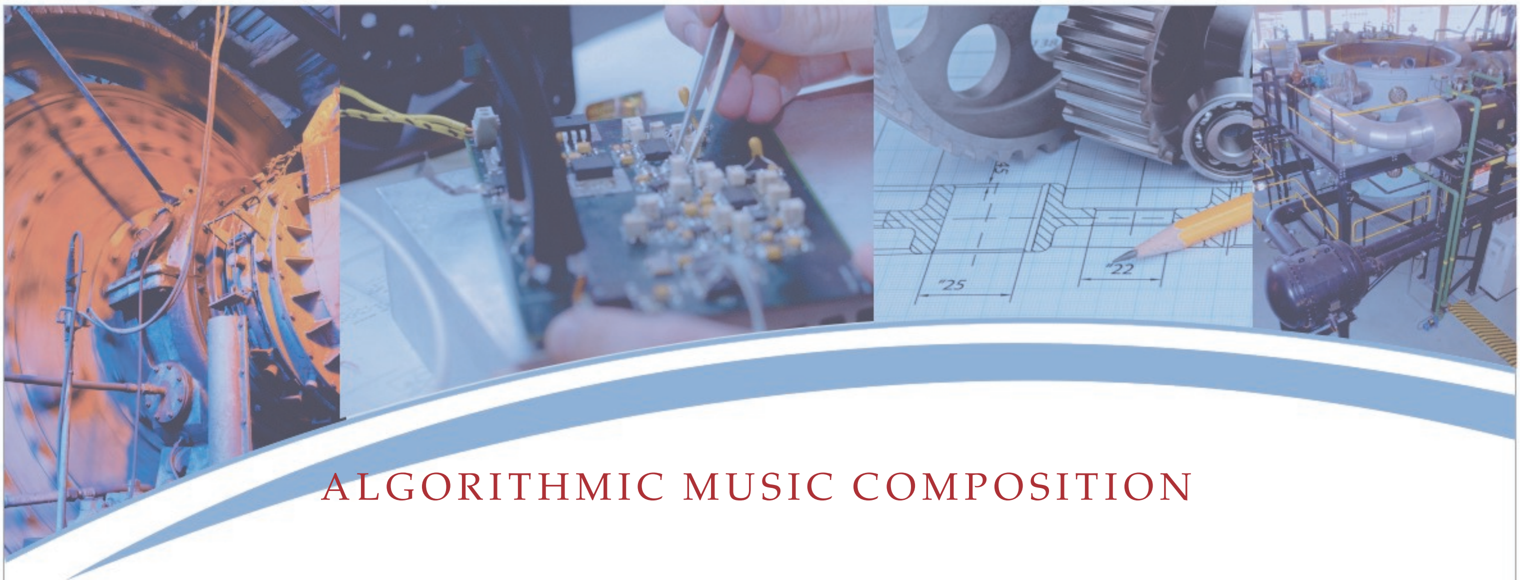

ALGORITHMIC MUSIC COMPOSITION

by:
S. Jacholke 23671750

submitted in persuit of the degree

BACHELOR OF ENGINEERING
in
COMPUTER AND ELECTRONIC ENGINEERING

North-West University Potchefstroom Campus

Supervisor: Mr. A.J. Alberts
Potchefstroom
October 12, 2015
version 2.2

It all starts here

NORTH-WEST UNIVERSITY
YUNIBESITI YA BOKONE-BOPHIRIMA
NOORDWES-UNIVERSITEIT
POTCHEFSTROOM CAMPUS

Stefan Jacholke: *Algorithmic Music Composition,* Composing music algorithmically, © October 12, 2015

ABSTRACT

Obtaining the required music licensing may not be viable for certain individuals and business owners. Alternatively the music can be composed and produced, however this requires time and skill.

For this project another alternative is investigated, to produce music using computer algorithms.

The field of algorithmic music composition and is not new. The problem of algorithmically composing music is difficult and there is a wide range of research on the topic.

For this project, the relevent research is investigated. Possible algorithms were identified and implemented. Algorithms producing pleasant results include: Markov Models, a simple feed forward neural network, Markov Chains and Genetic Algorithms. These algorithms were kept and implemented in a end-user application.

The Windows application developed features a friendly user interface that allows the user to compose, playback and store simple monophonic melodies in a certain style. In order to optimize the end user experience the algorithms were pretrained were possible.

Since it is difficult to quantify the pleasantness of a song, due to the inherent subjective experience, it was necessary to rely on pattern recognition and machine learning techniques. A weakness of the algorithms implemented is that they do not have a overarching theme or structure.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

ANN Artificial Neural Network

GA Genetic Algorithm

HMM Hidden Markov Model

MIDI Musical Instrument Digital Interface

NCD Normalized Compression Distance

NID Normalized Information Distance

IGA Interactive Genetic Algorithm

SRN Simple Recurrent Network

LSTM Long Short Term Memory

ART Adapative Resonance Theory

RNN Recurrent Neural Network

XAML Extensible Markup Language

WPF Windows Presentation Foundation

XML Extensible Markup Language

Part I

INTRODUCTION

1 ALGORITHMIC MUSIC COMPOSITION

1.1 INTRODUCTION

Music is the art form of sound. Certain characteristics and patterns can be identified that belong to certain types of music.

There has been a lot of work done in computer generated music and computer assisted music composition. Recent research into music composition by means of evolutionary genetic algorithms and other machine learning algorithms such as neural networks have met some success.

Music generated algorithmically by computers might some day share the success known by modern and historic music. In a hypothetical time period artists might compete with algorithmic composers.

Computer generated music is an old idea, however existing solutions that generate music use outdated procedural algorithms and do not match the quality of proper historic music. The variety and quality of existing solutions is limited. Most current solutions were implemented to accommodate the research being done and are not suitable for end-users.

This section will outline the problems that are encountered with conventional music composition and a solution is proposed. Some of the ideas behind systems that learn music composition will be outlined and described. The problem of implementing and creating a solution will be detailed as well as the steps required to complete such a project.

1.2 PROBLEM

In order to avoid copyright infringement music has to be licensed for use in commercial applications. Small business owners, independent developers and other smaller organizations may not have the required funding in order to obtain the relevant licensing.

The alternative solution is to produce the required music self, however this requires time and skill.

A solution to this problem is to generate music by artificial intelligent means. Algorithmic music composition is of theoretical interest and research has been done for composing music by genetic algorithms and neural networks, amongst other algorithms.

A software application that is able to compose and play monophonic melodies according to a certain style by means of machine learning algorithms could solve the licensing and resource problem.

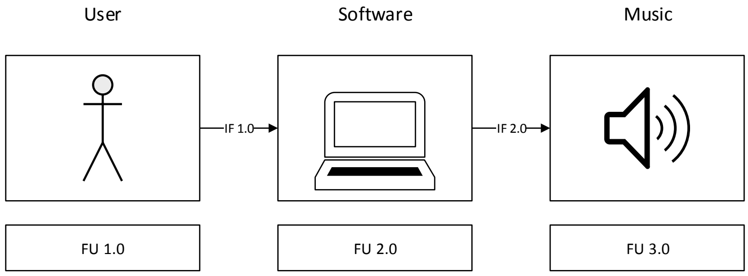

Figure 1: Figure of high-level architecture of project

1.3 SOLUTION

The solution to the problem is to create an application that is capable of composing and playing simple monophonic melodies using machine learning algorithms. Algorithms other than learning algorithms may be used, however these algorithms are narrow in application. Using learning algorithms and pattern recognition allows the application to produce melodies according to a selected style.

There is a need for a user-friendly application that can be used to generate simple monophonic melodies according to a certain style or category of music. The application is required to have the following features :

1. Melodies are to be algorithmically composed acccording to a certain style.
2. Provide playback functionality as well as saving and loading of previously composed songs.
3. The user should be able to select the style
4. The application should utilize a MIDI library which will be used as training data for the machine learning algorithms.

Figure 1 indicates the interaction between the user and the software.

Since the quality and aesthetics of a musical melody is subjective, it is beyond the scope of the application to ensure that each melody produced is objectively good[1]. Composed musical pieces will take the form of simple melodies.

1.4 ALTERNATIVE SOLUTIONS

A wide variety of research has been done and in this section we will briefly look at some of the existing solutions and their drawbacks

1 The focus is rather on the generated musical pieces not being objectively bad

1.4.0.1 *NEUROGEN*

NEUROGEN is a system that was designed to compose small diatonic, western-type, four part harmony compositions based on a training set of example musical fragments. NEUROGEN tries to produce coherent music of the type that is typically found in traditionally hymns [2]. However NEUROGEN is only tries to fulfill its main research goals.

1.4.0.2 *CONCERT*

CONCERT incorporates psychologically-grounded representations of pitch, duration and harmonic structure in order to compose novel melodies. CONCERT uses a neural network in order to do note by note predictions.

CONCERT struggles to compose natural music as the music produced was not globally coherent [3].

CONCERT also only tries to fulfill its research goals, that is, to compose music on a note by note basis usign a neural network.

More recent studies into recurrent neural networks based on LSTM indicate that it is indeed possible to compose music using recurrent neural networks that have global structure [1].

1.4.0.3 *GenJam*

GenJam - Genetic Jammer is a IGA that learns to improvise jazz [4]. The system is able to:

1. Play full-chorus improvised solos
2. Respond interactively when the trumpet is played to trade fours or eights
3. Perform a smart echo of improvisation

The main drawback of GenJam is that it is an interactive algorithm and as such requires input from the user. The creates a performance bottleneck as it is time consuming for the user to rate musical pieces. See section 5.6 for work on IGAs

1.4.1 *jMusic*

jMusic is an application and library that fascilitates musical composition in Java [5]. The application is designed to provide composers and developers a plethora of compositional tools that can be used for generative music, instrument building, interactive performance and music analysis. In addition the software provides the following:

- Familar music data structures
- Organizing, manipulating and analyzing music data
- Real time playback of music scores

- Read, write Musical Instrument Digital Interface (MIDI), Extensible Markup Language (XML) and .jm files

jMusic is mostly a developer tool to fascilate composition. It does not feature a user interface which can be used to easily generate compositions. Furthermore since it is mostly a tool to aid music composition it does not employ machine learning algorithms. The library requires exact and descriptive input in order to generate melodies, and as such requires domain knowledge.

1.4.1.1 *Other*

Most of the work done in the field has been aimed on narrow application of the ideas being investigated. Thus algorithms and applications constructed were aimed only to experimentally study the ideas being investigated and not to serve as a end-user product.

Some other small application exist which employ Interactive Genetic Algorithms (IGAs) such as:

1. Evolutune[2]
2. Song Builder[3]
3. DarwinTunes[4]

Even though more recent research into algorithmic music composition yields promising results there is no single application combining the more recent research into a coherent application that is able to play monophonic melodies[5].

1.5 FEASIBILITY

There has been a plethora of research done into algorithmic music composition. Some of the more recent research that utilizes machine learning algorithms is investigated in chapter 18.

Some machine learning methods to compose music algorithmically include:

- Recurrent neural networks [1, 3, 6]
- Genetic algorithms using NCD as fitness functions [7, 8]
- Genetic algorithms using neural networks as fitness functions [2, 9, 10]
- Genetic algorithms employing Zipf's law and cosine similarity [8, 11, 12]
- Interactive genetic algorithms [4, 9, 13, 14, 15]

2 http://askory.phratry.net/projects/evolutune/
3 http://www.compose-music.com/
4 http://game.darwintunes.org/
5 In addition there is no accessible user-friendly application that expose machine learning based algorithmic composition to the user

- Markov Chains for music composition [16, 17].
- Hidden Markov Models for harmonization [18].
- LSTM neural network for composing blues [1].

A variety of ideas and techniques in algorithmic music composition have already been researched. The difficulty arises in the implementation of the ideas (as the process is only fundamentally documented), representation of music in data structures and the implementation of the relevant machine learning algorithms.

1.6 METHODOLOGY

In Royce's classical waterfall model the following steps are followed when developing software:

- Requirements specification
- Design resulting in the software architecture
- Construction
- Integration
- Testing and debugging
- Installation
- Maintenance

Since there are a variety of algorithmic music composition strategies, and no formal analysis that compares the quality of the resulting music of each strategy (difficult as it is a subjective matter) the software prototyping methodology could be used to test these different ideas and to construct a trade-off study to determine the most feasible techniques.

Software prototyping is the development of prototypes - incomplete versions of the software. Some basic principles of software prototyping include:

- Prototype selected parts
- Breaking project into smaller segments
- User or client may be involved
- Iterative modification to meet user demands

The main ideas of the systems engineering process, as applicable to software will be followed. Ideas from the waterfall model and software prototyping will be incorporated.

The basic formulation of the design of the project is as follows:

1. Conceptual design
    - Identify problem
    - Identify requirements
    - Resource allocation

    - Literature study
    - Feasibility analysis
    - Trade off study
2. Preliminary design
3. Detail design
    - Design of software architecture
    - Prototyping of different strategies and algorithms
4. Construction
    - Code is written
    - Code is tested
    - Iteratively improve until project meets specifications
5. Phase out, support, maintenance

In order to identify suitable algorithms a literature review will be done. The various algorithms will be prototyped and tested and the best few resulting algorithms will be used. Software development will follow the waterfall model.

1.7 CONCLUSION

Since the act of generating music using algorithmic means is of theoretical interest there is a plethora of research available, however most of these implementations only aim to explore the main idea being researched. Thus there is a lack of accessible, user-friendly applications that are able to generate music algorithmically.

The aim of this project is to construct an application that is able to compose music algorithmically. The application will utilize machine learning algorithms in order to compose monophonic melodies according to the style of a certain author or theme. These ideas have already been researched and thus the application is feasible. The only cost is time and effort.

The methodology for successfully completing the project is outlined and a schedule for completing each goal is presented. This should assist in meeting the deadlines and completing the project successfully according to the specifications.

Part II

LITERATURE REVIEW

2 MUSIC AND MUSIC-REPRESENTATION

2.1 MUSICAL ELEMENTS

Music is an art form for which the medium is sound. The common elements in music are:

- Pitch - Subjective sensation reflecting lowness and highness of sound, also represented more objectively as frequency.
- Rhythm - Arrangement of sounds and silences.
- Dynamics - Execution of a given piece (speed, volume)
- Timbre - Tone

Music composition refers to the creation and recording of music through a medium that others can interpret. Music can be composed for repeated playback or it can be composed on the spot.

A melody is a set of notes (or rests) that are performed in series. Each note may have a different length and different stress. These notes are arranged in a certain rhythmic pattern.

Notes were traditionally given a letter to represent the pitch of the note. The names come from the set $\{A, B, C, D, E, F\}$. Notes may have their pitch modified by additional symbols such as a sharp(#).

Notes and rests may have different lengths.

Li and Sleep have found that a given piece of music remains recognizable when the length of the notes are randomized [19].

2.2 MIDI

Music can be stored digitally in a variety of formats and encodings. This allows repeated performance of the same track and also eases distribution of the music.

MIDI is a standard which describes the protocol, digital interface and connectors that allows electronic instruments to communicate with one another.

The MIDI 1.0 detailed specification fully describes the MIDI interface [20]

The MIDI file format describes the way MIDI information is stored in a file. Each MIDI file starts with a header chunk that describes the time division and the number of tracks. After the header chunk multiple track chunks occur. Each track contains multiple MIDI events.

