CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Apply evolutionary algorithms for music generation

Create a modular algorithm for research into the capabilities of evolutionary algorithms to create interesting musical structures

Bachelor's thesis

THE DEPARTMENT OF COMPUTER SCIENCE

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Bachelor's thesis 2022

Apply evolutionary algorithms for music generation

Create a modular algorithm for research into the capabilities of evolutionary algorithms to create interesting musical structures

Martin Bergström
Linnea Johansson
Emilie Karon Klefbom
Axel Lundberg
Viktoria Löfgren
Matilda Sönnergaard

CHALMERS

Department of Computer Science
DATX02 - group 52
Chalmers University of Technology
Gothenburg, Sweden 2022

Apply evolutionary algorithms for music generation
Create a modular algorithm for research into the capabilities of evolutionary algorithms to create interesting musical structures

Supervisor: Palle Dalhstedt, Associate Professor at University of Gothenburg
Examiner: Morten Fjeld, Professor at Chalmers University of Technology

Bachelor's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2022

Abstract

This bachelor's thesis aims to explore the possibilities of generating new music pieces with the help of algorithms and artificial creativity. The work explores the potential of generating music with computers and reflects on whether the generated music has any musical relevance. To generate pieces of music a genetic algorithm with a tree structured representation was used. For each generation, individuals are evaluated through a combination of human and automatic components which may allow the piece to affect further generations. To investigate the musical relevance; a questionnaire with generated pieces was distributed on social media to people with different musical backgrounds. The results show that our tree structure may have musical potential. The results also shows the comparison of different rating methods and genetic operators and their performance.

Sammandrag

Detta kandidatarbete har som syfte att utforsta möjligheterna att skapa nya musikstycken med hjälp av evolutionära algoritmer. Arbetet utforskar datorns potential att skapa musik och reflekterar över den genererade musiken har någon musikalisk relevans. För att skapa musikstycken användes en genetisk algoritm samt träd som musik representation. För varje generation utvärderas individer genom en kombination av mänskliga och automatiska komponenter som sedan låter stycket påverka kommande generationer. För att undersöka den musikaliska relevansen distribuerades ett frågeformulär med de genererade musikstyckena på sociala medier till personer med olika musikaliska bakgrunder. Resultaten visar att vår trädstruktur besitter viss musikalisk potential. Resultaten visar också jämförelsen av olika genetiska operatorer och fitness-funktioner och deras prestanda.

Keywords: Evolutionary algorithms, Genetic algorithms, Music generation, Python, Machine learning, Optimization, Artificial Intelligence, Genetic representation.

Acknowledgements

This report is a bachelor's thesis written at Chalmers Technical University the spring of 2021. The authors are six Bachelor students studying Computer Science, Information Technology and Engineering Mathematics.

We would like to give our thanks to our supervisor Palle Dahlstedt who has with great patience supported the project. With his extensive knowledge in music generation and experimentation he has guided us with appropriate sources and general ideas.

Finally, we would also like to thank everyone who took the time to respond to our questionnaire and provided feedback during the project.

Martin Bergström
Linnea Johansson
Emilie Karon Klefbom
Axel Lundberg
Viktoria Löfgren
Matilda Sönnergaard

Gothenburg, May 2021

Contents

Acronyms

AI Artificial intelligence.

ANN Artificial Neural Network.

EA Evolutionary Algorithm.

GA Genetic Algorithm.

LSTM Long Short-Term Memory.

MIDI Musical Instrument Digital Interface.

RNN Recurrent Neural Network.

Glossary

Chord A set of two or more simultaneous notes.

Crossover Crossover is the process of combining two individuals and generating a new individual with a combination of each parent's genes.

Dynamics Refers to variations of intensity or volume.

Gene A gene is one of the most fundamental elements in heredity and encodes a certain function.

Generation A generation is one iteration of the genetic algorithm which corresponds to one unique set of individuals. These individuals will become the parents of the next generation.

Genome The genome is an individual's complete set of genes.

Genotype The genotype describes an individual's unique set of genes.

Grid search Parameter search method where several combinations of parameter values are tested in a grid [27].

Hyperparameter Hyperparameters are parameters used to control how the learning algorithm should work. This could be the number of generations or mutation rate.

Individual An individual is a musical tree structure that encompasses a set of genes.

Key signature The key signature defines the sharps or flats in a piece of music.

Learning rate The learning rate determines how quickly the generator converges.

Melody Series of notes perceived as an entity.

Mutation Mutation is the process of changing one individual's set of genes.

Note A sound event with a specific pitch and duration.

Phenotype The phenotype is the observable set of traits that the genotype encodes.

Pitch The perceived frequency of sound waves.

Scale A set of pitches in a relative order that is predefined.

Solution space Set of all possible answers to a problem.

Time signature Refers to the number of beats in a measure of music.

1

Introduction

Since the earliest civilizations, human culture has been intertwined with music. As early as 35 000 years ago there has been evidence of flutes that would have been used as an instrument for playing music [9]. Today, people listen to music for 18 hours a week or 2.6 hours a day on average [25].

The computer had its debut in the art of music in the early 1950s when the Commonwealth Scientific and Industrial Research Automatic Computer (CSIRAC) played music for the first time [13]. The concept of algorithmic composition, with music being the result of algorithms, was introduced with project ILLIAC 1 which became the first computer to be used for composing music in 1957 [20].

The first instance where neural networks was considered in the field of automatic music generation was in 1988 when Lewis and Todd used Recurrent Neural Network (RNN) in music composition [28]. In 2002 the first use of so called Long Short-Term Memory (LSTM) networks occurred, and this has since become one of the standard ways of implementing neural networks for music composition [14].

As computers have become a considerable part of music generation and composition, there has been a debate alongside the development. The majority of the programs used for automatic music composition mainly look for similarities in the data it was trained on to create similar pieces. Creativity, however, is about producing *new* patterns, and subsequently the models in use often lack creativity [11]. Margaret Boden, British philosopher and computer scientist, claims that while many approaches to music generation lack transformational creativity, a form of creativity can in fact occur when using evolutionary AI programs [7]. Evolutionary algorithms could help on the way to solve the problem of creativity by implementing generative methods based on evolutionary models [32].

1.1 Purpose

The purpose of the project is to explore different aspects of a generative musical program. The focus is on learning about different music representations in programming, variations of evolutionary algorithm methods as well as neural networks.

1.2 Relevance

So why is the project relevant at all? Will other people find use of our work? We can look at a company like AIVA who has built their whole business using this exact technology [1]. Their target group is, for example, game developers who want to use customized music for hours of gameplay, and it is often unreasonable for small developers to hire humans to write and produce this music. Instead, algorithms could be used to allow for more economic options for creators of all sorts that want to have custom music for their projects. By further improving creative algorithms, it could also be applied to text, video and art or where there is a need for a large amount of customized content. Algorithms that have creative elements may also be developed as a tool to allow more people to get involved in creative processes without the need for formal training. AI could be the new essential tool for artists in the future.

1.3 Previous work

One example of using a generative genetic representation in an EA was made by Richard Dawkins [12]. He used a genotype consisting of nine genes taking on integer values. Initially, Dawkins was able to use this algorithm to generate tree structures where different variations of the genes controlled things such as the angles of the branches or how large the tree would become, and was positively surprised when the algorithm started generating structures resembling creatures. He used these representations as a basis to discuss what creativity is. While a computer only does what we, the programmers, tell it to do, the search- or solution-space, which is the space of all possible individuals, is very large. The creativity, as Dawkins describes it, comes from defining a method that can efficiently search for valuable and desirable candidates.

An important source of inspiration for us was Karl Sims' work in computer graphics, which is an early example of the use of EA in art. Sims created an EA that could generate images [37]. By storing genetic information as a mathematical formula, he was able to compute the colour of each pixel in the image. He primarily used user interaction for determining the fitness of the generated images, which inspired our implementation of interactive fitness.

An example of using an EA to generate music can be seen in Palle Dahlstedt's Ossia, which works by storing musical information in binary tree structures [10]. Every leaf node corresponds to a note, and every other note in the tree contains information about whether to play its left and right sub trees simultaneously or in sequence. The tree can also contain branches with recursive properties, effectively expanding the tree by repeating the recursive branches. The expansions of the recursive branches can be assigned operators, such as multiplying amplitudes, durations and pitches. The music is evaluated through a set of properties, such as *note density*, *repetitivity* and *silence*, with target values defined by Dahlstedt.

For more reading on previous work on evolutionary computation in music, we refer to the book *Evolutionary Computer Music* by Eduardo Reck Miranda and Al Biles [32].

1.4 Scope and delimitations

One delimitation is to handle music represented as musical scores rather than raw audio, since raw audio generation is often found to be more complicated and would not be a realistic choice with our time constraint and knowledge. Musical scores are easier to combine with EAs because of their symbolic representation. For these reasons, the project is focused on *composition* rather than *synthesis*.

Evolutionary Algorithms is a wide term referring to any algorithm that mimics biological processes. To narrow the scope, the focus is on using one type of Evolutionary Algorithms — Genetic Algorithm (GA). This type of algorithm has been shown to be useful in creative tasks such as music composition [17, 7].

2

Theory

Artificial intelligence (AI) is a set of tools that have great versatility in problem solving. The typical applications are optimization and prediction, but there is also a field of research trying to explore AI as part of creative processes such as music composition. This chapter will introduce different techniques within AI which are necessary to understand the content of this thesis. The chapter will also present some music theory that may be helpful to understand the rating of the music pieces.

2.1 Evolutionary and genetic algorithms

A Genetic Algorithm (GA) is an algorithm used for stochastic optimization influenced by how nature works, Darwinian evolution, and was first introduced by John Holland [23]. It is a type of the more general Evolutionary Algorithm (EA), and what differentiates the GA from other types of EAs is that it uses both crossover and mutation as genetic operators.

The algorithm defines a population set containing *individuals* and tries to replicate survival of the fittest by using a scoring function. Individuals with greater fitness scores are selected to move on to the next generation, the next iteration, for reproduction. These are called the parents, and by using a *crossover* function to combine genes of different parents, the algorithm produces a set of children who form the new generation. Individuals from the old generation may also be preserved in the new generation through something called *survivor selection*. All this is done through random processes, hence the stochastic nature of the GA. GAs rely on genetic operators such as selection, crossover and mutation to search the solution space for different optimization tasks. The algorithm is split into five phases: initialize the population, fitness evaluation of each individual, selection, crossover and mutation. Figure 2.1 shows a flow chart structure of the algorithm.

The initialization process depends on the nature of the problem, but the initial population is usually randomly generated to increase the probability for the individuals to be widespread in the search space. However, it would also be possible to initialize the population with a predefined group of individuals, which may increase the convergence speed or result in a narrower search space.

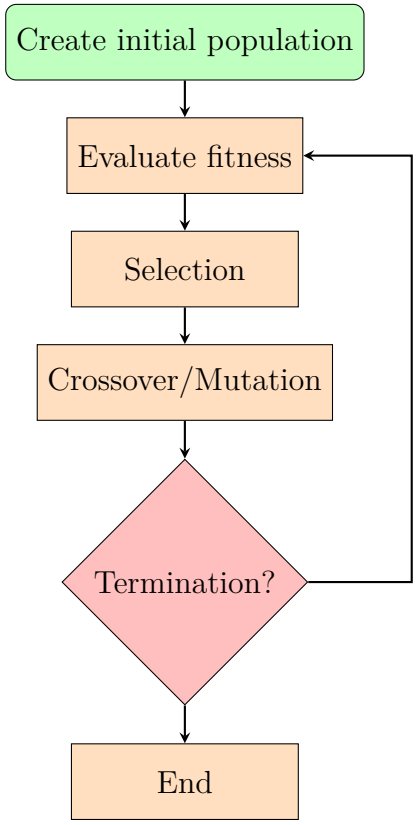


Figure 2.1: Flow chart of the genetic algorithm.

When using a GA to solve optimization problems, the fitness function is often designed in accordance with the constraints involved. However, when applying a GA in a creative field such as music generation, the fitness function is more difficult to formulate because musical value comes from subjective experience. One approach is to directly involve a user in the scoring process to guide the algorithm with their immediate response, see Section 1.3. Another approach is to train a neural network with existing sample data to act as the scoring function [15].

Independent of the underlying logic, the score of individuals will determine how likely they are to "mate" which in turn determines how likely their genes are to survive to the next generation. The *parent selection* process has a significant role and could be constructed in a variety of different ways as seen in Section 3.4.3.

After individuals have been selected, they are processed using different genetic operators such as *crossover* and *mutation*. The crossover operator picks a subset of genes from two or more individuals and combines them to form a new individual or what we would call a *child*. These children are thereafter mutated to introduce new features, before moving on to form the next generation. Each individual is thus a solution on its own, and the fitness score corresponds to how close the individual is to an optimal solution.

The iterative process of generating new individuals is repeated until a termination condition is met. One criterion could be to let the algorithm run for a fixed number of generations. Another could be to stop the iterations when the solutions satisfy some minimum score threshold. A more suitable approach for solutions requiring human interaction is manual inspection.

2.2 Neural networks

Artificial Neural Networks (ANNs) are influenced by how biological neural networks operate. They are based on interconnected networks of artificial neurons, a term first introduced by Warren McCulloch and Walter Pitts [30]. In its most basic form, the neural networks consist of only dense layers in which the artificial neurons are simple classifiers called perceptrons. The concept of the perceptron was first introduced by Frank Rosenblatt [35] and is in the context of neural networks the simplest feed-forward neural network model. It operates as a linear classifier that classifies weighted input by separating them into two categories. In other words, it produces a single output based on its inputs by using a linear combination. A multilayer perceptron is a system with more than one perceptron and is also called a deep neural network. They have an input layer, an output layer, and one or more hidden layers. An example of a perceptron can be seen in Figure 2.2.

