affective information and add mode bias accordingly; Using affective information and a mode translation such as in figure 11, a slight bias (of about 10%) is injected in the probability matrices to manipulate the mood of the music.

3. Evaluate operational space by consulting the rhythmic evaluation; If the rhythmic evaluation result is near the spectrum extremity then the generation process is simplified and a branch below is followed. Else, a mixture is randomly created resulting in melodies with strong rhythmic accents.

4. If the piece is predominantly melodic, then generate a rhythm:
    a) Beginning at higher order states from the progression probability table, construct new note(s) starting with pitch value; The progression matrix provides a pitch agnostic result that tends to cyclic repetition which is ideal for generating accompaniment. This is further accentuated by merging the generated results back with the matrix.
    b) Pitch shift the generated notes to the correct key and mode; Considering the progression matrix rooted in *C*, the pitch needs to be shifted to the correct key to sound properly.
    c) Consulting the velocity probability table, generate the note(s) velocity value;
    d) Consulting the duration table, generate the note(s) duration value;
    e) Subtract the duration from the available generation time and if there is still time left, return to step a).
5. If the piece is predominantly rhythmic, then generate a melody:
    a) Insert bias in the pitch probability tables from the detected key profile;
    b) Beginning at higher order states from the pitch probability table, construct new note(s) starting with pitch value;
    c) Consulting the velocity probability table, generate the note(s) velocity value;
    d) Consulting the duration table, generate the note(s) duration value;
    e) Subtract the duration from the available generation time and if there is still time left, return to step b).
6. Merge generated music in the system own state data. Generated music is merged back into tables and variables so that motifs and repetitions can be reinforced, while reducing the copy effect of Markov chains.
7. Return generated event list.

The results are sent back to the *Core* class of the system. The generated event list is stored in disk and sent to the *DeviceManager* class for reproduction.

## 4.3 SUMMARY

The final system operation is rather linear and cyclical so it can be easily understood by following the data. Starting at the input MIDI device where a musician is playing, events are generated containing information about played notes duration, velocity and pitch. These events need to be grouped together for analysis as performing analysis on every new event is not currently possible on current hardware. There are also many metrics that only make sense with a set of data. As such, a unit of time of one second marks the retrieval of a new bundle of data which is then further aggregated in bundles of 5 and 30 seconds. These bundles should provide enough representative time windows to capture different musical behaviours, such as the piece overall tone and motifs all the way to sudden changes in music.

The three bundles of data are then taken by the feature extraction subsystem to reduce the music information into meaningful smaller chunks of information. These features can be anything from descriptive statistics such as mean and variance up to higher abstraction features that depict musical structure such as key and mode information using MLP ANN. They should be representative enough as to allow general behavior and patterns to be inferred from these values.

A problem that surges at this point is one of synchronicity. As information can be a full second old, generated music will always be created with an outdated basis. The system will lag behind the artist and will be especially noticeable when a significant change occurs in the music. To combat the problem, a predictive system is in place, using ANN, to predict what the music will be like in the very near future. By acting on predicted data, the system will not lag behind as long as the predictions are accurate. To perform the prediction, a large network using over a hundred features is trained to predict a subset of needed features in the near future. To improve accuracy and adapt to each artist style, the system takes measures the difference between what was predicted and what was observed to tweak the network during execution.

With a vast amount of data about the music available, the compositional process generates MIDI events to add to the human performance, taking in account musical structure, piece affective connotation and role space to fill in the composition. Music is generated using modified Markov chains and handmade rules to guide the process.

# 5

# CASE STUDY

## 5.1 EXPERIMENT SETUP

Since showing musical results in a paper format is very hard to do without relying on a subjective survey format, focus was shifted on to the experimental verification of the prediction subsystem results. The goal is to verify if and how much the prediction subsystem improves results versus not having it enabled at all. Three musical pieces were prepared with different characteristics:

1. Single song

   A single song that stays on the same key throughout the music. Most examples of western popular music fall under this category.

2. A medley

   A medley of three very distinct songs, so the sharp change in music structure and style throws the system off.

3. Random notes

   An assortment of notes, literally played by a two year old child with no previous musical experience.

$$\|\mathbf{x}\| := \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}.$$

Figure 12.: Euclidean distance formula for N dimensions.