The header chunk is described as follows:

```
"MThd" + [header_length] + [format] + [n] + [division]
```

where

[header_length] always 6 bytes

[format] 0 - single track format, 1 multiple track format, 2 multiple song format

[n] number of track chunks

[division] unit of time for delta timing

After the header chunk [n] track chunks follow. Each track chunk is composed as follows:

```
"MTrk" + [length] + [track_event] <+ [track_event1] + [track_event2] +
   [...] + [track_eventn]>
```

where

[length] number of bytes in track chunk

[track_event] sequenced track event, described next

The track chunk can be seen to contain multiple track events. Each track event consists a delta time and either a midi event, meta event or sysex_event:

```
[v_time] + [midi_event] | [meta_event] | <sysex_event>
```

The meta-event contains additional information such as text which can be displayed, instrument names, lyrics and so on.

Table 1: Table containing some midi events

Command	Meaning	parameters	param 1	param 2
0x80	Note-off	2	key	velocity
0x90	Note-on	2	key	velocity
0xA0	Aftertouch	2	key	touch
0xB0	Continuous controller	2	controller #	controller value
0xC0	Patch change	2	instrument #	
0xD0	Channel Pressure	1	pressure	

Table 1 shows some of the possible midi events. Each event has a command and additional arguments.

The full midi specification can be ordered online at midi.org[1]

2.3 REPRESENTATION

Representation of music is important to achieve successful generation of a correct solution [2]. However the search space of machine learning algorithms may be too large if limitations are not introduced [21].

Figure 2 shows the music sheet of an excerpt of Bach's BWV0525 Sonata using modern notation.

1 http://www.midi.org/

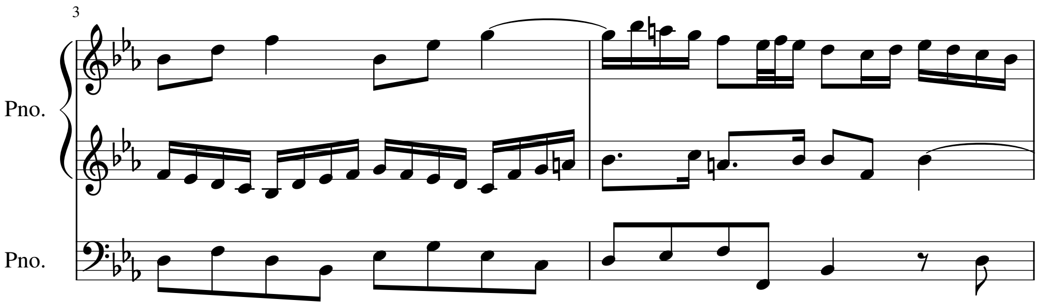

Figure 2: Excerpt of Bach's BWV 05225

The musical data from musical pieces need to have a proper representation in order to be used by a machine learning algorithm.

In [1] Eck a form of 12-bar blues was used with 8 notes per bar. He used the following representation for the **LTSM!** system:

1. 12-bar musical pieces were used
2. 8 notes per bar were used
3. The same chords were used between songs, see figure 5
4. The melody notes were built using the pentatonic scale
5. The neural network inputs indicate whether a certain note is on or off

A melody would be represented differently in a linear genetic algorithm. A simple melody consisting only of a few notes could be represented as seen in figure 3. Note the duration of a note is not defined and could be consider fixed.

However it might be preferable to encode more events such rests, chords, dynamics, note duration and so on.

Tree based representations are found in genetic programming [22]. Minsky has argued that tree representation is more suited to music as it mimics the hierarchical structure that is found in music [23].

A melody could be represented in tree form as seen in figure 4. The bottom leaves denote the pitch and duration. The higher level nodes perform operations.

Johanson and Poli employ several different operators including mirroring, concatenation, repetition and mirroring [15]

Tree based representations do not have a fixed size, as is commonly found in linear genetic representations. Care must be taken to not let the structure become too large. This is typically done by specifying the maximum depth of the tree.

Figure 3: Simple melody represented as a bit string

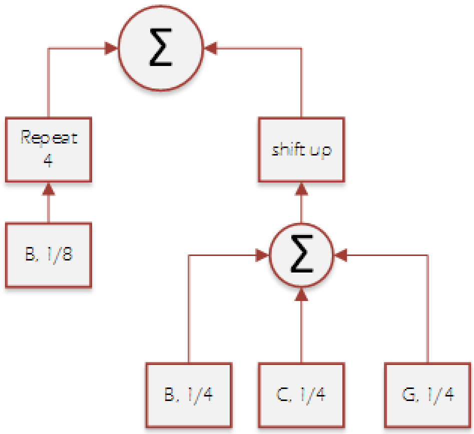

Figure 4: Simple melody represented in tree form

Figure 5: Training chords used in [1]

3 MUSIC COMPOSITION ALGORITHMS

3.1 HIDDEN MARKOV MODELS

HMM are stochastic methods that are used to model sequential data such as speech and gesture recognition. Under the Markov property the next observation is only dependent on the current state of the system. Such states may not be known and may be hidden from the observer as only the output values are observable. When a event is generated from a state, the model moves into a new state based on its transition probabilities. The term hidden is commonly used to indicate that many different state sequences can generate the same observed sequence of events.

The probability of any sequence of observations occuring when given a sequence of states is described by:

$$\Pr(x, y) = \prod_{t=1}^{T} \Pr(y_t | yt - 1) = \Pr(x_t | y_t)$$

Where $\Pr(x_t|y_t)$ denotes the probability of observing x_t given that the current state is y_t and $\Pr(y_t|y_{t-1})$ denotes the probability of being in current state y_t given that the previous state.

An transition and emission matrix are commonly used the calculate $\Pr(y_t|y_{t-1})$ and $\Pr(x_t|y_t)$ respectively.

A variety of learning algorithms exist which compute the structure of the model and also calculate the emission and transmission matrices. For space concerns these will not be discussed.

HMM have been applied with some degree of success to music composition. A system has been developed for producing a counterpoint line to a cantus firmis in the style of Palestrina [17]. Another approach utilizes a HMM for chorale melody harmonization [18].

3.2 MARKOV CHAINS

A Markov Chain is a stochastic model describing a sequence of events. The Markov Chain has the property that the probability of the next state depends only on the current state of the chain.

$$\Pr(X_n | X_1, X_2, \ldots, X_{n-1}) = P(X_n | X_{n-1})$$

Figure 6 is an example of a Markov Chain. For each state there is a probability of proceeding to the next state. For example if the current state is F, there is a 10% chance of the next state being F again, and a 90% chance of the next state being A.

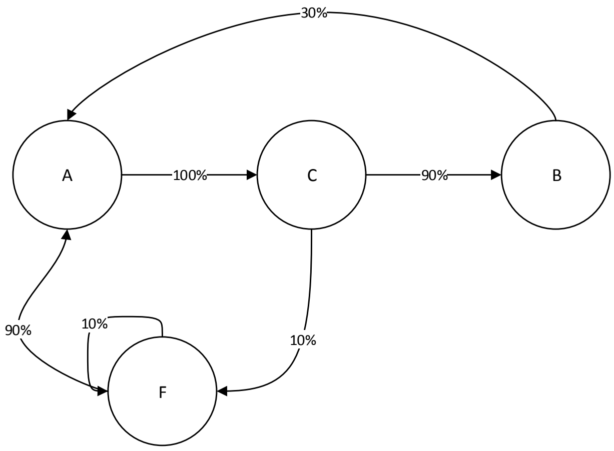

Figure 6: Example of a Markov Chain

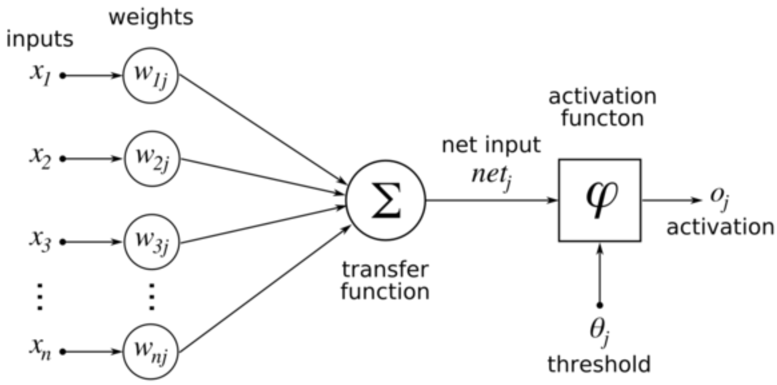

Figure 7: Model of artificial neuron

A Markov Chain can have a order. For a simple note such as figure 6 if the current note is B then the note A has a 30% chance of being next. This is a first order Markov Chain. A more complex chain of order 2 could have the current state being composed of two notes. Thus the next note is dependent on the previous two notes. A N-th order Markov chain can be represented by a transition matrix, which corresponds to a N+1 dimension probability table.

Markov Chains must be derived from existing music and generate output of the same style as the input. This reduces the compositional value and novelty however it suits the purpose of generating music in a specific style [24].

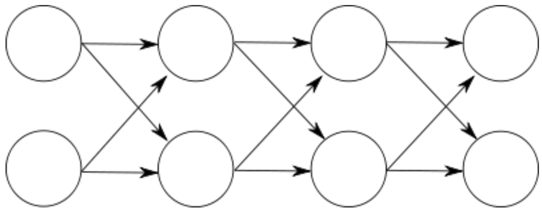

Figure 8: Figure of ANN in feed-forward topology

3.3 NEURAL NETWORKS

3.3.1 *Background*

An artifical neural networks consists of layers of artificial neuron[1] (see figure 7) that are connected to each other in a certain topology [25].

Figure 8 shows a multilayer ANN in feed-forward topology. The first layer is referred to as the input layer, the right-most layer is referred to as the output layer, the layers in between the input layer and the output layer are known as hidden layers.

Neurons have a number of inputs which are summed into an activation function. The activation function provides the output of a neuron. In figure 7 we have

$$o_j = \varphi\left(\sum_{i=1}^{n} x_i w_{ij}\right)$$

A common activation function is the continous sigmoid function given by

$$\varphi(t) = \sigma(t) = \frac{1}{1+e^{-\beta t}} \tag{1}$$

where β is the slope parameter. The derivative of the sigmoid function is easy to obtain and is given by:

$$\frac{\sigma(t)}{dt} = \sigma(t)(1-\sigma(t)) \tag{2}$$

for $\beta = 1$

In order to determine weights for a two-layer neural network we need to minimize the error between the output of the neural network and the given target value for a set of inputs.

Thus for the output neurons we have

$$E(\vec{w}) = \frac{1}{2}\sum_{d \in D}(t_d - \sigma_d)^2 \tag{3}$$

where D is the set of training examples and t_d is the target output for training example d and o_d is the output of the artificial neuron for training

1 neurons are commonly referred to as units

example d We need to minimize equation 3. Obtaining the deriviative of E with respect to w_i yields:

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_i d)$$

where x_{id} is the input for component x_i

Gradient descent is a optimization algorithm that takes steps proportional to the negative of the gradient. This we update the weights by:

$$\vec{w} \leftarrow \vec{w} - \eta \nabla E(\vec{w})$$

Applying the gradient descent algorithm we obtain a weight update rule of

$$w_i \leftarrow w_i + \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

For a multi layer network with multiple output units the back propagation algorithm needs to be used. The error needs to be summed over all the network output units

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - \sigma_{kd})^2$$

The backpropagation algorithm to determine the weights of a feed-forward neural network is given in appendix A

3.3.2 *Composition*

Neural networks have been used in music classification, in genetic algorithms as fitness functions and more complex neural networks have even been used in algorithmic music compositions.

Simple feed-forward neural networks do not contain a mechanism to remember past history.

In [3] Mozer created a recurrent connectionist neural network **CONCERT**[2][3]. The system works as follows:

- The network is trained on sample melodies from which it learns melodic and phrase constraints
- Representations of pitch, duration and harmonic structure that are based on psychological studies of human perception, based on Laden and Keefe's work [26]

The system yielded good results on simple structured artificial sequences however the system performed poorly on natural music[4]. Mozer described

2 CONCERT - connectionist composer of ERudite tunes
3 The ER may also be read as ERratic or ERsatz
4 One critic described the resultant melodies as compositions only a mother could love

the system as lacking musical coherency [3]. Furthermore the system performs poorly as the length of the pieces increases.

Mozer stated the reason for failure is likely due to the Recurrent Neural Network (RNN) not being able to track more distant events that build global structure [3] however a LSTM recurrent network is able to achieve this goal [1].

In order to solve the problem of global structure Douglas and Jurgen attempted to use a LSTM network to compose musical pieces [1]. In this attempt the network was successfully able to learn a form a blues music and stay close to the relevant structure. The system used cross entropy as the error rate:

$$E_i = -t_i \ln(y_i) - (1 - t_i) \ln(1 - y_i)$$

where y_i is the output activation and t_i the target value for the $i - th$ output unit. The topology of the network was arranged as follows:

1. Four cell blocks are connected to the input units for chords
2. The last four cell blocks are connected to the inputs units for melody
3. chord cell blocks have recurrent connections to themselves and melody cell blocks
4. melody cell blocks have recurrent connections to other melody cell blocks
5. output units for chords are connected to cell blocks for chords and to input units for chords
6. output units for melody are connected to cell blocks for melody and to input units for melody

The underlaying chord structure was kept fixed.

The results indicate that a LSTM network is able to compose with both local- and global structure from a set of training data [1].

3.3.3 *Conclusion*

Simple feed-forward neural networks do not have the functionality to remember past history and thus do not have the capability to evaluate repetitive rhythmic patterns.

Recurrent neural networks are able to encode temporal information though initial investigation by Mozer lead to limited success as compositions lacked global structure.

Further investigations by Douglas and Jurgen indicated that **LTSM!** (**LTSM!**) networks are able to compose with local and global structure.

3.4 GENETIC ALGORITHMS

3.4.0.1 *Background*

Genetic Programming, not to be confused with evolutionary or genetic algorithms is a evolutionary algorithm based methodology to find computer programs that perform defined tasks by simplistically mirroring biological evolution. The more fit programs carry on their chromosomes into future populations. Fitness is rated by a fitness function. Other genetic operators such as recombination and crossover are also usually applied. These evolved genetic programs are usually represented in tree form. Figure 9 indicates the function $(2.2) - (\frac{X}{11}) + 7\cos(Y)$ written in tree form.

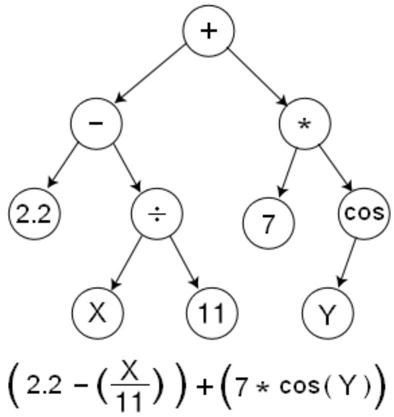

Figure 9: Figure of a example genetic program tree

A genetic algorithm consists of the following components:

1. Representation for chromosomes[5]
2. Initial population of chromosomes
3. A set of genetic operators to alter the population
4. A fitness function to assess the fitness of an individual
5. A selection method to determine which individuals in a population survive

The algorithm proceeds as follows:

1. Initial population is randomly generated
2. The fitness of each individual is assessed
3. Individuals are selected to which genetic operators are applied, e.g. Two parents are selected to generate a new child with crossover Random mutation occurs in individual
4. Various forms of selection are available that determine which individuals will be in the next generation

5 Also commonly referred to as an individuals

Genetic operators are used to generate diversity A genetic algorithm has a fixed set of genetic operators. Operators may include:

- Reproduction - A parent from the population is carried over to the next generation
- Crossover - genotype of both parents are combined using different procedures
- Mutation - A single mutation is applied to a chromosome at a set mutation rate

The choice of a **fitness function** is a big problem when using a genetic algorithm to compose music [27, 28]. There is no objective method to rate whether a melody is good or bad [29]. Traditionally when posed against such tasks the fitness function is provided interactively by the user, i.e. the user rates whether the piece is good or bad

The interactive GA approach is an approach to the fitness function where a human interactively rates the quality of a composition (fitness). A well known software that utlizes a neural network and a interactive interaction for a fitness function is GenJam [9]. The drawback of having the user interactively evaluate the fitness of individuals is that it is time consuming and poses a processing bottleneck [1]

Selection is the choice of which individuals will be chosen for the next generation. Selection concerns the reproduction and crossover operators. Some methods of selection include:

- Roulette wheel selection - chance of individual being chosen is proportional to fitness
- Tournament selection - tournament is staged between two individuals to detemine which one gets selected[6].