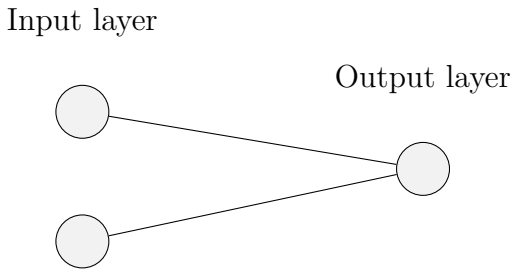


Figure 2.2: A perceptron.

Like neurons in a biological brain, these artificial neurons can transmit signals to other neurons. Connections between neurons are called edges, and they have associated weight. Increasing or decreasing the weight will affect the signal strength. A neuron will activate and pass on signals if the signals into the neuron exceed some threshold defined by each neuron. By assembling these neurons into layers and allowing signals to travel through the first layer, the input layer, to the last layer, the output layer, we create what is called a feed-forward neural network. The most significant property of a feed-forward neural network is that all information flows forward and no connections form loops. Figure 2.3 illustrates how a model of a feed-forward neural network with only dense layers could look.

There are many parameters to tune when designing a neural network, and we will not describe all of them, but three important parameters are layer activation functions, loss functions and optimizer. Activation functions define the output of a neuron, with the goal of introducing non-linearity into the output of a neuron, which is

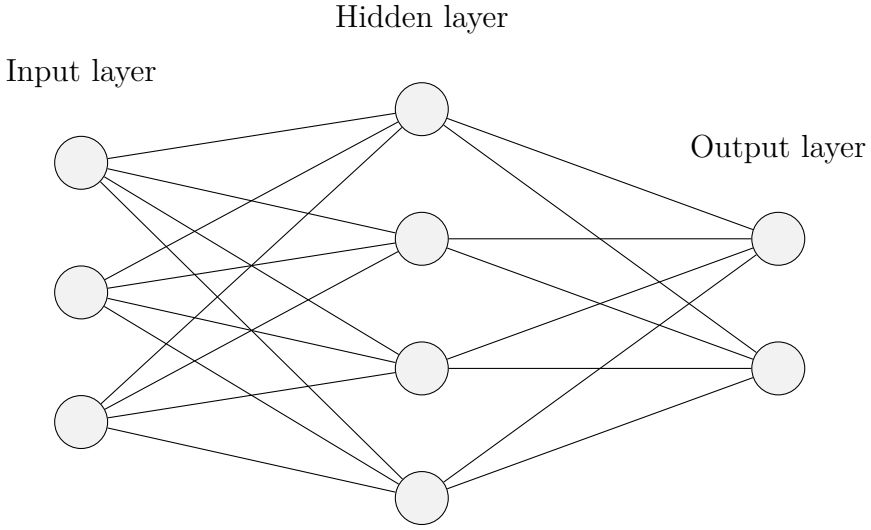


Figure 2.3: A fully dense neural network with three layers: one input layer, one hidden layer, and one output layer.

what allows neural networks to learn and perform more complex tasks. The loss function evaluates how well the algorithm does when it models the dataset; high loss corresponds to the algorithm not being able to model the dataset, whereas low loss indicates that it has found a good model for the dataset. Lastly, the optimizer defines how the model is updated in response to the output of the loss function. For further reading, we recommend the Keras documentation [8].

Supervised learning is a major learning paradigm in machine learning and corresponds to a particular learning process. Every data point is labeled before the training phase by either manual labor or by an automatic process. The goal of a supervised machine learning model is to successfully predict a label or value to a corresponding output.

When there is a large amount of unlabeled data and when labeling data manually might be expensive, one can make use of another machine learning concept called active learning. Active learning is an iterative supervised learning process that puts a user in the center: human in the loop [39]. The user has an active role wherein the algorithm can query the user to label new data through an interactive process. This can have both positive and negative effects as it can be an easy way to get labeled data, but it can also be a slow process. This is especially the case when labeling media such as music because the annotator can only listen to one melody at a time. This is quite dissimilar to how one could process a large amount of visual data at the same time as in Karl Sims project [37].

Lastly, Recurrent Neural Networks (RNNs) are derived from feed-forward neural networks. Unlike feed-forward neural networks, connections between nodes may form cyclic directed graphs, which allows some interesting dynamic behaviour. In this way, RNNs possess an internal state to store data which they can use to process

sequences of inputs of variable lengths. As such, this concept is very powerful for processing sequences of notes, volumes, and time relations to generate new data of the same kind. In our project, we have used a certain type of RNN that is called LSTM. Long Short-Term Memory (LSTM) is a type of RNN suited for classifying or processing time related sequences, proposed by Sepp Hochreiter and Jürgen Schmidhuber [22]. It was developed to avoid the vanishing gradient problem [21], which can occur in other types of RNNs. LSTM networks work by introducing different cells, enabling the network to decide which information it needs to forget, update, keep or output.

2.3 Music theory

An approach to automatic music generation is to implement rules based on music theory, as done by Lejaren and Hiller [20]. This project, however, takes another approach since it does not consider traditional music theory, which gives the algorithm more freedom. As a consequence, not much knowledge in music theory is needed to understand this project. In this section, the necessary terminology is introduced. For further reading see *Harmony* by Walter Piston [34].

Two fundamental elements of music are *pitch* and *duration*. The pitch refers to the perceived frequency of the sound waves and is the property of a note that makes it sound "high" or "low". Duration is the time the note sounds. In this project, duration is given in beats, or quarter notes, which is a unit of time. The length of a beat is determined by the tempo, which is given in beats per minute (BPM). An example of four quarter notes rising in pitch is seen in Figure 2.4.

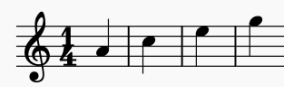


Figure 2.4: An example of notes rising in pitch.

A basic rendition of a music piece can be given as a series of notes with these two attributes. The notes can be played in sequence, as a *melody*, or simultaneously, as a *chord*. However, there are many additional properties that determine the character and nuances of a music piece. One of them is volume, and the variation of it, referred to as *dynamics*. Another is timbre or tone quality, which is the property that makes a violin sound different from a clarinet even though they play the same pitch.

Throughout this thesis, scientific pitch notation is used; for example C_4 where the letter indicates the pitch and the lowered number indicates the octave.

3

Method

The program is written in Python as the group had previous experience in the language. Python is one of the flagship languages in the area of machine learning which makes it a good choice to implement neural networks. This chapter presents the libraries used, how the fitness functions work, the implementation of the genetic representation and the genetic operators. The last part of this chapter describes how we tested different genetic operators and how it changed the performance of our generator.

3.1 Libraries and formats

Musical Instrument Digital Interface (MIDI) is a format for storing music, and it is the output format of our system. More specifically, it stores musical events as different MIDI messages. MIDI files do not store raw audio information, but rather instructions for creating a sound, somewhat analogous to sheet music. The actual sound then depends on what software is used to play the file. For live playback when running the code, we therefore use a Python package called PYO [4]. To read and write to MIDI files, the Mido [6] package was used, which includes functions for extracting and compiling information into MIDI files. Looking at Mido messages gives us a good idea of how the MIDI format works, and an example of what these messages might look like follows below.

```
track.append(Message('note_on', note=60, velocity=127, time=0))

track.append(Message('note_off', note=60, velocity=0, time=480))
```

The first message turns on the note with pitch 60, which correlates to the note C_4. `velocity=127` refers to how fast the note is struck, which effectively defines the volume of the note. The second message turns off the same note, and `time=480` means that it happens 480 ms after the previous event. In practice, this means that the note C_4 is struck with volume 127 and makes a sound for a duration of 480 ms. A more extensive guide on the MIDI format written by the MIDI Manufacturers Association can be found here [3].

The MAESTRO set [19] is used for calibrating and training the fitness function described later in Section 4.3.2. It contains a large set of recorded performances of piano pieces by classical composers such as Bach, Beethoven and Mozart in MIDI format.

To simplify the implementation of the neural networks, the libraries Keras [8] and Tensorflow [29] were used instead of implementing layers, activation functions and model pipelines from scratch, see Section 3.2. Lastly, the libraries Pandas [31] and NumPy [18] were used to decrease the run-time of the system, see Section 3.5.

3.2 Fitness

The objective of the fitness function is to evaluate each individual in every generation and give them a score reflecting how desired they are. Difficulties arise when evaluating music because the scores vary due to individual preference. We attempted to address this by using three different methods of rating the individual: sub-raters, interactive feedback and a neural network based on the interactive feedback.

The final fitness function uses a combination of the three methods where each method is used for a predetermined number of generations. Whenever a certain method is used for a number of generations, we call this a training cycle. For example, a sequence of three training cycles could be to first use the sub-raters for 10 generations, then interactive rating for 20 generations, and then the neural network for 10 generations. A sequence of training cycles is passed as an argument when running the generator.

3.2.1 Sub-raters

Our implementation of what we call sub-raters is inspired by previous works such as the work done by Jae Hun Jeong and Chang Wook Ahn [26] and a Bachelor's thesis with a similar purpose as ours [2]. These projects have been influential in what sub-raters we chose to implement. We started by implementing some of the sub-raters used in these projects and added a couple of our own sub-raters to solve problems we discovered along the way.

Every sub-rater measures a property of an individual, such as the number of unique pitches or the pitch range, and most sub-raters return a number between 0 and 1, depending on how prevalent the property is. It is important to note that this number is not the fitness of a piece of music, but rather a measurement of a property, and before the sub-raters can be used to determine the fitness of individuals, target values for each sub-rater need to be calibrated to determine if the presence of a feature is desired or not. These target values could be manually determined by a human, but we chose to calculate the scores of the sub-raters for already existing music and use the average of those scores as target values for each sub-rater. The fitness of an individual is then determined by measuring how close the scores from

each sub-rater are to the target values, according to Equation 3.1.

$$1 - \frac{\text{sum of the scores from each sub-rater}}{\text{number of sub-raters}} \tag{3.1}$$

All the scores from the sub-raters are then normalized into a single fitness score between 0 and 1. A fitness score close to 1 means that it is very similar to the training data and many sub-rater values are close to the target values. A fitness score closer to 0 means that the sub-rater values of that individual differ a lot from the target values. A short summary of each sub-rater implemented in this project follows below.

Neighboring pitch range: Calculates the ratio between the number of times neighboring pitch values are greater than 24 (two octaves), and the total number of notes. In Figure 3.1, this occurs three times and there are seven notes in total. Thus, the score from this sub-rater is $\frac{3}{7}$.

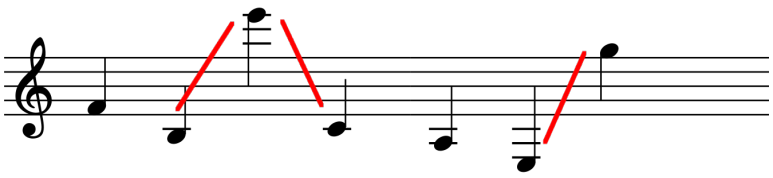


Figure 3.1: Example showing the sub-rater for neighboring pitch values. The lines show where the neighboring pitch exceeds two octaves.

Since large jumps in pitch may sound jarring or out of place, the purpose of this sub-rater is to limit the number of times that this occurs. However, if the training data contains many large jumps, the target value will instead learn to encourage more of this behaviour. As opposed to other methods, *e.g.*, calculating the average change in pitch between notes, we believe that this is less restrictive since it only considers big jumps in pitch.

Direction of melody: Calculates the ratio between the number of consecutive notes in the same direction after at least two notes have already been played in the same direction. When there are multiple notes playing at the same time, only the first note is included in the melody. In Figure 3.2, there are two instances of notes in the same direction, so the score from this sub-rater is $\frac{2}{9}$.

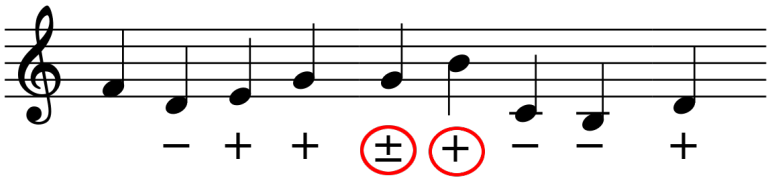


Figure 3.2: Example showing the sub-rater for direction of melody. The signs under each note indicate the direction of the melody, and the circled signs under the notes count towards the score.

Stability of melody: Calculates the ratio between the number of changes in direction and the total number of notes. When there are multiple notes playing at the same time, only the first note is included in the calculation. In Figure 3.3, you can see how the direction changes four times, so the score from this sub-rater is $\frac{4}{9}$.

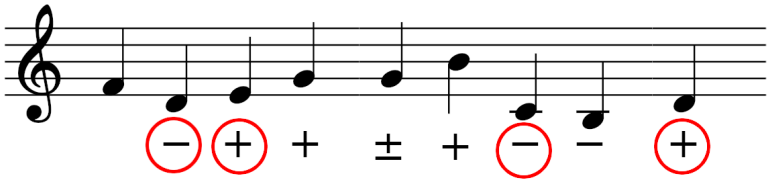


Figure 3.3: Example showing the sub-rater for stability of melody. The signs under each note indicate the direction of the melody, and the circled notes count towards the score.

Pitch range: Calculates the ratio between the minimum and maximum pitch value. We wanted to measure and control the size of the pitch range. It can however be noted that the sub-rater does not take into account the register of the pitch range. For example, if the lowest note is C_2, which has the MIDI pitch value 36, and the highest note is C_5, which has the MIDI pitch value 72, we get a score of 0.5. But if the lowest note is G_2, which has the MIDI pitch value 43, and the highest note is D_6, which has the MIDI value pitch 86, we also get a score of 0.5, even though the second range is wider and in a slightly higher register.

Unique notes: Calculates the ratio between the number of notes with a unique pitch and the total number of notes. This sub-rater ensures that there is a balance in the variety of pitches.