The musical pieces were trimmed to three minutes exactly each, performed live with the system (not recorded). The system was modified to run with and without the prediction module active for each song, with the values passed on to the generative process

being compared to the next cycle real observed results. This setup compares how accurate the prediction was versus the real observed value and also compares each successive data extraction as if there was no prediction module enabled. The comparison was made by calculating the euclidean distance (figure 12) between the $N$ features present in both sets. All features were scaled to the 0 to 1 range to improve accuracy [33] as was the distance result, since the actual value is not as important as the comparison between the results. The results were averaged over five second blocks to present smaller tables and less erratic graphics. Results are provided as annexes. All testing was done on a Yamaha PSR-e413 61-key MIDI controller and using Commons Math version 3.6.1 and Neuroph version 2.94 libraries.

## 5.2 RESULTS

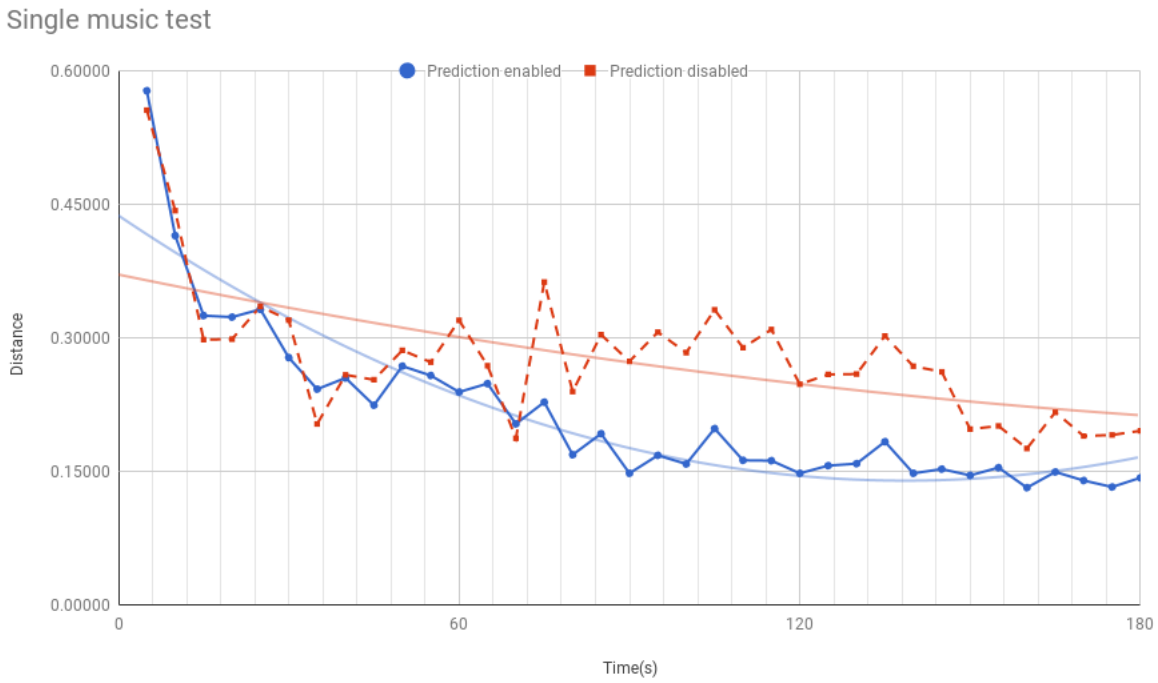



Figure 13.: Testing results chart for a single song.

