CHALMERS  UNIVERSITY OF GOTHENBURG

Generation of music through genetic algorithms

Bachelor of Science Thesis in Computer Science and Engineering

VIKTOR ANDERLING
OLLE ANDREASSON
CHRISTOFFER OLSSON
SEAN PAVLOV
CHRISTIAN SVENSSON
JOHANNES WIKNER

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2014

Generation of music through genetic algorithms

Viktor Anderling
Olle Andreasson
Christoffer Olsson
Sean Pavlov
Christian Svensson
Johannes Wikner

Examiner: Jan Skansholm

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000
Department of Computer Science and Engineering
Göteborg, Sweden June 2014

Abstract

The focus of this bachelor thesis is to generate appealing music segments algorithmically. Since its creation, the art of music has constantly evolved, developing new genres and styles over time. Computers have long been recognized for their potential in discovering new music, but a computer has yet to produce a truly appealing piece of music.

This thesis employs an evolutionary approach, generating large amounts of musical segments and selecting the best ones. This selection is made by a group of programmed raters with different specializations. This method aims to emulate the process of natural selection.

While the generated results may not have been top hits in themselves, many interesting segments were created. The created music was diverse, original and could in many cases be considered to be appealing.

This project was able to produce decent results in short segments but there is definitely room for improvement. It is recommended to add more raters to make the rating process as precise as possible.

Sammanfattning

Syftet med detta kandidatarbete är att generera tilltalande musiksegment algoritmiskt. Alltsedan musikens uppkomst har denna varit under konstant utveckling, med nya genrer och stilar som resulat. Datorers potential inom skapandet av ny musik har länge varit känt, men en dator har ännu inte lyckats skapa ett verkligt tilltalande musikstycke.

Detta arbete använder sig av en evolutionär metod där stora mängder musikstycken genereras varpå de bästa väljs ut. Valet görs av en grupp programmerade betygsättare med olika specialiseringar. Målet med denna metod är att efterlikna det naturliga urvalet.

Trots att de genererade resultaten inte direkt var listettor har många intressanta musikstycken skapats. Den skapade musiken var divers, originell och kunde i flera fall anses tilltalande.

Detta arbete lyckades producera godkända resultat i korta stycken men det finns definitivt utrymme för förbättring. Det är starkt rekommenderat att utöka antalet betygsättare för att göra betygsättningen mer precis.

Glossary

Genetic algorithm Algorithm designed to emulate the process of evolution through the use of natural selection.

Individual A single musical segment used in the genetic algorithm.

Generation A set of individuals, created once during every iteration of the genetic algorithm.

Parent An individual of a specific generation that has contributed to the creation of the next generation.

Child An individual of a specific generation generated by a parent.

Sibling An individual's relation to another which is in the same generation.

Crossover The process of taking a set of parents and combining them to generate a set of children.

Mutation The process of changing specific properties of an individual, based on preset rules.

Rater Grades the musical qualities of an individual. The rater consists of a set of sub-raters.

Sub-rater Grades a very specific characteristic of an individual.

Target rating A value between 0 and 1 that serves as a target value for the rater.

Selection The process which decides which individuals will be chosen as parents.

Initialization The method for generating the first generation.

MIDI A standardized data format for representing sheet music.

Seed MIDI data used for initialization through crossover or Markov chains.

Markov chains Probabilistic initialization method using one or more seeds.

Track A musical segment which plays in parallel with other tracks in the same song. Contains its own instrument and channel.

Track tag A type of metadata attached to a track, describing its role in the musical piece.

Contents

1 Introduction

The art of music has been performed for thousands of years, with the oldest findings of instruments dating as far back as 6600BC. The evolution of music is a relevant subject as it is not only extensive but also extremely diverse. This becomes strikingly clear in the statement of musicologist Jean-Jacques Nattiez:

> The border between music and noise is always culturally defined—which implies that, even within a single society, this border does not always pass through the same place; in short, there is rarely a consensus [...] By all accounts there is no *single* and *intercultural* universal concept defining what music might be.[1, pp.48,55]

The undefined nature of music in combination with its diversity makes musical analysis a very complex subject. To simplify the analysis, it becomes necessary to focus upon one single culture of music. Within this single culture, a myriad of somewhat defined styles and genres exist. Many of these have evolved from a merge of multiple predecessors, one example being how the western culture's genre of Rock n roll evolved from a mix of Jazz and Country. This kind of evolution, where differing music merges together to form something new is at the heart of *algorithmic music*. Through simulating evolution, new music can be created algorithmically with an almost infinite number of possible outcomes.

1.1 Purpose

The main purpose of the project is to create an application with the ability to generate original and appealing music segments. As appealing is a very subjective definition, the user should be given the ability define it. By using music which the user considers appealing the application should be able to create new music with similar characteristics.

1.2 Method

The project applies an algorithmic approach to generate music. First the user selects a method of initialization, including a choice of seed if necessary. A *genetic algorithm*, emulating the process of natural selection, employs the chosen method of initialization to create the parents of the first generation. By mixing the characteristics of the parents, a new generation will be created and rated. The best individuals of the generation will be chosen to repeat the cycle from the mixing stage. After a chosen number of iterations, an output which should resemble the characteristics that the raters desire is created.

To decide which individuals are the best, the raters will first rate a set of musical segments that are considered to be good. The generated individuals that are rated closest to the values of the good segments are chosen to become the parents of the next generation.

1.3 Scope and limitations

To narrow the scope to a reasonable level, the project focuses heavily on generating music and avoids analyzing it as much as technically possible.

The music format needs to have a well defined structure whilst not being too complex, a niche the standard MIDI format fills nicely. In difference to other formats, MIDI is divided into well defined tracks where pitch and timing values for each note can be extracted easily. A thorough description of the format can be found in section 3.1.

2 Technical background

The area of algorithmic composition can be described as "the process of using some formal process to make music with minimal human intervention"[2, p.1] and is commonly performed using computational methods. However, the idea of using formal instructions and processes to create music dates much further back than the era of computers.

2.1 Before the computer

The ancient Greeks constructed their musical systems upon formalisms, which consisted of mathematical properties derived from nature and theorietical applications of numbers. Pythagoras believed that there was a direct relation between the natural laws and musical harmony:

> The word music had a much wider meaning to the Greeks than it has to us. In the teachings of Pythagoras and his followers, music was inseparable from numbers, which were thought to be the key to the whole spiritual and physical universe. So the system of musical sounds and rhythms, being ordered by numbers exemplified the harmony of the cosmos and corresponded to it.[3, p.843]

With their numerical representation of music, the Greeks set the mathematical baseline for musical composition and further experimentation would follow throughout the centuries.

The famous composer Wolfgang Amadeus Mozart created his own simple automated composition technique with his Musikalisches Wurfenspiel, a musical game based on dice rolls. The game "involved assembling a number of small musical fragments, and combining them by chance, piecing together a new piece from randomly chosen parts.".[2, p.2]

2.2 Computational pioneers

The introduction of computers opened up a new world of possibilities within algorithmic composition. The potential of computed music was recognized as early as the 19th century by Ada Lovelace, the inventor of the calculating engine.

> Supposing, for instance, that the fundamental relations of pitched sound in the signs of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.[4, p.4]

The first computer-generated composition was created by Lejaren Hiller and Leonard Isaacson at the University of Illinois in 1957. It was named the Illiac Suite after the Illiac High-speed digital computer, which was used to create the composition. The composition was completed in three steps: First, raw musical material was created with the computer, secondly the materials were modified according to a variation of functions and finally rules were applied to select the best results. Such computational compositions which follow the methodology of creating, modifying and rating were to become known as *rule-based* compositions

Another pioneering method of computational algorithmic composition which empolyed a different methodology was created by Iannis Xenakis. Iannis made use of the computer's computational speed to calculate multiple different probability theories which he then applied to create his compositions.[5]

More specifically, the program would deduce a score from a "list of note densities and probabilistic target ratings supplied by the programmer, leaving specific decisions to a random number generator"[2, p.3]. The method of creating compositions purely based on probability and random numbers became known as *stochastic composition*.

2.3 Similar projects and studies

Multiple projects utilizing genetic algorithms for musical composition have been performed, where most of them share one common trait; The individuals of each generation are rated using a user input.[6] This approach naturally generates pleasing results in the opinion of the user, but rating every individual becomes extremely tedious work.

Another common approach regarding the fitness function is to implement some kind of neural network combined with user input to teach automated raters.[7] While this approach has yielded good results, the amount of data necessary to teach the neural network and its automated raters to detect good music, also involves a large amount of manual work.

2.4 Relevance to project

As mentioned in the introduction, this project applies a genetic algorithm containing multiple steps. All steps use a combination of the stochastic and rule-based composition. Each step has specific rules by which it abides but they are all affected by random number generators which have the chance to drastically change the outcome. By combining both of these pioneering computational methods with natural selection, the results can be ensured to be diverse whilst upholding quality.

This project differs from the ones mentioned in the in the section above by implementing a new way of user input. Instead of having the user rate each individual of a generation, the application recieves a group of songs that the user considers to be good. The application then aims to create songs that are alike the user input songs. This approach assures that the results are pleasing to the user while reducing the amount of manual work.

