Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Music improvisation in Python using a Markov Chain Algorithm

William Clinton
Supervisor: Assistant Prof. Glenn Strong

May 3, 2019

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Masters in Computer Science (MCS)

Declaration

I, William Clinton, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Signed: _____ Date: _____

Summary

This dissertation deals in the area of music generation in term of algorithmic composition. Using the principle of Markov Chains, a method of improvisation is successfully implemented into a Python system. The method implemented has previously been used in a MatLab program and this dissertations aim was to see if it was possible to emulate the MatLab algorithm in Python. The reason for using Python was to bring automation into the process. In the MatLab version every aspect of the system was hard coded.

In the Python system the music being inputted for improvisation was create in the MIDI format which could be processed by Python using the 'Mido' library. Processing music in this fashion immediately adds automation to the method as any random MIDI file can be read and analysed in the same way.

The music improvised is based off the backing chords for select bars of music (9 to 12). Having the backing chords hard coded into the system, the improvised bars could be created with little issue while following the original improvisational method. Certain aspects of this method needed some modification to work with MIDI instead of the data representation used in the MatLab program. Music notes were manually created using sine wave value which differed depending on the given frequency of the unique note in question. Using this method, there was little to no automated and no room for scaling the system. Meaning to get the full range of notes, they would all have to be manually defined. Compared with using the MIDI format, all notes are defined with a numbering system between 0 and 127 which of course is much simpler to work with.

There are other aspects of the improvised method that need further research if the entire method is to become a viable tool for an artist to use. The algorithm being investigated uses the principle of a Markov Chain to create new notes. The duration and the pitch of each notes is decided using this method. The process works well to create the music autonomously but unfortunately due to the simplification place on the data used, the Python system can not work fully without human interaction.

Expanding the simplified areas would result in a much more complex improvisation system which would need many more moving elements. Additional techniques of neural nets and a backing chord identification method would be needed to give the program a complete set of accurate data set.

This dissertation is successful and does in fact prove the concept being investigated. This confirmation opens up a new viable area for further investigation in the field of music generation. This specific aspect of music generation has not been thoroughly explored which means that there many opportunities for further development.

Acknowledgements

- I would like to express my gratitude to my supervisor Assistant Prof. Glenn Strong, for his continued support over the year. Without his advice and guidance, this dissertation would never have been finished.
- My brother and domain expert who worked with me through the creation of the dissertation.
- My fellow classmates who got me through this year.

Contents

List of Figures

Nomenclature

MIDI	Musical Instrument Digital Interface
WAVE	Waveform Audio File Format

1 Introduction

1.1 Music Generation

For years computers have been used to create music. They have been successful in this venture and the popularity of using computers in this field does not look like it will slow down any time soon. Music is an integral part of the lives of people. Every year there are more and more methods to create music development. For TV, film, video games etcetera, all of which use technology in their production processes. Music evokes that emotional response that few things can achieve.

This dissertation will focus on algorithmic music composition. It is a field of research that has been in the works since the 1950s. There are many areas of algorithmic composition to look at, however, this dissertation deals specifically with the field of music improvisation.

As stated above, music composition using computers has become the norm. No longer are musicians simply writing out their music on sheet music paper, manually changing things with pencil and rubber in hand. Music is digitised and displayed with computers. Changes to the music can be done with a press of a button and exported in a music file that can be sent anywhere, then opened in a different software and modified there.

1.2 Motivation

The main motivation for this dissertation is to develop a framework to investigate the possibility of using a Markov Chains algorithm for music improvisation. Markov chains have been falling in popularity in recent years and this dissertation will delve deeper into the subject of Markov Chains in the background section.

This dissertation aims to create a study that would involve something engaging both sonically and academically.

Computer generated improvisation is an aspect of music generation that artists add to their pieces to give them interesting ideas and create uniqueness. Improvisation has been looked

at extensively but there are so many places the area can still go to and need to be explored.

The motivation for this is not just academic but more curiously. If new methods can be created that one day could lead to the creation of beautiful sounding music, is it not worth looking for them? Not every artists has the know-how to create programs which they can use to compose music. Some do exist but they are few and far between.

This dissertation is working from an algorithm previous devised in a piece of academic work (1). The dissertations focus is to automate the process so that any person can use the system to create music of their own. It is not known if this new algorithm can easily transfer to a Python system and if additional methods need to be added.

The aim of this dissertation can be simply defined as research for proof of concept. If the concept is usable, the system can be furthered explored, otherwise this area of music generation can be ignored and focus can be put towards other areas.

Initially the work previously done would be recreated in such a way that there is less manual input and the project can move in the direction of no human interaction except for inputting music files.

This dissertation is not trying to work out the mathematical methods to create good music. It is seeing if mathematical methods can be combined to create a product that artists can use as a tool in their composing process.

1.3 Dissertation Overview

The structure of this dissertation is as follows: Section 2 provides background information on the topics discussed in this dissertation. It will give you the tools require to understand the area the dissertation is working in. Section 3 will go over the design of the project. This dissertation is inspired from another piece of work as previously stated and the method used will be discussed and the new methods that were used will be introduced. Section 4 is the implementation stage where the project produced is broken down into its different parts and described in detail. Section 5 is the evaluation stage where the output of the finished program is looked at. The strengths and weaknesses are detailed. In section 6 the conclusion of the dissertation is defined and the recommendations for future work are given.

2 Background

2.1 Music Generation

There has been many aspects of music generation done in the field of computer science since the field began in the 1950s. This use for computer sciences has branched out in many different branches.

'The Analysis of Generative Music Programs'(2) is an article that focuses in particular music generation as an art form and what is meant by the term 'generative art'. Margaret Boden outlined a taxonomy of electronic and interactive art devised with Ernest Edmonds (Boden 2007). In their terms, G-art (generative art) is composed of works that are 'generated, at least in part, by some process that is not under the artist's direct control'.(2)

A further sub-category of CG-art refers to specifically computer-generated work, 'produced by leaving a computer program to run by itself, with minimal or zero interference from a human being'.(2)

The final hurdle before we progress to consider real examples is to mention the process versus product debate. Art itself is replete with examples of the claims of a human being for an object's 'artistic status' leading to the grant of that status and a transformation of the bounds of art. Some might see this cultural negotiation as having gone to breaking point in the 20th century. A cogent and passionate defence of the status of a generative program rather than an individual production as an artwork has already been presented by Ward and Cox (1999) (3) in discussing Adrian Ward's Autoshop application. (2)

'AI Method in Algorithms Composition - A Comprehensive Survey'(4) is a survey that gives a comprehensive account of research on algorithmic composition, presenting a thorough view of the field for researchers in Artificial Intelligence. The different methods that will be discussed in this section are the core aspect. If you desire more detail then the original survey has all you will need.

Traditionally, composing music has involved a series of activities, such as the definition of melody and rhythm, harmonisation, writing counterpoint or voice-leading, arrangement or

orchestration, and engraving (notation). Obviously, this list is not intended to be exhaustive or readily applicable to every form of music, but it is a reasonable starting point, especially for classical music. All of these activities can be automated by computer to varying degrees, and some techniques or languages are more suitable for some of these than others (Loy and Abbott, 1985) (5).(4)

For relatively small degrees of automation, the focus is on languages, frameworks and graphical tools to provide support for very specific and/or monotone tasks in the composition process, or to provide raw material for composers, in order to bootstrap the composition process, as a source of inspiration. This is commonly known as computer-aided algorithmic composition (CAAC).(4)

The survey mainly focuses more on techniques, languages or tools to computationally encode human musical creativity or automatically carry out creative compositional tasks with minimal or no human intervention, instead of languages or tools whose primary aim is to aid human composers in their own creative processes. These three scenarios (automated expressiveness, algorithmic sound synthesis and non-linear music) are sparingly mentioned in this survey.(4)

The range of methodological approaches used to implement algorithmic composition is notably wide, encompassing many, very different methods from Artificial Intelligence, but also borrowing mathematical models from Complex Systems and even Artificial Life.(4)

These papers will go over a number of methods to give a overview of the field in general and then it will focus a closer on the methods used in this project. Some of these methods are combined together in different music generation system. In general, one method does not seem to be enough to create a ideal system. Samples of these systems are described later in this section.