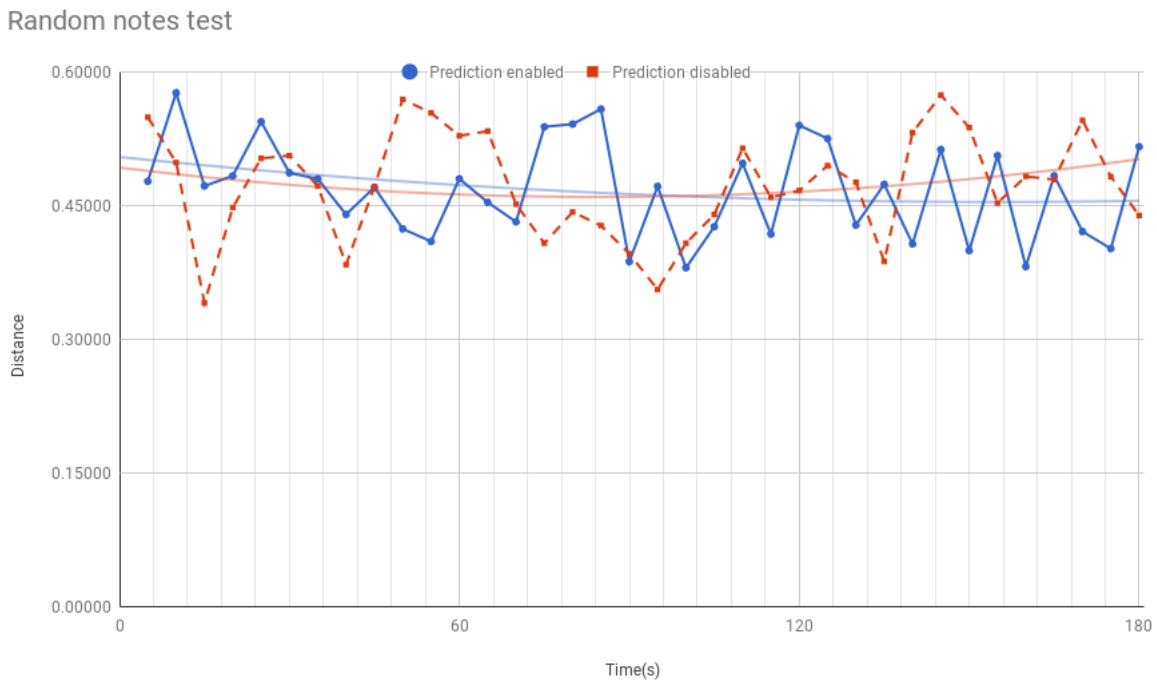



Figure 14.: Testing results chart for randomly playing on the keyboard.

## 5.3 DISCUSSION

There are some shared expected behaviors. Every time a new piece starts the error is very large, even when using the pre-trained prediction network. This is expected as the in the start, the data bundles are not full, which makes statistics dependant on cardinality very different from the usual values and musical structure features like key-finding returning very erratic results due to low sampling. However, the error goes down rapidly as more data is collected and the actual results can be seen when the data windows are full. At this point, every test with prediction disabled stabilizes and behaves erratically depending only on how the piece changes from second to second. However, especially on a stable music such as in figure13, it can clearly be seen that with prediction enabled, the results are much more stable and overall lower.

The random notes figure 14 test is, as expected, pretty random. Without any consideration for music structure or coherence, the results are very similar between both setups, equally erratic and presenting an overall high degree of error. The slight lower tendency is quite probably result of chance rather than the prediction module being able to improve the result.