3 Technical Model

To fully understand the methodology of this project it is necessary to delve into the technical model. This section covers the general techniques, data structures and datamodels used in this project.

3.1 Musical data structures

The standard MIDI format is a way to serialize musical representation electronically by associating pitches with different time events.[8] The timed pitches are put into tracks which represent different musical instruments. Tracks are in turn part of the entire score that contains various metadata such as the tempo and author of the musical piece.

To work efficiently with the standard MIDI format, the programming library *jMusic* has been selected as an interface for interaction and manipulation of the data. jMusic employs a specific data structure for representing the music, visualized in figure 1 below:

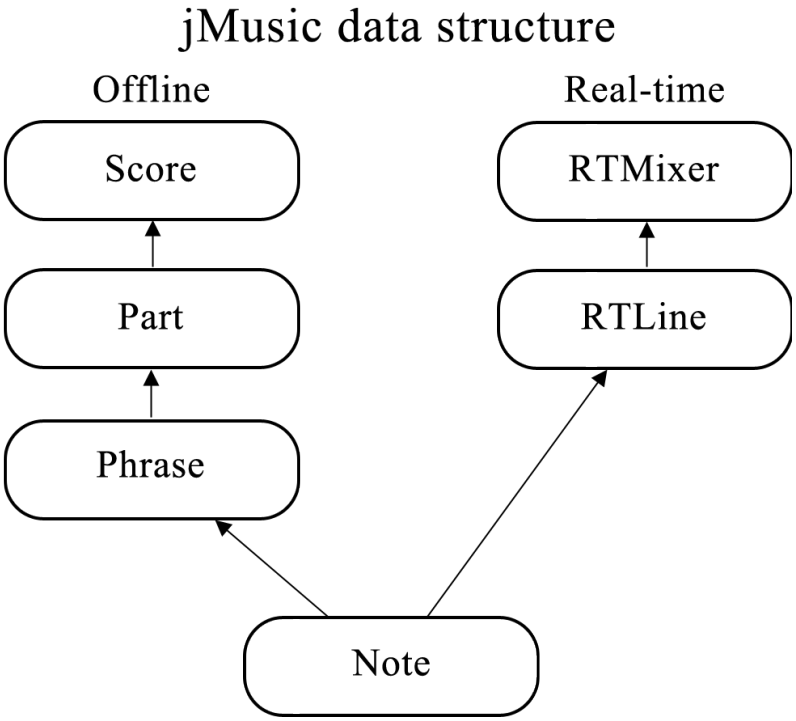

Figure 1: Data structure of jMusic. A score consists of multiple parts which consists of multiple phrases which consists of multiple notes.[9]

3.2 Data model

The jMusic score and part implementations are closely associated with the song and track implementations respectively, which have been specifically constructed for this application's purpose. These have additional properties and functionality that simplifies common operations. Figure 2 visualizes the dependence of the package "evoMusic.model.song", which holds the song and track representations.

Song representations are created from the MIDI songs inserted into the application and *tags* are assigned to the different tracks by the user. The tag determines what function a specific track has in the song. The currently used tags are: melody, chords, rhythm, beat, drums, baseline and none. The none tag is useful for excluding unwanted tracks as they will be excluded from the generation process. After a musical segment has been created it is added to a database, allowing it to be easily recreated between the runs of the application.

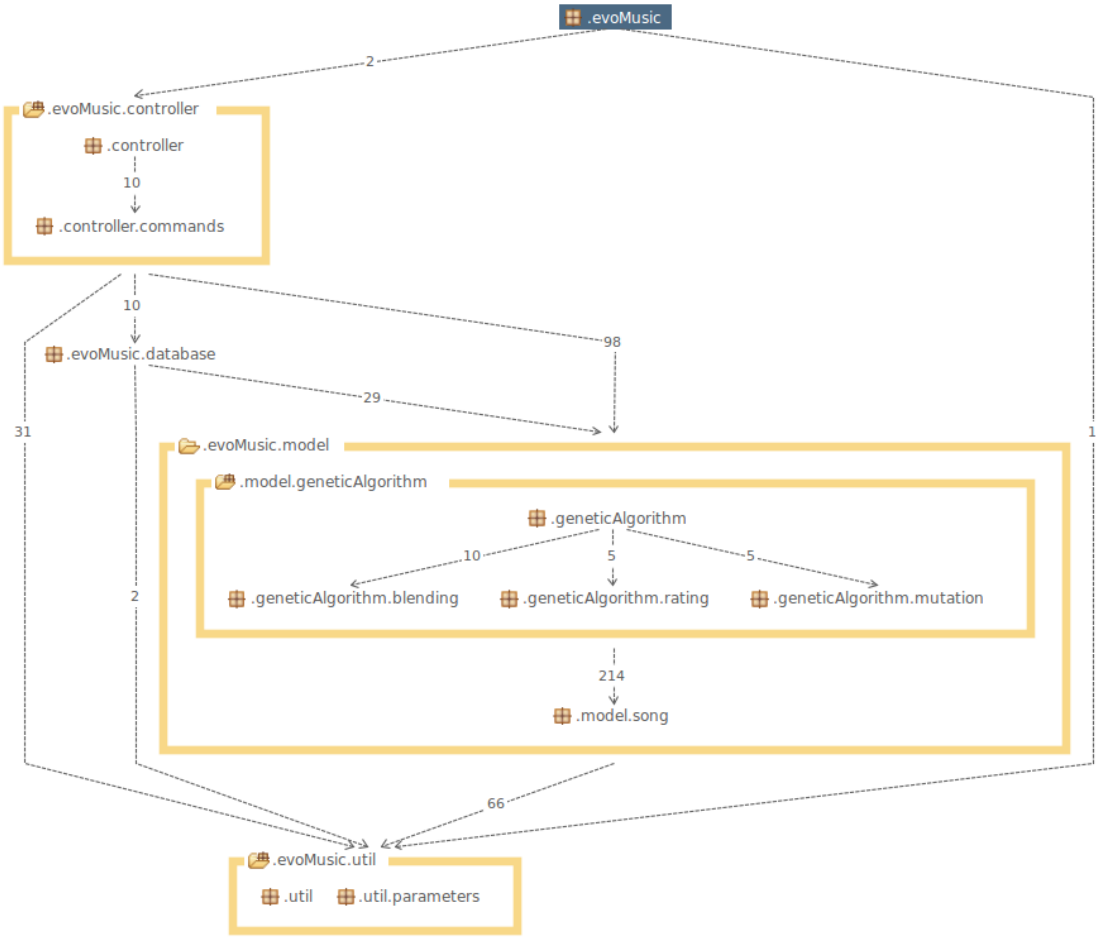

Figure 2: The composition of the data model of the application. The package "evoMusic.model.song" is a central part of the data model as it holds the representations for Songs and Tracks.

4 The genetic algorithm

The genetic algorithm lies at the heart of the project and aims to emulate the process of natural selection. The algorithm contains five major steps that define its functionality: Initialization, crossover, mutation, rating and selection; All of which will be explained in greater detail in their own subsections.

The algorithm creates the first generation with the chosen initialization method whereupon the individuals within it are rated. A set of user-defined size containing individuals with the best rating is created. These individuals are defined as *elites*. After finding the elites, a set of individuals from the first generation are chosen through selection to be sent through crossover and mutation. The crossover and mutation process creates the next generation, which is subsequently rated. The best individuals of the new generation are then compared with the elites, if they are superior they become the new elites, otherwise the old ones are kept. The set of elites are then sent through the selection phase, completing the iterative cycle of the algorithm. The algorithm will continue to iterate until a user-defined amount of generations have been created, whereupon it will output the best individual of that final generation. Figure 3 below visualizes the process:

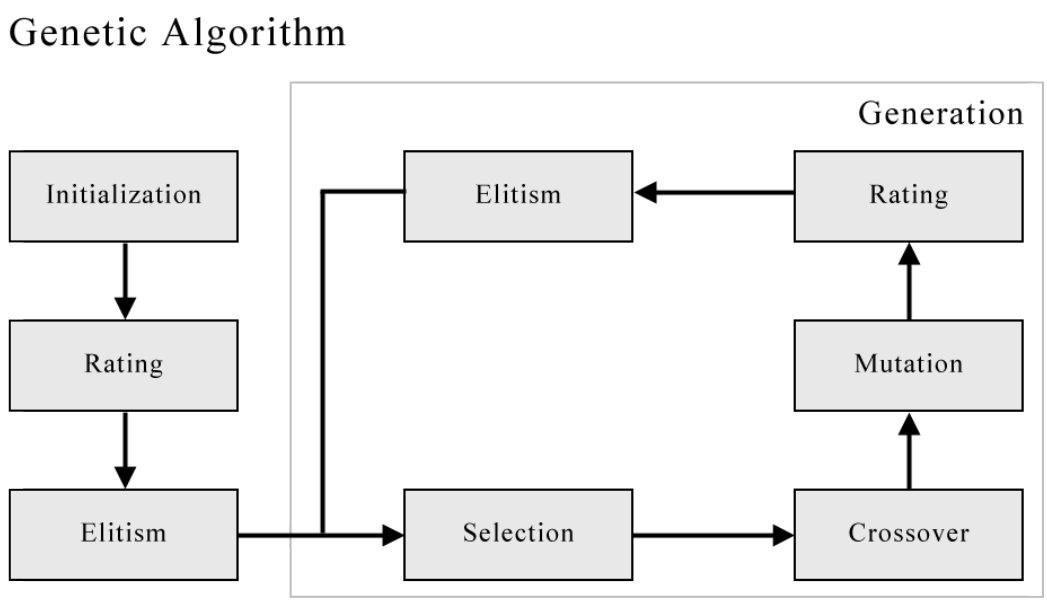

Figure 3: Flow chart depicting the functionality of the genetic algorithm

5 Initialization

Initialization is the first step of the genetic algorithm and this project employs three possible methods. The purpose of the initialization step is to ensure that the genetic algorithm has a pool of individuals to use during the first iteration. Two of the three methods require a seed in the form of one or more musical segments, while the third one creates individuals using random generation.