3.4.0.2 *GA approaches to music composition*

Genetic algorithms have already been used in a variety of work in algorithmic music composition. Some of these include:

- Thematic bridging [30]
- Composition systems using IGAs [31]
- IGAs to improvise jazz solos [9, 4]
- Integration between interactive genetic algorithms and genetic programs [32]
- Hybrid approaches employing statistical, connectionist and evolutionary elements [11]
- Various work into different fitness functions

In [33] the generated musical pieces had the style of well-known authors even when the fitness function only took relative pitch envelope into account

6 This is tournament selection in its simplest form

and all generated note lengths were of fixed duration.

Thematic bridging is the application of an initial music pattern to a final pattern over a specified duration [30]. In this approach Horner modified or reordered elements in a music pattern through various operations. For example:

Given the initial note pattern of:

```
Gb Bb F Ab Db
```

and a final note pattern of:

```
F Ab Eb
```

the musical output could be:

```
Gb Bb F Ab Bb F Ab Gb Bb F Ab Eb F Ab F Ab Eb
```

by means of various operations such as mutation, rotation, deletion and so on. For thematic bridging a composite fitness function was used which rates how close the developed pattern matches the final pattern and whether the ordering of the elements are correct

Similarly a system of **variations** was developed Jacob that proceeds as follows [21]:

1. Define a primary set of motives
2. Compose phrases by layering and sequencing new motives
3. New motives are created by variations of motives already in the phrase
4. Phrases are combined together

Jacob's system had a human judge evaluate the individual chromosomes.

NCD has also been commonly used as a fitness function, for more information about the normalized compression distance see sections 4.2 (in music classification) and 5.5 (as a fitness function). Alfonseca proposed the following genetic algorithm scheme for composing melodies [33]:

1. Define a set of M musical pieces for a guide set G
2. Encode both the guides and individuals in the population as pairs of integers where the first integer represents the note interval and the second the length as a multiple of the minimum unit of time.
3. See eq. 5 on page 32 for the fitness function used
4. The 16 lowest fitness genotypes of every generation is removed
5. The 16 highest fitness genotypes of every generation are paired by means of genetic operations.

3.4.0.3 *Conclusion*

There are various variations on genetic algorithms and genetic programs, however evolutionary algorithms are a viable means to compose music.

The encoding of a musical piece as a chromosome affects the interactions of the genetic operators on the musical piece and most authors encode the problem differently.

It is important to restrict the domain of problem otherwise the search space for the genetic algorithm may be too large [21]. Most of the studies listed in this document had restricted goals. For example, using only two octaves for the notes significantly reduces the size of the search space and many real melodies comply with it [33]

The fitness function is an important part for having the genetic algorithm result to good melodies. See section 5 for insight literature has on fitness function for evolutionary algorithms. The representation of melodies for the algorithm is arguably just as important.

For this project the focus is on evolutionary algorithms and as such other procedural means of music composition will be neglected. There is too much work in music classification and as such the focus will be on only a few possible algorithms.

Fitness functions for genetic algorithms are considered in chapter 5

4 MUSIC CLASSIFICATION

4.1 INTRODUCTION

In this section we will investigate methods of music classification. If some algorithm is a good method to rate the closeness of a song to a genre or style then it may also serve as a good fitness function for a evolutionary algorithm.

4.2 NORMALIZED COMPRESSION DISTANCE

4.2.1 *Background*

The Kolmogorov complexity of piece of text is the measure of the computable resources needed to specify the text. The complexity of a string is the length of the shortest possible description of the string in a fixed universal description langauge [34].

The information distance between two string x and y is defined as the length of the shortest program p that can compute x from y and y from x The length of p can be expressed using Kolmogorov complexity [35]:

$$|p| = \max\{K(x|y), K(y|x)\}$$

The information distance p is a absolute measure. A more useful similarity metric is one that expresses the distance in relative terms. The Normalized Information Distance (NID) is given by [36] :

$$\mathrm{NID}(x, y) = \frac{\max\{K(x|y), K(y|x)\}}{\max\{K(x), K(y)\}}$$

The concept of NID is important, however it is not computable. An approximation of the normalized information distance is commonly used. $K(x)$ is approximated by $Z(x)$ where $Z(x)$ is the binary length of a data x compressed by a compressor Z.

$$\mathrm{NCD}(x, y) = \frac{Z(xy) - \min\{Z(x), Z(y)\}}{\max\{Z(x), Z(y)\}}$$

where $Z(xy)$ is the length of $x + y$ compressed by Z. Any good compressor may be used for Z such as

- gzip
- bzip2
- Lempel-Ziv and its variations

4.2.2 *Literature*

Normalized compression distance has been used in a variety of cases. It has been used in applications of general clustering and classification of data in arbitrary domains. This includes music classification [31].

Cilibrasi and Vitiyani used the Normalized Compression Distance to approximate the Kolmogorov Distance between different musical pieces as a method to compute clusters of music [31]. The MIDI files were pre-processed such that when two notes occur at the same time only the note with the highest pitch is kept. The music was represented as a string and the distances between different musical pieces was computed.

Ctaltepe, Sonmez and Adali used the normalized compression distance and used it to classify music pieces using k-nearest neighbors [37]. The training data has a label associated. The closest k training data (by NCD) to a song is obtained and the most frequent label in the k set is used to classify the musical piece. Ctaltepe Sonmez and Adali found that the distance measure works better when more training data is available and the performance is dependent on how the input data is pre processed. The best results were obtained when the midi files were sampled at 1ms and the $k = 1$ nearest neighbor identification was used. The music was represented in the following format: outputting the first note and then the difference in pitch between consecutive notes. Using the above means a classification accuracy of 79% was achieved on 57 midi files.

Li and Sleep have also found that the 1-nearest neighbor with a Lempel-Zip compressor outperformed more complex statistical methods and compressors. Using relative pitch intervals in the music representation outperformed using absolute pitches. A performance of 92.35% was obtained [19]. The midi files were organized into four categories Beethoven (302 files), Haydn (261 files), Chinese (80 files) and Jazz (128 files). The dataset is unbalanced and the study does not make it clear which partition of the dataset was used as training samples and what partition was used for verification.

In [33, 38, 19, 7] it was found that the normalized compression distance serves as a promising fitness function for genetic algorithms for automatic music generation. Thus NCD may be viably used as a fitness function for a genetic algorithm and as a metric to help classify music.

4.3 NEURAL NETWORKS

McKay investigate using K-nearest neighbor techniques and artificial neural networks in order to classify MIDI music by genre. He included some of the following metrics as input to the neural network [39]:

- Number of notes - standard deviation of number of notes activated in each channel
- Note duration - standard deviation of total duration of notes
- Dynamics - standard deviation of average volume of notes

- Melodic Intervals - average melodic interval
- Simultaneity - average number of notes that are played concurrently
- Note density - average number of notes per second
- Average time between attacks - average time between note activations
- Initial tempo - tempo in beats per minute
- Pitch variety - number of pitches used at least once
- Most common pitch class - Most common pitch divided by number of possible pitches

For a full list please see his MacKays dissertation on MIDI classification [39] Using the complete list of metrics neural networks had a success rate of 83%.

5 FITNESS FUNCTIONS

5.1 INTRODUCTION

In this section we will briefly review some functions that may be used as a fitness function for a genetic algorithm.

Algorithms that help classify music could possibly be used as fitness functions, as this would rate the similarity of the evolved piece of music to a genre or style. These algorithms are found in section 4. If work has been done that has used the music classification algorithm as a fitness function it will be listed in this section.

Fitness functions can be categorized as follows:

- Interactive - The user provides the fitness of a piece of music
- Expert systems - The musical fitness is assigned according to best practices commonly studied in music theory
- Learning based functions - Fitness functions that learn from previous data

Learned fitness functions are sensitive to input data and the selection of features is a difficult task [8].

5.2 ZIPF'S LAW

Zipf's law states that the frequency of an event is inversely proportional to its statistical rank, that is:

$$f \propto r^{-a}$$

where f is the frequency of occurence of a particular event and r is the statistical rank. a is close to 1. Zipf's law can also be stated as:

$$P(f) \approx \frac{1}{f^n}$$

where $P(f)$ denotes the probability of an event of rank f and n is close to 1

One can determine whether melodic intervals follow Zipf's law by counting the melodic intervals in a piece. The result is usually plotted on a logarithmic scale and is known as a rank-frequency distribution

Zipf found evidence for the theory in music. An analysis of Mozart's Bassoon Concerto in Bb Major revealed an inverse relationship between the length of intervals between repetitions of notes and the frequency of their occurrence [40].

Several other different rank frequency distributions can be obtained for a musical piece. These include [12]:

- Pitch - distribution of MIDI pitches
- Chromatic tone - distribution of 12 chromatic tones
- Duration - note durations
- Melodic interval - distribution of melodic intervals
- Melodic bigrams - distribution of adjacent melodic interval pairs

Linear regression is performed to obtain the slope of the distribution. The coefficient of determination R^2 is also computed in order to determine how well the slope fits the data.

Figure 10 plots the rank frequency distributions of various metrics for the Beatles' song Let It Be. The figures were generated by Jensen for his thesis [8]. The metrics can be seen to follow a Zipfian distribution. Results by Manaris also indicate that most music pieces display near Zipfian distributions [12].

Zipf-based metrics capture essential parts of scaling properties in music. These metrics indicate that music follows a distribution balanced between near-zero slope and steep negative slope. Different styles of music have different slopes. There exists also a correlation between Zipf metrics and human preference [12].

Jensen used a Gaussian to define the target fitness as [8]:

$$f_m(a, b) = e^{(\frac{b-a}{-\lambda})^2}$$

where a is the metric slope of an evolved piece of music, b is the target slope and λ is the tolerance

Since there are several metrics for a given piece of music the fitness function should incorporate these. Jensen used the weighted sum of several metrics.

$$f(\vec{a}, \vec{b}) = \sum_{i=1}^{N} w_i f_i(a_i, b_i)$$

Jensen has found that Zipf metrics can be used to evolve pleasant music using a tree-based representation, however the majority of the evolved melodies were unpleasant [8]. Zipf metrics only capture the scaling properties of distributions and ignore the musical events that account for different frequencies. Zipf's law neglects musical content and can be seen as knowledge weak. Jensen concluded that the Zipf metrics are insufficient for musical fitness alone.

Manaris had more success with Zipf metrics however he used them as input to an artificial neural network to evaluate the fitness of melodies, however Manaris states it is wiser to use the fitness function in a partially interactive system [12]

1 Figure generated by Jensen for his thesis [8]

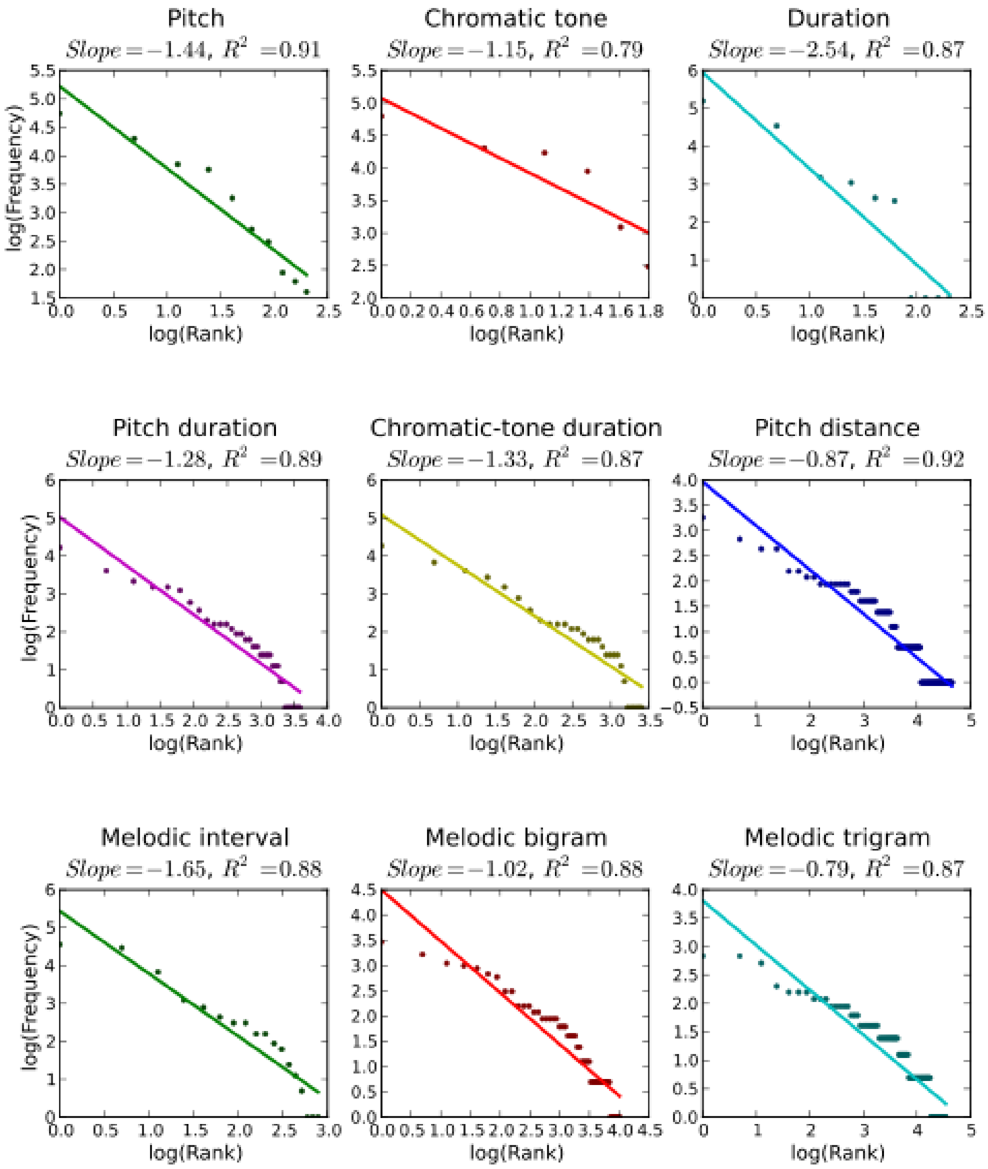

Figure 10: Rank frequency distributions and slopes of different metrics for The Beatles' Let It Be [1]

5.3 COSINE SIMILARITY

In Information Retrieval cosine similarity is commonly used to asses the similarity of two documents:

$$\mathrm{sim}(\vec{A}, \vec{B}) = \cos\theta = \frac{\vec{A} \cdot \vec{B}}{|\vec{A}||\vec{B}|}$$

where $\vec{A}$ and $\vec{B}$ are two document vectors

In order to rate the similarity between music scores features such as pitches and melodic intervals are used.