Grammar

In broad terms, a formal grammar may be defined as a set of rules to expand high-level symbols into more detailed sequences of symbols (words) representing elements of formal languages. Words are generated by repeatedly applying rewriting rules, in a sequence of so-called derivation steps. In this way, grammars are suited to represent systems with hierarchical structure, which is reflected in the recursive application of the rules. (4)

To compose music using formal grammars, an important step is to define the set of rules of the grammar, which will drive the generative process. The rules are traditionally multi-layered, defining several subsets (maybe even separated in distinct grammars) of rules for different phases of the composition process: from the general themes of the composition, down to the arrangement of individual notes. (4)

Knowledge-Based Systems

Knowledge-based system is used as an umbrella term encompassing various rule-based systems under several different paradigms, with the common denominator of representing knowledge as more or less structured symbols. Since knowledge about musical composition has traditionally been structured as sets of more or less formalised rules for manipulating musical symbols (Anders and Miranda, 2011) (6), knowledge-based and rule systems come as a natural way to implement algorithmic composition. (4)

Markov Chains

Conceptually, a Markov chain is a simple idea: a stochastic process, transiting in discrete time steps through a finite (or at most countable) set of states, without memory: the next state depends just on the current state, not on the sequence of states that preceded it or on the time step. In their simplest incarnations, Markov chains can be represented as labelled directed graphs: nodes represent states, edges represent possible transitions, and edge weights represent the probability of transition between states. However, Markov chains are more commonly represented as probability matrices. (4)

When Markov chains are applied to music composition, the probability matrices may be either induced from a corpus of preexisting compositions (training), or derived by hand from music theory or by trial-and-error. The former is the most common way to use them in research, while the latter is more used in software tools for composers. An important design decision is how to map the states of the Markov chain to musical objects. The simplest (but fairly common) mapping just assigns a sequential group of notes to each state, with the choice of just one note (instead of a larger sequence) being fairly common. (4)

It is also common to extend the consideration of the "current state": in an n-th order Markov chain, the next state depends on the last n states, not just the last one. As a consequence, the probability matrix has n + 1 dimensions. (4)

Many systems use Markov models to generate finite-length sequences that imitate a given style. These systems often need to enforce specific control constraints on the sequences to generate. Unfortunately, control constraints are not compatible with Markov models, as they induce long-range dependencies that violate the Markov hypothesis of limited memory. (7)

The paper 'Finite-Length Markov Processes with Constraints' (7) describes a novel and efficient approach to controlled Markov generation for a specific class of control constraints that 1) guarantees that generated sequences satisfy control constraints and 2) follow the statistical distribution of the initial Markov model.

The main issues to addressed in this paper are 'Zero-frequency problem' which is when a

system reaches a state with no more transition (8). What should the system do at that point? Start again (9). 'End point or drift problem' (10) which concerns the fact that the generated sequence can violate musical constraints holding, e.g., on the pitch range of the melody. Finally 'Control Constraints' which are arbitrary conditions that a user may select, e.g. wanting a improvised selection to end on a certain note. (7)

In general, it is not possible to find such a Markov process because control constraints violate the Markov property, as outlined by [Pachet and Roy, 2011] (11)f. However, when control constraints remain within the Markov scope, which shows that such a model exists and can be created with a low complexity. (7)

'Learning Jazz Grammars'(12) talks about about an educational software tools that can generate novel jazz solos in a style representative of a body of performed work, such as solos by a specific artist. This paper talks about the combination of Grammars and Markov Chain methods.

The approach of the paper is to provide automated learning of a grammar from a corpus of performances. Use of a grammar is robust, in that it can provide generation of solos over novel chord changes, as well as ones used in the learning process. Automation is desired in this paper as manual creation of a grammar in a particular playing style is a labour-intensive, trial-and-error, process. (12)

To construct a Markov chain with meaningful transition probabilities, the system needed a reasonable number of data points for each state. Before building the transition matrix, they grouped similar abstract measures together using the k-means clustering algorithm. (12)

To measure the methods effectiveness at style emulation, the paper set up an experiment to determine whether or not test subjects could match the styles of three prominent jazz trumpet players with solos composed in the style of each player. Which resulted in the system producing convincing imitations of the three artists. (12)

The paper 'Music Generator with Markov Chain - a Case Study with Beatme Touchdown'(13) aimed to generate music for composer and for others at real-time. To read MIDI file complaint to the Standard MIDI File (SMF) and outputs MIDI file too. The melody generator itself would use a Markov Chain/Model. The process was to read existing musics as samples, the program would then calculate the parameters for those musics samples and with those calculated parameters, a melody would then be generated. The aspects looked at in this paper were: pitch, duration, density, intensity and timbre.

The paper desired a further study to be conducted with more complex Markov Chains models coupled with AI algorithms to be able to produce hear-able musics. (13)

'Algorithmic Music Composition Using Dynamic: Markov Chains and Genetic Algorithms'(14) presents a novel method for algorithmic music composition using two techniques employed in

previous computational systems: Markov Chains and Genetic Algorithms. Genetic algorithms are employed as a search method that finds the set of Markov chains that yield the most pleasant sound music

The Genetic algorithm guiding composition is described as a system that consists of two different sections, a lower level system bases of Markov Chains that guide the selection of next pitch, rhythm and chord, based upon the current pitch, rhythm and chord. Markov chains consist of a sequence of state vectors which give the probabilities of transitioning from one state to another (DeFranza J., Gagliardi D., 2009) (15). In the case of this system, the current chord, pitch, and note duration (rhythm) were manipulated using a Markov Chain.(14)

Chord progressions are handled simply in this paper. The current chord is simply based upon the previous chord. However a simplification is made. In a traditional jazz lead sheet, the chord can obviously change at any time. The system is restricted to one chord per measure. Not done because it is impractical or difficult, but instead it allows the system to test its basic functioning of its algorithm without over-complication.(14)

The genetic algorithm used is inspired by the idea "survival of the fittest". If possible solutions can be represented in a search space as genomes (sets of Markov Chains that direct composition), they can then compare the fitness of a set of individual solutions and choose the best ones to continue on and create a new set genomes. If they have chosen correctly, they would expect our next "generation" to have a higher overall fitness. Automatic fitness functions that rate composition based upon training data exist. This implementation differs as the user is the fitness function and shapes the next generation directly.(14)

The system in this paper restricts its rhythm to only a set of common rhythms. Generally, in modern music the duration of a pitch is never restricted in any way. It can, in fact take on any real value number of beats. However, rhythm generally appears as a small set of different possibilities. The paper takes the liberty in restricting possible rhythms in this system which aides in simplicity. The system described in this paper was made to test a method and was not meant to be perfect. This meant that there were simplification made which would not be acceptable otherwise. (14)

One of the most commonly cited computer generated systems is the Hiller and Isaacson's (1958) Illiac Suite (16), a composition that was generated using rule systems and Markov chains, late in 1956. It was designed as a series of experiments on formal music composition. During the following decade, Hiller's work inspired colleagues from the same university to further experiment with algorithmic composition, using a library of computer subroutines for algorithmic composition written by Baker (also a collaborator of Hiller), MUSICOMP (Ames, 1987) (17). This library provided a standard implementation of the various methods used by Hiller and others.(4)

The paper 'Modeling music as Markov chains - composer identification'(18) described research for the development of a model that enables a computer to symbolically examine a piece of classical music which is never exposed to it, and tell who is the composer.