5.1 Crossover initialization

The crossover initialization takes the seed and performs a standard crossover on them. A more detailed description of crossover can be seen in section 6. The resulting individuals become the first generation. The first generation tends to keep a lot of the stucture and features of the seed, resulting in a higher initial rating. One drawback of this method is that the results are less diverse than the randomly initialized ones.

5.2 Random initialization

Random initialization is the most commonly used method for genetic algorithms. Each individual in the first generation is generated randomly, resulting in notes with random pitch and length. The randomization can be slightly controlled to result in individuals that are likely to be more appealing. These configurations include methods such as distributing the pitches more towards the center, so that the individuals contain less pitches with extreme values. While the populations generated using random initialization are extremely diverse, they tend to lack in musical quality.

5.3 Markov chains

Named after the Russian mathematician Andrey Markov, a Markov chain in its basic form is a stochastic memoryless system. The chain consists of states where for each state there is a specific probability of either traveling to another state or staying in the current one. These probabilities are collected in a matrix which represents the system as a whole and only depend on the current state. For a formal mathematical definition of the basic markov chains, see [10, pp.1-5]. Figure 4 below represents a 3-state markov chain with probability matrix P:

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \end{pmatrix}$$

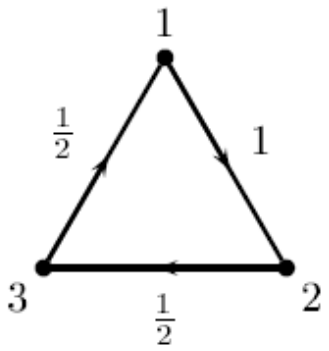

Figure 4: Basic 3-state markov chain with probability matrix P.[10, p.2]

5.3.1 Application within algorithmic composition

When applying Markov chains to algorithmic composition of music, the basic functionality must be changed. Instead of the system being memoryless the probabilities of traveling between states should depend not only on the current state, but also on a number of previous states. The amount of previous states that are considered is known as the *order* of the Markov chain.[11]

To apply the system of higher order Markov chains, the states are set to represent sequences of pitch values. This results in a matrix containing the probability of notes following each other. By increasing the order of the chain, recurring patterns in a song can be discovered.

5.3.2 Application within this project

There is a major issue with the standard application of Markov chains where states represent pitch values. When merging musical pieces with the same key signatures, the standard implementation using pitch values as states is not a problem. However, if the only difference between two melodies is their transposition, a problem arises. A person would not be able to hear the difference between them if listened to separately, while the basic Markov chain would categorize them as completely different melodies.

To combat this effect, the Markov chain within this project analyzes the occurrence of intervals between notes instead of the pitch values, as seen in figure 5. This means that any two pitch changes of the same magnitude will be represented the same way in the Markov matrix, regardless of their transposition.

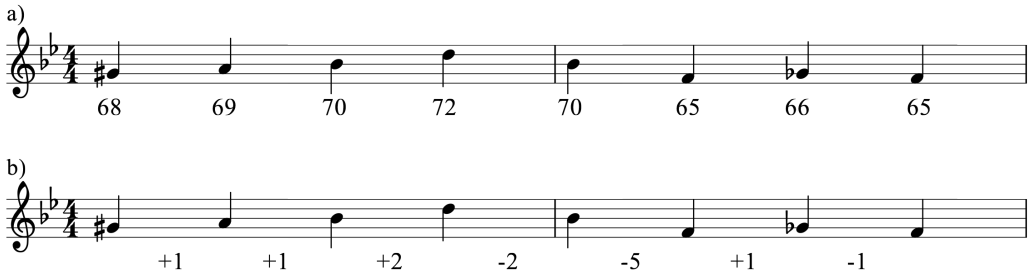

Figure 5: a) shows a melody where the pitch of each note represents a state. b) shows the same melody where the difference in pitch between each note and its next neighbor represents a state.

This project implements multiple Markov chains, one for each note property deemed relevant. These properties are: the relative interval to the next note, the duration that the note is played, the loudness of the note and finally the difference in start time between the current and previous note. The separation of the note properties allows the multiple Markov chains to work by employing a layer-by-layer approach, considering only one note property at a time.

As explained in section 5.3, Markov chains are efficient state-machines employing matrices of probabilities for travelling between states. The method employed within the project results in multiple matrices representing separate note properties. The first matrix contains probabilities of going from a current state of intervals to a successor state of intervals. The following matrices contain probabilities of going from a current state of

intervals to a state containing the values of a certain note property. This method results in all note properties being dependent on the interval structure of the song. For a formal mathematical definition of the interval state transitions, see appendix A.

The method handles notes playing at the same time by representing them as a series of notes with a start time difference of zero. Additionally, each property is mapped to its corresponding track in the song, which allows them to be separated with the same instruments and channels that their seeds had. A final detail is that the Markov chains focuse heavily on pitch values rather than properties like rhythm. The reason for this being that the pitches of a melody tend to be more recognizable than the rhythm. However, it would definitely be possible to implement the Markov chain to focus on other properties and test the impact upon the result. While theoretically possible, such an implementation has not been made due to time constraints.

6 Crossover

The purpose of the crossover is to create children by recombining parts from a set of parents. The crossover plays a vital part in the genetic algorithm, being more powerful and common than random mutations. However, excluding recombinations may have remarkably good impact on the results if the population is very small.[12]

There are multiple methods for executing a crossover, varying the amount of parents or crossover points.

6.1 Application within this project

This project employs a crossover that can handle an arbitrary number of parents of arbitrary length. The parents are adjusted to have matching structure concerning lengths and tracks. The amount of children generated by the crossover will be the same as the number of input parents and will furthermore have the same lengths and number of tracks.

Initially, the adjustment of the parents is accomplished by cutting off the end of all longer parents to match the shortest one. Next, the different track tags for each parent is counted. The parent with the fewest number of tracks with a certain tag, determines the amount tracks with that tag that the other parents should have. If any parent holds more tracks than this number, the tracks are merged until the desired number is reached. This normalization only occurs once as the structure created will be maintained throughout the entire generation process.

Once it's confirmed that the parent have the same structure, their tracks are split into segments at regular intervals of rhythmic beats. Secondly, the segments are distributed equally into children consisting of tracks with the same tags and lengths as the parents. The result of a crossover of a track visualized by figure 6 below:

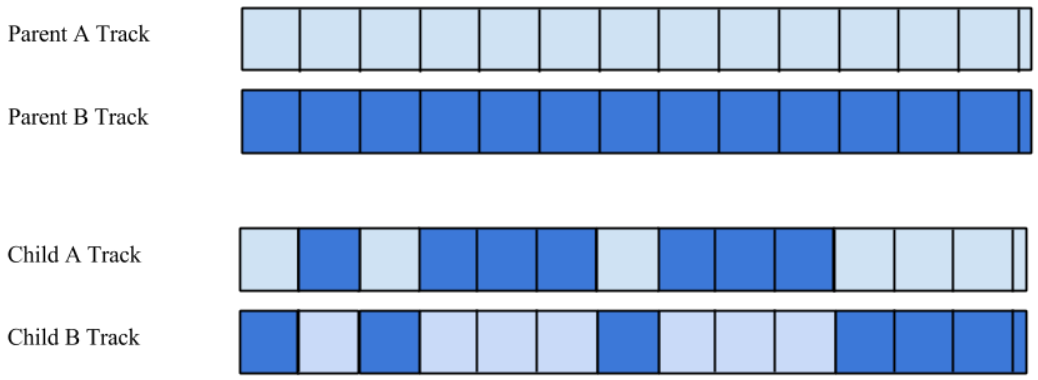

Figure 6: Resulting children of a crossover between two parents

7 Mutation

This process aims to emulate an accelerated version of the natural occurrence of mutations. The mutations will induce changes in the structure of the music which may be either positive or negative. The mutation is carried out by multiple different sub-mutators, explained in detail below.

7.1 Sub-mutators

Upon being mutated, the individual will iterate through each sub-mutator.Each of these sub-mutators contain a local mutation probability value P, preset by the user. P represents the probability of a note in the individual being mutated and its value is affected by a changing multiplier M.