As with Zipf's law the fitness is the weighted sum of similarity measures:

$$f(\vec{A_1}, \ldots, \vec{A_n}; \vec{B_1}, \ldots, \vec{B_n}) = \sum_{i=1}^{N} w_i f_i(\vec{A_i}, \vec{B_i}) \tag{4}$$

The fitness function rates the fitness of the evolved individual with a target piece.

Jensen conducted multiple experiments using the fitness function in equation 4. At first only a single metric was included and thereafter multiple metrics. As more metrics were included the evolved melodies became more similar to the target piece. More pleasant melodies were evolved when the target piece was The Beatle's Let It Go than Mozart's Piano Sonato No. 16. There was no correlation between melody and rhythm. Metrics included for the fitness function were:

- Pitch
- Chromatic tone
- Melodic interval
- Melodic bigram
- Rhythmic interval
- Rhythmic bigram

Jensen concluded that the results obtained by the cosine simalirity fitness function were more pleasant than those obtained by Zipf's law as Zipf's law rates music on scaling properties only [8].

5.4 NEURAL NETWORKS

Different forms of networks have been used as a fitness function for evolutionary algorithms. Some of these include:

- Adaptive resonance theory neural networks using binary classification patterns [10]
- Recurrent neural networks

- Cascade correlation neural network designed to reduce GenJam bottleneck [9]

Some common problems with neural networks as fitness functions is that they require a lot of time to be trained, require a good representation for a set of inputs to map to an output and the structure of the neural network is fixed after training [10].

In [9] Biles tried to design a cascore neural network to rate musical scores. Since a neural network outputs fitness based on the input parameters the choice of input metrics are important. Metrics that included the number of new note events in a measure, the number of unique new note events, the size of the maximum interval, the number of changes in a direction between adjacent notes failed to capture the fitness for the Artificial Neural Network (ANN) [9].

Biles argues that the reason for this is that humans listen to music in more complex ways and that simple statistical measures fail to capture this. Zipfs law, which only captures the scaling properties of music also yielded poor results as a fitness function for similar reasons [8] (See section 5.2).

An **Adapative Resonance Theory (ART) network** has been used as a fitness function whereby a Genetic Algorithm (GA) utilizes clustered representations of rhythm styles to interactively generate rhythm patterns to according to a certain style [10].

An adaptive resonance theory network utilizes unsupervised learning and clustering algorithms to recognize patterns. New clusters are created if a pattern cannot be associated with existing clusters. Another characteristic of ART networks is that new training does not cause loss or corruption of old training data [41]

A ART1 network clusters binary vectors. The basic structure of an ART1 network involves (See figure 11) :

1. Input processing field - F1
2. Cluster units - F2
3. Mechanism to control degree of similarity of patterns in the same pattern
4. weighted bottom-up connections between F1 and F2 layers
5. weighted top-down connections between F2 and F1 layers

Burton had the ART network fitness function operate as follows [10]:

1. Each individual in the population is an input to the ANN
2. The network determines the winning cluster
3. The degree of similarity between the individual and the cluster is determined
4. If the degree of similarity is above a certain threshold the individual is added to the cluster

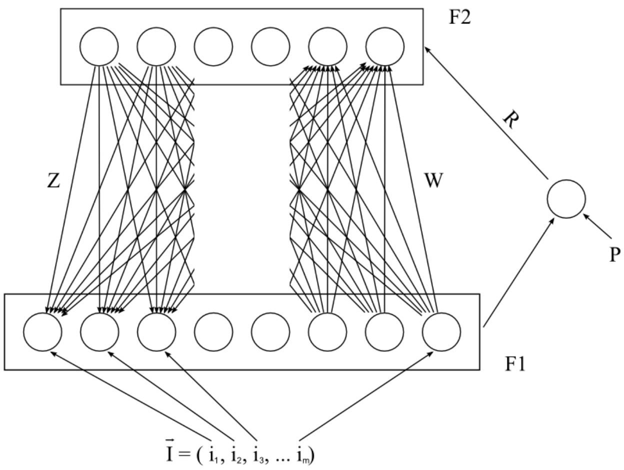

Figure 11: Figure of ART ANN topology

5. If no clusters match the individual closely enough a new cluster is created
6. Fitness is assigned as a degree of similarity to a cluster.

NEUROGEN is another attempt at using a neural network as a fitness function to compose small diatonic, western, four part harmony compositions. The system has shown limited success however it was able to produce 4 bars of music [2].

Chen used a **Simple Recurrent Network (SRN)** with composition rules on tonality and rhythm as a fitness function for a GA [6]. The simple recurrent network has an input layer that represents a measure at time T with the output layer representing the measure at time $T + 1$. A recurrent network is used as a single step predictor to compose music. The network predicts notes at $T + 1$ using the notes at time T. After the network has been trained it can be seeded with inital values to generate novel compositions. The following constraints were used:

1. Pitch diversity constraint - number of measures with unique pitch sequences
2. Rhythmic diversity constraint - number of measures with unique signatures
3. Measure density constraint - ratio of number of notes to maximum notes
4. Pentatonic pitch class constraint - number of notes that belong to pentatonic pitch class
5. Cell structure - ratio number of times cell pattern occurs to maximum number of patterns

The system was able to generate melodies with systematic structure however it lacks global structure. Individual measures sound pleasant and diverse however there is a lacking structure as a whole

5.5 NORMALIZED COMPRESSION DISTANCE

As noted in section 4 the Normalized Compression Distance has been used to help classify music genres. However Normalized Compression Distance has been explored as a possible fitness function for evolutionary algorithms [33, 38, 7]

The fitness function used by Alfonseca et. al for an individual x and a guide set G was defined as [7]:

$$f(x) = \left(\sum_{g_i \in G} \mathrm{NCD}(x, g_i) \right)^{-1} \tag{5}$$

Where g_i is a guide in guide set G containing M musical pieces and x is the set of differences between consecutive notes.

Alfonseca encoded the chromosomes as N vectors containing a pair of integers. The first integer denotes the note interval and the second represents the length as a multiple of the smallest unit of time [33].

5.6 INTERACTIVE FITNESS FUNCTIONS

Genetic algorithms which employ user interaction as a means of rating the fitness of are known as Interactive Genetic Algorithms (IGAs).

In [13] constructs a system that composes 16-bars music using a GA. The user rates individual chromosomes, new chromosomes are applied by genetic algorithm and the user is asked to rate the individuals again. Should the user find a good piece they may favorite it. The fitness of the chromosomes were seen to increase as the generations increased, however it is unknown whether the melodies were pleasant.

Using an interactive fitness function may lead to better results than most other fitness functions however it is a tedious and demanding process and may also lead to inconsistencies in evaluation. Some researches try to reduce this effect by constructing ANNs which learn the user's ratings, and as such may be used in place of the interactive fitness function [9, 14]. See section 5.4 for the use of ANNs as fitness functions.

Johnson and Poli had a user rate individual sequences and trained a neural network base automatic rater, which may replace the user in larger runs. The musical pieces generated by the automatic rater were pleasant but they were not as pleasant as the musical pieces generated by the user interactive runs [15]

The superiority of a interactive fitness can be seen, as a person can rate the pleasantness, or fitness of a song much more accurately than current quantitative fitness functions. However this imposes a bottleneck on the system as it is time consuming. A partially interactive system may yield a good compromise [9].

6 CONCLUSION

We have investigated numerous methods to compose music algorithmically. The two most prominent methods currently to compose music is through genetic algorithms and neural networks.

Genetic algorithms require proper music representation and a good fitness function. Multiple work has gone into investigating various possible fitness functions. The three most promising candidates are NCD, ANN and cosine similarity.

Interactive genetic algorithms yield good results although this imposes a performance bottleneck on the system, as the system is required to wait for user input. This is a time-consuming process although a partial interactive system might be viable.

Simple feed-forward neural networks are unable to compose music due to their inability to encode temporal information. Recurrent neural networks were investigated and early findings yielded poor results, though LTSM networks seem promising.

Algorithmic music composition is viable although there is large room for improvements to be made. Currently only short musical pieces sound pleasant. Longer pieces tend to be repetitive and lack global structure.

Most current methods restrict their domain in order to investigate only the main research questions.

A hybrid approach may yield good results.

Part III

CONCEPTUAL DESIGN

7 INTRODUCTION

In this part of the document a conceptual design is proposed to solve the problem of composing music algorithmically. Composing music algorithmically can be done in a variety of ways. Different types of algorithms can be utilized:

- Algorithms that utilize expert knowledge. Expert systems utilize knowledge from a specific domain in order to make decisions.
- Algortihms that learn. These algorithms utilize pattern recognition to learn from data in order to make decisions.

For an expert system to be used in algorithmic music composition knowledge from music theory is utilized in order for the algorithm to make decisions. When too many constraints are imposed on the algorithm the variety of the type of music produced is lessened.

For this project however, the focus is on algorithms that can learn. In order to design an application that utilizes machine learning algorithms to compose music the project will first be decomposed into discrete units. The function and interaction of each unit will be investigated.

Different types of machine learning algorithms will be investigated and the best solution will be chosen that satisfies the requirements of the project.

8 SYSTEM ARCHITECTURE

In this section the project will be broken down into discrete functional components and the interaction between different components will be investigated.

Figure 13 shows the interaction of the user with the application. The user interacts with the application and selects a certain style of music to be composed. The application generates the music according to a certain algorithm and plays back the generated piece.

Figure 14 indicates a conceptual user interface with the primary elements. A user is able to select a certain style of music for composition. Once the piece is generated the user is able to save the music to a MIDI file.

The core functionality of the application resides in the algorithm that is responsible for composing a piece of music.

By incorporating all these elements we can concretely describe the project in discrete components in a functional architecture diagram, see figure 12.

8.1 FUNCTIONAL ARCHITECTURE

8.1.1 *Functional Unit 1 - Operator*

The operator is responsible for interacting with the application. The operator will select the style or category for composition and instruct to application to compose music. If a certain piece of music is to be save the operator will be

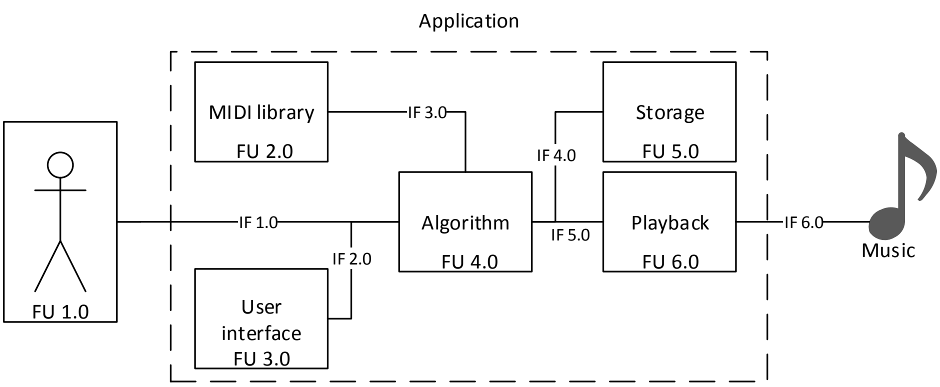

Figure 12: Functional architecture

responsible for instructing the application to do so and to select the location the MIDI file is to be saved

8.1.2 *Functional Unit 2 - MIDI library*

The MIDI library is a large collection of MIDI files. The files in the MIDI library are organized into categories. Each category represents a certain style of music The algorithm will utilize a subset of the library (a category) as input.

8.1.3 *Functional Unit 3 - User Interface*

The user interface is the front end of the application. The operator interacts with this functional unit in order to instruct the application what to do.

The user interface should be designed in a user-friendly manner in order to accommodate the operator. A conceptual user interface is shown in figure 14.

This user interface allows the user to select a certain style, instruct the algorithm to compose music and to save or play back a piece of music once it is composed.

8.1.4 *Functional Unit 4 - Algorithm*

The algorithm is the central part of this project. The algorithm converts the input MIDIs into output MIDI.

The algorithm will utilize a category of MIDI files from the MIDI library in order to compose a new piece of music not in the MIDI library.

The output piece of music will represent the style of music that was used as input. The possible types of algorithms were discussed in section

8.1.5 *Functional Unit 5 - Storage*

This unit is responsible for storing the output MIDI from the algorithm into a MIDI file.

8.1.6 *Functional Unit 6 - Playback*

This unit is responsible for playing back the output MIDI from the algorithm through an audio output device

8.2 INTERFACES

The interfaces indicate the interaction between different functional units. The interfaces have the following functions:
Interface:

1. Interface between the user and the user interface. This input would be from a input peripheral device.
2. Interface between the user interface and the algorithm. The interface calls the algorithm with the parameters supplied by the user such as when to start and what type of style was selected
3. Interface between the MIDI library and the algorithm. The input to the algorithm is a selection of MIDIs. The interface converts MIDI files into a format required by the algorithm
4. Interface between the algorithm and storage. This interface converts the output from the algorithm into the MIDI file format
5. Interface between the algorithm and playback. This interface converts the output from the algorithm into a format that is ready for playback through the playback functional unit.
6. Interface between the playback and the music. This represents the audio output device and it's workings.

9 OPERATIONAL FLOW

The operational flow indicates the interaction between the operator and the application and how the operator should use it.

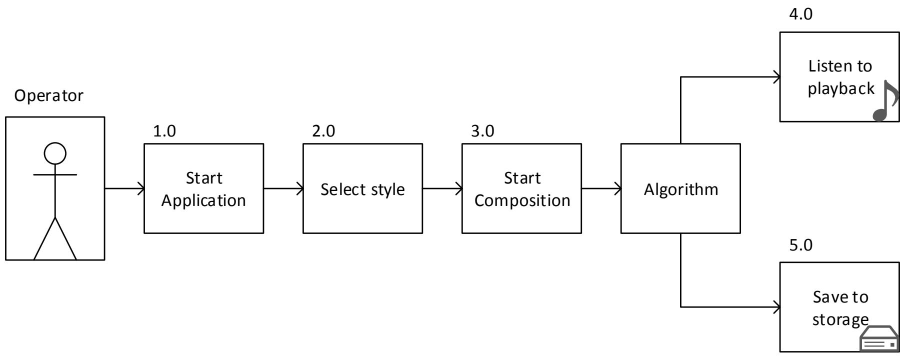

Figure 13: Operational Flow

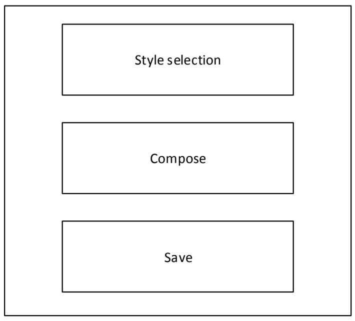

Figure 14: Conceptual user interface

Figure 13 shows how the operator interacts with the application. Figure 14 shows a conceptual user interface with which the operator would interact.

For this application, the operator selects the style of music to be composed; instructs the application to compose the music according to the selected style and then to instruct the application to either play back the piece of generated music or save it to storage.

10 CONCLUSION

In order to solve the problem of generating music algorithmically the task was broken down into its functional architecture. From this each functional unit and its interaction is made apparent. The core component of this project is the algorithm which is used to compose music

Part IV

ALGORITHMS

11 INTRODUCTION

The system operates on MIDI files. MIDI events are processed and the resulting notes are stored in a sequence where the pitch and duration of each note are recorded. A note is represented in MIDI as a number from 0 to 127. Thus for each MIDI track a monophonic melody can be extracted.

A large set of MIDI files (600) were collected for style of music. The following categories were used:

1. A mixture of Classical music (665)
2. A mixture of retro video game music (395)
3. A mixture of Pop and Top40 (1370)
4. A mixture of Classic Rock (2274)
5. A mixture of Jazz music (114)
6. A mixture of Dance and Techno (214)

These categories are not the final list of styles that will be included into the end user application.