Equal consecutive notes: Calculates the ratio between the number of equal consecutive pitches and the total number of notes. Note that the first note in a sequence of equal consecutive notes does not count towards the score — otherwise we would

get a point for every note in the song. Many repeating pitches in a row can sound fatiguing, but some repetition may be desirable, so the purpose of this sub-rater is to balance the pitch repetition. In Figure 3.4, there are three instances of equal consecutive pitches, so the score from this sub-rater is $\frac{3}{9}$.

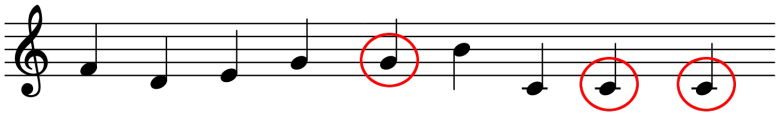


Figure 3.4: Example showing the sub-rater for equal consecutive notes. The circled notes count towards the score.

Unique rhythms: Calculates the ratio between the number of notes with unique durations and the total number of notes. If a piece has a small ratio of unique rhythms, there may not be enough rhythmic variation to make the piece interesting. On the other hand, if a piece contains a large ratio of unique rhythms, this might be because it contains many very long or very short notes, which may not be pleasant or interesting to listen to. This sub-rater tries to find a desirable balance between these two extremes.

Duration of melody and Total pitches: The first of these sub-raters calculate the duration of the piece in seconds, the second the total number of notes in an individual. The purpose of these sub-raters are to ensure that the final piece is long enough. Without these, the final piece may be only a few seconds long or may only contain one or two notes.

Number of chords: Counts the number of *chord events* and the total number of *events*, where an *event* refers to either a single note or a chord being played. The score is given by the ratio between chord events and all events. In Figure 3.5, there are three chords and a total of seven events, so the score from this sub-rater is $\frac{3}{7}$. This sub-rater ensures that the ratio of chords to single notes in the generated music is similar to the ratio in the training data.

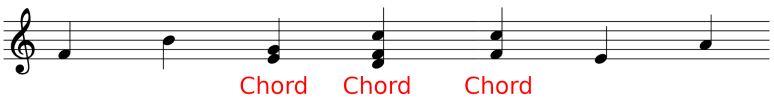


Figure 3.5: Example showing the sub-rater for the number of chords.

Stability of velocity: Takes the sum of the differences in velocity between the notes and divides it by the highest possible change in velocity between two notes, which is limited to 127 by the MIDI format. Then, it divides this by the number

of notes in the individual to get the average. The equation for this can be seen in 3.2. The purpose of this sub-rater is to create a similar sense of dynamics as in the training data.

$$\text{stability of velocity} = \frac{\text{total change in velocity}}{127 \cdot \text{number of notes}} \tag{3.2}$$

Note density: Calculates the density of notes in an individual, according to Equation 3.3. Regulating the note density controls how many notes are played per second, which can vary considerably between different types of music.

$$\text{note density} = \frac{\text{number of notes}}{\text{duration}} \tag{3.3}$$

Limit chord size: If a chord contains five notes or more, the number of notes past five notes counts towards a penalty (*e.g.* if a chord contains seven notes, the penalty goes up by two). The penalty is divided by the number of *events* (see *number of chords sub-rater* on page 13). The purpose of this sub-rater is to limit the number of chords contain five or more notes. But if the training data contains many chords that are larger than five notes, the generator will learn from that behaviour instead. An example can be seen in Figure 3.6, where there are two chords giving a total penalty of six points, and a total of eight events. The score is then $\frac{6}{8}$.

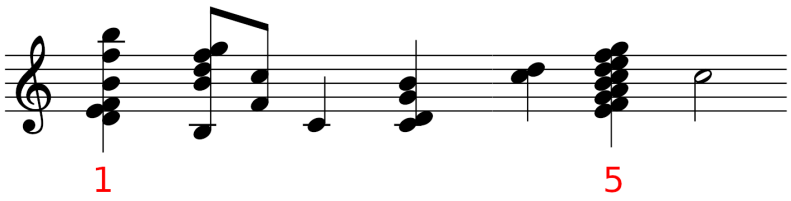


Figure 3.6: Example showing the sub-rater for limiting chord size.

3.2.2 Deep Neural Network for interactive and automatic rating

The sub-rater scoring method is limited since it can only assess the features looked for in the different sub-rater functions. There are still many features that are missing, but finding these features manually may be difficult for a human. A more automatic solution to feature extraction is using deep neural networks, see Section 2.2. A challenge with using NNs is that a label is needed to our training data. In our case, this means that to train the NN a data set of music pieces was needed and corresponding scores that indicate how desirable that piece is. Since this score is subjective, an interactive rating system was implemented.

This system allows a user to set the scores of the training data by listening to the individuals in a generation and scoring them between 0 and 100. The score is then normalized to a value between 0 and 1 before being passed to the neural network. To reduce the time it takes to score monotonous individuals, the user could enter a score before the individual has finished playing. After each generation, the neural network would retrain its weight using the newly scored individuals as part of the training data. This scored data set can also be saved and used as an initial data set later on.

Theoretically, all generations could be scored by a human, but this is time consuming and the changes from one generation to the next may be small. This is where the neural networks are used to automatically score a couple of generations between interactive scoring cycles. The network would learn how a user rates a typical melody to mimic that process. This allows the generator to run for more generations, which both creates more variation in the pieces the human has to listen to and also increases the likeliness of the generator to converge to a higher fitness score, see Section 3.5.

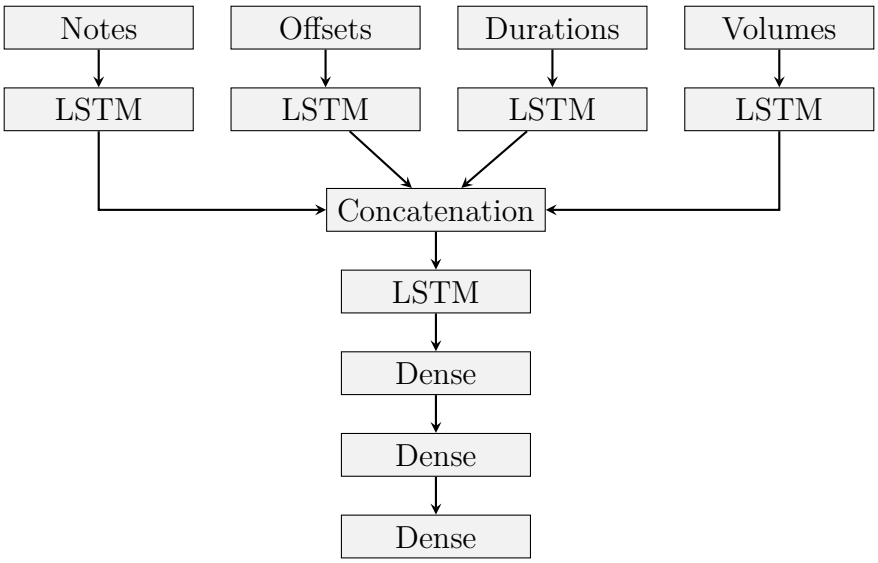


Figure 3.7: The layout of the neural network.

The model that is used is illustrated in Figure 3.7. We chose *mean-square-error* as our loss function and *adam* for our optimizer since these work well with regression tasks such as ours. The model starts off with four input layers that take in pitch, offset, duration and volume of all notes in an individual. These four layers were chosen since they make up the accessible information that can be attained from the MIDI format. Notably, the duration data is taken directly from the MIDI format, but calculated with the help of the offset data. Since the input layers need to be of the same size, 20 notes are read in at a time. If a piece is shorter than 20 notes, the piece is padded with zeros in all input layers. If a piece is at least 20 notes, the last input to the neural network is the last 20 notes of the song. For example, if a piece is 25 notes long, notes 1-20 would be the first inputs to the network and notes 5-25 would be the next and last input to the network for that individual.

All of the inputs would then get passed through individual LSTM layers with 16 nodes each, before being concatenated into one layer that gets passed through another LSTM layer with 100 nodes. The use of LSTM layers was mainly due to inspiration from Sigurður Skúli [36], and Jordan Bird [5] who used LSTM to "predict" what would be the next notes in a melody and give a score based on how close the generated results were to the predicted results. After the LSTM layer, we use batch normalization before two dense layers with 100 and 64 nodes, and ReLU as activation functions. The *batch normalization* normalizes the contribution of a layer in mini-batches. This reduces the number of training epochs required to train the network. Lastly, there is one more batch normalization before the output layer. The final layer is a dense layer with a *sigmoid activation function* that outputs a score between zero and one. Predictions closer to zero are considered to be worse.

3.3 How we represent music in code

When representing music in an EA, both the genotype and phenotype have to be considered. If we take humans as an example, the genotype refers to our DNA, whereas the phenotype refers to the human body. In the initial phase of this project, both the phenotype and the genotype were lists of note objects. Lists are easy to work with and intuitive to understand, but this meant that the genotype and phenotype were essentially the same. If there is a more complex relationship between genotype and phenotype, such as with human bodies and human DNA, a small change in the genome (the genotype for a specific individual) may cause bigger and potentially more interesting changes in the phenome (the phenotype for a specific individual). This is why we decided to use trees as our genotype, and this is what we refer to as our genetic representation. Another advantage with trees, compared to one-dimensional lists, is that trees have an inherent ability to contain structural information since parent nodes can pass information to their children. In this chapter we will explain how the phenotype and genotype for the individuals in our generator were designed.

3.3.1 Phenotype (Musical representation)

The phenotype in our case refers to how we store musical information that can be directly translated into a MIDI output. There are two parts to the musical representation we use. Firstly, how to represent the individual notes and, secondly, how to store all notes as a musical piece. Notes are represented by the class `Note`, which contains the attributes `pitch`, `duration`, `volume`, and `next_note`. Most of these are explained in Section 2.3 except for `next_note`, which is the time offset to the next note. For example, an offset of 0 means that the notes are played simultaneously, while an offset of 2 means that the next note is played two beats after the first note. A piece of music can simply be stored as a list of notes. When evaluating an individual, its genotype will be rendered into a list of `Note` objects, which is then passed to the fitness function.

3.3.2 Genotype (Genetic representation)

The genotype is what the generator uses to initialize a population, mutate individuals and perform crossover operations. To be able to get a comprehensive output, the genetic representation has to be "translated" into our phenotype representation. Our genetic representation stores information of a song as a binary tree. The root node contains a `Note` object, referred to as the root note, and all the other nodes contain one or more node functions. A node function typically contains a parameter that would change some aspect of a note. What node function or functions a node contains is randomized upon initialization or during the mutation stage.

To "translate", or render, the tree into a musical piece, the root note is passed down through all of the branches of the tree. Every time the note passes through a node, the node function of that node is applied to the note, changing what note the child nodes will receive. The notes that are passed through the tree are added in order to a list of notes. This means that every node corresponds to a note, however, this is further complicated by the introduction of more complex behaviours, such as the Repeat function, which may affect more than one note. A short description of all the node functions we implemented follows below:

- **Pitch**: Adds or subtracts a value to the pitch. The value is randomly generated from Gaussian distribution with $\mu = 0$ and $\sigma = 3$, rounded down to the closest integer. Due to the restrictions of the MIDI format, the pitch value is restricted to be between 0 and 127, or C_{-1} and G_9. The probability of this function occurring is 0.8.

- **Duration**: Multiplies the duration by a value. The value is 2^p, where p is randomly generated from Gaussian distribution with $\mu = 0$ and $\sigma = 1$, rounded down to the closest integer (so the duration can be either doubled or halved). The duration is restricted to be between 1/128 and 4 whole notes to make the pieces easier to listen to. The probability of this function occurring is 0.5.

- **Volume**: Adds or subtracts a value to the volume. The value is randomly generated from Gaussian distribution with $\mu = 0$ and $\sigma = 5$, rounded down to the closest integer. Due to limitations of the MIDI format, the volume is restricted to be between 0 and 127. The probability of this function occurring is 0.5.

- **NextNote**: Specifies an offset to the next note. If this function is not present, the default is to set the offset to be the duration of the current note (*i.e.* the next note is played when the previous note has finished playing). The parameter is 2^p, where p is randomly generated from Gaussian distribution with $\mu = 0$ and $\sigma = 1$, rounded down to the closest integer. The probability of this function occurring is 0.3.

- **Repeat**: This node function causes all of its children to be repeated once or twice when translating the tree. The number of repetitions is selected with equal probability. The probability of this function occurring is 0.1.

- **RepeatInverse**: Works similarly to the *repeat* node function, but the children are only repeated once and the order of the repeating children is reversed. The probability of this function occurring is 0.1.

- **Dynamics**: Changes the volume or velocity of all children of that node linearly (results in a crescendo or diminuendo). The increment or decrement is a multiple of 10 between -30 and 30, excluding 0, chosen with uniform distribution. The probability of this function occurring is 0.1.

The size of a tree is not given at initialization. In order to generate a new individual, we start with a root node with random node functions. The probability that a node generates children decreases in proportion to the size of the tree.

An example of a genetic representation of an individual can be seen in Figure 3.8, which translates to the musical fragment shown in Figure 3.9. Note that the tree only contains nodes with a single function, however, it is possible for a node to contain multiple functions. Note also that Figure 3.9 does not show the volume of each note, but the volume is stored in our final output.