The user specifies the rate of change of the multiplier by defining a multiplier step M_{step}. The multiplier has an initial value of 1 and the multiplier step is subtracted in every iteration. An example of the process is detailed by equation 1 below:

$$\begin{aligned} P &= 0.9 \\ M_{step} &= 0.05 \\ P_{gen1} &= M * 0.9 = 0.9 \\ P_{gen2} &= P_{gen1} - M_{step} = 0.85 \\ P_{gen3} &= P_{gen2} - M_{step} = 0.80 \\ &\vdots \end{aligned} \tag{1}$$

The multiplier is necessary as it will make the algorithm go from favoring exploration to exploitation, beginning with heavy mutation while steadily lessening with each iteration. The efficiency of this methodology has been proven through the particle swarm optimization algorithm.[13] The last user-set value is the multiplier minimum which prevents the mutation probability from ever becoming zero.

7.1.1 Note Pitch

The purpose of the RandomNotePitch sub-mutator is to mutate note pitches. For each note, the sub-mutator compares the local mutation probability against a random number between 0 and 1. If the local probability is equal or larger than the random number the note will be mutated. Figure 7 below visualizes a possible result:

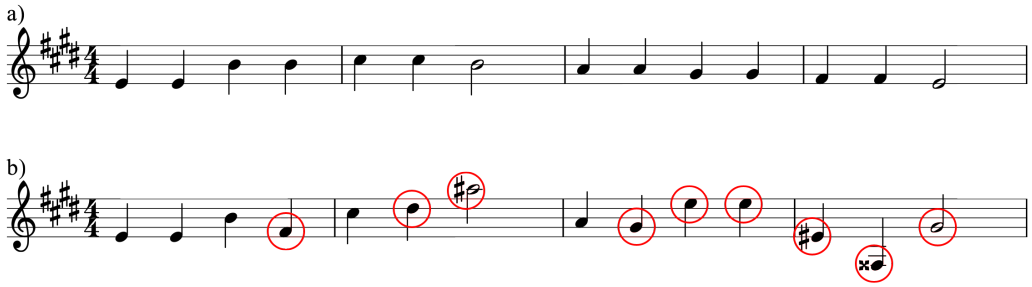

Figure 7: a) represents the core melody of *Twinkle Twinkle Little Star*. b) represents the same melody with encircled notes having been mutated by RandomNotePitch with a local probability of 0.5.

The user must also define how powerful the mutation will be by setting a mutation range parameter. The parameter defines the maximum possible change of the pitch value when a note is subjected to mutation. The equation is designed so that it is most likely that the mutated pitch will be within two half-tones of the original. Such a distribution is desirable as neighboring notes generally tend to have pitch values close to each other. Without such a distribution the result risks only consisting of pitches which differ greatly, resulting in a very unappealing sound. Equation 2 below details the functionality further:

$$
\begin{aligned}
\text{Range parameter, r} &: 0 \leq r \leq 127 \\
\text{Current pitch value, c} &: 0 \leq r \leq 127 \\
\text{Number of half-tone steps, s} &: d(c) \leq s \leq u(c) \\
\text{Max positive pitch change } u(c) &= \begin{cases} r & : c + r \leq 127 \\ 127 - c & : c + r > 127 \end{cases} \\
\text{Max negative pitch change } d(c) &= \begin{cases} -r & : c - r \geq 0 \\ -c & : c - r < 0 \end{cases} \\
\text{Distribution function } f(i) &= \begin{cases} 0 & : i = 0 \\ r - (i - ((i-1) \bmod 2) & : i > 0 \\ r - (|i| - ((|i|-1) \bmod 2) & : i < 0 \end{cases} \\
\text{Probability of taking } s \text{ half-steps } P(s) &= f(s) \Big/ \sum_{i=d(c)}^{u(c)} f(i)
\end{aligned}
\tag{2}
$$

7.1.2 Start time

The purpose of the Rhythmvalue sub-mutator is to mutate note start-times. For each note, the sub-mutator compares the local mutation probability against a random number between 0 and 1. If the local probability is equal or larger than the random number the note will be mutated. The user must also define how powerful the mutation will be by setting two mutational parameters. The first parameter decides how many measures long a mutation step should be, while the second determines the maximum number of steps possible. Figure 8 below visualizes a possible result:

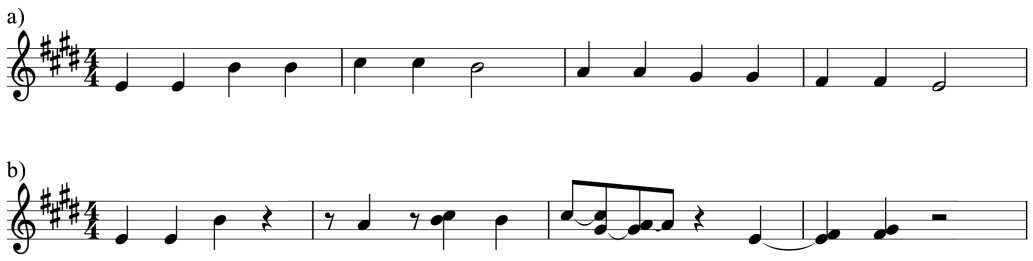

Figure 8: a) represents an unmutated music segment. b) represents the resulting music segment after being mutated by the start time mutator.

To produce an appealing output it is necessary that the parameters are set so that the mutated notes are not moved further than one measure. This is necessary to prevent a note from being put completely out of its context. Moving multiple notes very far in time has proven to be too drastic of a mutation.

7.1.3 Simplify

The purpose of the Simplify sub-mutator is to flatten melodies. For each note, the sub-mutator compares the local mutation probability against a random number between 0 and 1. If the local probability is equal or larger than the random number the note will be mutated. The pitch value of the mutated note will be changed to equal the value of the previous one. A possible result of mutating with Simplify is shown in figure 9 below:

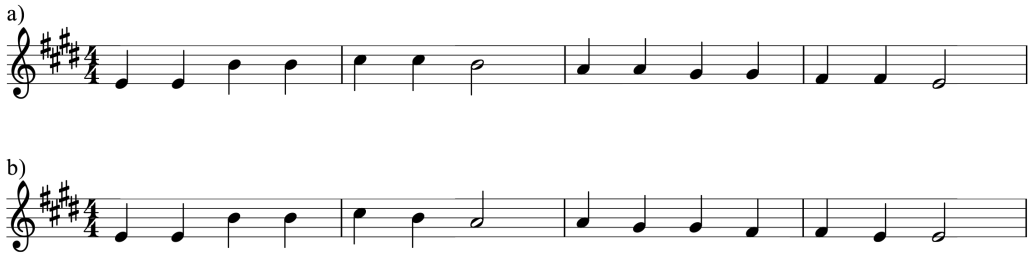

Figure 9: a) represents an unmutated music segment. b) represents the resulting music segment after being mutated by Simplify.

7.1.4 Swap segments

The purpose of the SwapSegment sub-mutator is to swap the position of two different, randomly selected segments of the same length in a track. First the sub-mutator selects the length and different start times for the two segments to be swapped. These segment values are randomly selected within calculated intervals, described by equation (3) below:

$$
\begin{aligned}
L &\sim \mathrm{U}([0.25, Tracklength/2]) \\
S1 &\sim \mathrm{U}([0, Tracklength - (L * 2)]) \\
S2 &= \begin{cases} S1 + L & : Tracklength - S1 = 2L \\ \sim \mathrm{U}([S1 + L, Tracklength - L]) & : Tracklength - S1 \neq 2L \end{cases}
\end{aligned}
\tag{3}
$$

L represents the length of the segments, $S1$ the start time for segment one and $S2$ the start time for segment two. $\sim \mathrm{U}[x, y]$ represents the uniform distribution of x and y. This method ensures both segments fit within the length of the track. When using random selection for the segment values, the length could range anywhere between one fourth of a beat to half the length of the music segment. The placement of the segments can range anywhere from within the first beat to the very end of the track. Figure 10 below visualizes a possible result:

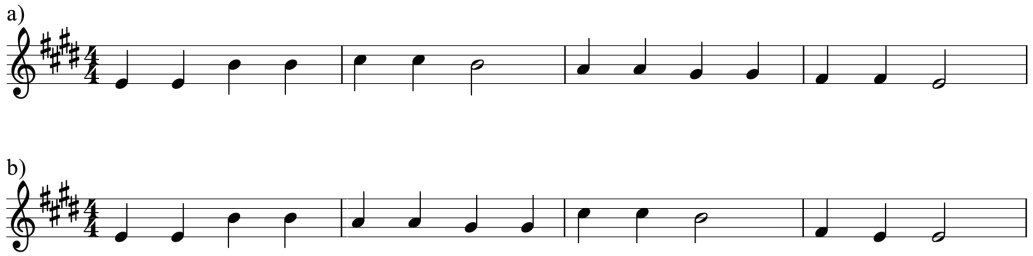

Figure 10: a) represents the unmutated music segment b) represents the resulting music segment where the second and third segment have been mutated by SwapSegment.