12 REPRESENTATION

This section discusses how music is represented internally in the system and how the data structures are designed. Proper data structures are extremely important for good algorithms [1].

12.1 ASSUMPTIONS AND SIMPLIFICATIONS

For this system our model of music is simplified and we do not make provisions for chords. Melodies are monophonic and are simply a list of notes of their durations.

Furthermore, we assume all MIDI files that correspond to **Track type** 1.

12.2 NOTE

A note is the most basic element. A note consists of:

1. Note duration - A whole note, half note, quarter note and so on.
2. Note pitch - A number representing the note pitch, in MIDI scale. $0 \leqslant P < 128, p \in \mathbb{Z}$

From these properties the following can be derived:

1. Octave - Every twelve note pitches represent one octave
2. Chromatic tone - An integer from 0 to 11. Also corresponds to the name of the note for example C, C#, G, A#

These properties are related to the Note pitch by:

$$P = 12o + p$$

where o is the octave and p is the chromatic tone

12.3 MELODY SEQUENCE

A melody sequence is simply a list or sequence of notes:

$$\{N_1, N_2, \ldots, N_n\}$$

Where N is a note.

Most algorithms operate only on the melody sequence and output a melody sequence. Some algorithms may optionally require information on the instrument in which the melody sequence is played.

1 Bad programmers worry about the code. Good programmers worry about data structures and their relationships. Linus Torvalds

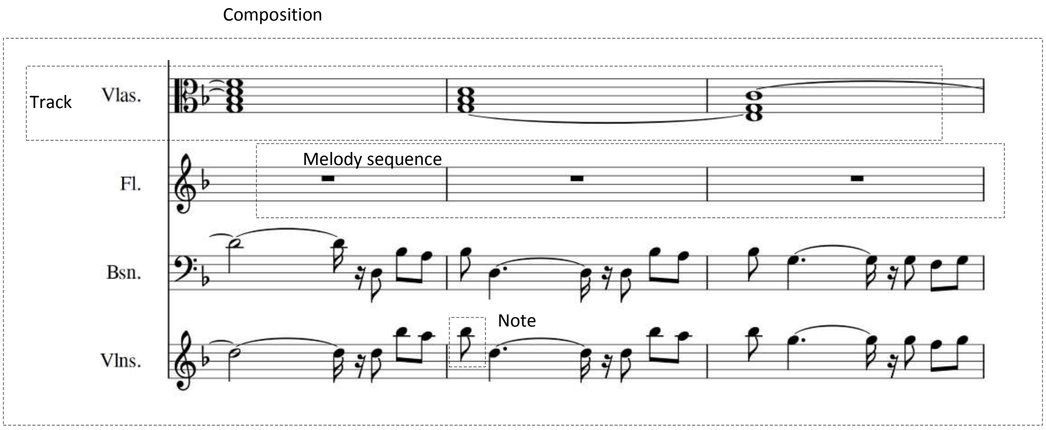

Figure 15: Illustration of a composition and its elements

12.4 TRACK

A track contains a sequence of melody sequences and is also described by an instrument in which the sequence are played.

12.5 COMPOSITION

A composition is a sequence of tracks which can be played in parallel. MIDI files are converted into compositions.

Figure 15 illustrates the relation between the above mentioned structures for a midi file.

13 MELODY GENERATION WITH A HIDDEN MARKOV MODEL

13.1 INTRODUCTION

An attempt was made to use a hidden markov model to generate melody sequences.

A HMM is a statistical Markov model where the assumption is made that the system being modeled is a Markov process with hidden states. See section 3.1 for more information on hidden markov models for melody generation.

13.2 METHODOLOGY

For this experiment a simple HMM is constructed with a forward structure. All uniques notes are linked with a unique integer id with the use of a dictionary. The amount of states of the HMM was constrained to 100. Notes were constrained to only be in the 5th Octave, this helps reduce the unique number of symbols. In addition the duration of all notes were restricted to be in {tn, sn, en, qn, hn, wn } where wn is a whole note, qn is a quarter note and so on.

In order to train the HMM, that is to find the emission and transmition matrix K-Means clustering was used. K-Means clustering partitions n observations into k clusters into which each observation gets assigned to the cluster with the nearest mean.

13.3 RESULTS

The training time increases greatly as the number of symbols increases and thus it is necessary to constrain the number of symbols. The Viterbi algoritmh is expensive and requires time proportional to the product of the number of states and number of edges in the model, other algorithms are even slower. Overall the HMM produced melodies without structure.

Figure 16 shows the melody generated with the HMM. The resulting melody sounds chaotic and lacks overall structure.

Figure 16: Melody generated with a HMM

14 INSTRUMENTAL GENERATION WITH MARKOV CHAINS

14.1 INTRODUCTION

For this experiment a compositional engine will be designed that outputs melodies for a specific instrument.

In order to construct such an instrumental generator an algorithm is required that can predict time-series data.

A large advantage of recurrent neural networks over Markov chains and HMM is that neural networks have greater representational power and can take into account syntactic and semantic features. A RNN does not make the Markov assumption and is able to take into account long term dependencies.

Markov chains have the advantage of being much simpler to implement and are extremely fast and efficient with the proper implementation. Recurrent Neural Networks are more difficult to implement and to train. Various training algorithms exist to train a RNN. Training a recurrent neural network is slower than a regular feed forward network. The poor results obtained from the LSTM network (See chapter 15) dissuade us from using them for the instrumental generator.

This concept will utilize Markov Chains to construct a probabilistic model for a sequence of notes. The probability of the next note occuring depends on the previous state of the chain. See section 3.2.

14.2 METHODOLOGY

For this technique a different Markov Chain will be constructed for each instrument. For a certain instrument, all the notes for that instrument in a MIDI file were added to the Markov chain. This chain is used to generate notes according to a certain instrument.

A third order Markov Chain was used. That is, the previous three notes constitute the state of the model. A lower order would result in more novel and random melodies where a higher order would produce melodies with higher similarity.

14.3 RESULTS

Figure 17 shows a melody generated with a Markov Chain for the Piano. A third order Markov chain was used with this melody and overall it sounds pleasant. Melodies produced with Markov Chains dont have an overarching theme but do have a local structure and theme.

Melodies produced with Markov Chains are in the same style as the input songs. This reduces compositional value and novelty, especially for higher

Figure 17: Instrumental melody generated with a Markov Chain

```
Data : MIDI files that accord to a certain style
Result : Markov Chain for a particular instrument
initialization;
for each MIDI file do
    for each Instrument do
        if instrument is selected instrument then
            add notes to Markov chain;
        end
        ;
    end
end
```

Algorithmus 1 : Markov Chain for a particular instrument

order Markov Chains. A low order Markov Chain produces more novel although more chaotic and random melodies.

14.4 POTENTIAL IMPROVEMENTS

- Define which instruments sound good with other instruments
- Pick which instruments can be slotted together
- Ensure different tracks accompany each other and are in harmony

15 ACCOMPANIMENT GENERATION WITH A LSTM NETWORK

15.1 INTRODUCTION

The LSTM network is a Recurrent Neural Network (RNN), however it is more optimized and better suited to classify, process and predict time series in the case when there are long lags of indeterminate length between important events. In this experiment a LSTM network is used to generate an accompaniment track for a main melody track

15.2 METHODOLOGY

The LSTM accompaniment is also generated for a specific instrument. Since it is difficult to determine which track is the main melody of a composition an assumption is made that the main melody track is the first track found within a MIDI file. The accompaniment track is another track within the MIDI file.

For the LSTM network all activation functions were tanh, the network had 2 input units, 15 hidden units, 150 output units, a learning rate of 0.1 and momentum of 0.9.

A list of all possible notes were obtained and the problem was treated as a classification problem. For a set of n possible notes the network has n possible active output units and the active output unit indicates the index of which note was activated. This was done in order to reduce the possibility of incorrect notes as even small error might cause problematic results in alternative representations (if the pitch and duration were used as output).

The input for the neural network is the note pitch and duration of each note in the melody track and the output is the index of the note activated in the accompanied track at the same index.

In order to reduce the dimensionality of the output vector filtering is applied. There are bound to be events that occur rarely which can be seen as noise, and the emphasis should be on more prominent notes. We apply a simple filtering to discard notes which have a normalized frequency below some threshold in the dataset.

$$\frac{f}{N} < k$$

where f is the frequency of the note, N the total number of notes and k the threshold.

Since a melody track is stored as a sequence of notes, a note in the melody track and a note in the accompaniment track at the same index might not

Data : MIDI files that accord to a certain style
Result : LSTM training for a particular instrument
initialization;
for *each MIDI file* **do**
    for *each track other than melody track* **do**
        if *instrument is selected instrument* **then**
            for *each note in melody track and respective note in accompaniment track* **do**
                add normalized pitch and -duration of current melody note as inputs to dataset;
                add output note to list of uniques notes;
                add output note index as normalized output to dataset;
            end
        end
    end
end

Algorithmus 2 : Training set for LSTM network

occur at the same time. Thus we iterate through the melody track and use the note closest in time in the accompaniment track for the algorithm.

Training LSTM and other recurrent neural networks is slow, as such the weights and structure of the network are saved to storage and is precomputed for each instrument and style of music.

15.2.1 *Results*

Since the network requires an input melody track to construct an accompaniment track it does not face the same problems such as lack of structure which most other non-recurrent neural networks do.

The output accompaniment can sometimes be pleasant, however a beter timing relation is required between the input and output notes. Most of the produced melodies are repetitive, see figure 18. The network does not produce an accompaniment which is in harmony or in phase with the main melody track.

The results of the LSTM network were surprisingly poor, in comparison with the Markov Chains. A RNN has greater representational power and can take into account syntactic and semantic features. A RNN also does not make the Markov assumption and is able to take into account long term dependencies. The poor performance of the LSTM network is attributed to the implementation and the large training time required for good results.

Figure 18: Accompaniment melody produced with a LSTM network

16 ACCOMPANIMENT GENERATION WITH A FEED FORWARD NETWORK

16.1 INTRODUCTION

In this experiment an attempt is made to generate an accompaniment track for a melody using a feed forward network. One of the reasons for this experiment was to reduce the training time, as recurrent networks have a very large training time.

16.2 METHODOLOGY

For this prototype we construct a neural network with the following parameters:

1. all units use the sigmoid activation function $\frac{1}{e^{-x}}$
2. 1 input layer, 1 hidden layer, 1 output layer
3. 2 input units
4. 2 output units
5. 10 hidden units

The data for the network consists of the melody track and the accompaniment track. For each note in the melody track the note pitch and duration is given as inputs to the network, the output is the note pitch and duration of the corresponding note in the accompaniment track.

16.3 RESULTS

17 ACCOMPANIMENT GENERATION WITH A MARKOV MODEL

17.1 INTRODUCTION

For this experiment an attempt is made to produce an accompaniment based on the frequency of certain notes occuring in the accompaniment track given the melody track.

More specifically, for each note in the melody track the odds are calculated for each possible accompaniment note that may occur at that time. Given an input melody, for each note in that melody an accompaniment note is produced according to the calculated probabilities.

17.2 METHODOLOGY

Data : MIDI files that accord to a certain style;
Result : Accompaniment melody sequence;
for *each MIDI file* **do**
  Get main melody;
  for *each accompaniment track in file;*
  do
    for *each note* N_m, i *and previous note* $N_{m,i-1}$ *in main melody and corresponding note* N_a, i *in accompaniment melody;*
    do
      Increment frequency of N_a, i in frequency table for $\{N_m, i, N_m, i-1\}$;
    end
  end
end

Algorithmus 3 : Constructing frequency table for model

Figure 19 illustrates a Markov Model. The states s_i denote the notes of the main melody (for a first order Markov Chain) and the states y_i denote the accompaniment notes for melody track. In the case of a HMM he states s_i are commonly hidden, however they are known in this case. The states y_i are the output notes that we are interested in generating.

For this experiment the states are setup up as a second order Markov Chain; that is the next note is determined by the previous two notes. For this model we are not interested in determining thee next state s_{i+1} thus a transmission matrix is not learnt. The probabilities of the output notes y_i are learnt instead.

Data : Main melody sequence
Result : Accompaniment melody sequence
initialization;
Create empty accompaniment sequence;
for *each note* N_m, i *and previous note* $N_m, i-1$ *in main melody sequence* **do**
  Lookup frequencies of possible output notes in frequency table for $\{N_m, i, N_m, i-1\}$;
  Obtain probabilities for next notes in frequency table;
  Obtain resulting note according to roulette selection;
  Add resulting note to accompaniment sequence;
end
Return accompaniment sequence;

Algorithmus 4 : Obtaining accompaniment melody

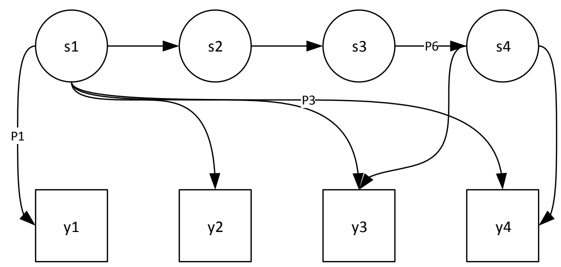

Figure 19: Illustration of a Hidden Markov Model

Figure 20: Accompaniment generated with Markov Model

Since

17.3 RESULTS

Accompaniment generation with a Markov Model produces better results than with a LSTM network. The states of the model are the notes of the main melody and the observations are the notes of the accompaniment. For a short input melody sequence the resulting accompaniment sounds rather pleasant and fits with the main melody. For longer melodies the accompaniment diverges from the main melody and is not in harmony.

18 GENETIC ALGORITHM FOR MELODY GENERATION

For this project a quantitative approach is taken toward algorithmic music composition. In particular quantitative metrics will be used in the fitness functions of the genetic algorithm.

In section the following types of music composition algorithms were investigated:

1. Neural Networks
2. Genetic Algorithms

In the literature review, it was found that the pieces generated by neural networks lack musical coherency and perform poorly as the length of music increases. Some other attempts have met slightly more success although the overarching view for neural networks in music composition seems grim[1].

The decision was made for genetic algorithms as the main composition algorithm for the following reasons:

1. They allow for great flexibility in implementation and music representation
2. The majority of research into machine learning music composition has been into genetic algorithm fitness functions
3. Great amount of variety made possible by different fitness functions and by the representation of music used in GAs

Thus to reiterate, the algorithm employed in the application will be a GA. As stated above a large amount of research has been into the fitness functions of different GAs.

In section 5 the following fitness functions were covered:

1. Zipf's law
2. Cosine similarity
3. Neural Networks
4. Normalized Compression Distance
5. Interactive evaluation[2]

Figure 21 indicates the flow for a genetic algorithm/program.