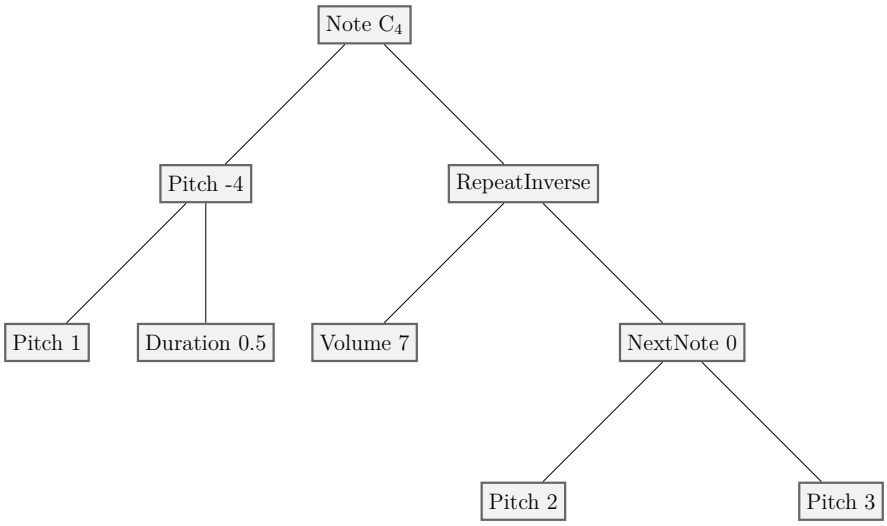


Figure 3.8: An example of how the genetic representation of an individual may look. The text in each node indicate what note or node function that node stores and the value stored for that function.

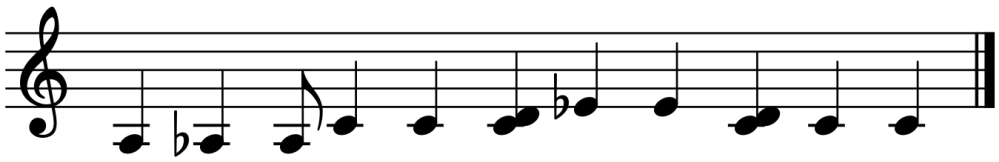


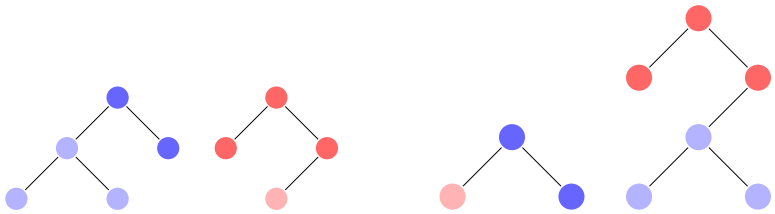
Figure 3.9: Sheet music illustrating how the genetic representation in Figure 3.8 translates to an actual piece of music.

3.4 Genetic operators

There are several ways of implementing the genetic operators, and in this section we describe the ones we have implemented in detail.

3.4.1 Crossover

Crossover is used to change the order of features by combining features from two parents. Two types of crossover are used: single point and uniform crossover. In both techniques, two parents are used to create two mirrored children so that no information of the parents are lost. In single point crossover, the children are created by swapping a random node from each parent. An example can be seen in Figures 3.10a and 3.10b.



(a) Parents. **(b)** Children.

Figure 3.10: An example of single point crossover.

Uniform crossover can be thought of as a single point crossover, but with a probability of happening for every common node. In uniform crossover we "toss a coin" for each common node of the parents, and if we get heads we perform a single point crossover in that node. We define common nodes as a node position that occurs in both trees.

3.4.2 Mutation

Mutations are what introduces new behaviour that was not present in the initial population. Once in every generation, for every individual, there is a probability that it will be mutated, *i.e.* some aspect of its genetic material is changed. This introduces new features or values in the population. In our project we implemented the following mutations for our genetic representation:

- **Change root note:** Changes the note in the root node of the tree, effectively transposing the whole piece or changing the tempo or volume.
- **Swap functions:** Randomly selects two different nodes in the tree and swaps their node functions.
- **Mutate function:** Selects a node in the tree and generates a new node function and for that node.
- **Mutate parameter:** Generates a new parameter for a function of a node, without changing the function type.
- **New tree branch:** Selects a node in the tree and replaces it with a newly generated sub tree.

3.4.3 Parent selection

We implemented a number of different methods in our project. It is possible to select which generator to use by passing it as an argument when running the generator. A comparison of the effect these different methods had on our generator is given in Section 4.1. The parent selection methods used are described in detail below.

Tournament selection: Randomly select a portion of the population. Pick the individual with the highest fitness out of these. An example is shown in Figure 3.11

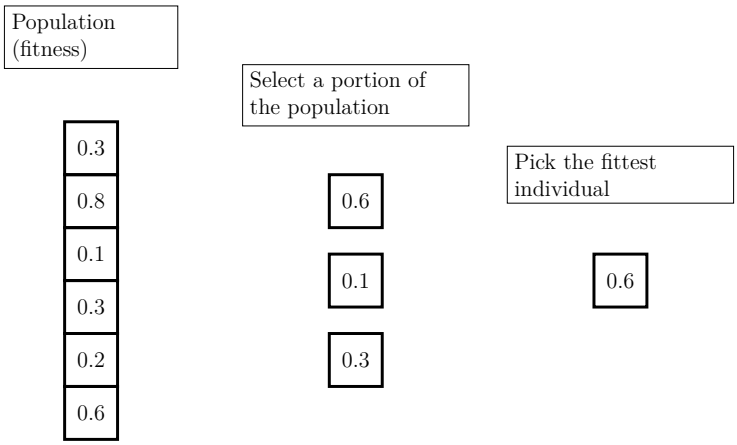


Figure 3.11: An illustration of how tournament selection might look like.

Roulette wheel selection: Let the fitness score of each individual represent a probability of being picked. This can be imagined as a roulette wheel where each partition represents an individual and the size of each partition is proportional to the fitness of the individuals, as seen in Figure 3.12. The wheel is then “spun" once to select one individual.

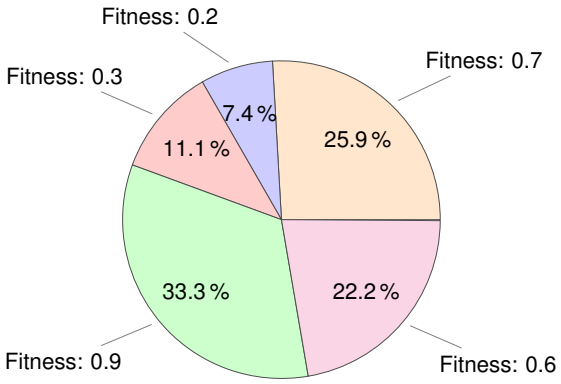


Figure 3.12: Visualization of probabilities in roulette wheel selection.

Stochastic universal sampling: This is similar to roulette wheel selection, but instead we pick two parents from one spin of the wheel. This is useful to ensure that different individuals get selected.

Rank selection: Order the population in an array based on their fitness. Give each individual a rank, given by its position in the array. Assign the probability of each individual being picked for recombination depending on its rank. It can also be thought of as roulette wheel selection, but basing the probabilities on the rank instead of the fitness; compare Figure 3.12 and 3.13. Rank selection is useful when the fitness scores are very similar. In roulette selection, if the fitness is almost the same for all individuals, it becomes a uniform distribution chance.

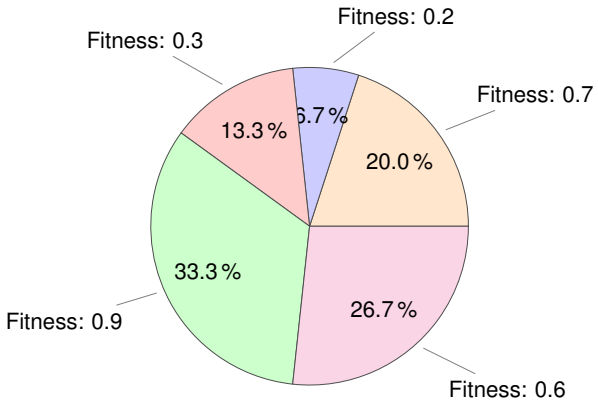


Figure 3.13: Visualization of probabilities in rank selection as a pie chart.

3.4.4 Survivor selection

In addition to the generated children, some individuals may be carried over to the next generation unmodified. This ensures that not all data from the previous generation was lost due to a sudden high rate of mutations. When choosing survivors, it is important to choose some of the best individuals so that the data being saved contains value. Saving the individuals with the highest fitness score in a population is called *elitism*. Another common method is to use tournament selection which also ensures that the individuals selected have some value. As with parent selection, the choice of the survivor selection method can be passed as an argument to the generator and a comparison of these methods can be seen in Section 4.1.

3.5 Testing different genetic operators

Previous works using EAs for music generation that we have encountered have not compared the performance of different genetic operators; see Section 3.4. We wanted to test how different genetic operators would affect the system and how they would work with our genetic representation. Therefore, the generator was designed to make it easy to switch between different genetic operators. To be able to do more complex comparisons, we first needed to reduce the run-time of the generator.

3.5.1 Reducing run-time

As with most machine learning models, EAs tend to converge to a higher fitness score if the size or complexity of the model is increased, and we let the model run for a longer time. In our model, this would correspond to increasing the population size or the number of generations. These allow the algorithm to explore a larger part of the solution space and increase the probability of finding an optimal solution. But increasing these parameters also increases the number of operations the generator has to perform. To reduce the run-time we have used the libraries Pandas [31] and NumPy [18], which provide more efficient alternatives to Python's built-in list operations. We also replaced the built-in copy functions with custom functions for some of our classes. Additionally, the genetic representation was designed to minimize tree traversals by storing a list of its nodes in the class directly, updated only when necessary. This reduced the run-time of operations such as randomly choosing nodes from the tree.

3.5.2 Diversity

Diversity refers to how different individuals are to each other within a population, and it is essential to keep a high level of diversity to prevent the generator from converging prematurely. If this happens, the generator may not explore a large enough part of the solution space, which may give us a non-optimal solution and reduce the learning rate of the generator. In this project we also want variation in our results to get more out of the interactive evaluation, so that a user will not have to listen to and compare many similar individuals. As such, maintaining diversity

is essential. This can be achieved in a number of ways. Different crossover functions, parent generators and survivor selections encourage the generator to converge slower or faster by increasing or reducing diversity. But there is usually a trade-off between maintaining diversity and increasing learning rate since random behaviour can counteract the learning process. Other aspects of the generator also affect diversity; mainly the distribution of the randomly generated parameters and the fitness function, but these will not be discussed in this report.

Since we could not find an established standard for measuring diversity of music pieces, we have attempted to find our own means of measuring diversity by comparing the best and worst fitness scores in each generation. We found this to be a rudimentary but not computationally intensive way to measure diversity. In Figure 4.1, the bottom graph shows how this difference changes depending on the parent generator used. We can see that the difference does not seem to reduce over time. This is good since we want to maintain diversity even in our final population. We can also see that the range varies for the different parent generators. We have tried to take diversity into account when selecting optimal genetic operators, but since our diversity measure is rather ambiguous, it has not been the sole deciding factor.

3.5.3 Optimizing performance

When optimizing the performance, the goal is to improve the highest achieved fitness score as well as the learning rate for the generator. By improving the learning rate, the time it takes for the generator to converge is also reduced. To get a more objective measure of the performance, we use a custom fitness function that tests if the generator can mimic a predetermined piece, in this case a C major scale, instead of the ordinary fitness functions. This is done to ensure that the full range of scores from 0 to 1 are possible, which may not be the case for our sub-rater and neural network due to trade-offs between some aspects of the music.

When trying to find the optimal genetic operator, we have compared different alternatives sequentially. This may cause problems if some of the genetic operators are greatly affected by one another. A more thorough method of testing would have been a grid search [27], but with limited time and computational power, we decided that sequential testing was a more feasible option.

4

Results

The project is evaluated in three main areas: the performance of the generator and its different genetic operators, the effectiveness of the fitness functions, and the musical potential of the genetic representation. The genetic representation is evaluated by a survey and the rest by running tests and measuring the performance of the generator.

4.1 Performance of the generator

The performance of the generator is mainly measured by how good the generator is at attaining a high fitness score. This refers to both best and worst case scenarios, as well as how quickly it seems to converge. The following tests aim to get a good grasp of the performance of the generator, and we will also find an optimal set of genetic operators. In the tests, the fitness function described in Section 3.5.3 is used.

In Figure 4.1 we can see a comparison of the different parent selection methods described in Section 3.4. Note that the purple curve labelled *parent generator selection* is a random selection, each generation, from all parent selection methods. Since there are many random variables in the generator, each data point is produced by taking the average value from five identically initiated generators. This reduces the randomness in the output, and the results become more robust.

From this we can infer that tournament selection has the highest maximum fitness, learning rate and diversity and can therefore be considered the optimal parent selection method. We can also see that the generator seems to converge to a maximum fitness score at around 30 generations, which allowed us to test a lower number of generations in our following tests.