8 Rating

Rating represents the specific implementation of the fitness function in this project. The rating is performed by several sub-raters and is an integral part of the genetic algorithm as it has a major impact on the success of the entire process. The individuals obtained by the previous steps must be rated to decide which parts are fit to be used as parents in the creation of the next generation. If the rater does not function properly the results might not represent the desired properties that the subraters define, or in the worst case cause the overall fitness of the songs to worsen for each generation, nullifying the entire evolutionary purpose of the algorithm.

8.1 Application within this project

To be able to critically analyze and rate the musical segments created, the rating is performed by multiple sub-raters. These sub-raters all return a result varying from 0 to 1 representing how well the musical segments match their characteristics.

As stated in the introduction, the purpose of this project is to generate appealing music segments. To assure that the sub-raters work towards this purpose they are given a target rating, which is calculated by making each sub-rater rate a set of songs stored in the database. This target rating acts as a goal for the genetic algorithm to strive for, if the user wishes to create a good rock song, the sub-raters should first be set to rate a group of rock songs that the user finds appealing. When a sub-rater has rated all of the songs, its mean rating value will be calculated. The value will act as the target rating and is calculated using equation 4 below:

$$\begin{aligned} D &= \text{A set of songs from the database} \\ \text{Target rating} &= \sum^{D} rate(s)|_{s \in D} \Big/ |D| \end{aligned} \tag{4}$$

By comparing the target rating to the rating recieved from the sub-rater, the quality Q can be calculated. Equation 5 below details this process:

$$\begin{aligned} i &= \text{an individual} \\ Q &= abs(\text{Target rating} - rate(i)) \end{aligned} \tag{5}$$

These values give an accurate representation of what sub-rating values a song should aquire to be appealing to the user. The best individuals will be the ones which match as many subraters as close to the respective target ratings as possible. The sub-raters can be visualized as separate channels in an equalizer, where each sub-rater can be adjusted to produce the results the user desires. If a characteristic is not representative, the rating value should not have the same influence as that of a characteristic that is. To assure that the target rating represents the input pieces, the influence I takes the spread into account by utilizing equation 6 below:

$$I = 2 * (0.5 - min(\text{Target rating} - \text{Min rating}, \text{Max Rating} - \text{Target rating}) \tag{6}$$

The final rating for the song is computed using the concepts above. All sub-ratings are summed together and normalized to produce a final rating R between 0 and 1. Equation 7 below details the final rating for n sub-raters:

$$R = \sum_{k=1}^{n} Q_k * I_k \Big/ \sum_{k=1}^{n} I_k \tag{7}$$

8.2 Basic sub-raters

Listed within this section are all of the basic sub-raters implemented so far. Basic sub-raters all have the common purpose of identifying simple musical features. When coupled together, they become a tool for defining the musical structure of the segments they rate.

8.2.1 Neighboring Pitch Range

The purpose of this sub-rater is to detect notes with a pitch value at least two octaves apart compared to the previous note. To make music more pleasant, it is favorable to have notes with pitch values relatively close to their neighbors. With this in mind, the rating will be higher if less so called crazy notes are detected. Where a crazy note is one with a pitch value at least two octaves apart from the previous note. The rating is calculated by equation (8) below:

$$R = \frac{\text{Number of crazy notes}}{\text{Total number of notes}} \tag{8}$$

8.2.2 Direction Of Melody

The purpose of this sub-rater is to determine the direction of a melody. The direction is defined by an upward or downward trend for the note pitches and is considered to be a basic structural characteristic. The sub-rater iterates through every note in a track, counts the number of times it finds a note whose pitch value is higher than the one before it, and calculates the rating using equation (9):

$$R = \frac{\text{Number of pitches higher than the one before it}}{\text{Total number of notes}} \tag{9}$$

If the rating is above 0.5 the melody has an upward trend , while a value of less indicates a downward one.

8.2.3 Direction Stability Of Melody

The purpose of this sub-rater is to determine the amount of changes in pitch direction within a track. The sub-rater iterates through every note in a track, counts the number of times the pitch value changes direction, and calculates the rating using equation (10):

$$R = \frac{\text{Number of pitch direction changes}}{\text{Total number of notes}} \tag{10}$$

8.2.4 Variety Of Note Density

The purpose of this sub-rater is to analyze the note density by determining the amount of notes that share the same start time. The sub-rater iterates through every note in a track, analyzes how the density varies in every beat, determines the smallest and average density, and calculates the rating using equation 11 below:

$$R = \frac{\text{Least number of notes in a beat}}{\text{Average number of notes in beats}} \tag{11}$$

8.2.5 Syncopation Notes In Melody

The purpose of this sub-rater is to analyze the syncopation, defined in musical theory as "the displacement of the normal musical accent from a strong beat to a weak one.".[14] The sub-rater defines a syncopation note as one whose duration sustains across to the next beat. The sub-rater iterates through every note in a track, calculates the amount of syncopation notes, and calculates the rating using equation 12 below:

$$R = \frac{\text{Number of syncopation notes}}{\text{Total number of notes}} \tag{12}$$

8.2.6 Pitch Range In Melody

The purpose this sub-rater is to determine the pitch range between the highest and lowest pitch values in a track. The sub-rater iterates through every note in a track, finds the highest and lowest pitch, and calculates the rating using equation 13 below:

$$R = \frac{\text{Lowest pitch value}}{\text{Highest pitch value}} \tag{13}$$

8.2.7 Variety Of Rest Note Density

The purpose of this sub-rater is to determine the variety of rest notes, where a rest note is defined as a silent note. The sub-rater iterates through every note in a track, calculates the average and lowest rest note density, and calculates the rating using equation 14 below:

$$R = \frac{\text{Least number of rest notes in a beat}}{\text{Average number of rest notes in all beats}} \tag{14}$$

8.2.8 Continous silence

The purpose of this sub-rater is to identify longs segments of silence. Long silent segments are not dersireable, making this sub-rater very necessary for assuring quality. First, the sub-rater determines the longest distance between phrases, Secondly the rest notes are identified. The length of each section containing only rest notes is added to a total sum and compared using equation 15 below:

$$R = 1 - \frac{\text{Length of total silence interval}}{\text{Length of input segment}} \tag{15}$$

8.2.9 Unique Note Pitches

The purpose of this sub-rater is to determine how many unique note pitches that are being used in the music segment. It rarley occurs that music contains only one or two different note pitches, a decent variety of them is necessary to assure quality. The sub-rater iterates through every note in a track, finds all unique notes, and calculates the rating using equation 16 below:

$$R = \frac{\text{Number of unique note pitches}}{\text{Total number of notes}} \tag{16}$$

8.2.10 Equal Consecutive Notes

The purpose of this sub-rater is to rate how many times two consecutive notes share the same pitch value. The sub-rater iterates through every note in a track, finds all note pairs who share the same pitch value, and calculates the rating using equation 17 below:

$$R = \frac{\text{Number of two consecutive notes with the same pitch}}{\text{Total number of notes}} \tag{17}$$

8.2.11 Unique Rhythm Values

The purpose of this sub-rater is to rate the variety of rhythm values, which can be defined as the duration of the notes. The necessity of this rater is based on the fact that different genres differ greatly in rhythm diversity. Simple melodies tend to keep the same rhythm while more complex music can differ greatly throughout a song.

The sub-rater iterates through every note in a track, finds each unique rhythmvalue, and calculates the rating using equation 18 below:

$$R = \frac{\text{Unique rhythm values}}{\text{Total number of notes}} \tag{18}$$

8.3 Advanced sub-raters

Listed within this section are the advanced sub-raters with their specific purpose and functionality described in detail. These sub-raters rates more advanced musical features.

8.3.1 Repetition rating

The purpose of the repetition sub-raters is to identify repetitions in the individuals. They consist of three sub-raters, each considering a separate of the beat,chord and melody tracks. The reason for this separation of three relatively equal tasks is first and foremost the fact that a repetition is of differing importance depending on the track type. In addition, the melody and chord repetition sub-raters observes repeating pitches while the beat repetition subrater observes repeating rhythm values.

Western musical structure relies heavily on repetition divided into multiple choruses. Because of this, repetition is a favorable characteristic that is necessary to detect with sub-raters to ensue that the music produced is appealing.

The repetition sub-raters employ a modified version of a longest repeated pattern algorithm which is based on suffix arrays, originally used for finding repeating patterns of characters in strings[15]. The sub-rater's method for calculating the rating consists of multiple steps. First, they indentify all repeating patterns whose length must be at least equal to one percent of the total track length, which was determined by extensive testing to find an optimal percentage which would work with a large range of track lengths; Second, they sort the patterns depending on how much of the entire track they cover; Third, the sub-raters iterate through the sorted set of patterns and identify the pitch or rhythm values within them; Finally, the rating is calculated using equation (19). The repetition sub-raters favor tracks where most of the pitch or rhythm values are found within repeating patterns.

$$R = \frac{\text{Number of pitch or rhythm values within patterns}}{\text{Total number of pitch or rhythm values}} \tag{19}$$

8.3.2 Scale Pattern

The purpose of this sub-rater is to identify the occurence of the common diatonic scale pattern. Western music usually stays in the same key throughout the entire song which implies that the majority of pitches should follow the same scale pattern.[16] Diatonic scales consist of seven notes and a repeating octave. A simple analytic model is therefore to consider the amount of notes in the song that follow the diatonic pattern.