Artificial Neural Networks (ANNs)

ANNs are typically used as a machine learning method, using a set of examples (input patterns) to train the network (i.e., to set the weights of the connections between neurons), in order to use it to recognise or generate similar patterns. Effectively, this means that neural networks need a pre-existing corpus of music compositions (all of them in a very similar style, generally); therefore they can at most imitate the style of the training examples. Most papers use a supervised learning approach, meaning that the examples in the training set are associated with a signal, and the ANN learns this association. An important aspect of ANN design is the modelisation of musical composition, that is, the mapping between music or music notation and the inputs and outputs of the network. Another important aspect is the way in which compositions are fed to the ANNs: they may be presented as temporal patterns in the network inputs, which are usually windowed in segments, but in some cases they are fed at once (as wholes) to the ANNs. These implementations do not scale well, due to big ANNs need to model long compositions.(4)

In the paper 'A Hybrid Neural-Markov Approach for Learning to Compose Music by Example'(19) a combination system of Markov chains with Neural Nets is used to try and bridge the gap of from manually composed music next to a method that only uses Markov chains. Markov chains can be great for reading patterns in music, when it comes to high-order systems. They can not great at connecting the patterns to create cohesive pieces.

An example of this method is thus (19). This systems approach uses musical training sequences to learn to compose, thus incorporating human-composed musical patterns in an algorithmic composition technique. Looking at other methods for music generation, a programmer has to manually encode the basic music theory into the system so it can create music. In contrast, this work does not provide any music theory rules to the program. Instead the system learns them from the training sequences. The aim of this approach is not only to capture music theory rules and the exceptions to the rules that create variety in a composition. (19)

This approach learns to compose music by example. It first extracts patterns from the training sequences. A pattern is a sequence of intervals that may start at different baseline pitches, and has different baseline durations. It then constructs a Markov chain in which each state represents a pattern and transitions between states which represent allowable sequencing of patterns. It trains a neural network to select the baseline pitch and duration for the pattern of

intervals represented by each state of the Markov chain. It uses this model to compose novel musical sequences. This systems Markov model could be viewed as a higher-order Markov model, since it outputs sequences of notes at each state. (19)

The system had used a probabilistic emission function that depends only upon the current state. However, this resulted in a disadvantage of not accounting for the pitch and duration of the previous pattern. In the completed work, it has chosen to learn the distribution of baselines at the current state conditioned on those of the previous state using a neural network. The neural network then models the conditional distribution on pitch and duration baselines for each state, given the output of the previous state. Note that this implies that our Markov model is first-order. (19)

In regard to the results of this paper, the main motif, or theme, in the original pieces occurs multiple times in the output, indicating that the approach was successful in extracting and using the signature patterns of pieces. The method successfully maintained smooth flow between consecutive notes rather than making large interval jumps. Finally, the pieces respect the common music theory concept of key, in that most of the notes are in the key, with a few accidental notes to create variety. (19)

Evolutionary Methods

Most evolutionary algorithms (EAs) approximately follow a common pattern: a changing set of candidate solutions (a population of individuals) undergoes a repeated cycle of evaluation, selection and reproduction with variation. The first step is to generate the candidate solutions of the initial set, either from user-specified examples or in a more or less random way. Each candidate is then evaluated using a fitness function, a heuristic rule to measure its quality. The next phase is selection: a new set of candidate solutions is generated by copying candidate solutions from the old one; each candidate solution is copied a number of times probabilistically proportional to its fitness. This step decreases the diversity of the population, which is restored by applying (to a fraction of the candidate solutions) some operators designed to increase the variation (for example, mutation or recombination operators). These steps are applied in an iterative fashion; as a result, best and mean fitness gradually tend to increase.(4)

Cellular Automata

A cellular automaton (CA) is a discrete (in time, space and state) dynamic system composed of very simple computational units (cells) usually arranged in an ordered n-dimensional (and potentially unbounded) grid (or any other regular tiling). Each cell can be in one of a finite number of states. In each discrete time step, each cell's state is deterministically updated, using a set of transition rules that take into account its own state and its neighbours'

states. Although this definition can be generalised in multiple ways, it represents a good first approximation.(4)

2.2 Artist focused Methods

'Beyond the Cybernetic Jam Fantasy The Continuator'(20) is a paper which describe a system where its sole purpose is to be for musicians. The system is called 'The Continuator' (20) that uses techniques from interactive and automatic learning systems. This system learns and plays interactively with a user. The system plays in the style of the user by analysing the music as it is being played. The system works in real-time. This system was used to explore a new avenue for artists to compose.

2.3 Music Improvisation

This is masters paper called 'Music Improvisation using Markov Chains'(1) from the university of Maastricht in the Netherlands which deals with adding improvisation into an existing piece of music using Markov Chains. The aim of this paper was to create new improvised bars for a piece which were recognisable from the old piece. As there are many methods of improvisation this paper decided to deal with only Baroque style music. The piece of music used in this piece was "Ode to Joy" by Ludwig van Beethoven.

This paper inspired this dissertation as the field of improvisation using Markov chain has been exhausted to an extent in recent years and this paper looked at an area that did not have any comparisons. The method of implementation was using the programming language MatLab. This describes the method of improvisation in great detail but it is clear that the product of this paper was merely a concept. There code create only worked for one piece and could not easily be modified to use multiple different pieces.

2.3.1 MatLab

MatLab is a programming platform designed specifically for engineers and scientists. The heart of MatLab is the MatLab language, a matrix-based language allowing the most natural expression of computational mathematics.(21)

The MatLab language, apps, and built-in math functions enable you to quickly explore multiple approaches to arrive at a solution. MatLab lets you take your ideas from research to production by deploying to enterprise applications and embedded devices, as well as integrating with

Simulink (block diagram environment for multidomain simulation) and Model-Based Design (Desktop version capabilities).(21)

2.3.2 Backing Chords

In information described in this section was learned by working with a domain export in the first of music composition. They supervised the design before the code for this Python program was created.

In basic terms a backing chord are the harmonic interval progressions a piece of music gravitate towards. Backing chords are governed by harmonic note principles, rhythm and key of a piece. If we take the key of C major there are seven notes in this scale (C, D, E, F, G, A, B) and twelve intervals (C, C, D, Dflat, E, F, F...). Look at figure 2.1 which display the C Major backing chord. Each note has a tonal significance to C and are labelled thus as minor, major, augmented or diminished (C major,D minor,E minor,F major,G major,A minor,B diminished).

This is based off the interval between each note and C (i.e. the distance between each note and C). Triad chords were typically used in Barque style music and as intervals are built up on the relationship between two notes triads means the built up from three. Depending on the interval distance a different sound quality or tonality is achieved. C,E,G is the chord of C major and if we count the distance between the different notes from C we get four and seven. From D minor we get three and seven. You can see the interval of the middle notes decides its sound.