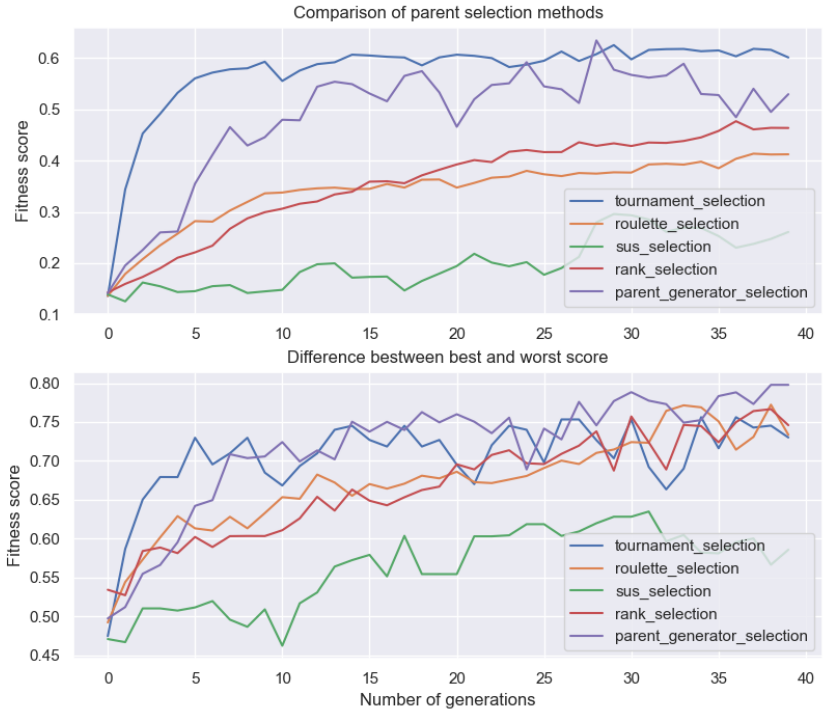
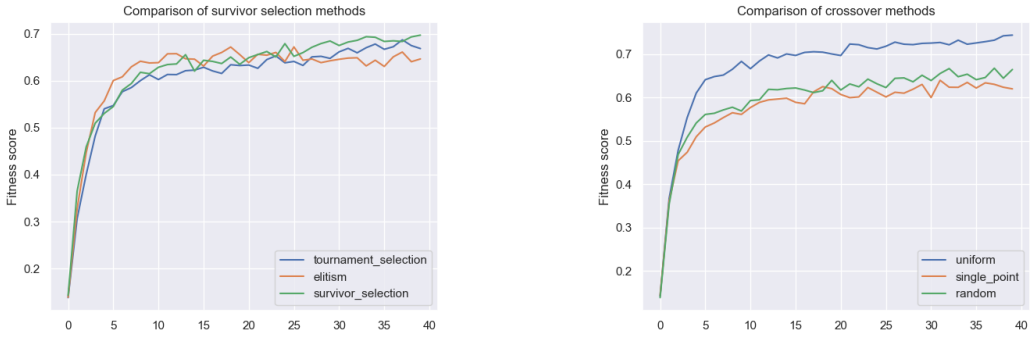


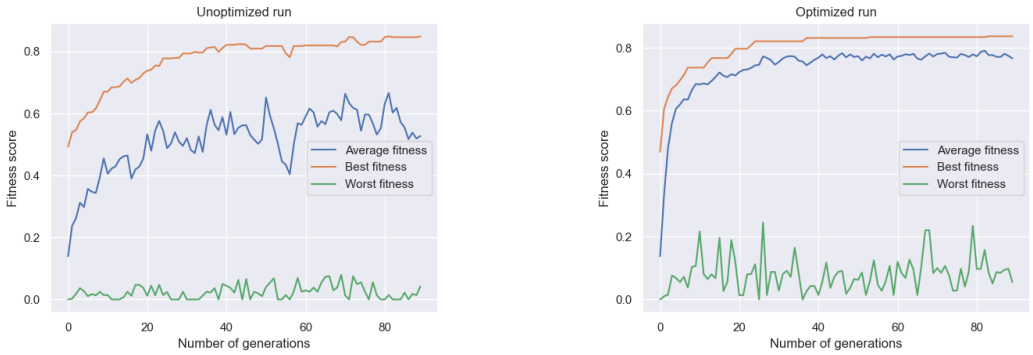
Figure 4.1: Comparisons of average fitness for different parent selection methods



(a) Comparison survivor selection **(b)** Comparison of crossover

Figure 4.2: Comparison of different survivor selection methods and crossover methods. The crossover test was done using random survivor selection.

In Figure 4.2a we see a comparison of the survivor selection methods after 15 runs. Note that the *random* selection has a random chance each generation of using either tournament or elitism. As can be seen in Figure 4.2a, the selection methods seem to perform equally well. Since the results are not decisive, we chose to use the *random selection* for the optimal generator to avoid possible undetected bottlenecks by using only one of them. Finally, in Figure 4.2b we can see that *uniform crossover* significantly outperformed both *single point crossover* and *random selection*.



(a) Unoptimized generator **(b)** Optimized generator

Figure 4.3: A generator using random selection of genetic operators in (a), compared to a generator using the genetic operators found to be optimal in (b). Both generators use the fitness function described in Section 3.5.

A comparison of the optimized generator and a generator using randomized selection can be seen in Figure 4.2. The optimized generator uses *tournament selection* for parent selection, *random selection* for survivor selection and *uniform crossover*. Both learning rate and average fitness increase significantly without any major loss of diversity. It can be noted that the maximum fitness score is still not optimal.

Figure 4.4: Fitness of population over time using the optimal genetic operations and sub-rater fitness.

Figure 4.4 shows a similar comparison but using our sub-rater fitness function. Notably, the range of fitness values between the best and worst runs in regard to fitness is reduced. It takes about 15 generations for the generator to converge when using a population size of 100. The generator converges to a fitness of around 0.85.

4.2 Questionnaire

The musical potential of the genetic representation was evaluated with a survey. The survey was distributed through social media by the members of the group and our supervisor, as well as some music students. By including quantitative questions on musical experience and opinions on automatically generated music we were able to analyze the answers based on the respondents' experience with music and attitude towards automatic composition.

The main part of the survey consisted of listening to six pieces selected by us from 60 randomly generated samples. These pieces were generated using only the genetic representation and were not trained or evolved at all. We selected pieces that we thought had different musical qualities. Most of the questions were qualitative and kept open to encourage the respondents to comment on what they found interesting or relevant. We kept all information correlated to the pieces such as names and images as neutral as possible as not to influence the respondents' opinions. The full questionnaire is included in Appendix D.

We got 57 replies, but during preprocessing we deemed two of these answers unusable due to misunderstanding of the questions and not answering the survey seriously. The distribution of the quantitative answers can be seen in Appendix B.

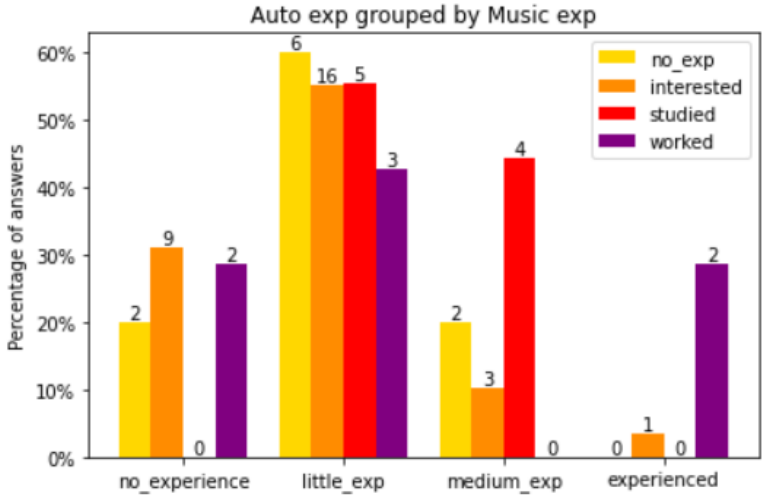


Figure 4.5: Bar graph of experience with automatically generated music, grouped by previous music experience. The y-axis indicates what percentage of the music experience group the bar makes up. The numbers above each bar indicates the number of answer that bar represents. A darker color represents a more musically experienced subgroup.

From the distributions in Figure 4.5 we can determine that most respondents had little to no experience in music and little to no experience with automatically generated music. We can also see that these groups are correlated, and respondents with little to no music experience tend to also have little experience with automatically generated music. Music students tend to have **some experience** and respondents who work in music have **varying experience** with automatically generated music. From this, we can determine that music students tend to be the most experienced group within both fields.

From the responses, we can also determine that the answers for the favourite piece are almost uniformly distributed across all pieces. There is a slight tendency to prefer any of the first three, but since the order of the pieces was fixed and not random, this could be explained by question order bias. When grouping these responses, we found that the piece a respondent selected as most interesting did not have any strong correlation to what previous experience they had.

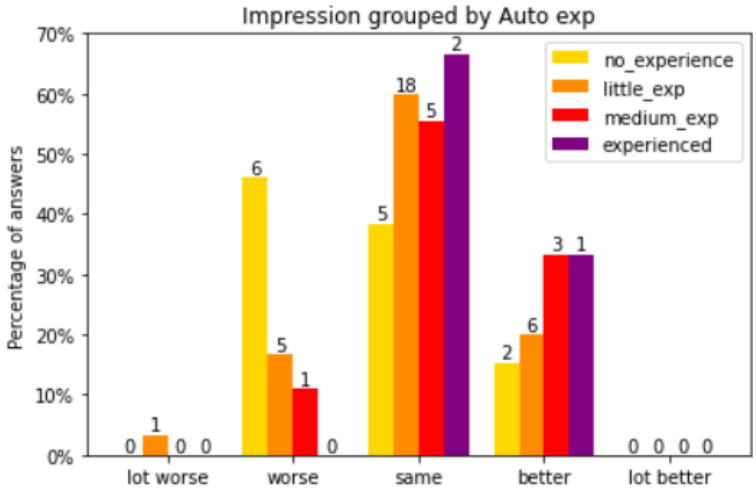


Figure 4.6: Overall impression of the generator grouped by experience with automatically generated music

In Figure 4.6 we can see that respondents who have at least some experience with automatically generated music tend to appreciate the samples more than those with no previous experience. We did not find any strong correlation between music experience and the overall impression of the generator.

The qualitative answers were grouped by music experience when analyzing to see if there would be any correlation between answers. Most respondents were interested in the possibilities of automatically generated music, but some were concerned about the impact it would have on the music industry. More experienced musicians tended to focus on the possibilities of automated music being used as a tool in music composition, and it was not generally believed that it poses any threat to human composition in the music industry. The opinions of the individual pieces were mixed within and across all experience groups. In all experience groups the respondents mentioned feelings and associations arising from listening to the music. Most com-

monly mentioned were jazz, movie music, and thematic music in general. Many respondents commented on an uncomfortable or scary feeling in the music, and one of them suggested that this may be because the generator creates dissonance, which is used to build tension in music.

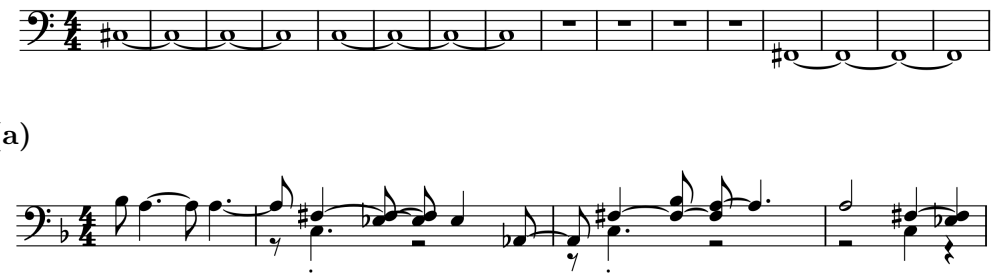
The qualitative answers of respondents who did not like the music or did not like a particular piece often indicated that the pieces did not sound like something a human would compose. The respondents were also negative to rapid changes or breaks in style or melody. Overall, there were many comments on fragments that people found good or interesting, but they considered the pieces unfinished and said that they lacked direction and structure. Rhythm and dynamics were frequently mentioned as points of improvement.

4.3 Fitness

As the fitness functions were not evaluated in the survey that was sent out, they were considered and evaluated individually.

4.3.1 Sub-rater

To evaluate the functionality of the sub-raters, we consider each sub-rater in isolation, as the resulting melody can be presented without the disturbance of the other sub-raters. We think that evaluating three different sub-raters is enough to get a good grasp of the whole sub-rater system. The three sub-raters — *note density*, *direction of melody* and *equal consecutive notes* — are chosen because of the distinct structure each of them may create. Each sub-rater test contains two pieces which were generated with different target values of each sub-rater. Scores of the resulting pieces are shown in Figure 4.7, Figure 4.8 and Figure 4.9.



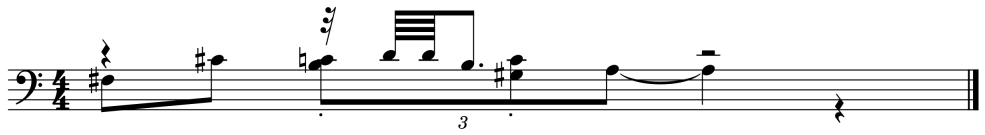
(a)

(b)

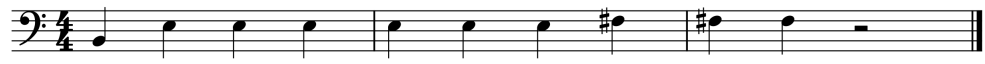
Figure 4.7: Examples of generated pieces using only the sub-rater *note density*. In (a) the target value is 0 — as low density as possible, while in (b) the target value is 1 — as high note density as possible.

However, it is not completely true that each test is evaluated in isolation, but rather each sub-rater in combination with the sub-rater *total pitches*. This is to avoid the tendency for the genetic algorithm to convert to just one note, which was a

recurring theme without *total pitches*. This is probably because the presence of more notes introduces more "noise", and the algorithm will tend to go for the simpler outcome.



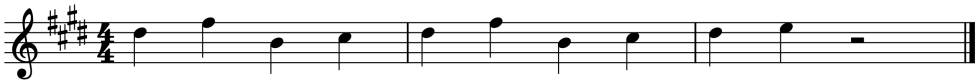
(a)



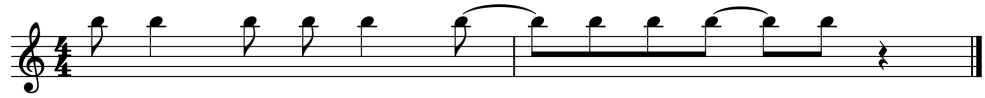
(b)

Figure 4.8: Examples of generated pieces using only the sub-rater *direction of melody*. In (a) the target value is 0, which means that the melodies will oscillate more between each pitch. In (b) the target value is 1, which means that the melody will have a few changes in direction.