To compute the rating, the sub-rater finds the best way to fit the music segment's notes into the scale pattern. After the optmial fit is found, the number of notes that fit perfectly into this pattern is divided by the total amount of notes in the music segment. The lowest possible outcome is represented by the fraction $\frac{N}{12}$, where N represents the amount of notes in the pattern. The reason for this is that the maximum amount of notes within an octave cannot exceed 12. This lowest outcome is utilized to normalize the linear equation 20. Following the normalization, the final rating is calculated using equation 21. Figure 11 visualizes a scale pattern matching using the diatonic scales, where $N = 7$.

$$R(x) = kx + m = \begin{cases} 0 & x = N/12 \\ 1 & x = 1 \end{cases} \Leftrightarrow R(x) = \frac{12x - N}{12 - N} \tag{20}$$

$$\begin{aligned} r &= \frac{\text{Maximum notes fitted into pattern}}{\text{Total number of notes in the music segment}} \\ R(r) &= \frac{12r - N}{12 - N} \end{aligned} \tag{21}$$

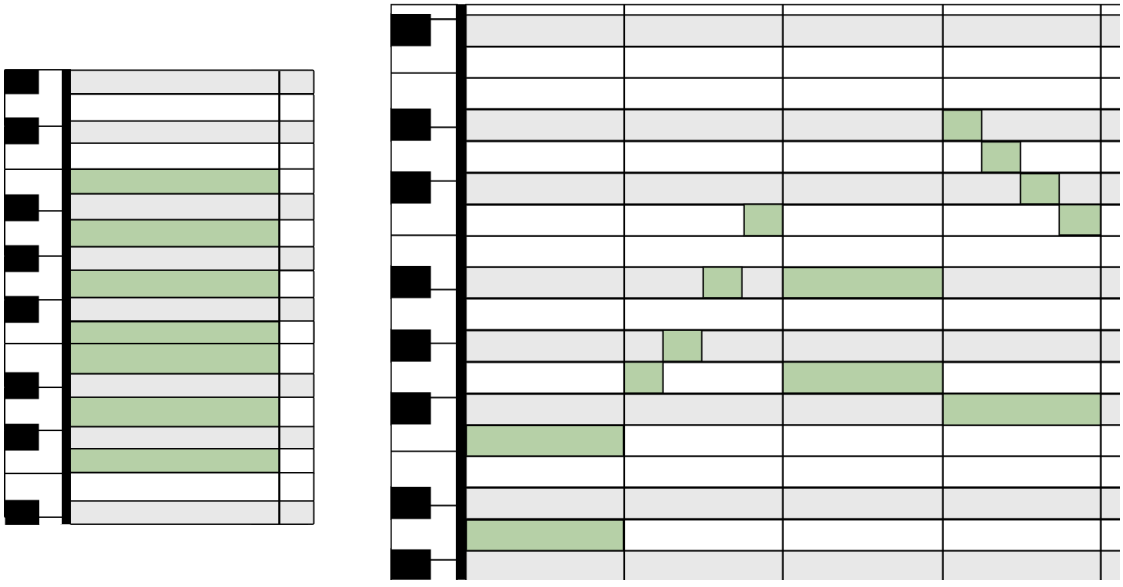

Figure 11: The left image represents the diatonic pattern. It is shifted 12 times to find the maximum number of matches with the example melody pattern to the right. Both melodies will always be transposed to stay within the range of one octave. The maximum number of matches in this case is found when the pattern is shifted one semitone up.

8.3.3 Window based scale pattern rating

The purpose of this sub-rater is to act as an extension to the scale pattern sub-rater (see section 8.3.2). It utilizes the scale pattern sub-rater to rate smaller segments. The smaller the segments are, the bigger their impact on the resulting rating. The reason for this is that it is easier to notice if notes do not fit into the same scale pattern if the time difference between the notes is smaller.

The small segments are created and rated recursively by starting at a maximum segment size that is decreased by half until a minimum segment size is reached. The size ranges are defined by the user, an example of the segmentation can be seen in figure 12 below:

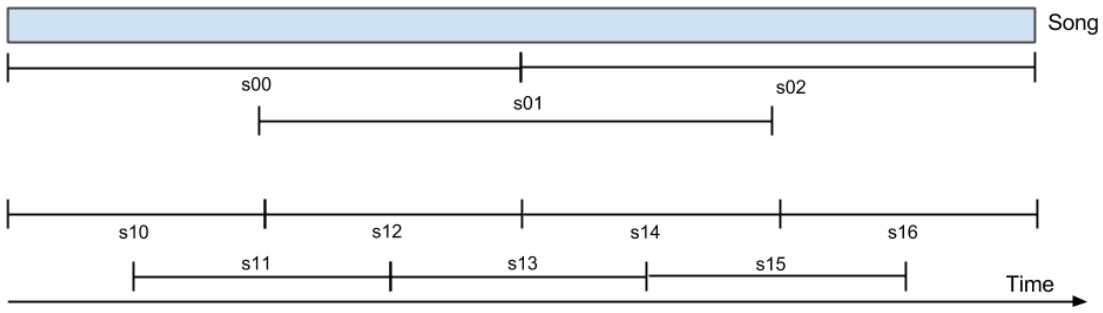

Figure 12: The illustration shows the timeline of a larger segment. First, the new segments s00-s02 are created, after which they are split into the smaller segments s10-s16.

8.3.4 Zipfs law

The purpose of this sub-rater is to determine how well the occurences of pitch values follow the distribution function given by Zipf's Law.[17] Zipf's law was originally used for determining the frequency of words in litterature, but studies have shown that these distribution curves are relevant in musical theory as well.[18]

Zipf's law is based on a ranking system in respect to occurences[19], when applied in music the pitch values are ranked depending on how often they occur. The theory behind zipf's law in music states that there is a relationship between the different occurences of pitch values corresponding with their respective rank. According to this theory, the number of occurences of the most occuring pitch value multiplied with its rank should give a value equal or close to the number of occurences of the pitch value which occurs second most multiplied with its respective rank and so on.

Studies for zipf's law in musical theory have shown that a piece of music which follows this law is more likely to sound pleasant and thus gives a way of predetermining the quality of music before listening to it.[18]

This sub-rater computes the rating by first calculating the optimal zipf's law value for every unique pitch in a music segment, ranking by number of occurences of each pitch. Finally the sub-rater utilizes equation 22 below:

$$D = \sum\nolimits_{P \in U_p} \triangle(O_d P, A_d P) \tag{22}$$

D represents the sum of the distances between the optimal distribution O_{dP}, given by Zipf's law, and the actual distribution A_{dP}, for each pitch P in all unique pitches U_p. Finally the rating is calculated by subtracting the division of the distance D, by the maximum possible distance D_{max}, from 1. This is visualized by equation 23 below:

$$R = 1 - \frac{\mathrm{D}}{\mathrm{D}_{max}} \tag{23}$$

8.3.5 Dissonance and consonance

The purpose of this sub-rater is to identify dissonance between note pitches within melody and chord tracks. Too much dissonance in a song is often considered to be unpleasant, this rater identifies how many simultaneous note sets are cosonant or dissonant. This method is based upon a purely mathematical approach.[20] Including a more mathematical viewpoint on how well notes play together is necessary to keep the sub-raters diverse.

By observing the logarithm of the lowest common multiples of the note pitches played simultaneously, the dissonance and consonance level between them can be calculated. The rating is calculated by equation (24) below:

$$R = 1 - \frac{\text{Average dissonance}}{\text{Average worst case dissonance}} \tag{24}$$

9 Selection

This process emulates natural selection and its purpose is to choose the best individuals but there is an element of randomness involved. The selection process will not simply pick the best individuals and send them into the next step of the algorithm, such a procedure would have a complete lack of diversity. Instead, a roulette wheel selection is employed.

9.1 Roulette Wheel Selection

This selection method is a direct analogue to a casino roulette, where a proportion of the wheel is assigned to every possible selection. The genetic algorithm utilizes the rating of each individual to determine how large their part of the wheel will be. This process ensures that it is most probable that the best individuals will be selected, but there is still a chance for lesser individuals to be chosen instead. The selection process is visualized in figure 13 below:

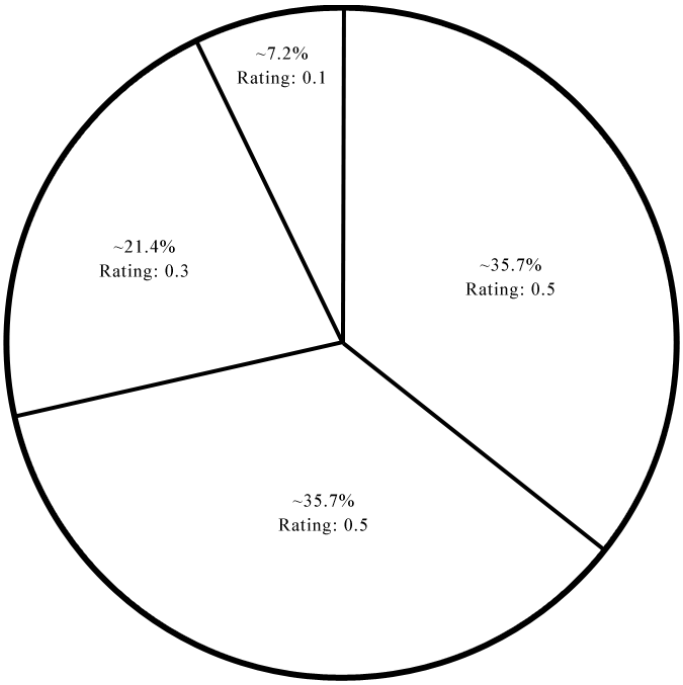

Figure 13: Roulette wheel probability distribution employed on 4 individuals

10 Results

The results are represented by the final output generated by the genetic algorithm. The output consists of the best remaining individual after the final iteration. The final result varies greatly depending on the seed and method of initialization.