1 Although neural networks as functions in genetic algorithms have had better success
2 An interactive fitness function imposes a bottleneck on the performance of the system

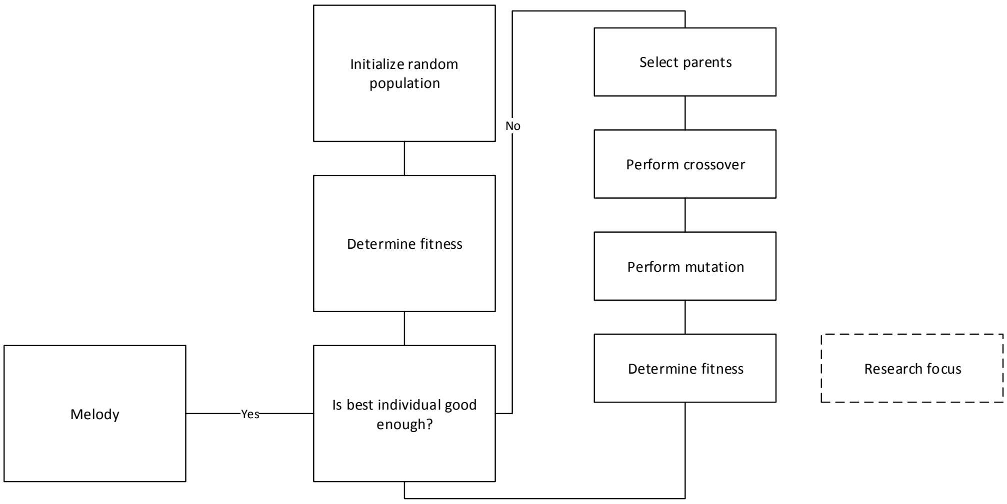

Figure 21: Flow diagram of the operation of a genetic algorithm

18.1 INTRODUCTION

Evolutionary algorithms come in a variety of different types. The two most common types found are Genetic Algorithms and Genetic Programming. Genetic Programming has been found to be more suitable for composing music than genetic programming due to music forming a hierarchical structure.

A flexible genetic programming model will be developed that is able to function with the investigated fitness functions.

The two largest problems for a evolutionary computing problem is:

1. Obtaining a good representation of the problem
2. Obtaining a good fitness function

A genetic algorithm modifies the structure of an individual, if the structure is poorly chosen then a optimal solution wont by found by the algorithm. An ideal fitness function is able to quantify the fitness of an individual. An ideal fitness function in music composition would map the human perception of pleasantness into a fitness value.

18.2 FITNESS FUNCTIONS

The fitness function is a primary research interest in genetic music composition. An ideal fitness function captures the human perception of pleasantness in music.

Of the set of investigated fitness functions only the following functions will be implemented:

1. Cosine similarity
2. Neural networks

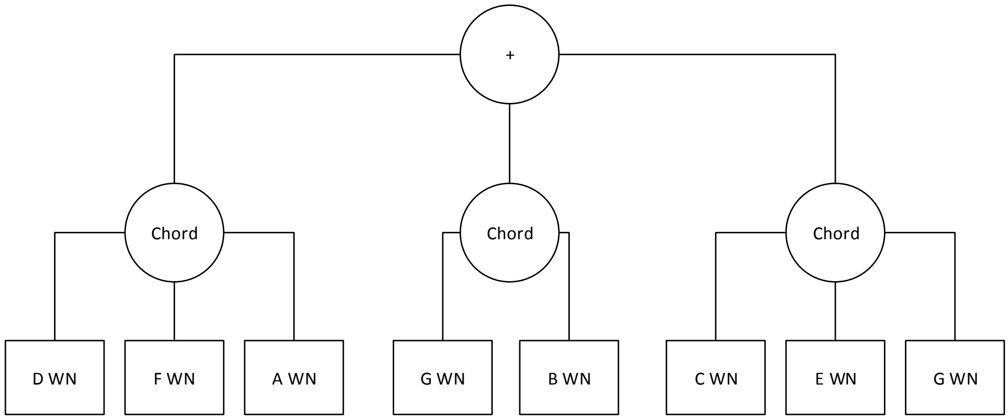

Figure 22: A music piece in a tree structure

3. Normalized Compressions Distance

Since interactive evaluation is slow and Zipf's law is superseded by Cosine similarity.

18.3 MUSIC REPRESENTATION

A flexible music representation will also be developed for the Genetic Programming algorithm. The representation will model MIDI events and manipulation of them. For example:

```
(d wn :=: f wn :=: a wn) :+:
(g wn :=: b wn) :+:
(c bn :=: e bn :=: g bn)
```

Where the :=: operator indicates pieces are played in parallel (chords) and :+: indicates pieces are played in series. a,b,d,e,f,g indicate the note pitch and wn indicates a whole note. Figure 22 indicates this in tree form. Strictly a chord is a set of three or more notes that are played simultaneously. Note that the second branch is only constituted of two notes.

Minsky and Laske [23] argued for a tree representation of music since the tree represents the hierarchical nature of music. The tree representation is much more complex than data structures such as vectors that are used in GAs.

Some authors [4] limit the search space by ensuring that melodies are in a certain scale

18.4 GENETIC OPERATORS

Several different operators can be performed on the tree structure. In figure 22 the serial concatenation and parallel concatenation operators were shown. Some additional operators which may be employed include:

- Repetition - Repeating a segment a given number of times
- Shift note pitches - Shift all pitches of notes by a certain amount
- Duration elongation or contraction - For example a slow operation doubles the duration
- Transposition - Moving note positions relatively
- Retrograde - Reversed order of notes

Genetic operations such as mutation and crossover may be used conventionally.

18.5 METRICS

Feature extraction is required to reduce the search space and to provide the fitness function with musically meaningful measures with which to rate the fitness. In this section we cover some quantitative metrics that may be used as measures to identify or represent music.

The frequencies of a metric is used in the Cosine Similarity fitness function.

The different types of metrics that will be used include:

1. Pitch - List of pitches
2. Pitch differences - Store first pitch, thereafter list consecutive pitch differences
3. Chromatic tone - 12 pitch class. Notes are reorganized into 12 classes.
4. Note durations - durations of notes
5. Pitch distance - intervals between repetition of pitches
6. Chromatic tone distance - intervals between repetition of chromatic tones
7. Melodic interval - intervals between the current note and the previous note
8. Melodic bi-gram - Pairs of melodic intervals
9. Rhythm - duration of a note in addition to the following rest
10. Rhythmic interval - relationship between adjacent note rhythms
11. Rhythmic bi-gram - Pairs of adjacent rhythmic intervals.
12. Chromatic tone duration - A pair of the chromatic tone and the duration

Let p denote the pitch of the note, where $0 \leqslant p < 128$. Then the chromatic tone is given by:

$$c = p\%12$$

where % indicates the modulo operation. The melodic interval is given by:

$$mi_k = p_k - p_{k-1}$$

The melodic bi-gram is given by:

$$b_k = (mi_k, mi_{k+1})$$

Let r indicate the note duration with rests Then the rhythmic interval is given by:

$$ri_k = \frac{r_k}{r_{k-1}}$$

and the rhythmic bi-gram

$$rk_k = (ri_k, ri_r)$$

In this manner we can build a metric vector, let $m_i(A)$ denote a metric's value at position i for musical piece A. Then the metric vector is given by $\vec{M}_A = \{m_0(A), m_1(A), \ldots, m_n(A)\}$

18.6 FITNESS FUNCTIONS

In this section two fitness functions are described which can be used for the Genetic Algorithm. NCD and Cosine Similarity are described here.

The fitness functions described in this section require the phenotype of an individual. Thus it is necessary to parse the genotype, the genetic tree of the individual down into a sequence of notes.

18.6.1 *Cosine similarity*

Some fitness functions such as Cosine similarity and Zipf's law operate on the features of music.

Cosine similarity can be applied to the metrics listed in section 23. Let $\vec{M}_A$ denote the vector of metric values according to a metric m for a music piece A and $\vec{M}_B$ denote the vector by m for a music piece B then the similarity between A and B is given by:

$$\text{similarity}_m(A, B) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|}$$

A set of metrics may be used. The fitness function is then given by a weighted average:

$$f = \frac{1}{N} \sum_{k}^{N} w_k \times \text{similarity}_{mk}(A, B)$$

where w is a weight assigned to metric mk.

18.6.2 *Normalized compression distance*

In order to utilize NCD both pieces being tested need to be encoded in the same way. Musical pieces may be encoded as metric vectors as listed in section 23. More complex metrics may be utilized however there has been no thorough investigation into this.

The NCD as an estimate to the NID was covered in section 4.2.

In order to utilize the NCD as a fitness function the following steps are taken:

1. Encode a set from the MIDI library according to a metric. Let $\Omega = \{\vec{M}_0, \vec{M}_1, \ldots, \vec{M}_n\}$ for musical pieces 0 to n in the MIDI library that accord to a certain style.
2. Encode the population individual x according to the metric (Given by $\vec{M}_x$).
3. Employ the fitness function

The fitness function that will be used is:

$$f(x) = \left(\sum_{\vec{T} \in \Omega} \mathrm{NCD}(\vec{M}_x, \vec{T}) \right)^{-1}$$

18.7 METHODOLOGY

In this section the setup and parameters of the experiment to produce melodies with Genetic Algorithms are outlined.

In order to compose a melody with a genetic algorithm a fitness function is required to rate the fitness of an individual. The following fitness functions were investigated:

1. NCD
2. Cosine similarity

A proper representation is required. A melody is represented in tree form. The tree is given a maximum depth of $(\log_2 x) + 3$ where x is the average number of notes in a melody.

The following genetic operators were used:

1. Concatenation - The addition of two nodes
2. Duration Shift - Scale the duration with a factor
3. Pitch Shift - Move all note pitches a certain number up or down
4. Repeat - Repeat a note a certain number of times
5. Swap - Swap the positions of two notes

The following metrics were investigated: The different types of metrics that will be used include:

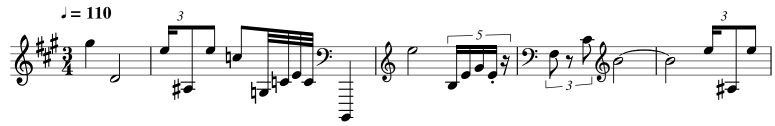

Figure 23: Melody produced with a Genetic Algorithm using the NCD fitness function

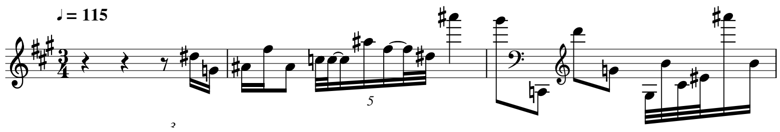

Figure 24: Melody produced with a Genetic Algorithm using the cosine similarity fitness function with chromatic tone duration and rhythmic bigram metrics

1. Pitch differences
2. Chromatic tone duration
3. Chromatic tone distance
4. Pitch distance
5. Melodic interval
6. Melodic bi-gram
7. Rhythmic interval
8. Rhythmic bi-gram

The duration and pitch a note may take on is limited to the available note pitches and durations in the dataset.

The mutation rate was set to 0.1 and the crossover rate was set to 0.9. The population is shuffled every epoch. The GA was run for 1000 generations and the individual with the highest fitness was used as the resulting chromosome. Since the result is in tree form it needs to be parsed into a simple sequence of notes.

18.8 Results

Figure 23 indicates an excerpt of a melody produced with the Genetic Algorithm using the NCD fitness function. The NCD fitness function produced the worst results. The resulting melodies sounding chaotic and random. NCD is

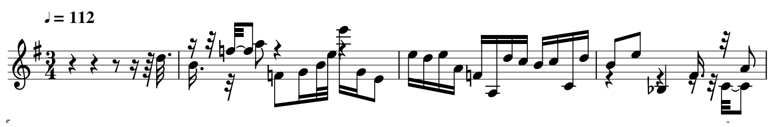

Figure 25: Melody produced with a Genetic Algorithm using the cosine similarity fitness function with all frequency metrics

also slow, as each individual needs to be converted to its text representation and then be compressed in order to obtain its compressed distance.

The melody produced in figure 24 using the cosine similarity fitness function was rather pleasant and the rhythmic bigram and chromatic tone duration -frequency metrics produce pleasant melodies.

The melody produced in figure 25 used the cosine similarity fitness function using all the frequency metrics. At times the melody is pleasant however off-sounding or out of key notes are a common occurence. The melodies also dont have structure. This indicates that some of the frequency metrics either dont work well in conjunction with each other or some frequency metrics have a negative effect on the pleasantness of melodies.

The following frequency metrics produced good or pleasant results:

1. Chromatic tone distance - intervals between repetition of chromatic tones
2. Melodic interval - intervals between the current note and the previous note
3. Rhythmic bigram - Pairs of adjacent rhythmic intervals
4. Chromatic tone duration - A pair of the chromatic tone and the duration

18.9 CONCLUSION

The decision was made to utilize a genetic programming algorithm since the tree structure accommodates the hierarchical nature of music. Genetic programming provides flexibility, variety of possible styles and a large amount of research has been done one fitness functions for genetic algorithms.

Fitness functions require good measures that make it possible to rate musical pieces quantitatively. A set of metrics were developed in which musical pieces can be measured.

Two fitness functions, namely Cosine similarity and NCD were developed to incorporate these metrics.

The results of different fitness functions were compared. NCD performed the worst. Not all of the frequency metrics required for the cosine similarity fitness function produced good results. A subset of good frequency metrics were explored.

Part V

DETAIL DESIGN

19 INTRODUCTION

In this part the design of the application the end user will use will be discussed in detail. Topics covered include the final selection of algorithms in the application. The design of the user interface. A high level overview of the data structures used. The functionality and features of the application and finally some additional considerations will be reviewed.

20 TECHNOLOGY, PLATFORM AND PROGRAMMING LANGUAGE

Some algorithms and features lend themselves to being implemented more easily in certain languages than other. Some implementations of the algorithms considered are notoriously difficult and external libraries were used in this. Thus a programming language needs to be chosen that does not constrain the functionality of the application. The three languages with most support include:

1. Python
2. C# (and other .NET languages)
3. Java
4. C++

Ranked according to subjective estimates on the amount of programming time required, with C++ taking the most time.

Machine learning libraries exist for all of the above mentioned languages. Python has poor support for rich user interfaces. Java is a good contender and is cross platform. Thus in order to make a final selection the consideration is made to the end user. .NET languages fascillate the Windows Presentation Foundation (WPF) - a graphical system for rendering user interfaces, which will allow a rich user experience to be developed.

Winner: *C#*[1]

This additionally has the following advantages:

1. Ability to easily create rich user interfaces.
2. C# runs on the CLR and allows access to the large .NET library[2]
3. Powerful language constructs such as operator overloading, event handling, delegates, powerful multithreading support and generics.

1 The algorithmic code is cross platform and can be run through Mono, even though the graphical front end developed in WPF cannot.
2 Additionally the IKVM project allows access to Java libraries

21 SELECTION OF ALGORITHMS

In this chapter the selected algorithms for the end user application are chosen. The algorithms for consideration are:

1. Melody generation with Genetic Algorithms
2. Melody generation with Hidden Markov Models
3. Accompaniment geneartion with Hidden Markov Models
4. Accompaniment generation with a LSTM network
5. Accompaniment genereation with a feed forward ANN
6. Instrumental generation with Markov Chains

The selection of algorithms will be done by means of elimination. Algorithms that yielded sub minumum results will not be selected for inclusion into the end user application.

22 FEATURES AND FUNCTIONALITY

The main goal of the application is to generate melodies according to a certain styles. The user is to selection a certain category of music and the application will generate a musical piece in that style. The premise is to generate the melodies in a certain style using machine learning techniques that will use the data present in the corpus of MIDI files.

Since training machine learning algorithms take up a lot of time it is not feasible to train the algorithm on the fly. Training will take place beforehand and the resulting data structure of the algorithm will be cached to a file. For example a Neural Network will save its topology and weights to a file; this allows the application to load the file, present inputs to the neural network and output a result without spending time learning the optimal weights for the network.

Additional features would be to allow the user to save a generated melody to a file (in MIDI or a custom file format), load or play an existing MIDI file (for comparison or other aims), re-randomize a generated composition and view more information or properties of the generated music piece.

The application is to be used by end users which do not have background on machine learning. The application is be as user as friendly as possible while still having flexibility in composing songs. The user interface is to visually display the song and playback of it. Playback functionality such as playing and stopping a song are mandatory.