It might be difficult to verify the independence of each combination of sub-raters, but intuition tells us that *total pitches* should be one of the most independent sub-raters in combination with any of the others because it does not directly modify the structure of the melody, rather the length. Consequently, the combination should not influence the results negatively.



(a)



(b)

Figure 4.9: Examples of generated pieces using only the sub-rater *equal consecutive notes*. In (a) the target value is 0, which means that the melody will avoid consecutive notes with the same pitch. In (b) the target value is 1, which means that the melody tries to have as many consecutive notes with the same pitch as possible.

These figures show that the interaction between the generator and the sub-rater successfully works. The generated music sheets correspond to a structure that seems reasonable given the intended functionality of each sub-rater.

4.3.2 Automatic rating

We constructed three different test cases: Trial 1, Trial 2 and Trial 3. Each trial uses parts of the MAESTRO set [19]. The first two trials attempt to evaluate how the neural network performs given a standardized and earlier used dataset with a lot of data already preprocessed. Trial 3 combines the MAESTRO set with pieces generated by our genetic algorithm. The MAESTRO set contains a myriad of different piano pieces by composers such as Bach, Mozart and Beethoven and some useful metadata. The integration of the dataset with our neural network model is done by giving each track a score on the rating scale 0.0-1.0. Each score was dependent on the composer rather than the track itself, to give every track of the same composer the same score. The results of each trial can be viewed in Table 4.1.

Trial 1 could be looked at as a binary classification (with some minor detours). Each piece in the MAESTRO set is scored with a value of 0.0, except pieces by Bach which are scored with a value of 1.0. As Table 4.1 shows, the accuracy is quite high even for the baseline. This has to do with the quantity of tracks rated zero ($\sim$ 90%) in proportion to the tracks rated one ($\sim$ 10%) and the fact that the baseline will always score zero no matter what input it processes. Our model performs better than the baseline, around 7 percentage points and has an accuracy of 96.83 %.

Test	Accuracy	Baseline	Baseline modifier	Threshold
1	96.83 %	89.68 %	0.0	0.5
2	84.13 %	51.69 %	0.3	0.2
3	86.80 %	13.20 %	0.5	0.2

Table 4.1: The baseline always predicts the value in the baseline modifier column, and thus acts as a dummy predictor. The threshold is the acceptance of the predicted value, *e.g.* if the label is 0.5 and the predicted score is 0.7 is is considered a valid prediction if the threshold value is greater than 0.2 ($|\text{score} - \text{label}| < \text{threshold}$).

Trial 2 is similar to Trial 1 in the sense that we still use the MAESTRO set and assign Bach pieces a score of 1.0. However, for every other composer we instead create a mapping of that composer to a random score between 0.0-0.6. This results in every musical piece composed by the same composer receiving the same score. This approach assumes that there is some inherent relation with the mapping of a musical piece and its composer, and the same goes for Trial 1. It is a reasonable assumption since each composer tends to have their own style and an observer might score a musical piece based on its composer. The baseline works a little differently than in Trial 1. Instead of always assigning 0.0 to every musical piece, it assigns 0.3 since that is the mean value of the allowed range for the random score — maximizing the range of valid predictions.

Test	Accuracy (only ones)	Threshold
1	69.23 %	0.5
2	7.14 %	0.2
3	100.00 %	0.2

Table 4.2: This accuracy only considers labels equal to a value of one with the same scoring technique used in Table 4.1.

Trial 3 attempts to combine the data of the MAESTRO set with the data generated by our genetic algorithm to mimic how it will work in practice. Despite the substantial difference in the data size for Trial 3 in comparison with Trial 1 and Trial 2, the results look promising with an accuracy of 80.80 %. Although, one interesting observation is that the model seems to predict every 1.0 correct with an accuracy of 100 % suggesting that there is some unwanted bias, see Table 4.2.

We can conclude that the best accuracy of 96.83 % is achieved in Trial 1. However, compared to the baseline, Trial 3 seems to have the highest overall score where the difference of the percentage between the accuracy and the baseline is 73.6 %. Although, as table 4.2 shows, this may be the cause of some bias, as the model has an accuracy of 100 % when predicting the melodies of Bach.

4.3.3 Interactive rating

Due to the way the interactive fitness process works, it can be difficult to verify whether it works correctly or not because of the subjective properties it has. However, one attempt would be to deduce its performance based on the results of the automatic fitness function and argue by induction that the model will adapt with better performance as the dataset increases. One way to construct a test is to retrain the neural network after an observer has manually scored a set of generated melodies multiple times and plot the accuracy and the size of the dataset. Figure 4.10 depicts the result of such a test. The accuracy of the model seems to gradually worsen the more data points the model is trained with. The accuracy seems to peak at 88.52 % at 19129 data points.

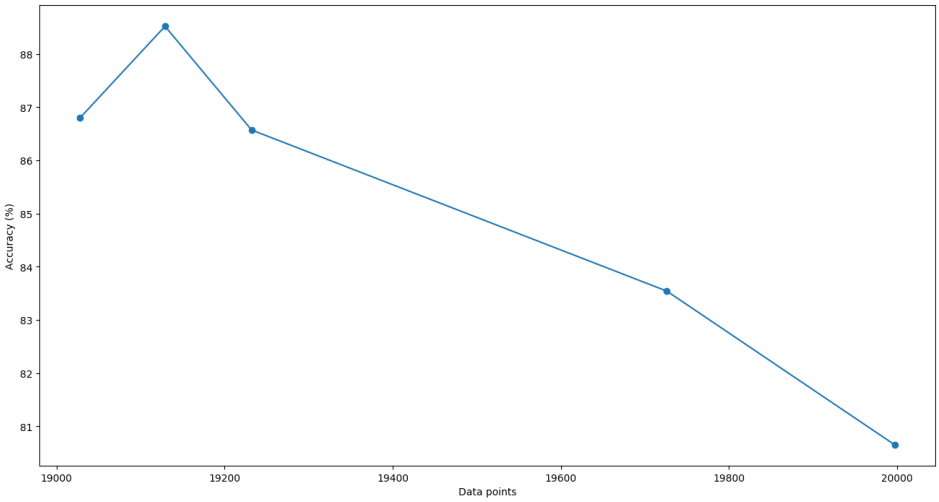


Figure 4.10: The accuracy of the neural network with incremental new data points.

One note should be made regarding the collection of new data points, that is the short time interval in which the data were acquired, the generator's inherent ability to create very similar pieces in each generation and the repetitiveness of the task. This is further investigated in Section 5.2.2

4.4 Functional tests and overall performance

To see how the fitness functions interact with each other, the generator was set to use the combination configuration, which allows for switching between fitness functions after each generation. A sequence cycle that switches between automatic fitness and sub-rater every fifth generation was used for a total of 100 generations. The sub-rater was calibrated on the MAESTRO set, using only the portion from the year 2006. The automatic fitness used the trained model from Trial 2 in Section 4.3.2. As with the interactive fitness evaluation, the end result can be difficult to evaluate, but since we use the MAESTRO set for both fitness functions, we can compare our results and check for similarities. The scores shown in Figures 4.11 and 4.12 were rendered automatically from MIDI files with Musescore [33], which explains the difference between the score shown in Figure 4.11 and the original score.

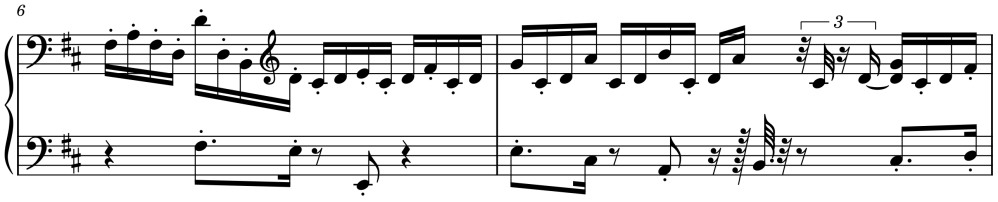
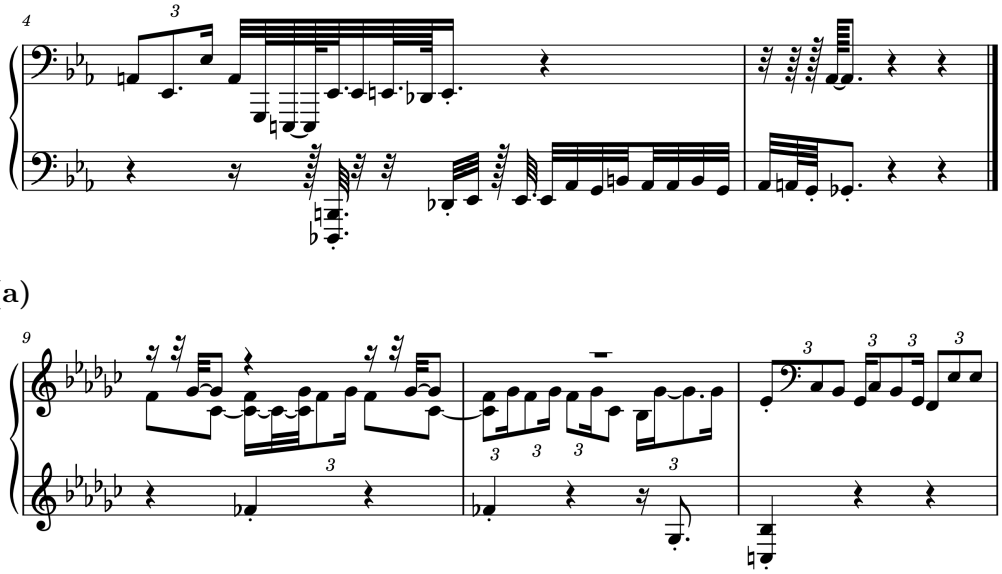


Figure 4.11: Excerpt from Prelude in G Major, WTC I, BWV 860. Score automatically generated from one of the MAESTRO MIDI-files.

An important thing to note is that the piece shown in Figure 4.11 is only a small excerpt from a randomly chosen piece by Bach. However, Bach has an easily identifiable style [16], so we feel that there is at least some relevance to this comparison. The other pieces in Figure 4.12 are also excerpts from longer pieces, but seem to be good representations of the pieces as a whole.

Without delving too deep into music theory, there are some things that are easily spotted, perhaps the most obvious being the presence of two voices. While the pieces are short, making it difficult to make any statement about counterpoint melodies, it does look like the piece in 4.12b in particular has a similar structure of a top and bottom melody playing at the same time, whereas the piece in 4.12a rather has two melodies played sequentially. Additionally, the Bach piece contains some degree of repetition, which both of the generated pieces also possess.



(a)

(b)

Figure 4.12: Excerpts from two generated melodies.

A more straightforward comparison could be to compare some of the properties that the sub-raters measure. It is possible to see that some sub-rater values are similar across the pieces. For example, they all seem to have a similar *stability of melody*, as all of them are changing direction often. The size of their pitch ranges are also relatively similar to the Bach piece (10 for the Bach piece and 16 and 15 respectively, and for the *pitch range* sub-rater, as described in Section 3.2.1, 0.645, 0.490, and 0.711 respectively), in particular the piece in 4.12b, and they have about the same number of unique rhythms. More interesting, perhaps, is that their note densities are very similar. The note density is 5.125 notes/beat for the Bach piece, 4.388 notes/beat for the piece in 4.12a, and 3.889 notes/beat for the piece in 4.12b. However, while the scores are relatively similar, for the piece in 4.12a, it may not have worked as intended. The Bach piece is very consistent and has about the same note density across the whole piece, whereas the piece in 4.12a mainly has a large spurt in the middle, leaving two beats completely quiet at the end. Then again, it is important to note that the sub-raters were trained on the whole of the 2006 dataset, not just compositions by Bach, and only used every fifth generation. As such, their impact on the final result should be smaller and considered less important than that of the automatic fitness. The conclusion is that the generated pieces share some similarities with the Bach piece, but the results are varying.

5

Discussion

This project explores many different aspects of music generation. In this chapter, we discuss the results of using our genetic representation and fitness functions. Additionally, we discuss the potential uses of this type of program, as well as the ethical implications of using artificial intelligence in a creative field.

5.1 Genetic representation

What is the DNA of music? The choice of genetic representation of the music greatly affects the evolution. The early prototype of this project uses a list of notes as the genetic representation of music. While this is a sufficient and intuitive way of storing musical information, it does not contain any structural information. The tree, as used by Sims [37] and Dahlstedt [10], is an interesting alternative. With trees, the development process becomes more complex, and the genetic operators, while simple, have more potential to bring interesting variation in the music.

Interesting variation is also brought by our use of relative node functions, described in Section 3.3. Music is in many ways relative: if a piece is transposed from one key to another, all pitches stay the same relative to each other. A simple operation such as changing the pitch parameter in a node function then results in transposing a section of the music.

Other examples of musically interesting node functions are the repetition and inversion functions. Many of the generated pieces contain repeated passages, which create a sense of cohesion and opportunities for motifs to appear. An example of repetitive elements can be seen in Piece 4 from the survey, which is included in Appendix D. Its use of repetition is mentioned by survey respondents as a reason why they chose this piece as their favourite. One says: *"It was more cohesive than the other pieces, with similar elements that appeared several times."* and another says: *"It was the most well thought out piece and had repetitive elements"*. However, the musical structures achieved by our genotype are often too elementary on a larger scale. This is also mentioned by several respondents. Many of them complain about the lack of direction in the music. A respondent says *"In general I think the parts, while often okay by themselves, don't really seem to belong together."*, while another

draws parallels to text generation: *"Just like with generated text, the music lacks structure thus meaning, thus always on the border of nonsensical"*.