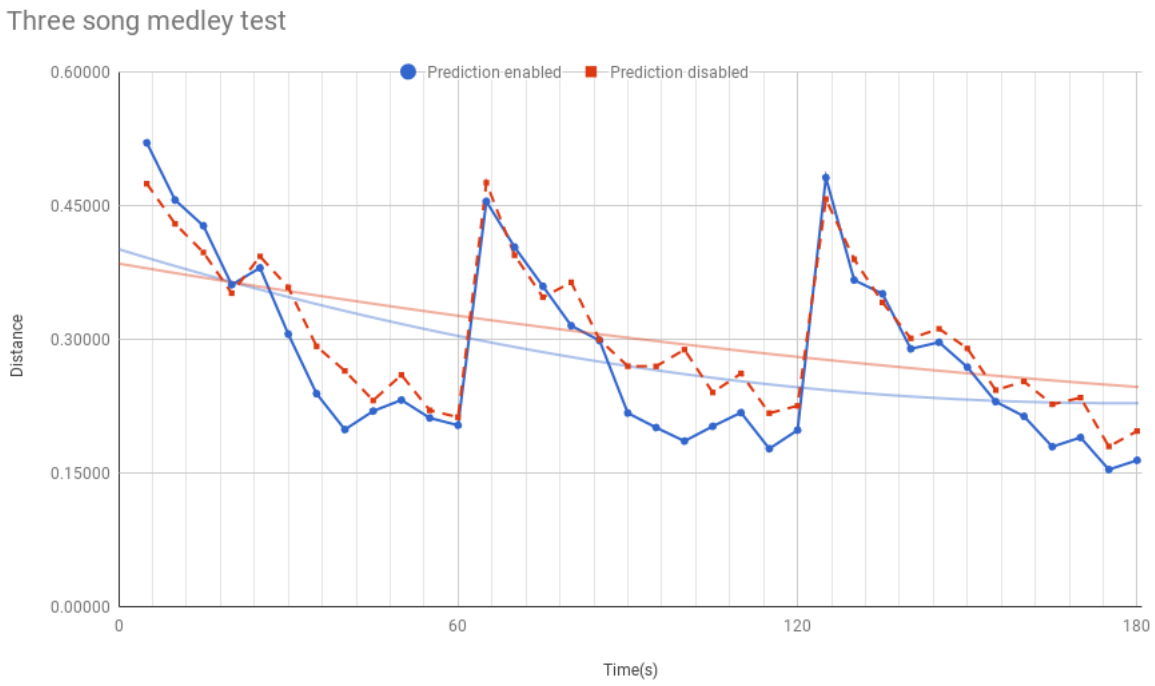



Figure 15.: Testing results chart for a three song medley.

The results from the medley in figure 15 are very interesting. As expected, when the music changes drastically, error skyrockets as suddenly all values still present on the data bundles are completely outdated and conflicting with newer values. However, a clear tendency can be seen for faster reduction in error when prediction is enabled. Overall, the results are very positive. The prediction system can not anticipate drastic changes in music but can attenuate regular music change and overall tends to reduce error significantly as more data is acquired. On stable music without very drastic changes in tone or structure, it can provide very good results. For example in a improvisation scenario that tends to long sessions exploring a certain musical motif or chord progression, the results can be drastically better using a prediction ANN.

# 6

# CONCLUSION

## 6.1 CONCLUSIONS

The initial two objectives for this project, namely the implementation of a musical analysis computational system and a coherent music composition subsystem were achieved.

Although based on previous work, the current analysis system has been completely redone with vastly more features (over 160 per second versus 32 in the previous one) with more higher abstraction features and each trained with more than an order of magnitude more test cases than the single network used before. There was a focus in the predictive system to solve a problem in a way that the results could be shown. It is unfortunate that results of other parts of the system are very hard to show on paper.

As for the second objective, even though showing the results is hard due to the medium and inherent subjectivity, the objective of generating musically coherent music is shown by the very process used to generate music, always inside the framework defined by the input piece musical structure and style. However, a public survey acquiring opinions on the results for various musical pieces generated or not by the program might be a good way to, in the future, evaluate the musical qualitative performance. Such endeavour has shown to be too time consuming to conduct during this project timescale.

The project was ambitious and was a fantastic opportunity to explore computational musical analysis and generation, and as will be discussed next, there are many improvements and further work to be done on this system.

## 6.2 PROSPECT FOR FUTURE WORK

Future work on this specific system can be done through different vectors. One clear area of improvement would be to train all used ANN with higher quality data-sets. During this project the various networks data-sets were built by generally giving descriptive values (such as rather arbitrary arousal and valence values for affective classification) and the program gathered features during execution and matched with those hard-coded values.

Especially on a system that performs analysis on such short blocks of data, using a blanket value for the whole musical piece will produce less than optimal data-sets. Carefully hand-crafted data-sets built on top of many short pieces of music with generally agreed upon results (such as from public opinion surveys) will surely provide much better results.

Another area to be explored is a study of the impact on the results quality and accuracy of using different network architectures, configurations and learning algorithms. While MLP networks provide a capable and generalist framework, different configurations might produce better results, suited to the various types of data. Also another possible exploration would be on the input features that actually have an important impact on the different higher abstraction features. Fewer input values result in better performance and thinner data-sets. It could be the case that removing unrelated metrics actually improves the result quality as well.

On the topic of musical composition, an exploration of a more deliberative system running asynchronously from the musical analysis system would be very interesting. Reaction to changes could be attainable on such system with very fast reaction rules listening for triggers in the analysis. Another interesting project would be to heavily focus on machine creativity and generating music without human crafted rule assistance.

Using the same architecture, it might be worthwhile to explore variable cycle time system. This would enable the system to adapt the analysis process to faster or slower musics, maybe even synchronizing with the music tempo for improved results and faster reaction to changes. However, such system would require much larger data-sets, as different features would be incompatible if obtained on different time windows. Window width would be an input node on every network and data-sets would need to be expanded.