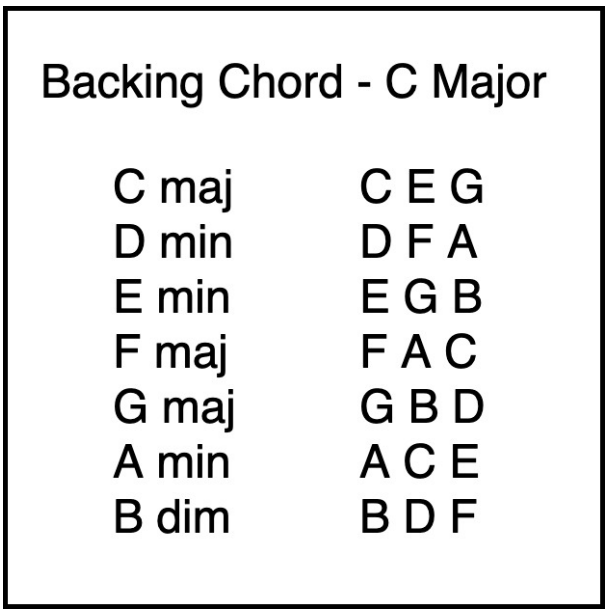

Figure 2.1: Backing chord example for the key of C Major

When using backing chords the arranger must take the key of the piece into consideration then look at the notes in a given melody line. Depending on what comes before and after a

particular note she/he must see if he can create a chordal progression and there are no rules strictly speaking, yet in Barque music one would typically see 'C, F, G, G' or 'C, C, F, G' and rarely anything more sinister 'C, F, Bflat, D' (Which involves key changes) for example.

However, backing chords were generally decided by the melody line. For example in "Ode to Joy" in bar one we see the chord G. In the melody line we have B,B,C,D and if we add a G backing chord and dismiss the C note we get a nice full G chord sound. When searching for what chord to use one looks for the many notes in common with a chord, then usually a nice progression can be obtained but a good knowledge of music theory, as you can see, is needed to carry it out. This above details simple backing chords only yet there are first, second and third inversion chords which consist of the same notes of the simple chords just in different orders: C,E,G would become E,G,C in its first inversion.

This gives a brief insight into the world of backing chords which encompasses music theory and shows why a computer generated algorithm can be an advantage in this complex system.

2.4 MIDI

The Standard MIDI File (SMF) is a file format specifically designed to store the data that a sequencer records and plays (whether that sequencer be software or hardware based).(22)

This format stores the standard MIDI messages plus a time-stamp for each message. The format allows saving information about tempo, pulses per quarter note resolution, time and key signatures, and names of tracks and patterns. It can store multiple patterns and tracks so that any application can preserve these structures when loading the file.(22)

MIDI messages are broken up into three subcategories. Channel Voice Message which instruct the receiving instrument to assign particular sounds to its voice Turn notes on and off Alter the sound of the currently active note or notes. Channel Mode Messages which determine how an instrument will process MIDI voice messages. Finally System Messages which carry information that is not channel specific, such as timing signal for synchronisation, positioning information in prerecorded MIDI sequences, and detailed setup information for the destination device.(23) (24)

2.4.1 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python is simple, easy to

learn syntax emphasises readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.(25)

Mido

Mido is a library for working with MIDI messages and ports. It's designed to be as straight forward and Pythonic as possible. This library is specifically designed to work with MIDI files so they can easily manipulated using the Python language. This library is continuously updated to work with any updated to the MIDI standard.(26)

3 Design

This section is going to overview the process that went into designing and finally implementing the method for this project. As there was a previous project to work from there are many aspects to look at and see how the design needed to change for this project.

Figure 3.1: 16 bars of Ode to Joy

The method for designing an improved version of the the original method (1) required a deep understanding of the paper. The initial step was to reproduce the work already created. Doing this would allow for a baseline to be established and therefore allow for results produced at the end to be compared. This required obtaining the original MatLab code from the paper. However, a pristine version of the code was not easily accessible. In the paper itself there was a version of the code but there was no repository to acquire the full version.

The code on the paper was imperfect and lines of code were cut short. An attempt was made to reach out to the author of the paper and contact the supervisor of the paper as this

happened to be a masters paper. Unfortunately, no response was received and completely understanding the algorithm defined in the paper became the force behind this dissertation. The method was what inspired this dissertation and therefore, having a firm understanding of the original methodology was enough to continue with the system.

The original system created the improvised section for the four sixteen bars of the music piece 'Ode to Joy' by Ludwig van Beethoven. The system would replace bars 9 to 12 with new improvised bars. These bars would be based off the backing chords of the original bars and by doing so this would allow the new bars to be still recognisable compared with the original. If you look at 3.1 you can see this information clearly.

Using MIDI formatted files allow the music to be read straight from the file and the different notes played can be isolated and viewed. Methods for doing so are straightforward and do not require many lines of code. For this reason, MIDI was the chosen to be used in the Python system.

3.1 Framework

The original system created the piece of music manually in MatLab. This was done by getting the frequency of each note that was going to be used in the program. For example, the music note A4 (Note A in the fouth octave) has a frequency of 440Hz. These frequencies were then used to create the notes which were created in the form of a sine wave. Each note would then be created at that very instant of insertion with an assigned duration. These notes would then be joined together for each bar depending on the bars specific structure. The bars would then be put into a vector which would at last be converted into a WAVE file (.wav).

Every aspect of this above MatLab systems was manually created, even the improvisation section. The bars selected for the improvisation had unique code. This meant that for every chord (e.g. the last bar contains four backing chords), there was repeated code in the sense that each method worked in the same way but differed in the values that they dealt with.

This project required the system to have a level of automation that can not be implemented in MatLab. Python was decided on as it is a lightweight programming that has all the benefits of object oriented programming. Using the Python library 'Mido', music could read from any music input as long as the file imported was a MIDI file. This would immediately allow a level of automation that the previous work did not have.

Using the MIDI format (.mid), music notes are assigned to number values. There is no need to look at the mathematics behind the sound of the note itself. We can look at a note in its simplest form which removes the complex understanding needed in the MatLab code. MIDI

files can be created for the output. This is much easier to deal with compared to WAVE files which can grow in size depending on the music piece being created.

The MIDI files used in this project were created in the software 'MuseScore'(27) which is a free notation software which allows for the creation of MIDI files. The setup of MIDI is based off how this software creates them. There was no need to delve into different this as MIDI files can be read by most modern music software systems.

3.2 Backing Chords

The improvisation was based off known backing chords . The triad of each chord was the focus range for the new notes as Baroque improvisation was many based off it (1). The first, third and fifth notes of a scale were more likely to be played and the second and fourth were less likely. The probability for these notes on the Markov Chain reflected this showing higher probabilities for the triad notes. The reason for the second and fourth notes were to allow for a little more variation in the music and would allow for sequential movement of the music which is seen in the original piece.

Knowing the sequence of backing chords will benefit the new system and allow for quicker music generation. These values do not change and can be considered constants. Automating the system for creating notes for each backing chord will not be as straightforward as could be hoped. When using MIDI, notes are thought of as different numbers. Depending on the key signature of a piece the exact note number values will be different as some keys have accidentals (flats/sharps/naturals). If you look at figure 3.2 you can see an example how accidentals work. The third instance of B note in the sequence gets changed to B flat which is not in the key of C Major and then the note immediately get turned back to a regular B note.

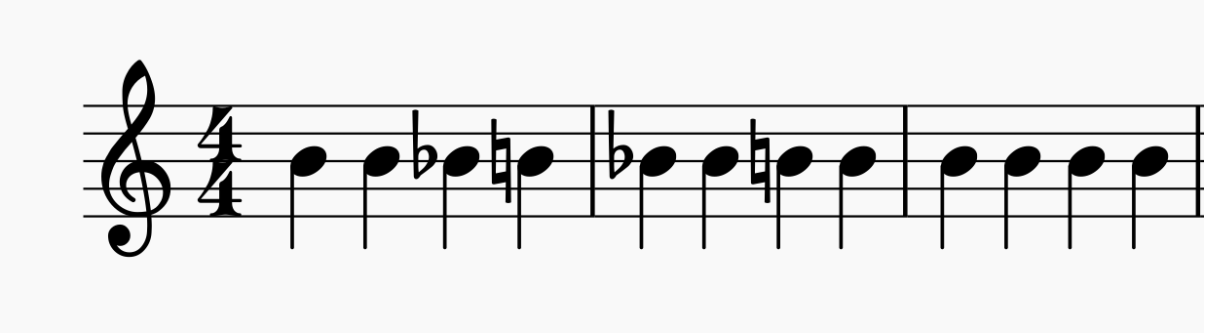

Figure 3.2: C Major key - Repeated B notes which accidentals

As this project is a proof of concept instead of a shipping product the set of notes used in the system were set to a number of known notes and there is no reason to create; a method to

read the MIDI file, identify the key signature and determine the appropriate notes. Working from the original project the range of the notes used for each backing chord were set and therefore were hard coded.