Note that the pieces used in the questionnaire were not evolved but randomly generated individuals. Some of the respondents' suggested areas of improvement, such as harmony, rhythm and dynamics, could be achieved by fine-tuning the fitness function instead of constraining the genetic representation. By not constraining the genotype, the algorithm is given more freedom since the search-space is kept relatively large. This improves the probability of finding musically interesting and original pieces.

However, the genotype is not completely unconstrained. We chose appropriate probabilities for each node function to appear, as well as appropriate distributions for their parameters. It would be interesting to include these parameters as evolvable traits instead of using fixed values. To integrate the parameters into the learning process, one could use another lifetime model for the EA such as a Baldwinian model or Lamarckian model [38]. This may be a solution to find hyperparameters that produce more musical value instead of only optimizing the performance of the algorithm.

5.2 Fitness

With a relatively unconstrained genotype, a suitable fitness function is essential. The fact that the fitness of music is highly subjective complicates it further. This section discusses the performance of the fitness functions used in this project.

5.2.1 Sub-rater

The results discussed in section 4.3.1 show that the sub-rater system performs well. It is good at filtering out pieces that are too far away from what we are interested in, *e.g.* pieces shorter than a few seconds or long pieces with only a few long notes. It is also responsive to the training data, and as the results show, changing the target value for the sub-raters results in a noticeable difference in the evolved pieces.

The sub-rater system was introduced to the program relatively early, before we had finalized the genetic representation, so the sub-raters were designed to work within the limitations of the less advanced genetic representation used at the time. Initially, we decided to put our focus on other rating methods and did not expand or update the sub-raters more than necessary. Perhaps most importantly, this means that many of the sub-raters are not well suited to notes playing simultaneously, which was something we noticed late in the project. An example of this is the *direction of melody* sub-rater, which may give an unexpected score to pieces with chords or several voices.

Another flaw of the sub-raters is that they only measure some aspects of a music piece and do not take the piece as a whole into consideration. Some short pieces can sometimes get an undeserved high rating because of the way that the sub-raters are implemented. The sub-raters could also have been designed from a more statistical perspective; there are many different ways of measuring the same property that may give vastly different results. Another area of improvement is the weight or bias for the different sub-raters. As of now, each sub-rater influences the fitness score with the same weight. An idea for further improvement could be to give each sub-rater individual weight.

Technical details aside, the sub-rater system also has aesthetic challenges. In some sense, comparing very broad aspects of the music to a target score is a way to guide the algorithm with existing music without explicitly imitating it. It might result in new and original music while still conforming to some norm. On the other hand, the sub-rater system does not consider aspects such as harmony or form. Even if it did conform to traditional rules found in music theory, there are still aspects of music that are hard to measure. This is the main reason why we implemented an interactive fitness to complement the sub-rater system.

5.2.2 Interactive and automatic rating

As mentioned, an interactive fitness has potential to assess aspects of music that might be difficult to measure. An interactive rating system is used by Karl Sims in his work in computer graphics [37]. However, it can be difficult to use in music generation since it is more time consuming to listen to a music piece than to glance at an image. Additionally, only one piece can be listened to at a time.

As the results of the interactive rating suggest, Section 4.3.3 and Figure 4.10, there may be a couple of factors that could interfere with the collection of interactive music samples. Small population sizes are often required to make the time situation more manageable, but this will directly inflict a low diversity throughout the generations. Since the task of rating each sample interactively is time consuming and repetitive, and many samples in each generation have a tendency to be very similar, it is unreasonable to expect a qualitative rating of each piece. This may induce some contradictory judgements where very similar pieces get different scores, which could hinder the neural network model to make sense of the data. Furthermore, how should one relate earlier generations with subsequent generations? Earlier generations may contain a lot of samples with randomized data which could be considered as noise. There is also an internal bias within every generation where each piece would be rated in relation to each other. Very different pieces from different generations could therefore get the same score even though the quality may differ significantly. Although, it is hard to limit the user's actions to avoid these conflicts, and it puts unnecessary constraints on the user which work against the idea of an interactive rating function.

However, the evaluation of the automatic rating suggests that the model is capable

of good performance compared to its baseline dummy, see Table 4.1, even though the baseline is quite simple in each case. When we compare the accuracy of each trained model when only considering the labels equal to one — all Bach pieces — Trial 3 stands out with an accuracy of 100 %. This may be the cause of the significant difference of the Bach pieces compared to the generated pieces from our generator, and the model would rather separate these from each other than evaluate each piece independently.

Many of these reasons mentioned along with the short time period in which the data were collected may have contributed to the decline of accuracy when new data points were introduced, see Figure 4.10. Even though there were not a lot of data points introduced, the decline of accuracy is quite significant, which may indicate that there is a lot of noise introduced that the model is volatile, or that this approach needs some tuning or refining. It is probably a combination of all three factors. Initially the neural network model was a response to having a "human in the loop" — only use the interactive fitness process with no relation to a neural network model — and an exploration of using a neural network model alongside a genetic algorithm instead of more traditional approaches like training a model on a big collection of music pieces. However, quite a long feedback cycle still remains and the model may have a hard time of "getting started" if not pre-trained on some existing dataset because of all the noisy data points it will get in the beginning.

To adapt the model to many of these issues, we could use more complex query methods: a subset of each generation, some samples with low, medium, and high fitness score or samples that the model may have a hard time distinguishing. This would effectively improve the quality of the samples we pass to the neural network model as well as reduce the number of melodies we need to score. There is also potential to aid the interactive fitness process with a more suitable user interface to improve the overall quality of the interactive process.

5.3 Challenges and potentials of automatic music generation

As mentioned in Section 5.1, a concern raised by many respondents of the survey is structure and coherence. The lack of direction might hinder the successful generation of interesting and coherent longer pieces. However, it may not be as much of a problem in contexts where the music is used as a secondary medium, such as in computer games or visual media. In those areas, there is a high demand for large quantities of customized music, and it might be unreasonable to hire a human composer to write it. This is something that is already seen with projects such as AIVA [1].

Our program could also be a tool for human composers, instead of being the creative agent itself. Many respondents express that they see musical value in the pieces, but that they need to be processed or arranged by a human composer. If the program is used as a source of inspiration and ideas, it could aid the composition process of a human composer. With an interactive tool, the user does not need much musical training in order to get started. This has potential to open up the music creation process to more people.

5.4 Ethics

The use of artificial intelligence techniques in creative tasks is becoming increasingly popular. The algorithms can be used both in the design process to generate parts to build up to the finished artwork or directly as the final result. In the latter, artists have little to no involvement. There are both pros and cons with this lowering of the required knowledge for creation. On the one hand, it allows more people to create, which leads to more opportunities of exploration within the field. On the other hand, one could say that there is less reason to learn the former required knowledge which could lead to the pieces of art becoming restricted to that platform.

5.5 Authorship and artistic integrity

Another issue of an algorithm creating a finished product is ownership of the artwork. Who has the right to claim the intellectual property when it is being used for profit? Should it be the person who used it at that moment or the people who created the algorithm? If the algorithm used preexisting songs as training data, should the original artists be able to claim it? Is the computer truly creative or should it be treated like using samples when creating songs? One could argue that the artificial intelligence is only using the songs from the training data in the same way artists use other songs as inspiration — of course, it could still be very close to an existing song and thus considered plagiarism.

5.6 Creativity

Is human induced creativity at risk of being lost? To be able to answer this, we have to define what creativity really is. That is of course a very hard question on its own; Margaret Boden, British philosopher and computer scientist, describes it like something of a mystery — almost paradoxical, but further classifies three cases in which creativity can happen [7]. These are: making unfamiliar combinations of familiar ideas, exploring conceptual spaces and transforming the space. Of the three cases, transforming the space has the most groundbreaking results, because they are often viewed as a “shock” of surprise as they stretch the unknown and impossible. However, transformation and combination tend to have a more elusive character than exploration. Thus, computer models based on exploration have had the most successes. Such models, however, have limited capabilities because they

explore different variations of already created artworks with similar results and can thus be seen as a source of inspiration rather than a suppressor of creativity. Boden estimates that professional human artists spend 95% of their time to explore and only 5% to transform, which implies that a computer model needs both [24]. Still, the problem with transformation is not the lack of novel ideas, but rather the lack of novel ideas with interest and value. AI has a big role to play here, according to Boden, but she further explains that many of the processes behind creative thinking still remain unknown. Hence, it is probably safe to say that creativity is not at risk of being lost in the near future. Although, should this be our goal if it ever gets technically feasible? Rob High, Vice president for IBM Watson, thinks we should focus on the techniques that inspire humans to be creative — an opportunity to push creative boundaries — as opposed to technology that can "create" without supervision or direction [24].

The more relevant questions to ask may instead be: to what extent will AI-generated art affect the way we create art and will it be for the better? Will we appreciate art in the same way? How about artistic integrity? Presumably, as the algorithms get better and easier to work with, less prior knowledge is required to create artworks. There will be a greater supply of art, but not necessarily with vast diversity — with the same starting point, exploration tends to converge to similar results. Perhaps we will attribute each work with less value. There might also be hidden motives and biases in the algorithms which could violate the integrity of the artist.

Whether our program truly is creative is a question that therefore is hard to answer. On the one hand, many of the aspects that Boden mentions are crucial for creativity can be found in the program. On the other hand — who are we to judge the creativity of our own program? The survey results did not clearly indicate one way or another, as such we can only speculate that the program at least contains a hint of creativity.

6

Conclusions

The purpose of this project was to explore different aspects of evolutionary algorithms in the field of music generation. The result is a program that uses an evolutionary algorithm, assisted by an artificial neural network and interactive rating to generate music. As discussed earlier, the experience of music is subjective, but we deduce that the program is of musical interest based on the answers in the survey.

As of now, the program is limited by the implementation of the various fitness functions and hyperparameters. The sub-rater system is too narrow to appropriately catch all the aspects of a music piece and the model for the neural network could have been further tested and developed. These are examples that were expected to limit the project due to the scope of the project as well as time constraints.

In order to improve the program, further work on the model for the neural network used could be done, as well as improving the existing hyperparameters. To ensure that the program and subsequently music pieces truly are creative, further testing and deeper understanding of creativity is needed.

Bibliography

[1] Aiva. URL: https://aiva.ai/ (visited on Mar. 15, 2021).

[2] Viktor Anderling et al. *Generation of music through genetic algorithms*. 2014.

[3] The MIDI Manufacturers Association. *An Introduction to MIDI*. URL: https://www.midi.org/images/easyblog_articles/43/intromidi.pdf (visited on May 13, 2021).

[4] Olivier Bélanger. *Pyo*. URL: http://ajaxsoundstudio.com/pyodoc/ (visited on May 13, 2021).

[5] Jordan Bird. *Keras LSTM Music Generator*. 2020. URL: https://github.com/jordan-bird/Keras-LSTM-Music-Generator (visited on Apr. 20, 2021).

[6] Ole Martin Bjørndalen. *Mido Documentation*. URL: https://mido.readthedocs.io/ (visited on May 13, 2021).

[7] Margaret A. Boden. "Creativity and artificial intelligence". In: *Artificial Intelligence* 103.1 (1998), pp. 347–356.

[8] François Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras (visited on Apr. 25, 2021).

[9] Nicholas J. Conard, Maria Malina, and Susanne C. Münzel. "New flutes document the earliest musical tradition in southwestern Germany". In: *Nature* 460 (2009), pp. 737–740.

[10] Palle Dahlstedt. *Autonomous Evolution of Complete Piano Pieces and Performances*. 2007.

[11] Palle Dahlstedt. *Big Data and Creativity*. 2019.

[12] Richard Dawkins. *The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe without Design*. Norton & Company, Inc, 1986.

[13] Paul Doornbusch. "Computer Sound Synthesis in 1951: The Music of CSIRAC". In: *Computer Music Journal* 28 (1 2004), pp. 10–25.

[14] Douglas Eck and Jürgen Schmidhuber. "Finding temporal structure in music: blues improvisation with LSTM recurrent networks". In: *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*. 2002, pp. 747–756. DOI: 10.1109/NNSP.2002.1030094.

[15] Majid Farzaneh and Rahil Mahdian Toroghi. *GGA-MG: Generative Genetic Algorithm for Music Generation*. 2020.

[16] Chris Garcia. *Algorithmic music — David Cope and EMI*. 2015. URL: https://computerhistory.org/blog/algorithmic-music-david-cope-and-emi/ (visited on Mar. 30, 2021).

[17] Andrew Gartland-Jones and Peter Copley. "The Suitability of Genetic Algorithms for Musical Composition". In: *Contemporary Music Review* 22 (2003), pp. 43–55.

[18] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[19] Curtis Hawthorne et al. "Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset". In: *International Conference on Learning Representations*. 2019.

[20] Lejaren Hiller and Leonard Isaacson. *Experimental music; Composition with an electronic computer*. New York, McGraw-Hill, 1959.

[21] Sepp Hochreiter. *The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions*. 1997.

[22] Sepp Hochreiter and Jürgen Schmidhuber. *Long Short-Term Memory*. 1997.

[23] John Holland. *Adaptation in Natural and Artificial Systems*. 1975.

[24] IBM. *The quest for AI creativity*. URL: https://www.ibm.com/watson/advantage-reports/future-of-artificial-intelligence/ai-creativity.html (visited on Mar. 20, 2021).

[25] IFPI. *Music listening 2019*. 2019. URL: https://www.ifpi.org/ifpi-releases-music-listening-2019/ (visited on Apr. 20, 2021).

[26] Jae Hun Jeong and Chang Wook Ahn. "Automatic Evolutionary Music Composition Based on Multi-objective Genetic Algorithm". In: *Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems*. Vol. 2. 2014, pp. 105–115. DOI: 10.1007/978-3-319-13356-0_9.

[27] Scikit Learn. *Tuning the hyper-parameters of an estimator*. URL: https://scikit-learn.org/stable/modules/grid_search.html (visited on May 13, 2021).