As for application, building an autonomous agent system that could collaborate in creating music with or without human contribution, with each agent fulfilling a role in the group would be very interesting. Or using a system of this nature to produce ambient music, from data gathered through sensors instead of a musician, adapting to the surrounding sound landscape and detected mood. Such system could find use in public spaces or offices to help continuously maintain a desired ambience. There are plenty of avenues to explore, be it on improving this particular system or in the field in general, and personally would love to continue working in this area.

# BIBLIOGRAPHY


[1] Cory McKay. *Automatic genre classification of MIDI recordings*. PhD thesis, McGill University Canada, 2004.

[2] George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *IEEE Transactions on speech and audio processing*, 10(5):293–302, 2002.

[3] Cory McKay and Ichiro Fujinaga. Improving automatic music classification performance by extracting features from different types of data. In *Proceedings of the international conference on Multimedia information retrieval*, pages 257–266. ACM, 2010.

[4] Zehra Cataltepe, Yusuf Yaslan, and Abdullah Sonmez. Music genre classification using midi and audio features. *EURASIP Journal on Advances in Signal Processing*, 2007(1): 036409, 2007.

[5] Jean-Julien Aucouturier. Representing musical genre: A state of the art. 32:83–93, 03 2003.

[6] António Pedro Oliveira and Amílcar Cardoso. A musical system for emotional expression. *Knowledge-Based Systems*, 23(8):901–913, 2010.

[7] Tuomas Eerola and Jonna K Vuoskoski. A review of music and emotion studies: approaches, emotion models, and stimuli. *Music Perception: An Interdisciplinary Journal*, 30(3):307–340, 2013.

[8] Jose D Fernández and Francisco Vico. Ai methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research*, 48:513–582, 2013.

[9] Jon Gillick, Kevin Tang, and Robert M Keller. Learning jazz grammars. In *Proceedings of the sound and music computing conference*, pages 125–130, 2009.

[10] Robert M Keller and David R Morrison. A grammatical approach to automatic improvisation. 2007.

[11] Kris Makoto Kitani and Hideki Koike. Improvgenerator: Online grammatical induction for on-the-fly improvisation accompaniment. In *NIME*, pages 469–472, 2010.

[12] Donya Quick. Generating music using concepts from schenkerian analysis and chord spaces. Technical report, Tech. rep., Yale University, 2010.

[13] Gerhard Widmer. Qualitative perception modeling and intelligent musical learning. *Computer Music Journal*, 16(2):51–68, 1992.

[14] Randall Richard Spangler. *Rule-based analysis and generation of music*. PhD thesis, California Institute of Technology, 1999.

[15] Ryan A McIntyre. Bach in a box: The evolution of four part baroque harmony using the genetic algorithm. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 852–857. IEEE, 1994.

[16] Karsten Verbeurgt, Mikhail Fayer, and Michael Dinolfo. A hybrid neural-markov approach for learning to compose music by example. *Advances in Artificial Intelligence*, pages 480–484, 2004.

[17] ManYat Lo and Simon M Lucas. Evolving musical sequences with n-gram based trainable fitness functions. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 601–608. IEEE, 2006.

[18] François Pachet. Interacting with a musical learning system: The continuator. *Music and Artificial Intelligence*, pages 103–108, 2002.

[19] François Pachet, Pierre Roy, Gabriele Barbieri, and Sony CSL Paris. Finite-length markov processes with constraints. *transition*, 6(1/3), 2001.

[20] Judy A Franklin. Multi-phase learning for jazz improvisation and interaction. In *Proceedings of the Eighth Biennial Symposium for Arts & Technology*, 2001.

[21] Petri Toiviainen. Modeling the target-note technique of bebop-style jazz improvisation: An artificial neural network approach. *Music Perception: An Interdisciplinary Journal*, 12 (4):399–413, 1995.

[22] Matthew I Bellgard and Chi Ping Tsang. Harmonizing music the boltzmann way. *Connection Science*, 6(2-3):281–297, 1994.

[23] Marco Scirea, Julian Togelius, Peter Eklund, and Sebastian Risi. Metacompose: A compositional evolutionary music composer. In *International Conference on Evolutionary and Biologically Inspired Music and Art*, pages 202–217. Springer, 2016.

[24] M Marques, V Oliveira, S Vieira, and AC Rosa. Music composition using genetic evolutionary algorithms. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 714–719. IEEE, 2000.