The sequence of backing chords for bars 9 to 12 were defined as 'D, G, D, G, D, B, Em, A, D, G' which can be seen in 3.1. Each unique chord in this list had a corresponding unique method which was used to create the improvised music. In Python this method could be streamlined so that the improvisation function could be reused and therefore reduce the size and complexity of the system, this would allow for more autonomy.

3.3 Note Duration

The tick value was assigned to the sequence. This varied depending on the number of backing chords in the bar. Each bar had the sum total of four ticks. For the first three bars each bar had two tick values but the last bars backing chords only had one (again look at music figure 3.1). Each tick represents one crotchet note value. This equals one beat of the bar. The number of beats in a bar is dependent on the time signature. The time signature for 'Ode to Joy' is four four or common time which means that there are four crotchet values in each bar. Four crotchets equals four ticks. The notes used in the improvisation were the semiquavers (0.25), quavers (0.5), dotted quavers (0.75) and crotchets (1). This is displayed in the figure 3.3

Note	Beat Value	Tick Value
	1	1
	0.75	0.75
	0.5	0.5
	0.25	0.25

Figure 3.3: How the note duration's are seen in MatLab

When using MIDI the method of setting duration is much more difficult. A single note needs two MIDI messages to define its length. To start a new note there is its 'note on' message and the note ends or ceases to sound with its 'note off' message.

'note off' messages are not used in the MIDI format for the files using in this dissertation. Instead 'note on' with a velocity of 0 is used instead. Velocity indicates how hard the note being play is being hit. The MIDI files were created in 'MuseScore' and the format was done in that manner. For the duration of this dissertation 'note off' messages will be described so to stop confusion. Check figure 4.1 to see how 'note on/off' messages work.

Each message had a duration element which would tell the system when the message would come into effect after the last message. These messages are sequential and the order matters. Each message would start after a certain number of ticks. This number would depend on the message that came before it. The exact start time of a note is determined on the cumulative duration of all past messages.

One of the main difference between MatLab and MIDI is the length of a bar. One bar in MIDI is 1920 ticks compare to four in MatLab. That means that 1 beat in the bar is worth 480 ticks in MIDI. Each different note has a unique 'note on' duration and a 'note off' duration. Each 'note on' and 'note off' duration needed to be identified and the method used in the original paper would need to be modified to reflect the difference in duration handling. See figure 3.4 to see how the 'note on' and 'note off' tick values come together to represent each type of note.

Note	Beat Value	Tick 'note on' Value	Tick 'note off' Value
	1	455	25
	0.75	341	19
	0.5	227	13
	0.25	113	7

Figure 3.4: How the MIDI note durations are seen in Python

The method goes through each backing chord separately which are in while loops which only end depending on the number of ticks assigned to the note. The improvised sections continue until the correct number of ticks is reached and the next chord starts and so on until the improvisation is complete.

Using the Markov method the duration is based off the last note played which is set up randomly in the beginning. Once the generation of the improvisation has started the system works autonomously. Different probabilities exist depending on the last notes duration. These probabilities are normalised, meaning that they sum up to 1. A random value is needed to

transition to the next state or in this case the next duration. Without the random value the Markov method could not work. When the random number is generated, it is compared with each value in the probability vector and against the remaining ticks. This makes sure that each bar stays with their tick value limit.

3.4 Note Pitch

The pitch for a note is selected in a similar manner to the duration method. There are set probabilities which govern these transitions. Depending on the current pitch for a certain backing chord there are different probabilities. The probability vector is isolated, a random value between zero and one is created and using this value the next state or pitch value is identified. This method continues as long as the duration method continues. The code displayed in the original paper is not implemented correctly as the current pitch is always set when a chord section is entered. See figure ?? which shows an segment of the original MatLab code with this bug. This means that when entering the D chord section, the D4 chord is always the last pitch which means that the probabilities used do not differ at all which means that the system is not following the Markov method.

```
if strcmp(current_chord,'D ')
    current_pitch = D4(L);

    tmpitch = [4/10 1/10 2/10 1/10 2/10;4/10 1/10 2/10 1/10 2/10;4/10 1/10 2/10 1/10 2/10];
     % create the vector that will be multiplied with the transition matrix,
     % such that only the row of the current note will be left

    if current_pitch == D4(L)
        multiplier2(1,1) = 1;
     elseif current_pitch == E4(L)
        multiplier2(1,2) = 1;
     elseif current_pitch == Fs4(L)
        multiplier2(1,3) = 1;
     elseif current_pitch == G4(L)
        multiplier2(1,4) = 1;
     elseif current_pitch == A4(L)
        multiplier2(1,5) = 1;
     end
```

Figure 3.5: Segment of original MatLab code with bug with the current pitch value

The Python project could handle the set probabilities with ease and setting the notes needed for a specific backing chord could be attached to the MIDI messages. The note pitch is the important factor for creating the note, it is what links the 'note on' and 'note off' messages. The duration creation is the most complex part of creating new notes. Pitches are straightforward in comparison. The same method from the original paper will be implemented in the

new project but instead there will not be the grave error from before which will mean the system will work with the Markov method.

4 Implementation

This section will go through how the system was implemented and gives a detailed description of how the methods were created and how the work together to create the desired output.

4.1 Getting Started

4.1.1 Handling Midi Files

The main objective of this system is to read a selected MIDI file and read through the information and replace the specified bars with bars of improvisation. The modified piece be outputted in a new MIDI file. My copying over the old file and replacing select messages, the new file could be created with little setup. This meant changes to the original piece would be as minimal as possible.

```
Williams-MacBook-Pro-3:midoTests williamclinton$ python readMidi.py
Track 0:
<meta message time_signature numerator=4 denominator=4 clocks_per_click=24 notated_32nd_notes_per_beat=8 time=0>
<meta message key_signature key='G' time=0>
<meta message key_signature key='C' time=0>
<meta message set_tempo tempo=500000 time=0>
control_change channel=0 control=121 value=0 time=0
program_change channel=0 program=0 time=0
control_change channel=0 control=7 value=100 time=0
control_change channel=0 control=10 value=64 time=0
control_change channel=0 control=91 value=0 time=0
control_change channel=0 control=93 value=0 time=0
<meta message midi_port port=0 time=0>
note_on channel=0 note=71 velocity=80 time=0
note_on channel=0 note=71 velocity=0 time=455
note_on channel=0 note=71 velocity=80 time=25
note_on channel=0 note=71 velocity=0 time=455
```

Figure 4.1: Midi messages as seen from the terminal

To do this the system needed to read through every message in the MIDI file and keep track of the number of bars. Knowing the time signature made sure that the system knew how long a single bar needed to be. The constant 480 is the set number of ticks for a crochet beat and by looking at the MIDI files meta message the number of crochet beats per bar could be set. This value was then check against the cumulative time of the file. This allowed the system to

recognise which bar was being read. Knowing this, the selected bars for improvisation could be replaced with newly created bars made up of improvised music. If you look at figure 4.1 you can see how the MIDI file structure.

4.2 Improvisation

4.2.1 Setup Chords

This system set hard coded elements to streamline the process of creating automation in the previously designed system. The backing chord data was set in global variables. Each backing chord value had a set of notes that could be used. These values were set in a 2D array. The system would extract the required data for a specific backing chord automatically and input it into the create message function which would create the desired messages.