To summarize, the application will:

1. Implement the selected algorithms and allow them to be used for song generation.
2. Allow saving and loading of MIDI files
3. Generate a monophonic melody according to a certain style. Allow the user to select which algorithm to use and adjust high level parameters
4. Playback functionality including playing, pausing and stop a song.
5. Feature a user friendly interface.
6. Visually display the generated melody.
7. Allow the user to randomize the song based on the selected settings.
8. Cache machine learning data to improve performance of the application

23 GRAPHICAL USER INTERFACE

In this chapter the design of the graphical user interface is presented, along with the features and functionality of the application.

As mentioned above the user interface is to have the following features

- Visually display the generated song
- Selecting the category of melodies to be generated
- Select algorithm and high level parameters
- Buttons to allow for saving, loading, pausing, pausing, stopping and playing of songs

The user interface will be developed using WPF and Extensible Markup Language (XAML).

Figure 26 indicates the starting interface of the application. This interface provides controls for playback functionality, is responsible for displaying a visual of the song being currently played, provides saving and loading functionality and allows the user to randomize the current song using the parameters set in the compose view.

Figure 27 allows the user to select the music style or category for music to be generated in. A track is a sequence of notes played in a particular instrument. A user is able to add new tracks using the provided button, which takes them to a new window as in figure 28. If a user double clicks on a specific track the frequency metrics of the track are shown as in figure 29.

Figure 28 allows the user to generate a track using a specific technique. This interface allows the user to select the type of technique for track generation and also displayes the progress. Once a track is generated it can be added to the list of tracks as in figure 27.

Figure 29 displays a list of frequency metrics about the track selected. An example metric is the frequency of musical intervals within a melody ($mi = |p_i - p_{i-1}|$, where p_i is the pitch at index i); more information about frequency metrics can be found in section .

The visualization blocks in figures 27 and 28 will be a simple visual displaying the notes over time. A simplified music sheet representation will be used, something in the vein of figure 30 (though simpler).

Figure 31 shows the state diagram of the user interface.

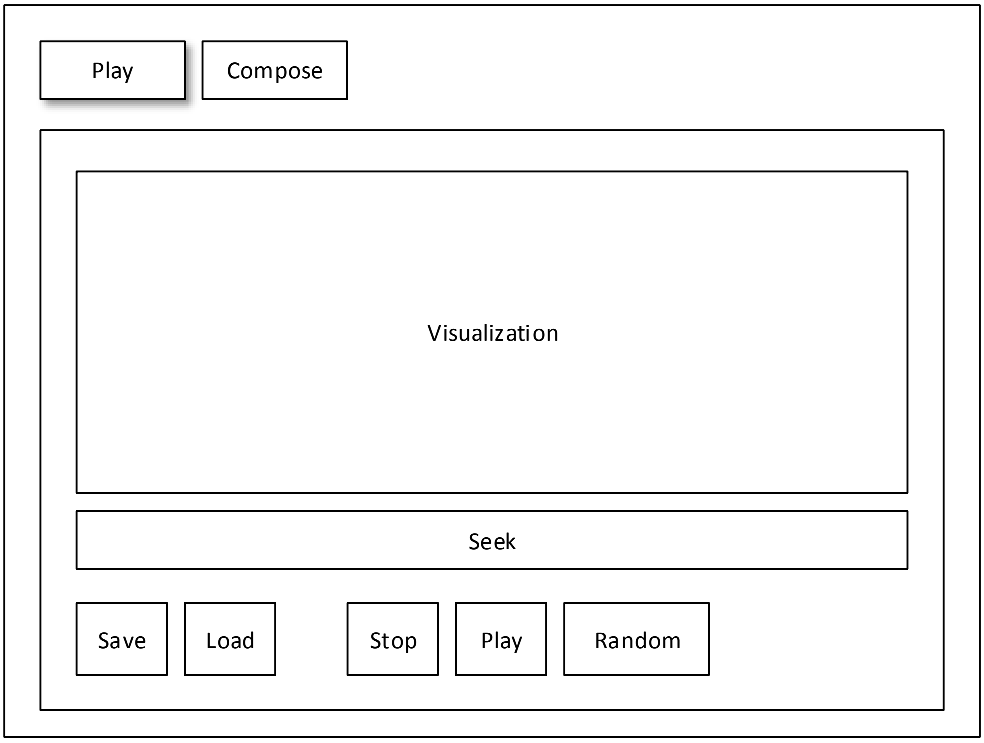

Figure 26: User interface for playback functionality

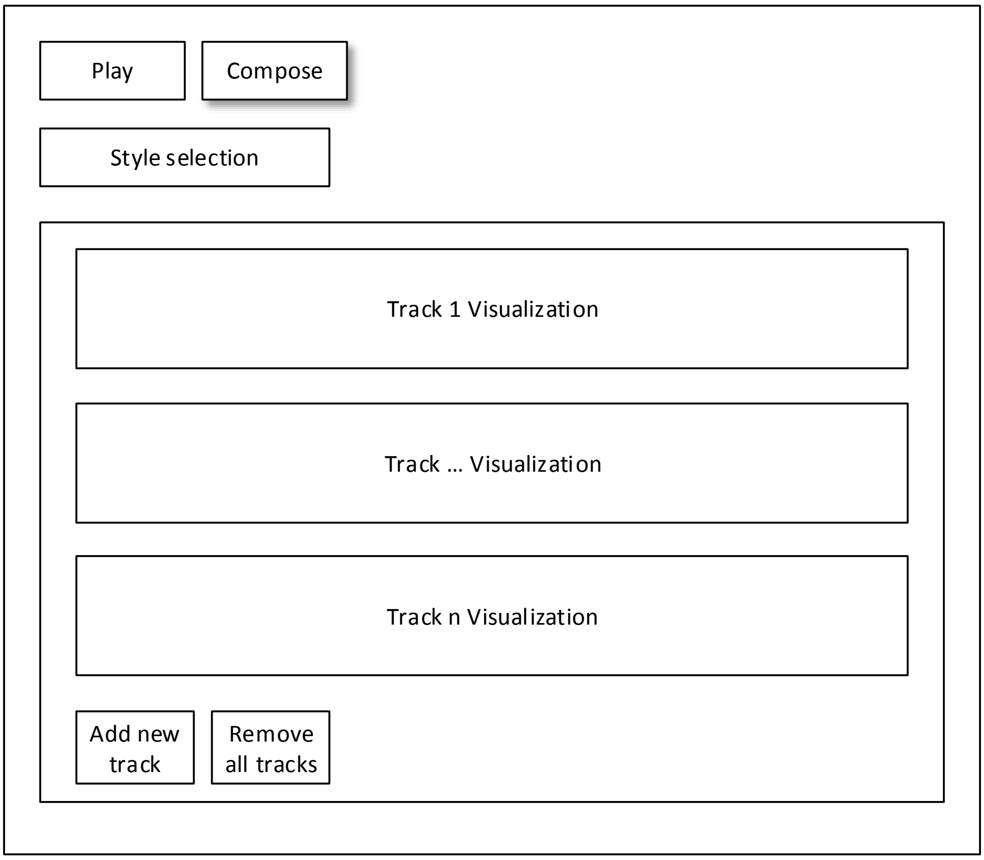

Figure 27: User interface for composition functionality

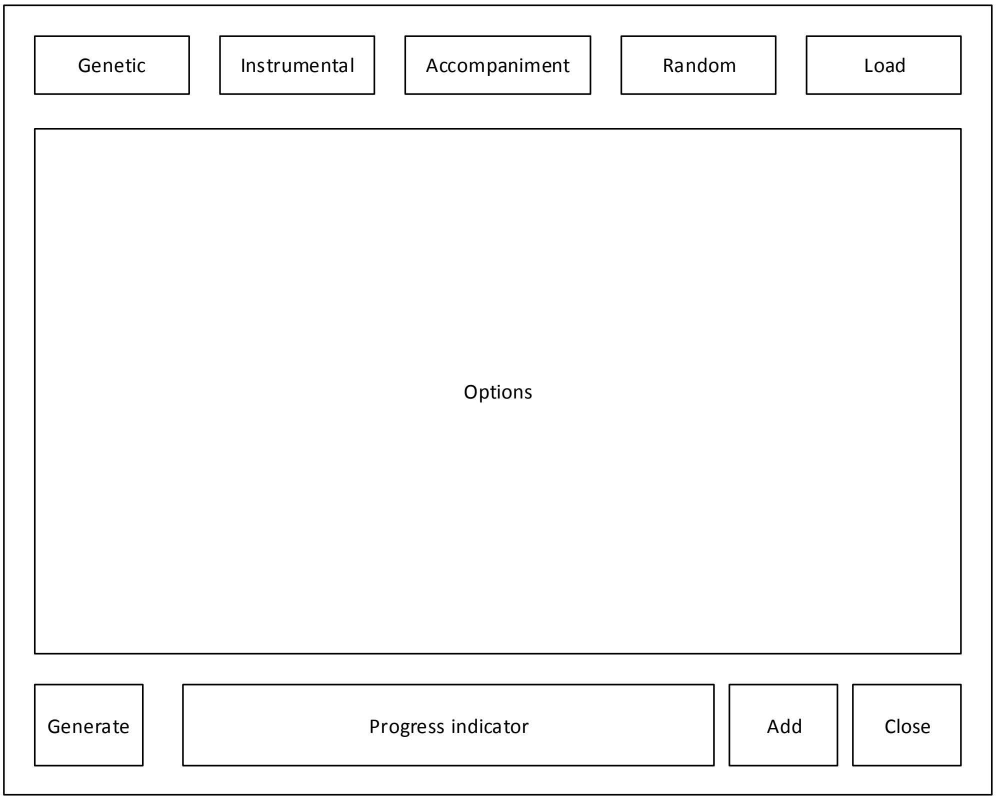

Figure 28: User interface for melody generation

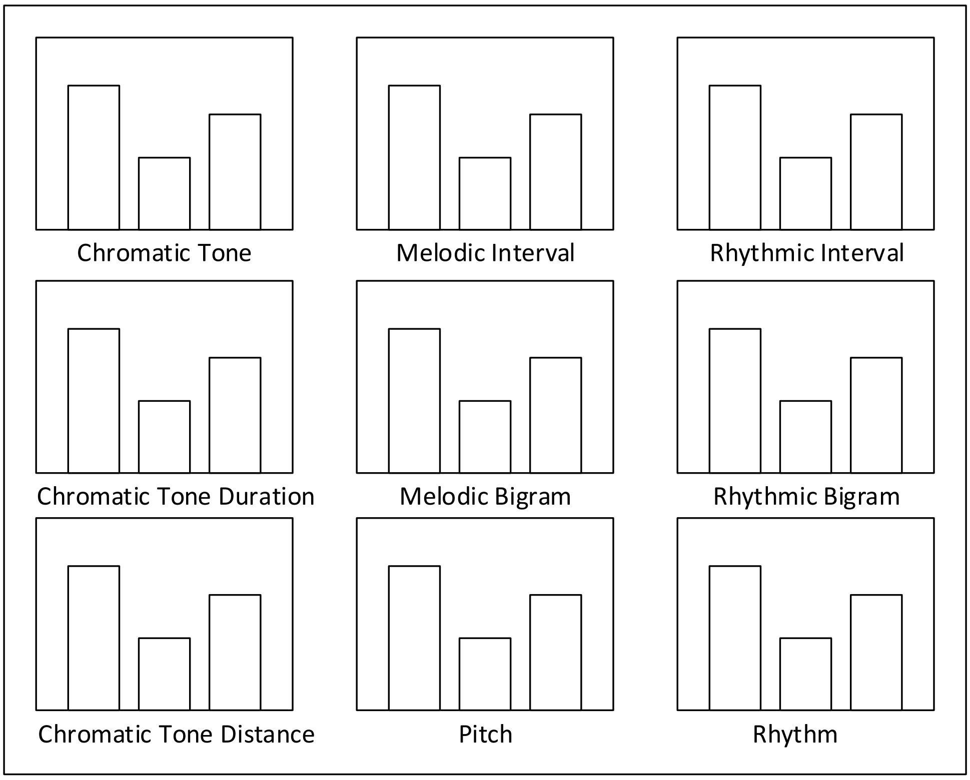

Figure 29: User interface for displaying metrics of a melody

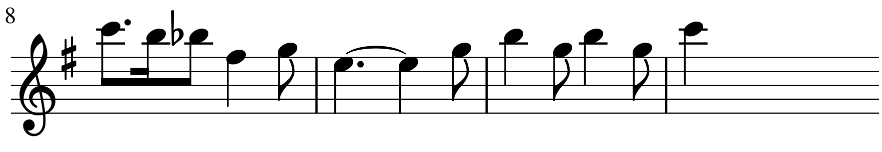

Figure 30: Visualization of notes

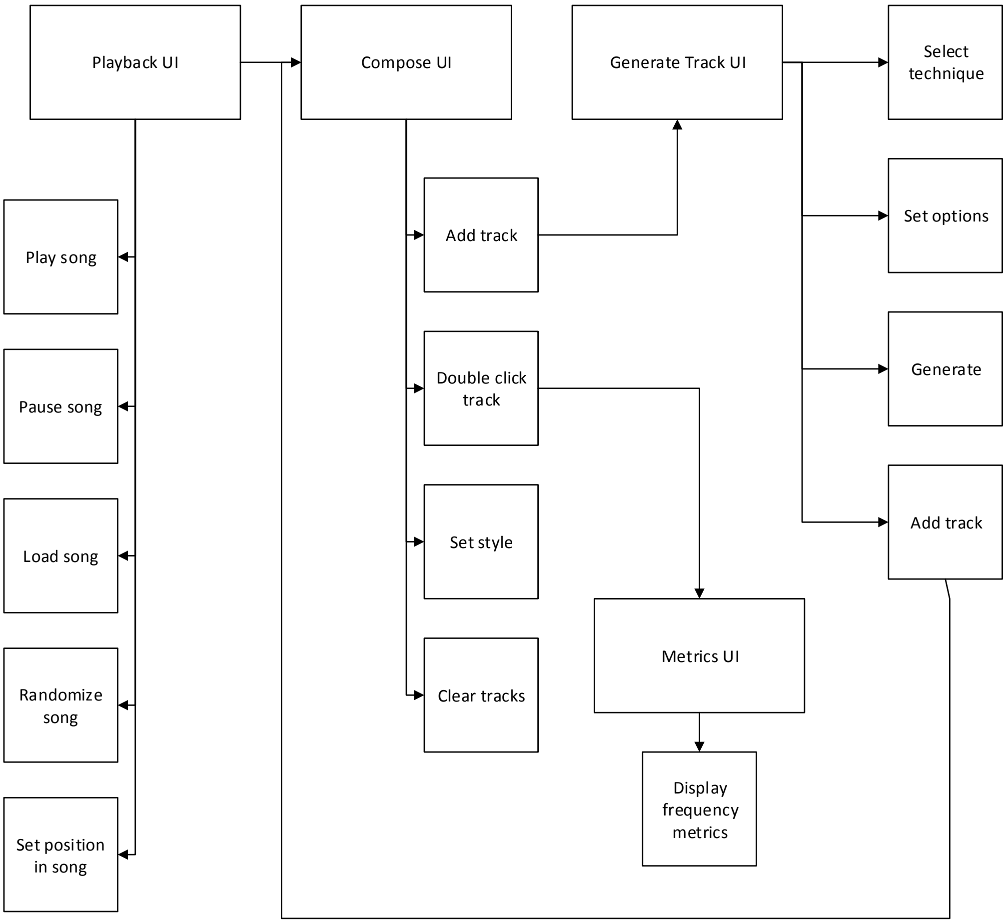

Figure 31: State diagram of the user interface

24 FINAL CONSIDERATIONS

Part VI

APPENDIX

A

ALGORITHMS

A.1 BACKPROPOGATION

For a feed forward network[1] with n_{in} inputs, n_{hidden} hidden units and n_{out} output units the back-progogation algorithm works as follows

1. Initialize network with random weights
2. Repeat until algorithm terminates
3. • $\forall(\vec{x}, \vec{t}) \in$ training examples do
   • a) Input instance $\vec{x}$ to network and compute $o_u \ \forall \ u \ \in$ network
     b) For each network output unit k calculate error term
     $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$
     c) For each hidden unit h calculate the error term δ_h
     $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$
     d) Update each network weight w_{ji}
     $w_{ji} \leftarrow w_{ji} + \eta\delta_j x_{ji}$

The complete derivation of the back-propogation algorithm for feed forward artificial neural networks can be found in Mitchell (2007) [42]

1 Other topologies exist with more complex methods for obtaining the weights

B PLANNING

Project breakdown and schedule

B.1 WORK BREAKDOWN STRUCTURE

Figure 32 indicates the work breakdown structure of the project. The front end design refers to the user interface and the interaction between the user and the application (IF 1.0).