[25] Andrew Gartland-Jones. Can a genetic algorithm think like a composer. In *Generative Art*, 2002.

[26] Manuel Alfonseca, Manuel Cebrian Ramos, and Alfonso Ortega. Evolving computer-generated music by means of the normalized compression distance. *WSEAS Transactions on Information Science and Applications*, 2005.

[27] Carol L Krumhansl. *Cognitive foundations of musical pitch*. Oxford University Press, 2001.

[28] David Temperley. *The cognition of basic musical structures*. MIT press, 2004.

[29] Inc. High Fidelity. How much latency can live musicians tolerate? Technical report, 2013.

[30] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[31] G Bandyopadhyay and S Chattopadhyay. Single hidden layer artificial neural network models versus multiple linear regression model in forecasting the time series of total ozone. *International Journal of Environmental Science & Technology*, 4(1):141–149, 2007.

[32] Adam Blum. *Neural networks in C++: an object-oriented framework for building connectionist systems*. John Wiley & Sons, Inc., 1992.

[33] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

# A

## DETAILS OF RESULTS

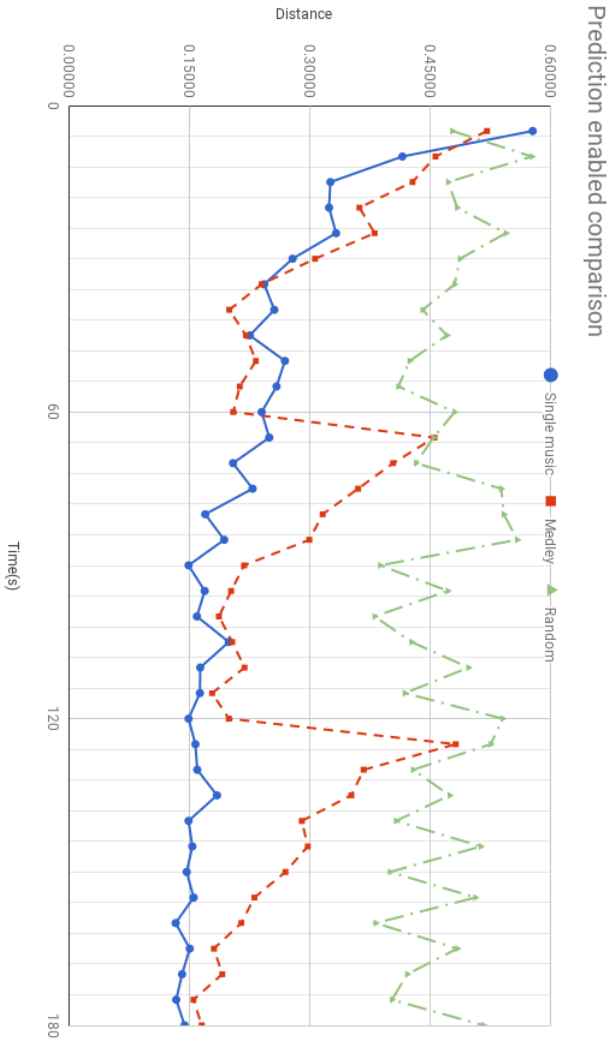



Figure 16.: Testing results chart with prediction enabled.