The duration allocated to each backing chord was set at this stage. This made sure that the message created would only use the time given to it. This again was past into the create message function.

The pitch probabilities were also set at this stage. These probabilities were taken from the original paper and set to the same backing chords. This values were deemed hard coded and serviced as the starting state for probability for these backing chords. The probabilities themselves did not change over the implementation of the system.

4.3 Create Messages

This function creates all the messages for a specified backing chord. The parameters past in were the backing chord number value and the number of ticks (duration) for the backing chord. This function is used for every backing chord and works perfectly in that purpose. There is no unique element to the code that is only used for a certain backing chord. This means that this function can replace any messages for a given backing chord. It does not work on the bases of one bar.

4.3.1 Duration

As the create message function created new messages, the allocated ticks remaining is decremented. This value is what keeps the create message function from ending. Only when ticks remaining equals zero will the function end. This means that there can be no margin of error when it comes setting the duration for a given messages.

As notes are broken up into two separate messages (as mentioned in the previous section) in MIDI placing the correct tick value in the correct message is vital. As each new message needs to know how long it needs to wait until it can be played. There needed a system to remember the last note that was create and place the 'note off' duration of the last note in the duration of the newly create note. The 'note off' message for the next note would then have the duration for the new notes length. The system would then hold onto the 'note off' length of the note if a note is created afterwards. This method continues for the whole message creation process. Check figure 4.1 to see how the 'note on' values are set.

When the first note for improvisation is being created, the last note in the bar that preceded the improvisation bar needs to be identified so that the first note starts at the correct time and there is not an issue. Initially the first duration of the first note was set to zero as in the testing phase the system only created the improvised section.

Setting the duration probabilities is slightly more laborious compare to MatLab which dealt with the data in matrices and vectors. This meant that required data could be isolated with simple mathematics. In the python project these values were set manually to give the same effect as the MatLab method.

4.3.2 Pitch

Once create message is called, the backing chord note values are set. This means that the five possible notes that can be made into notes are set. The function sets the pitch using the Markov method by initially starting the process off as working from the first note in the set of five notes. This note for the backing chord are in the same octave range of the original bars. Each of the five notes have a different set of probabilities.

For example, the first note has a higher probability of going to itself, its third note or its fifth note. If by some chance the note decided on is the second note when the system comes around to create a new note, the range of probabilities will be different to the first set so that the next note will be more likely the third or fifth note in the range which work better for the set backing chord.

Once the notes are selected they are used in the create message section and used to create the new notes. Once the function is called again to figure out the next note in the backing chord sequence, the probability for the give note is looked at and the process starts over until the create message function ends.

4.4 Difficulties

The main problem here was replace the specified bars perfectly. Using MuseScore to look at the music generated made it easy to identify when a problem was occurring. Knowing that the piece going into the system had sixteen bars at with a time signature of four four, if the output was faulty MuseScore would try and compensate and make the music make sense.

For example, when the music inserted was not the correct length MuseScore would add break symbols automatically to keep the shape of the piece. The sheet music looked fine but was not correct. Other times the time signature would change to three four which meant the number of bars would increase. The problem here is that it was not exactly clear where the issue was coming from and thus there needed multiple read through of the code to catch the bug.

There was an attempt to automate the process of identifying the correct range of five notes for the range of each backing chord. In the MatLab implementation they were hard coded and using Python it seemed that it could be possible to automate the generation different notes for every backing chord that could possibly come up in the key of G Major.

While investigating this method, it became clear that this method would not be straightforward. MIDI only deals with the number that represent the notes on the sheet music. It does not care if a note deviated from the specified key signature. This meant that the key signature would need to be read and the space of the octave notes would need to be set. Looking at figure 4.2 you can see that the notes do not have the same intervals between them. This is due to the various sharps and flats that are between normal notes.

Due to the method not having a clear method of identifying the intervals between MIDI notes for any key signature, this function was not developed further as it would not help with automated the system. The function could have been hard coded to deal with backing chord in G Major but it seemed unnecessary as there were already existing hard coded elements.

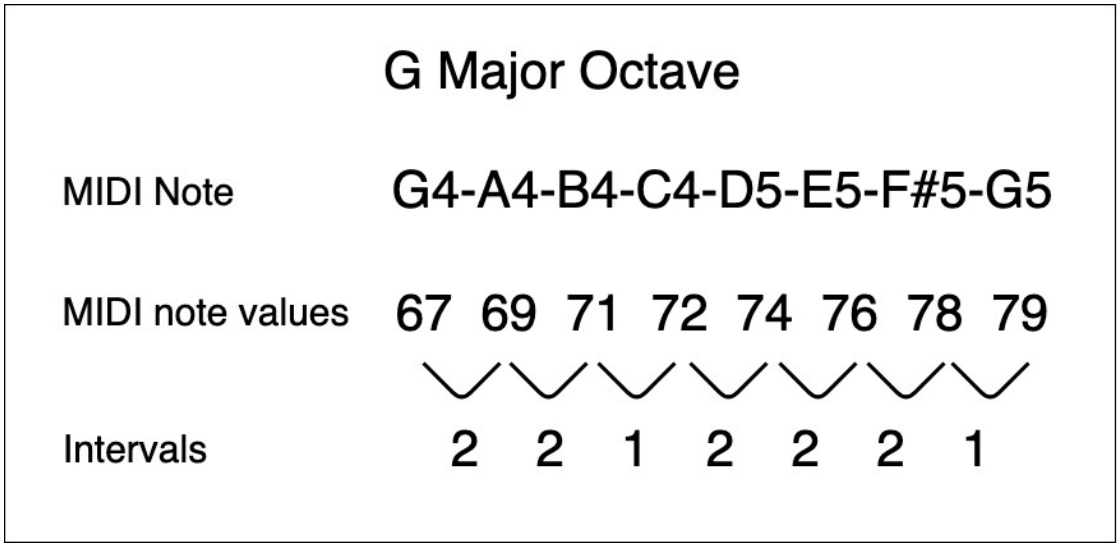

Figure 4.2: The octave of G Major in Midi

5 Evaluation

This section will describe the finished python system created. It will detail the output of the system implemented. Compare the system with the old and see how successful the system was to automate the various processes from the previous work. There was no experimentation of to evaluate the usability of the system. The number of automation implemented is how the dissertation looks at success.

5.1 Results

The system itself was successful in recreating the system that was implemented previously (1). The 16 bar exact of 'Ode to Joy' can be read into the system and a new file can be created which the selected bars for improvisation have been changed. If you compare figure 5.1 5.2 you can easily see how bars 9 to 12 are replaced with new improvised bars while the rest of the music has been left alone.