Despite the method of choice, the final results have been of varying quality. The algorithm is not to blame for this as the results are consistently rated higher the longer it iterates. This means that theoretically, all results would be equally appealing if the algorithm was iterated a large enough amount of times.

Another factor lies within the multiple user-defined parameters. A very large amount of testing would have to take place to decide the best parameter settings for generating a specific output. Quite alike the method of developing video games, while the game code might be finished, a lot of time must still be spent on balancing it to create a pleasant experience. Due to the time constraints of this project, parameters have not been balanced in a completely satisfactory way.

10.1 Crossover initialization results

Crossover initialization generates the first generation by performing crossover on a seed. The user may choose to initialize the seed multiple times, increasing the crossover effect. This method has generated the best results so far, mainly because of the musical structure granted by the seed.

10.2 Random initialization results

Random initialization has proven to produce the most diverse results whilst lacking in quality. The rating of these individuals tend to be lower than individuals initialized with other methods. This is due to the other initialization methods utilizing a seed which follow a musical structure, which naturally rates higher. In short, random initialization produces a larger amount of unique individuals, but the qualities of those tend to be questionable.

10.3 Markov initialization results

The Markov chain initialization method uses multiple Markov chains with one or more seeds to generate statistically similar music. This method was found to be the best one for merging different music into one coherent musical piece. The different parts in the music connected to each other seamlessly, compared to where the crossover could cause strange cut-offs. Additionally, this method was able to decide the exact length and randomness of the generated music pieces.

While the Markov chain method had the same advantage as crossover where the generated individuals were of high quality, they did not retain the structures of the original music segments where certain notes are accented at certain beats. Overall this method seems to have a good balance between diversity and structure, with the possibility to emphasize diversity if the user so desires.

11 Discussion

This section contains a discussion about the progress and challenges faced whilst trying to generate appealing music with genetic algorithms. It should be noted that the project should be seen as an experiment rather than a complete application. Hence, this section contains multiple recommendations for improving the current state of the application.

11.1 Creating the first generation

The initialization, explained in detail in section 5, is essential to the result as all individuals generated are based upon the first generation. It is understandable that changing the initialization method will cause the output to sound very different. The crossover initialization was the first method to be implemented, producing some quite appealing results. However, this method still lacks diversity as the produced results sound quite alike.

The lack of diversity motivated the creation of the two alternative initialization methods, random and Markov chains. The results vary greatly depending on the chosen method, therefore additional methods would be favorable to experiment further.

11.2 The non increasing rating over generations

In order to follow the progress of the algorithm, the best rating of each generation was consistently printed. A recurring pattern was noted where the rating would stop increasing after only a few generations. Since a genetic algorithm is partly stochastic, there is no guarantee that the result will increase with each generation. The amount of iterations necessary for improvement has a potential to increase beyond reasonable limits. Figure 14 illustrates an example of such behavior. If the highest rating found is at a), the genetic algorithm will most likely discard the individuals that are going towards b), since the area between a) and b) has a lower rating than a), effectively discarding individuals with undiscovered potential.

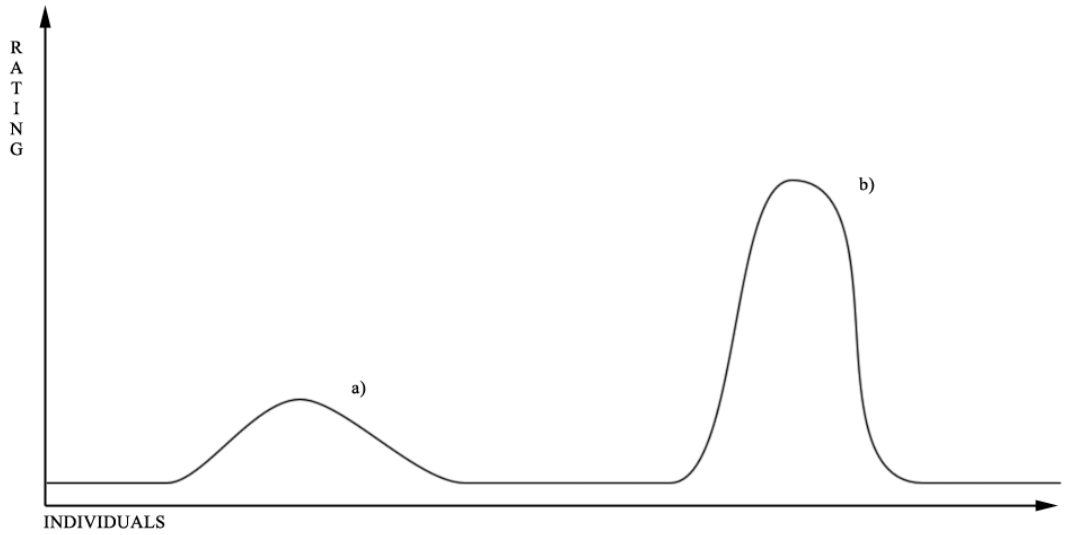

Figure 14: a) represents a local maxima of the rating while b) represents the global maxima.

11.3 Application complexity and issue solving

In order to test if the application was performing well, test-cases were used to check that every implementation was done in the right way. Unfortunately, odd behavior still occured even with the test-cases passing. The complexity of the application made covering every possible scenario in the test-cases very challenging. To improve this it is necessary to spend large amounts of time assuring that each test-case covers all possible scenarios.

11.4 Which sub-mutators to implement and use

The functionality of the sub-mutators was quite narrow initially, but they were expanded to affect the musical segments to a greater extent. This decision was based upon the realization that mutation is the only way for the segments to change their containing parts during an execution.

What followed was an implementation of four sub-mutators that together could handle all the editing of the individuals. A more detailed description of these can be seen in section 7. Theoretically, these mutators are able to edit an individual towards any desired structure. Possible improvements for the mutation process would be to include more mutators to increase diversity and perhaps ecourage more rule based mutation.

11.5 Which sub-raters to implement and use

The fitness function, being named the rater in this project, is the most integral part of the genetic algorithm. What sub-raters to use and implement is by far the most important decision to make, without good sub-raters the application will not generate anything of value.

Initially the group decided to work with the repetition raters (see section 8.3.1). This decision was based on intuition of what is important when creating good music. Even though the repetition raters are still a part of the application they do not represent the only features in music that are relevant. Following this, a document was found which gave inspiration of what additional raters to implement [21].

In total, sixteen raters were implemented, which is insufficient to consider all aspects of what good music is. However, the project has been created in a modular way which makes it easy to add new raters in the future. As the objective from the very beginning was to keep the application as modular as possibile the choice of having many sub-raters which each had a small responsibility and clear task was the best approach.

This was accompanied with a discussion about single- and multi-track sub-raters. A single-track sub-rater would only observe one track at a time while a multi-track sub-rater would observe multiple ones. The multi-track sub-rater would be able to rate how well different tracks fit in with respect to one another.

Implementing more sub-raters, especially multi-track ones, would significantly improve the application's ability to detect good music.

11.6 Single or multiple track-tags per track

An ongoing discussion was held around the use of multiple versus a single track-tag for each track. The discussion was based on the notion that a track could be considered to fill multiple functions. Since the application would become increasingly complex to develop if multiple track-tags were implemented, the decision was made to force the user to choose a single track-tag for each track.

Implementing multiple track-tags would be an improvement as tracks could fulfill mutiple roles, as long as the increasing complexity is handled properly.

11.7 Target rating usage and application

The theory behind the target ratings and the practical functionality of them was discussed and implemented quite early in the project. The purpose of the target rating was initially to determine a sub-rater's influence on the total rating.

This usage of target ratings were discussed later in the project and the decision was made to change it. In the original implementation, each sub-rater strived to achieve the maximum rating of 1.0 where the target rating acted as a weight on how much influence it had on the final rating. In the current implementation the rater instead strives towards the preset target rating. This approach increased the usefulness of the sub-raters now that a low rating was no longer instantly considered as a bad one. This implementation is more in line with our assumption that the computer cannot know what good music is and must be guided, here with the use of target ratings.