The back end design refers to the logic of the application and this includes reading the audio files, generating new music and playing music.

Research will be carried out in order to determine the algorithms to be used for algorithmically generating music and to find a suitable framework for developing the software (considering audio playback and similar factors)

Since there are a variety of algorithms, prototyping will be employed to test which algorithms are feasible by means of a trade-off study.

The work breakdown structure allows focus on critical elements of the project and their relation to the whole. This allows us to focus on discrete tasks that are realistic and achievable and keeps the project on track.

B.2 SCHEDULE

In order to successfully complete the project the following tasks must be executed:

1. Researching machine learning algorithms for music composition
2. Prototyping the algorithms in order to find a feasible subset
3. Collecting a library of audio files to be used as input for machine learning algorithms
4. Developing the required back end components for the software
5. Developing a user interface and designing the interaction components
6. Testing the application, fixing bugs and adding features until the specifications are met

Table 2 indicates the estimated length of these activities and the estimated time of completion. Figure 33 is a visual representation of the schedule using bar charts.

B.3 BUDGET

Since the project is a software application no external components will be required. The application will run on a platform that is capable of storing audio files and playing back sound.

Some unplanned costs that might arise include:

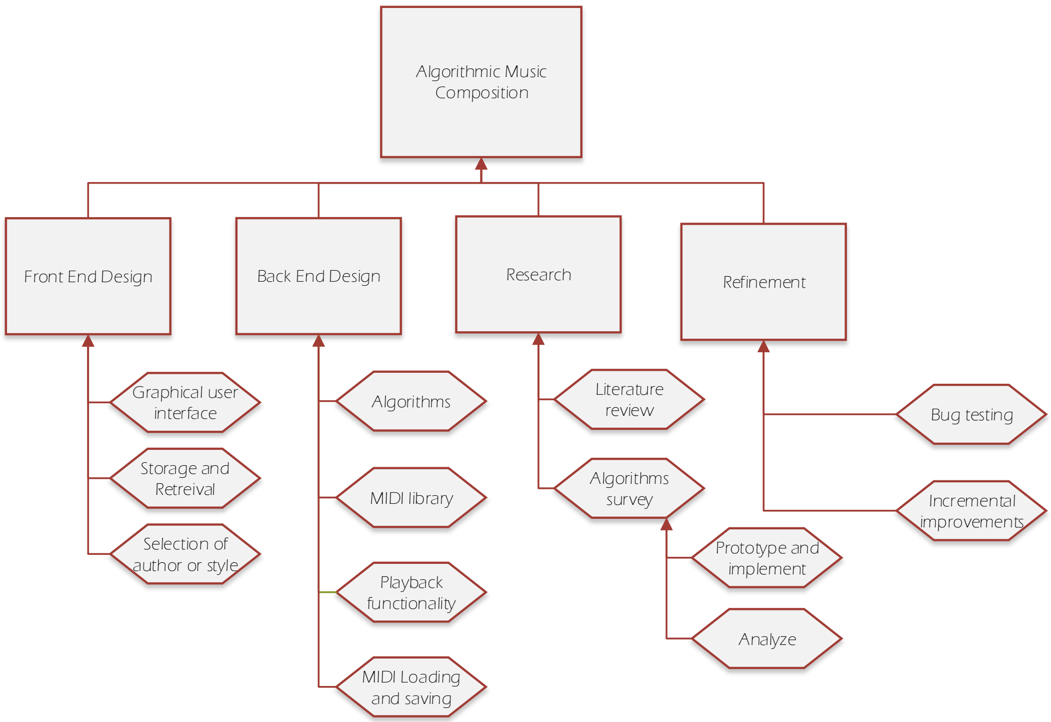

Figure 32: Figure of the work breakdown structure

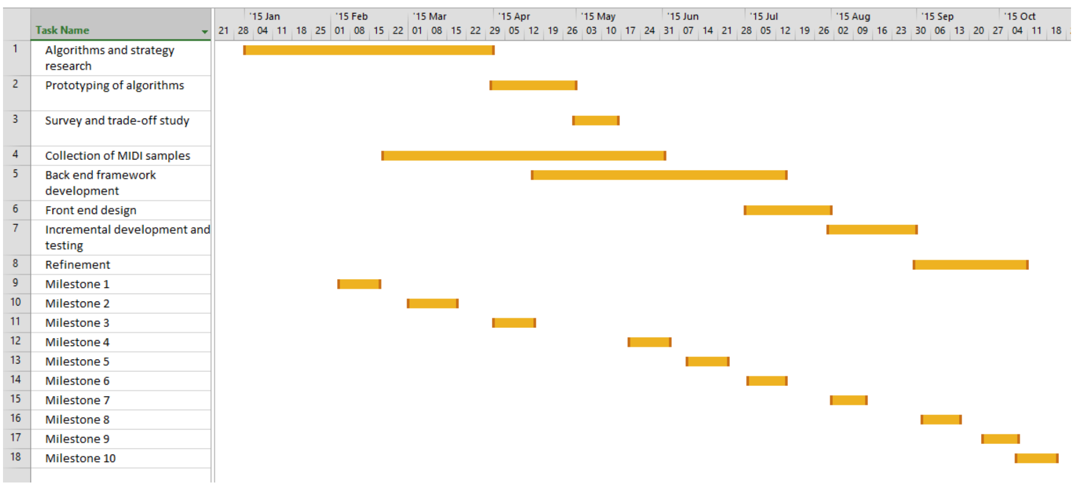

Figure 33: Gantt chart of project schedule

Table 2: Schedule for activities

Task Name	Duration	Start	Finish
Algorithms and strategy research	64 days	Thu 15/01/01	Tue 15/03/31
Prototyping of algorithms	23 days	Tue 15/03/31	Thu 15/04/30
Survey and trade-off study	12 days	Thu 15/04/30	Fri 15/05/15
Collection of MIDI samples	72 days	Fri 15/02/20	Mon 15/06/01
Back end framework development	66 days	Wed 15/04/15	Wed 15/07/15
Front end design	23 days	Wed 15/07/01	Fri 15/07/31
Incremental development and testing	22 days	Fri 15/07/31	Mon 15/08/31
Refinement	31 days	Mon 15/08/31	Sat 15/10/10
Milestone 1	11 days	Wed 15/02/04	Wed 15/02/18
Milestone 2	14 days	Sun 15/03/01	Wed 15/03/18
Milestone 3	11 days	Wed 15/04/01	Wed 15/04/15
Milestone 4	11 days	Wed 15/05/20	Wed 15/06/03
Milestone 5	11 days	Wed 15/06/10	Wed 15/06/24
Milestone 6	10 days	Thu 15/07/02	Wed 15/07/15
Milestone 7	10 days	Sat 15/08/01	Thu 15/08/13
Milestone 8	10 days	Thu 15/09/03	Wed 15/09/16
Milestone 9	9 days	Fri 15/09/25	Wed 15/10/07
Milestone 10	11 days	Wed 15/10/07	Wed 15/10/21

- Internet costs
- Acquisition cost of audio files
- Obtaining access to a study or article
- Audio equipment

BIBLIOGRAPHY

[1] D. Eck and J. Schmidhuber, "A first look at music composition using lstm recurrent neural networks," *Istituto Dalle Molle Di Studi Sull Intelligenza . . .* , 2002.

[2] P. M. Gibson and J. A. Byrne, "Neurogen, musical composition using genetic algorithms and cooperating neural networks," in *Artificial Neural Networks, 1991., Second International Conference on*, pp. 309–313, IET, 1991.

[3] M. C. Mozer, "Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing," *Connection Science*, vol. 6, no. 2-3, pp. 247–280, 1994.

[4] J. a. Biles, "GenJam: A genetic algorithm for generating jazz solos," *Proceedings of the International Computer Music Conference*, pp. 131–131, 1994.

[5] A. Sorensen, "Music Composition in Java."

[6] C.-C. Chen and R. Miikkulainen, "Creating melodies with evolving recurrent neural networks," *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, vol. 3, 2001.

[7] M. Alfonseca, M. Cebrian, and a. Ortega, "A Fitness Function for Computer-Generated Music using Genetic Algorithms," *WSEAS Transactions on Computers*, 2006.

[8] J. H. Jensen, *Evolutionary Music Composition*. PhD thesis, 2013.

[9] J. a. Biles, P. G. Anderson, and L. W. Loggi, "Neural Network Fitness Functions for a Musical IGA," *Architecture*, 1996.

[10] A. R. Burton and T. Vladimirova, "Genetic Algorithm Utilising Neural Network Fitness Evaluation for Musical Composition," in *In Proceedings of the 1997 International Conference on Artificial Neural Networks and Genetic Algorithms*, pp. 220–224, Springer-Verlag, 1997.

[11] B. Manaris, P. Roos, P. Machado, D. Krehbiel, L. Pellicoro, and J. Romero, "A corpus-based hybrid approach to music analysis and composition," *Proceedings of the National Conference on Artificial Intelligence*, vol. 22, p. 7, 2007.

[12] B. Manaris, P. Machado, C. Mccauley, J. Romero, and D. Krehbiel, "Developing Fitness Functions for Pleasant Music : Zipf's Law and Interactive Evolution Systems," *EvoWorkshops*, pp. 498–507, 2005.

[13] M. Unehara and T. Onisawa, "Construction of Music Composition System with Interactive Genetic Algorithm," *Evaluation*.

[14] L. Spector and A. Alpern, "Induction and Recapitulation of Deep Musical Structure," in *In Proceedings of the IJCAI-95 Workshop on Artificial Intelligence and Music*, pp. 41–48.

[15] B. Johanson and R. Poli, "{GP}-Music: An Interactive Genetic Programming System for Music Generation with Automated Fitness Raters," *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 181–186, 1998.

[16] K. McAlpine, E. Miranda, and S. Hoggar, "Making Music with Algorithms: A Case-Study System," *Computer Music Journal*, vol. 23, pp. 19–30, 1999.

[17] M. Farbood and B. Schoner, "Analysis and synthesis of Palestrina-style counterpoint using Markov chains," *Proceedings of the International Computer Music Conference*, pp. 471–474, 2001.

[18] M. Allan and C. K. I. Williams, "Harmonising Chorales by Probabilistic Inference," *Advances in Neural Information Processing Systems*, p. 2004, 2004.

[19] M. Ling and R. Sharp, "Melody classification using a similarity metric based on kolmogorov complexity," *Conference on Sound and Music Computing*, 2004.

[20] MIDI Manufacturers Assosciation, "MIDI 1.0 Detailed Specification," first edition of a recommendation, MIDI Manufacturers Assosciation, 1995.

[21] B. L. Jacob, "Composing with genetic algorithms," *In Proceedings of the 1995 International Computer Music Conference*, no. September, pp. 452–455, 1995.

[22] W. Langdon, *A Field Guide to Genetic Programing (Summary for Wyvern)*. No. March, 2008.

[23] M. Minsky and O. Laske, "a C Onversation With M Arvin M Insky," *AI Magazine*, vol. 13, no. 3, pp. 31–45, 1992.

[24] H. Jarvelainen, "Algorithmic musical composition," *Seminar on content creation, Telecommunications software and multimedia laboratory, Helsinki University of Technology*, p. 11, 2000.

[25] C. M. Bishop and Others, "Neural networks for pattern recognition," 1995.

[26] B. Laden and D. H. Keefe, "The representation of pitch in a neural net model of chord classification," *Computer Music Journal*, pp. 12–26, 1989.

[27] G. G. Wiggins, G. Papadopoulos, S. Phon-amnuaisuk, and A. Tuson, "Evolutionary methods for musical composition," *In International Journal of Computing Anticipatory Systems*, 1998.

[28] D. Matić, "A genetic algorithm for composing music," *Yugoslav Journal of Operations Research*, vol. 20, no. 1, pp. 157–177, 2010.

[29] M. Y. Lo, *Evolving Cellular Automata for Music Composition with Trainable Fitness Functions*. PhD thesis, 2012.

[30] A. Horner and D. E. Goldberg, "Genetic Algorithms and Computer-Assisted Music Composition," 1991.

[31] R. Cilibrasi and P. M. B. Vitanyi, "Clustering by compression," *Information Theory, IEEE Transactions on*, vol. 51, pp. 1523–1545, Apr. 2005.

[32] N. Tokui and H. Iba, "Music composition with interactive evolutionary computation," *Proceedings of the Third International Conference on Generative Art*, pp. 215–226, 2000.

[33] M. Alfonseca, M. Cebrián, and A. Ortega, "A simple genetic algorithm for music generation by means of algorithmic information theory," *2007 IEEE Congress on Evolutionary Computation, CEC 2007*, pp. 3035–3042, 2007.

[34] A. N. Kolmogorov, "On tables of random numbers," *Theoretical Computer Science*, vol. 207, no. 2, pp. 387–395, 1998.

[35] C. H. Bennett, P. Gacs, M. Li, P. M. B. Vitanyi, and W. H. Zurek, "Information distance," *Information Theory, IEEE Transactions on*, vol. 44, pp. 1407–1423, July 1998.

[36] M. Li, X. Chen, X. Li, B. Ma, and P. M. B. Vitanyi, "The similarity metric," *Information Theory, IEEE Transactions on*, vol. 50, pp. 3250–3264, Dec. 2004.

[37] Z. Cataltepe, A. Sonmez, and E. Adali, "Music Classification Using Kolmogorov Distance," *Representation in Music/Musical Representation Congress*, no. Bishop 57, 2005.

[38] M. Alfonseca, M. Cebrián, and A. Ortega, "Evolving computer-generated music by means of the normalized compression distance," in *WSEAS Transactions on Information Science and Applications*, vol. 2 of *SMO'05*, (Stevens Point, Wisconsin, USA), pp. 1367–1372, World Scientific and Engineering Academy and Society (WSEAS), 2005.

[39] C. Mckay, "Automatic Genre Classification of MIDI Recordings," *Technology*, vol. MA, no. June, pp. 1–150, 2004.

[40] G. K. Zipf, *Human behavior and the principle of least effort: an introduction to human ecology*. Addison-Wesley Press, 1949.

[41] G. A. Carpenter and S. Grossberg, *Adaptive resonance theory*. Springer, 2010.

[42] T. M. Mitchell, *Machine Learning*, vol. 4 of *McGraw-Hill Series in Computer Science*. McGraw-Hill, 1997.

DECLARATION

Put your declaration here.

Potchefstroom, October 12, 2015

Stefan Jacholke