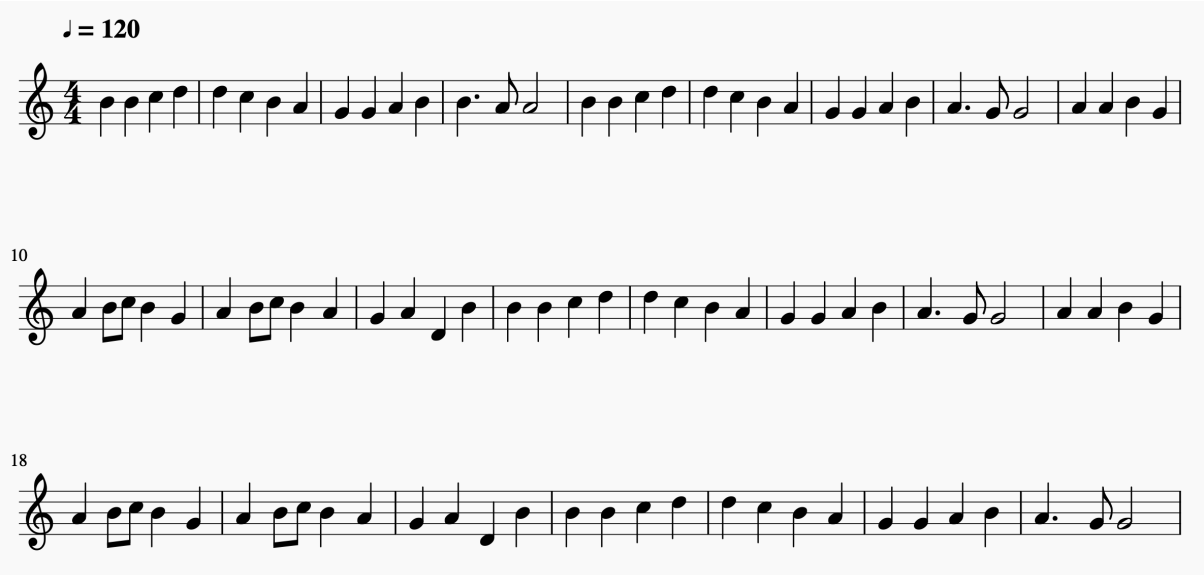

Figure 5.1: Unaltered MIDI File - Read in MuseScore

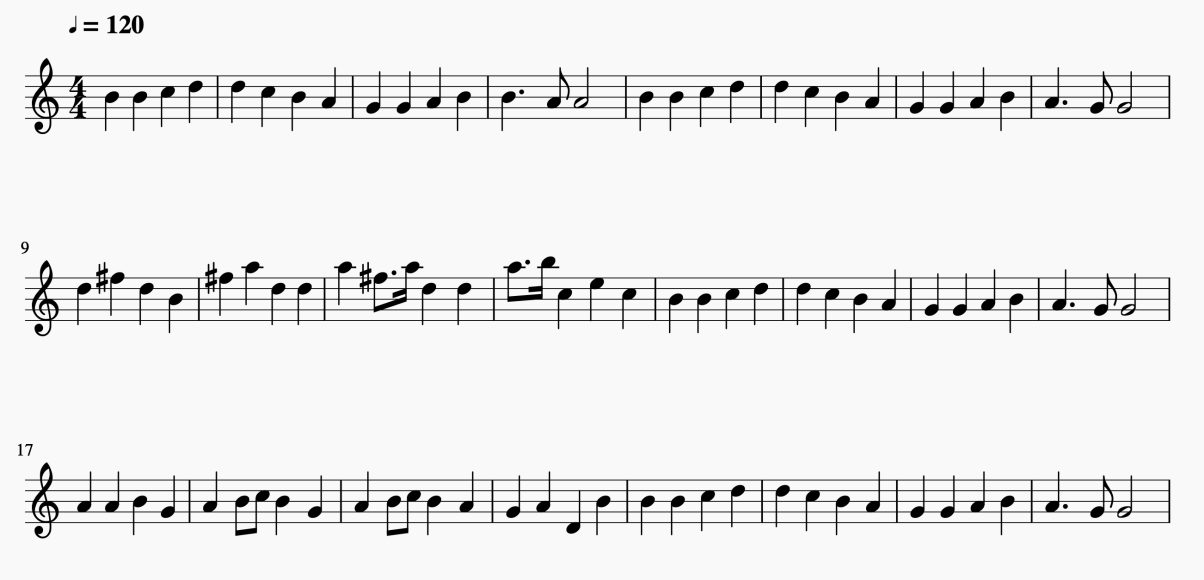

Figure 5.2: Altered MIDI File - Read in MuseScore

This shows that the first goal of whether it was possible to implement this technique of Music Improvisation using Markov Chains was a complete success. There was no need to modify any method to the extent that it was not recognisable from the original. Any changes that were made were needed to work with the MIDI format.

The problem with MIDI is that it does not think of sheet music in the exact same way that people think of sheet music. MIDI looks at note pitches as number from 0 to 127 (128 notes on a piano). With MIDI notes are not placed in selected bars but instead place them in sequential order and the system that reads the file has to display the music into human readable way.

Each backing chord section was created in the allocated time and there was no bleeding of notes into bars that were not supposed to be touched. This meant that when the output was opened using MuseScore there were no visual issues. The only change to the music from the original sheet was that system read the file in the key of C Major which has no accidentals. This meant that when viewing the output the sheet music would not have a key signature but instead showed off the accidentals. This was due the method of creating the MIDI files to be inputted.

When the file was exported from MuseScore there would be two meta messages, one to indicate the number of accidental which in 'Ode to Joy' is a single sharp. The meta message does not indicate major or minor key but in this version 'G' is given, which indicates G Major which means one sharp (F sharp). The second unnecessary message setting the key signature as C Major. MIDI leaves the software reading the file to understand the key signature. If you look back at figure 4.1 and you will see how it looks in the MIDI file.

Even when the original MIDI input is looked at in MuseScore the key is in C Major. This has

to do with how MuseScore creates MIDI files and the Python system created is not effecting it.

5.2 Strengths

This new system works much more autonomously compare to the original implementation. There is no need to hard code the original song into the system which is definitely better compared to the original implementation. As the inputted MIDI file is being copied over to the new file and only select bars are changing, it guarantees the format of the file stays the same.

The methods in the Python system use the same functions to generate new MIDI messages. When comparing to the original version, it had unique sections of code for every different backing chord. This made the system cluttered and difficult to understand. The new implementation is much more streamlined. It it easily understood and it allows for easy changes to be made to the system to be made.

Using MIDI the music itself is a lot more accessible which will allow people who do not have a good understanding of computer science to get involved in the process. It can be easily created so that any type of music could be used in this system in future if the correct methods are implemented.

5.3 Weaknesses

The music created follows the rules defined my the original method but there are many aspect that need to be addressed.

The main weakness with this system is that it can not be used for any other purpose then the original. It will only work for 'Ode to Joy' and it only improvises the bars selected in the original paper. This dissertation is to check proof of concept but at the same this system currently too specific to be useful to anyone.

The reason for this is that there are still too many aspects of hard coded element. They need to be addressed if this system is to work for more then one specific method. Unfortunately it is not as simple as implementing a few changes to the code. The needed changes will be described in future work in the next section.

The music does not flow in the same manner that the bars which are replaced do. In the figure 5.1 the unaltered bars have a shape to them which work together to create lovely flow as they connect. The music create as seen in figure 5.2, the system does not flow the same.

Each backing chord section is separate and due to that the music jumps a lot. This is issue in most part to the use of Markov chains.

The probability values that are used in this system are a big part of the problem. These probabilities are taken directly from the original paper and have been left touched. The reasons for using some probabilities over others are not clearly defined in the original paper. They were stated to be based off Baroque improvisation but that can be subjective depending on the person looking at Baroque. As improvisation system looks to create a general Baroque style as a whole it loses the personalised feel of the original music which was created by an individual.

This system knew the backing chords for the piece of music but of course not every piece of music has the same backing chord sequence. It is possible to update the code so that it has the complete list of backing chords and the position of those chords but this does not fix the underline need for a method to identify backing chords. Without it, this method can not move forward.

6 Conclusion

6.1 Summary

The aim for this dissertations was to see if it was possible to automate a method defined in a previous paper using the Python language. This involved having a deep understanding of the work created and investigating the many aspects that could be automated.

Using the Python library 'Mido' the music being read could be inputted in the form of MIDI files which are a format of music representation in a digital media. With the ability to read MIDI files implemented it set up a strong bases for the system being created to work from. Any music can be put into MIDI form and MIDI is set in a systematic way that a computer program can read through and analyse.