11.8 Diversity or appeal

The diversity and appeal of a musical segment has proven to be difficult to balance. An implementation striving for more diversity tends to result in a loss of appeal, and vice versa. As the application should be able to generate both diverse and appealing results, this posed a major problem.

Similar to the initialization, parameters have been added to let the user choose more freely which kind of output is desired. An interesting addition could be to allow some sub-raters to observe previous parents and other siblings in order to know the relation between the individuals in a generation. Such an implementation could theoretically improve the balancing of the results by benefiting individuals that stand out from the rest instead of solely producing the extremes of each side.

11.9 Choice of third-party modules

To be able to work towards the project goal efficiently, existing modules were utilized to lessen the workload. The free libraries and additional modules allowed the focus to lie on the actual project, rather than implementation of irrelevant parts not included in the scope of the purpose.

11.9.1 jMusic

As stated in section 3.1, jMusic is used for interaction and manipulation of MIDI data. Several libraries with similar properties were reviewed and the cholice lay between three possible candidates: The built in Java Sound API, jMusic and jFugue. One of the main reasons to use jMusic was due to the little effort that was put into getting started, thanks to its documentation of fundamentals and example code. The research did eventually prove to be too shallow and could have been made more thoroughly.

There is a frequently occuring problem when working with the structure of jMusic. A problematic scenario occurs when notes share the same start time. Such notes are separated by phrases which means that finding those specific notes require an iteration through each phrase.

A sort function was implemented to combat the phrase issue, sorting notes depending on start time. Additional layers were also added on jMusic's native implementations. The

structure of jMusic has caused complications that required additional time to compensate for while also increasing the complexity of the application. As the problems started occuring late in the project phase, a change of library was not an option. In hindsight, jMusic was not very well suited for the needs of this project and an improved version should consider another library.

11.9.2 MongoDB

The necessity of a database became clear early in the project as it was necessary to store songs and their properties. Upon creating the first generation, the application needs a set of stored songs in a database to aquire the target ratings. MongoDB is an open-source document database and has proved to be easy to both setup and use.[22] MongoDB has proven helpful when developing with an iterative approach, as it automatically handles situations where fields are added or removed to collections in the database.

11.10 Graphical User Interface

A Graphical User Interface (GUI) is an essential part of any application aimed at users other than the creators. Due to the limited time-frame of the project, little effort has been put in the GUI. still, it makes running the application more convenient, even though it is very simple.

```
--- EvoMusic shell version 0.1 ---
Enter '?' for help.

evomusic> ?
Available commands:
song [tags|select|play|delete|translate <path>|deselect|list]
generate <numberOfIterations>
?
evomusic> song list
Available songs(BPM):
0 -> Norway(72.00003051757812)
1 -> super_mario_bros_theme(200.0)
2 -> test_forest(120.0)
3 -> nyan_cat_cut(140.00013732910156)
4 -> m83(105.00010681152344)
Selected songs(BPM):
evomusic> song select 2 3 4
Selected song 2
Selected song 3
Selected song 4
evomusic> generate 2000
Start iterating
[#########                                ] 25% | Best rating: 0.8793947820
```

Listing 1: An example output from the command line interface of the application. The user enters commands after the prompt character

If the application would be developed into a consumer version, the GUI would need to be improved. The improved GUI should make it very simple for the user to provide the seed and generate new songs. The choice of using a terminal based user interface would also make the application easy to run on a server using it to implement a web application.

12 Possibilities and limitations for use in society

The varying results of the project shows that the employed approach of computational musicology still has a long way to go. Every single part of the project has to be carefully considered and analyzed in order to improve and balance the result. As this is not possible within the time constraint of project course, the application was created with a modular approach. If the project would become open-source, the modularity would welcome other developers to contribute to the project. This is a way of engaging developers to be part of a community, but would also effectively drive the research within the area forward. While releasing the project as open-source could negate the possibility of economical gain, it may still be of potential use for the music industry and artists. One of the most promising effect of this application is that it could potentially help artists of any experience level to get the inspiration they need to write new songs.

The project will probably have little to no impact on any environmental aspects. On the other hand, the time between starting to write music to getting a result can become a lot shorter with the application, which might suggest that the application is consuming less power than using a notation software to manually create a song. To be able to clearly determine if this is true, an environmental analysis has to be done, which is outside of our scope.

The project as a whole has explored many different posibilities within the study of computational musicology. Hopefully, others will benefit from the work done.

13 Bibliography

[1] J.-J. Nattiez, *Music and discourse: Toward a semiology of music.* Princeton University Press, 1990.

[2] A. Alpern, *Techniques for Algorithmic Composition of Music.* Hampshire College Press, 1995.

[3] D. J. Grout and V. Claude, *A History of Western Music.* New York: W.W.Norton & Company, fifth ed., 1996.

[4] E. Bowles, *Musicke's Handmaiden: Or Technology in the service of the arts.* Cornell University Press, 1970.

[5] I. Xenakis and S. Kanach, *Formalized Music: Thought and Mathematics in Composition.* New York: Pendragon press, pendragon revised ed., 1992.

[6] B. L. Jacob, "Composing with genetic algorithms," *International Computer Music Association*, vol. 1, 1995.

[7] B. Johanson and P. Riccardo, "Gp-music: An interactive genetic programming system for music generation with automated fitness raters," *Proceedings of the Third Annual Conference*, vol. 1, 1998.

[8] M. M. A. Incorporated, "Tutorial: The technology of midi." `http://www.midi.org/aboutmidi/tut_midifiles.php`, 1995. Accessed: 2014-05-18.

[9] A. Brown and A. Sorensen, "The jmusic data structure." `http://explodingart.com/jmusic/jmtutorial/x71.html`, 1998. Accessed: 2014-05-18.

[10] J. R. Norris, *Markov chains.* Cambridge University Press, 1998.

[11] K. Mcalpine, E. Miranda, and S. Hoggar, "Making music with algorithms: A case-study system," *Computer Music Journal*, vol. 23, no. 2, pp. 19–30, 1999.

[12] W. spears and V. Anand, *A STUDY OF CROSSOVER OPERATORS IN GENETIC PROGRAMMING.* Massachusetts Institute of Technology, 1991.

[13] M. Wahde, *Biologically Inspired Optimization Methods: An Introduction.* WIT Press, 1 edition ed., 2008.

[14] A. Latham, *The Oxford Companion to Music.* Oxford University Press, 2011.

[15] S. Aluru, "Lookup tables, suffix trees and suffix arrays," *Handbook of Computational Molecular Biology*, vol. 1, pp. 5–1–5–26, 2006.

[16] T. A. Johnson, *Foundations of Diatonic Theory: A Mathematically Based Approach to Music Fundamentals.* Key College, 2003.

[17] R. E. Wyllys, "Empirical and theoretical bases of zipf's law," *Library Trends 30*, vol. 1, pp. 53–64, 1981.

[18] B. Manaris, D. Vaughan, C. Wagner, J. Romero, and R. B. David, "Evolutionary music and the zipf-mandelbrot law: Developing fitness functions for pleasant music," *Lecture Notes in Computer Science*, vol. 2611, pp. 522–534, 2003.

[19] J. C. Sorell, "Zipf's law and vocabulary," *The Encyclopedia of Applied Linguistics*, vol. 1, pp. 1–6, 2012.

[20] E. Knobloch, "Euler transgressing limits:the infinite and music theory," *Quaderns d'història de l'enginyeria*, vol. 9, pp. 9–24, 2008.

[21] A. Brown, M. Towsey, S. Wright, and J. Diederich, *Statistical analysis of the features of diatonic music using jMusic software*. Research Institute for Humanities and Social Sciences (RIHSS), the University of Sydney, 2001.

[22] Open-source, "Mongodb overview." `http://mongodb.com/mongodb-overview`, 2009. Accessed: 2014-05-19.

A Formal mathematical definition of Markov chains applied in the project

I : A set of intervals

l : Length of musical piece (number of notes)

M : Melody of musical piece

n : Order of the Markov chain

s : Current state

s' : Next state

$At_M(s,k)$: State s at position k with length m

$R_M(s,s')$: s' is a successor of s

$Occ_M(s)$: Occurence of state s

$Occ_M(s,s')$: Occurence of state s' being a successor of s

$P_M(s,s')$: Probability for transitioning from s to s'

$$M : \{0,\dots,l-1\} \to I,\ M_0 = 0$$

$$At_M(s,k) \iff \exists m \leq n \big[s = (M_k,\dots,M_{k+m-1}) \big]$$

$$R_M(s,s') \iff \exists k \exists m \left[s = (M_k,\dots,M_{k+m}),\ s' = \begin{cases} (M_k,\dots,M_{k+m+1}) & : m < n \\ (M_{k+1},\dots,M_{k+m+1}) & : m = n \end{cases} \right]$$

$$Occ_M(s) = \{k | s = (M_k,\dots,M_{k+m})\}$$

$$Occ_M(s,s') = \{k | At_M(s,k) \wedge R_M(s,s')\}$$

$$P_M(s,s') = \frac{\#Occ_M(s,s')}{\#Occ_M(s)}$$