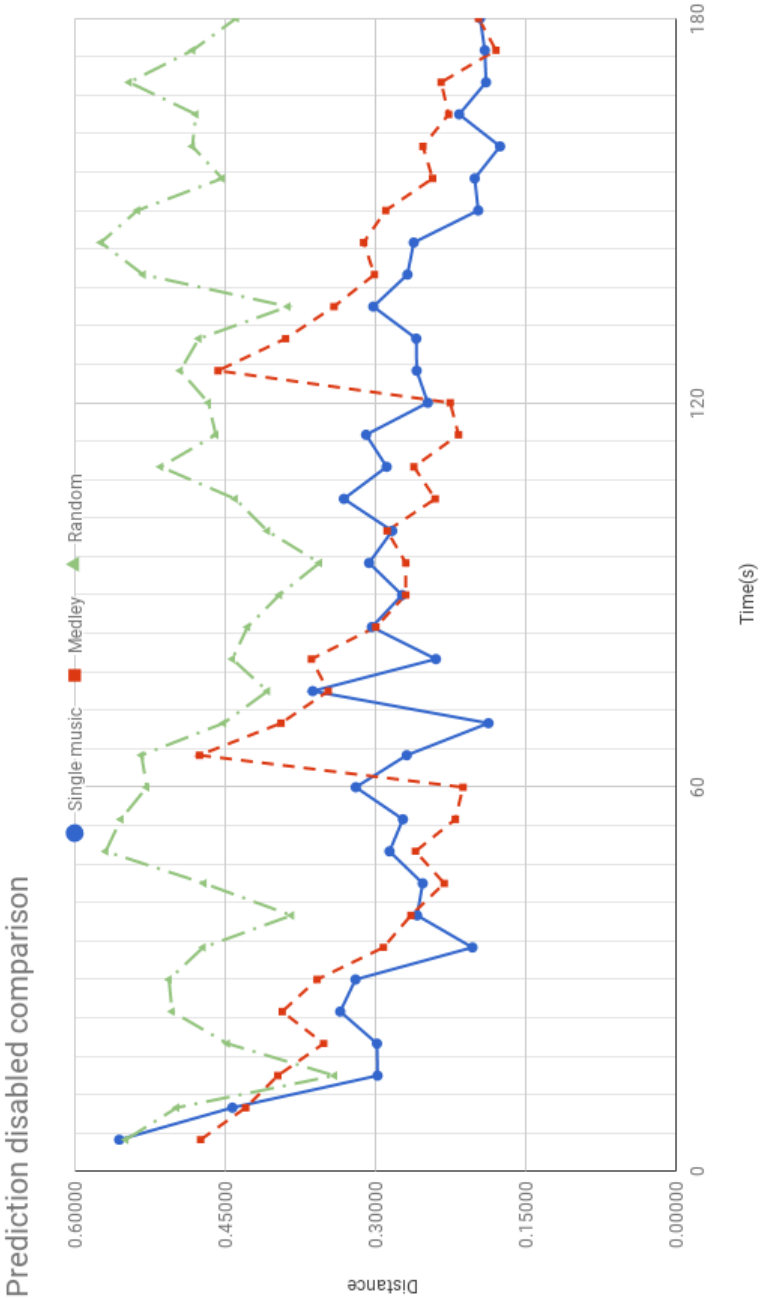



Figure 17.: Testing results chart with prediction disabled.

## A.1 SINGLE MUSIC TEST

| Time (s) | Prediction enabled | Prediction disabled |
|---|---|---|
| 5 | 0.57737 | 0.55562 |
| 10 | 0.41483 | 0.44283 |
| 15 | 0.32515 | 0.29819 |
| 20 | 0.32348 | 0.29875 |
| 25 | 0.33219 | 0.33537 |
| 30 | 0.27804 | 0.32013 |
| 35 | 0.24266 | 0.20362 |
| 40 | 0.25523 | 0.25874 |
| 45 | 0.22464 | 0.25320 |
| 50 | 0.26851 | 0.28625 |
| 55 | 0.25797 | 0.27291 |
| 60 | 0.23946 | 0.31989 |
| 65 | 0.24889 | 0.26906 |
| 70 | 0.20373 | 0.18762 |
| 75 | 0.22812 | 0.36281 |
| 80 | 0.16910 | 0.24001 |
| 85 | 0.19273 | 0.30384 |
| 90 | 0.14830 | 0.27376 |
| 95 | 0.16845 | 0.30664 |
| 100 | 0.15876 | 0.28361 |
| 105 | 0.19845 | 0.33176 |
| 110 | 0.16284 | 0.28912 |
| 115 | 0.16246 | 0.30962 |
| 120 | 0.14830 | 0.24813 |
| 125 | 0.15688 | 0.25922 |
| 130 | 0.15914 | 0.25960 |
| 135 | 0.18376 | 0.30238 |
| 140 | 0.14829 | 0.26839 |
| 145 | 0.15301 | 0.26211 |
| 150 | 0.14597 | 0.19785 |
| 155 | 0.15464 | 0.20137 |
| 160 | 0.13219 | 0.17611 |
| 165 | 0.14980 | 0.21685 |
| 170 | 0.14019 | 0.19009 |
| 175 | 0.13294 | 0.19130 |
| 180 | 0.14333 | 0.19595 |

Table 1.: Distance function evaluation of consecutive extracted features from a single music, with and without enabling the prediction module. Values range from 0.0 (perfect match) up to 1.0 (maximum error).