The previous work had hard coded methods to creating the improvised sections, most of which work in the same way. The new Python system needed to streamline this process so that the same method could be called to it which would give the same functionality that the separate hard coded sections could. This was achieved in the Python system as the data that was being processed was setup into data sets which could only be accessed at the appropriate time depending on the data being processed.

Hard coded elements still exist in the Python system. These are the pitch and duration probabilities which were taken from the original implementation. There was no need to change these values as they were not important in the grand scheme of this dissertation. This is due to the focus being on automating the process of the piece of music 'Ode to Joy'. This value were manually created and if another value were needed, they would still be as arbitrary as the original. Additional computer process would be need to create substantial data which in the scope of this project was too much. The dissertation is proving a concept and in that respect, things can be simplified.

Again there was simplification when it came to the backing chords being use to create the improvisation method. There are so many methods to generating backing chords and a deeper investigation into the best method of backing chord generation would be needed if the Python

system was going to automate the entire improvisation process. Backing chords are decided depending on a set of rule which means they can indeed be identified using a computer program. Knowing this, automation could be added to create the backing but with the scope and time frame of this dissertation, there was no need to move forward in this direction. Iterating the point, proof of concept is the most important part of this dissertation.

The range of notes used for the improvisation was useful to prove that the method could be implemented but when looking at it in terms of whether it sounds good, its a different story. The music does not sound as pleasant compare to the original music, this is due to the nature of using only Markov Chains to decide what notes need to be played. This issue is well known in the area of music generation with the suggestion of an additional method to give a better range depending on the type of music being looked at.

This dissertation was successful in proving that the music improvisation method being implemented can be implemented in Python and with more investigation, it can become a viable tool for artist to use. As there is little investigation in this niche area, this dissertation opens up the possibility for further development.

6.2 Future Work

The method of using Markov Chains can only bring this method so far. If this method is to be successful there needs to be additional modifications. The three main aspects detailed below are needed to move this concept into a position where it can be used by actual artists. A position where Baroque is not the only music being improvised.

6.2.1 Generating Backing Chords

Creating a method to analyse the MIDI file being inputted and generating the backing chords for every aspect of the piece. Parameters could be passed in so that the system can focus on certain bars which would save on processing time and continue working the same way.

The only draw about to this is that backing chords flow in a certain structure and looking a single aspect will not give the best result. There are rules that that need to follow and these rules can easily be implemented in the system. Determining the exact backing chords would be subjective to an extent as there many be many different backing chord sequences that could fit in a single piece of music. Different constraints could be set on the method so different types of backing chords generated and these constraints may depend on the type of music being improvised.

The implementation of this method should be the first priority of any future work. This method is dependent on the knowing the backing chords of a piece. If every other aspect of the system stayed the same but this method was introduced then any aspect piece of music in the style of Baroque could be inputted furthering the usefulness of the entire system and moving it in a strong direction.

6.2.2 Generating Probabilities

A method needs to be implemented to generate an accurate probability list for both pitch and duration. The automation failings of the Python system created in this dissertation is due to the limitation in these areas. There is no way that a person can manually define what these probabilities are for the style of Baroque music to an accurate degree compare with a computer program.

The implementation of a neural net system to analyse a large training sets of music could be used to create these probabilities. This means that any type of music could be looked at and the system would not be solely focused on the Baroque style. This would mean a certain amount of prepossessing but if an artist is working in the same genre of music, then this will not be an issue after the initial setup.

6.2.3 Change Note Range

In this system the range of notes used for each backing chord is extremely limited. They are based off a triad but the octave used for these notes is chosen arbitrarily. The ranges somewhat fit in the original range of notes of the original bars but they still do not suit very well.

Having the system understand the MIDI notes structure depending on the key signature will be important for this method. The different types of note interaction will vary in the backing chords but recognising which notes are appropriate for the key signature will ensure that the improvisation generate will not have mistake accidentals where they should not.

This method go hand in hand with generating probabilities. By expanding the range of potential notes that could be used for a backing chord then the probabilities would need to reflect that. This would change the method in such a way to smooth out the flow of the music generated and change the system into a tool that can be useful for anyone who want to create new music.

Bibliography

[1] Erlijn J Linskens and Gijs Schoenmakers. Music improvisation using markov chains. 2014.

[2] Nick Collins. The analysis of generative music programs. *Organised sound*, 13(3):237–248, 2008.

[3] Geoff Cox MA RCA. How i drew one of my pictures:* or, the authorship of generative art. 2004.

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

[5] Gareth Loy and Curtis Abbott. Programming languages for computer music synthesis, performance, and composition. *ACM Computing Surveys (CSUR)*, 17(2):235–265, 1985.

[6] Torsten Anders and Eduardo R Miranda. Constraint programming systems for modeling music theories and composition. *ACM Computing Surveys (CSUR)*, 43(4):30, 2011.

[7] François Pachet, Pierre Roy, and Gabriele Barbieri. Finite-length markov processes with constraints. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[8] Parag Chordia, Avinash Sastry, Trishul Mallikarjuna, and Aaron Albin. Multiple viewpoints modeling of tabla sequences. In *ISMIR*, volume 2010, page 11th, 2010.

[9] Shlomo Dubnov, Gerard Assayag, Olivier Lartillot, and Gill Bejerano. Using machine-learning methods for musical style modeling. *Computer*, 36(10):73–80, 2003.

[10] Stephen Davismoon and John Eccles. Combining musical constraints with markov transition probabilities to improve the generation of creative musical structures. In *European Conference on the Applications of Evolutionary Computation*, pages 361–370. Springer, 2010.

[11] François Pachet and Pierre Roy. Markov constraints: steerable generation of markov sequences. *Constraints*, 16(2):148–172, 2011.

[12] Jon Gillick, Kevin Tang, and Robert Keller. Learning jazz grammars. *Computer Music Journal*, 34:56–66, 01 2010.

[13] Zaky Hassani and Aciek Wuryandari. Music generator with markov chain: A case study with beatme touchdown. pages 179–183, 10 2016. doi: 10.1109/ICSEngT.2016.7849646.

[14] Chip Bell. Algorithmic music composition using dynamic markov chains and genetic algorithms. *Journal of Computing Sciences in Colleges*, 27(2):99–107, 2011.

[15] Jim Defranza and Daniel Gagliardi. *Introduction to Linear Algebra with applications*. Waveland Press, 2009.

[16] Lejaren A Hiller Jr and Leonard M Isaacson. Musical composition with a high-speed digital computer. *Journal of the Audio Engineering Society*, 6(3):154–160, 1958.

[17] Charles Ames. Automated composition in retrospect: 1956-1986. *Leonardo*, pages 169–185, 1987.

[18] Yi-Wen Liu and Eleanor Selfridge-Field. Modeling music as markov chains: Composer identification, 2002.

[19] Karsten Verbeurgt, Mikhail Fayer, and Michael Dinolfo. A hybrid neural-markov approach for learning to compose music by example. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 480–484. Springer, 2004.

[20] François Pachet. Beyond the cybernetic jam fantasy: The continuator. *IEEE Computer Graphics and Applications*, 24(1):31–35, 2004.

[21] What is MATLAB?, . URL `https://uk.mathworks.com/discovery/what-is-matlab.html`.

[22] What is MIDI?, . URL `http://valentin.dasdeck.com/midi/midifile.htm`.

[23] Summary of MIDI Messages, . URL `https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message`.

[24] MIDI Messages, . URL `https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node158.html`.

[25] What is Python? Executive Summary | Python.org, . URL `https://www.python.org/doc/essays/blurb/`.

[26] Mido - MIDI Objects for Python, . URL `https://mido.readthedocs.io/en/latest/`.

[27] MuseScore, . URL `https://musescore.org/en`.