## A.2 THREE SONG MEDLEY TEST

| Time (s) | Prediction enabled | Prediction disabled |
|---|---|---|
| 5 | 0.52038 | 0.47455 |
| 10 | 0.45620 | 0.42962 |
| 15 | 0.42746 | 0.39758 |
| 20 | 0.36133 | 0.35191 |
| 25 | 0.38032 | 0.39317 |
| 30 | 0.30609 | 0.35845 |
| 35 | 0.23939 | 0.29234 |
| 40 | 0.19916 | 0.26494 |
| 45 | 0.21987 | 0.23175 |
| 50 | 0.23225 | 0.26035 |
| 55 | 0.21215 | 0.22072 |
| 60 | 0.20399 | 0.21300 |
| 65 | 0.45479 | 0.47561 |
| 70 | 0.40344 | 0.39434 |
| 75 | 0.35965 | 0.34733 |
| 80 | 0.31551 | 0.36405 |
| 85 | 0.29890 | 0.29983 |
| 90 | 0.21763 | 0.27015 |
| 95 | 0.20138 | 0.26998 |
| 100 | 0.18623 | 0.28869 |
| 105 | 0.20279 | 0.24070 |
| 110 | 0.21818 | 0.26197 |
| 115 | 0.17771 | 0.21747 |
| 120 | 0.19857 | 0.22580 |
| 125 | 0.48141 | 0.45722 |
| 130 | 0.36660 | 0.38983 |
| 135 | 0.35126 | 0.34159 |
| 140 | 0.28957 | 0.30131 |
| 145 | 0.29694 | 0.31208 |
| 150 | 0.26904 | 0.29004 |
| 155 | 0.23041 | 0.24333 |
| 160 | 0.21406 | 0.25301 |
| 165 | 0.17993 | 0.22737 |
| 170 | 0.19029 | 0.23483 |
| 175 | 0.15437 | 0.18018 |
| 180 | 0.16475 | 0.19751 |

Table 2.: Distance function evaluation of consecutive extracted features, with and without enabling the prediction module. Values range from 0.0 (perfect match) up to 1.0 (maximum error).

## A.3 RANDOM NOTES TEST

| Time (s) | Prediction enabled | Prediction disabled |
|---|---|---|
| 5 | 0.47719 | 0.54897 |
| 10 | 0.57631 | 0.49829 |
| 15 | 0.47199 | 0.34113 |
| 20 | 0.48334 | 0.44784 |
| 25 | 0.54402 | 0.50289 |
| 30 | 0.48667 | 0.50604 |
| 35 | 0.48002 | 0.47199 |
| 40 | 0.44008 | 0.38399 |
| 45 | 0.47043 | 0.47111 |
| 50 | 0.42396 | 0.56908 |
| 55 | 0.40987 | 0.55393 |
| 60 | 0.48002 | 0.52816 |
| 65 | 0.45373 | 0.53344 |
| 70 | 0.43185 | 0.45162 |
| 75 | 0.53822 | 0.40782 |
| 80 | 0.54133 | 0.44289 |
| 85 | 0.55816 | 0.42771 |
| 90 | 0.38731 | 0.39597 |
| 95 | 0.47162 | 0.35608 |
| 100 | 0.38035 | 0.40795 |
| 105 | 0.42640 | 0.44001 |
| 110 | 0.49737 | 0.51428 |
| 115 | 0.41810 | 0.45935 |
| 120 | 0.53988 | 0.46676 |
| 125 | 0.52511 | 0.49482 |
| 130 | 0.42820 | 0.47626 |
| 135 | 0.47389 | 0.38746 |
| 140 | 0.40711 | 0.53177 |
| 145 | 0.51273 | 0.57387 |
| 150 | 0.39961 | 0.53746 |
| 155 | 0.50590 | 0.45301 |
| 160 | 0.38177 | 0.48272 |
| 165 | 0.48339 | 0.47916 |
| 170 | 0.42094 | 0.54564 |
| 175 | 0.40185 | 0.48242 |
| 180 | 0.51601 | 0.43897 |

Table 3.: Distance function evaluation of consecutive extracted features, with and without enabling the prediction module. Values range from 0.0 (perfect match) up to 1.0 (maximum error).