[28] Lewis. "Creation by refinement: a creativity paradigm for gradient descent learning networks". In: *IEEE 1988 International Conference on Neural Networks*. Vol. 2. 1988, pp. 229–233. DOI: 10.1109/ICNN.1988.23933.

[29] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/ (visited on Apr. 20, 2021).

[30] Warren S. McCulloch and Walter Pitts. *A Logical Calculus of the Ideas Immanent in Nervous Activity*. 1943.

[31] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[32] Eduardo Reck Miranda and John Al Biles. *Evolutionary Computer Music*. 2007.

[33] Musescore. URL: https://musescore.com/ (visited on May 13, 2021).

[34] Walter Piston. *Harmony: Fifth Edition*. 1987.

[35] Frank Rosenblatt. *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*. 1958.

[36] Sigurður Skúli Sigurgeirsson. *Classical Piano Composer*. 2020. URL: https://github.com/Skuldur/Classical-Piano-Composer (visited on Apr. 20, 2021).

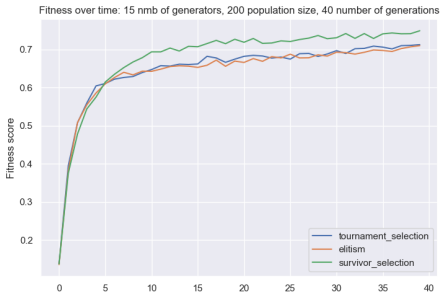
[37] Karl Sims. "Artificial Evolution for Computer Graphics". In: *SIGGRAPH '91 Conference proceedings*. 1991, pp. 319–328.

[38] Darrell Whitley, V. Scott Gordon, and Keith Mathias. "Lamarckian Evolution, The Baldwin Effect and Function Optimization". In: (July 1998). DOI: 10.1007/3-540-58484-6_245.

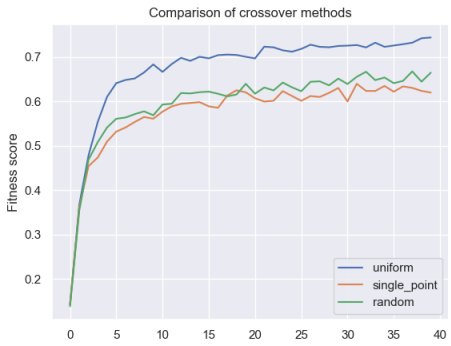
[39] Fabio Massimo Zanzotto. "Viewpoint: Human-in-the-loop Artificial Intelligence". In: *Journal of Artificial Intelligence Research* 64 (Feb. 2019), pp. 243–252. ISSN: 1076-9757. DOI: 10.1613/jair.1.11345. URL: http://dx.doi.org/10.1613/jair.1.11345.

A

Comparisons within the generator



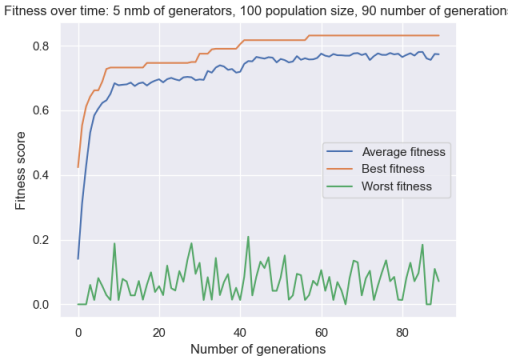
(a) Comparison survivor selection



(b) Comparison of crossover

Figure A.1: Comparison of different survivor selection methods in (a) and crossover methods in (b). The crossover test was done using random survival selection.

(a) Default parent selection



(b) Optimized model

Figure A.2: A generator using random selection of genetic operators in (a), compared to a generator using the genetic operators found to be optimal in (b). Both generators use the fitness function described in Section 3.5.

B

Results from the survey

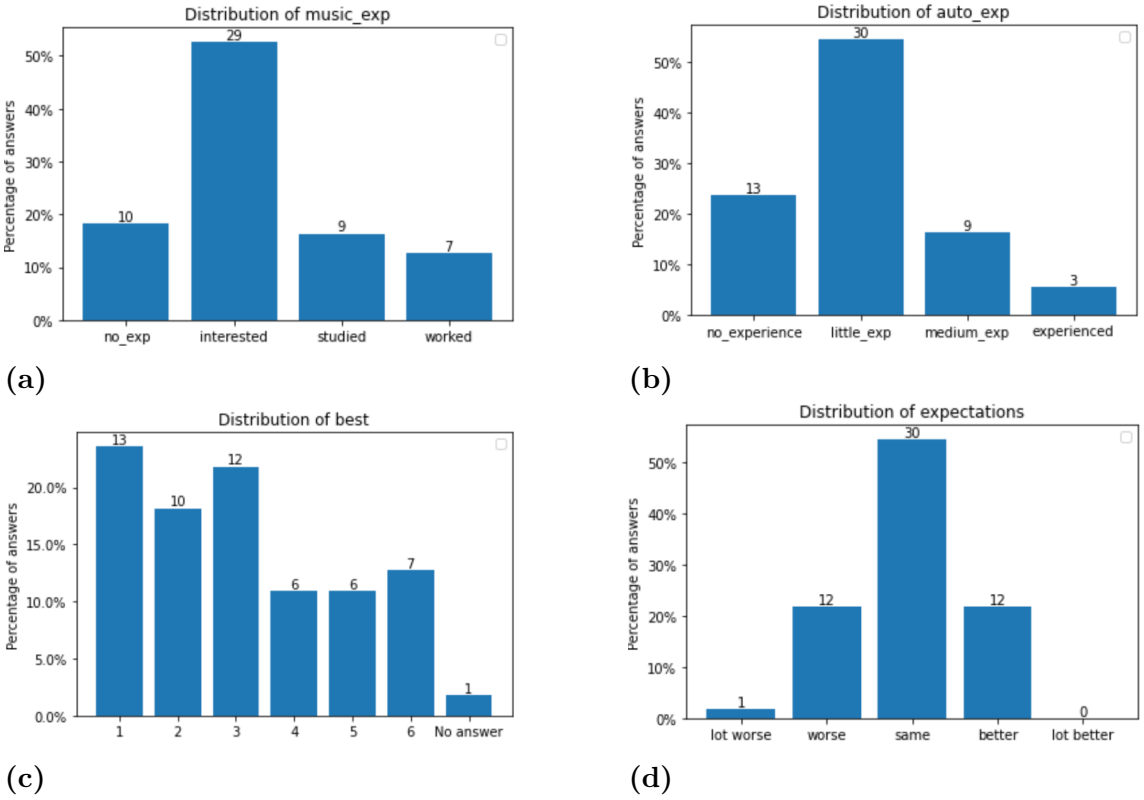
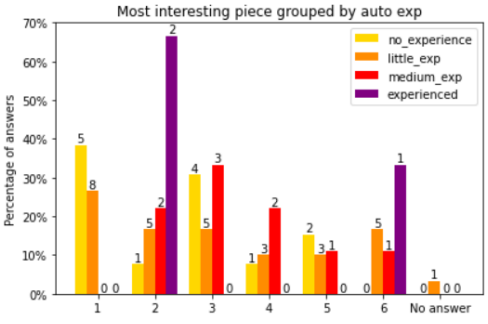
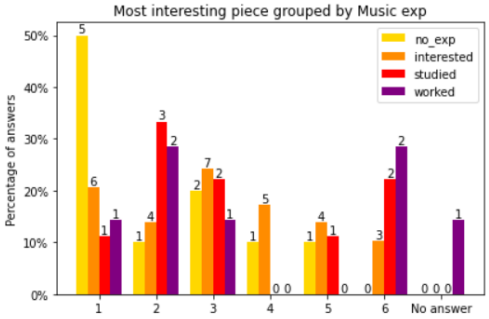


Figure B.1: Distributions of quantitative answers from English and Swedish form. The y-axis shows the percentage of the total answers for that category that a bar makes up. The numbers above each bar indicate the number of people who selected that answers. Some questions were optional so the total number of answers may differ for different plots.

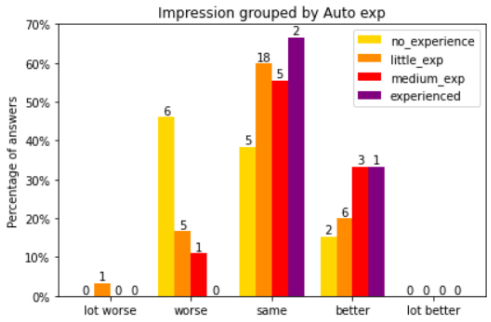


(a)

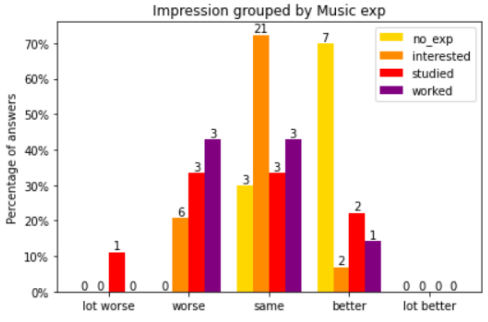


(b)

Figure B.2: Most interesting piece selected, grouped by experience with automatically generated music in (a) and previous music experience in (b). The y-axis indicates what percentage of the music experience group the bar makes up. The numbers above each bar indicates the number of answers that bar represents. A darker color represents a more experienced subgroup. From this we can see that what piece respondents selected as most interesting does not seem to have any strong correlation to what previous experience you have.



(a)



(b)

Figure B.3: Participants' impression of the generator, grouped by experience with automatically generated music in (a) and previous music experience in (b). The left axis indicates what percentage of the music experience group the bar makes up. The numbers above each bar indicates the number of answer that bar represents. A darker color represents a more experienced subgroup.

C

The questionnaire

Evaluating automatically generated music

Vill du hellre svara på svenska klicka på denna länk: https://forms.gle/S23oEuggX9CLYjiMA

Hello! We are a group of Chalmers students from the mathematics, IT and computer science departments. For our Bachelor's thesis we have constructed a program that can generate music. The purpose of the program is to generate something new and original rather than learning from a composer or genre.

Since we are engineers, the creative part of the code is mostly based on programming, math and AI rather than music theory. Therefore we would like your help to evaluate if the music we generate has some musical value.

In this form you will first answer some questions about your background in music as well as your opinion on automatically generated music. After that you will listen to 6 pieces that are between 10-60 seconds long and give some comments on each of those. Finally we wrap up with some questions about your overall impressions of the generator.

Good luck and thank you for your time!

*Obligatorisk

Opinions on automatically generated music

1. What is your previous experience with automatically generated music? Pick the option that fits you best. *

   Markera endast en oval.

   - ◯ Never heard about it before
   - ◯ Have heard about it but haven't read up on it
   - ◯ Read up on it and know about some bigger projects within the field
   - ◯ Have done my own projects and consider myself experienced

2. How would you describe your attitude towards automatically generated music?

Experience with music

3. What is your experience with music? Pick the option that describes your most significant qualification. *

   Markera endast en oval.

   - ◯ No particular experience *Fortsätt till fråga 6*
   - ◯ Interested in music (ex. Plays an instrument, Writes your own music) *Fortsätt till fråga 6*
   - ◯ Have a music education or currently studying music *Fortsätt till fråga 4*
   - ◯ Currently working in music or have worked with music in the past *Fortsätt till fråga 5*

Fortsätt till fråga 6

Education in music

4. Which music education do you have and for how long have you studied?

Fortsätt till fråga 6

Work experience in music

5. Can you give a short explanation of your work experience in music?

Evaluation of generated music pieces

It is finally time to listen to some music. You can listen to the same piece any number of times. We have decided to use only text answer so that you can comment on more than our suggested topics.

Some suggestions when commenting are to consider the music quality with respect to rhythm, melody and harmony. Did you find anything in the piece particularly good/bad? Would you associate any particular emotion, style or genre with that piece?

These are only some suggestions so feel free to write more/less and please write about what you found relevant and interesting for that particular piece!

All of these pieces were generated from our model. We have selected a couple we found more interesting but be aware that they have a lot of room for improvement and they are definitely more contemporary than pop.

If you would like to look at the music sheets you can find then through this link with stycke1 corresponding to piece #1 and so on. https://drive.google.com/file/d/1cx_RfXLaUKQZK1vMoRZmlORWvYodnHk9/view?usp=sharing

Good luck!

Piece #1

 http://youtube.com/watch?v=ocG_KlM8opE

6. Comments: Piece #1 (rhythm, melody, harmony, good/bad, emotion, genre, style)

Piece #2

 http://youtube.com/watch?v=c43hlA6GEKI

7. Comments: Piece #2 (rhythm, melody, harmony, good/bad, emotion, genre, style)

Piece #3

 http://youtube.com/watch?v=G8P9KjctYkg

8. Comments: Piece #3 (rhythm, melody, harmony, good/bad, emotion, genre, style)

Piece #4

 http://youtube.com/watch?v=sJNlkLIFJUQ

9. Comments: Piece #4 (rhythm, melody, harmony, good/bad, emotion, genre, style)

Piece #5

 http://youtube.com/watch?v=08zTKg6uZww

10. Comments: Piece #5 (rhythm, melody, harmony, good/bad, emotion, genre, style)

Piece #6

 http://youtube.com/watch?v=2foiM_jdYUE

11. Comments: Piece #6 (rhythm, melody, harmony, good/bad, emotion, genre, style)

12. Which piece did you like the most?

Markera endast en oval.

- 1
- 2
- 3
- 4
- 5
- 6

13. What made this piece better than the others?

Overall impression

14. Would you say that the generator that produced these pieces generally produces something with musical value?

15. How do you think the generator deals with melody, rhythm and dynamics in general?

16. How do you think the pieces held up to you expectations?

Markera endast en oval.

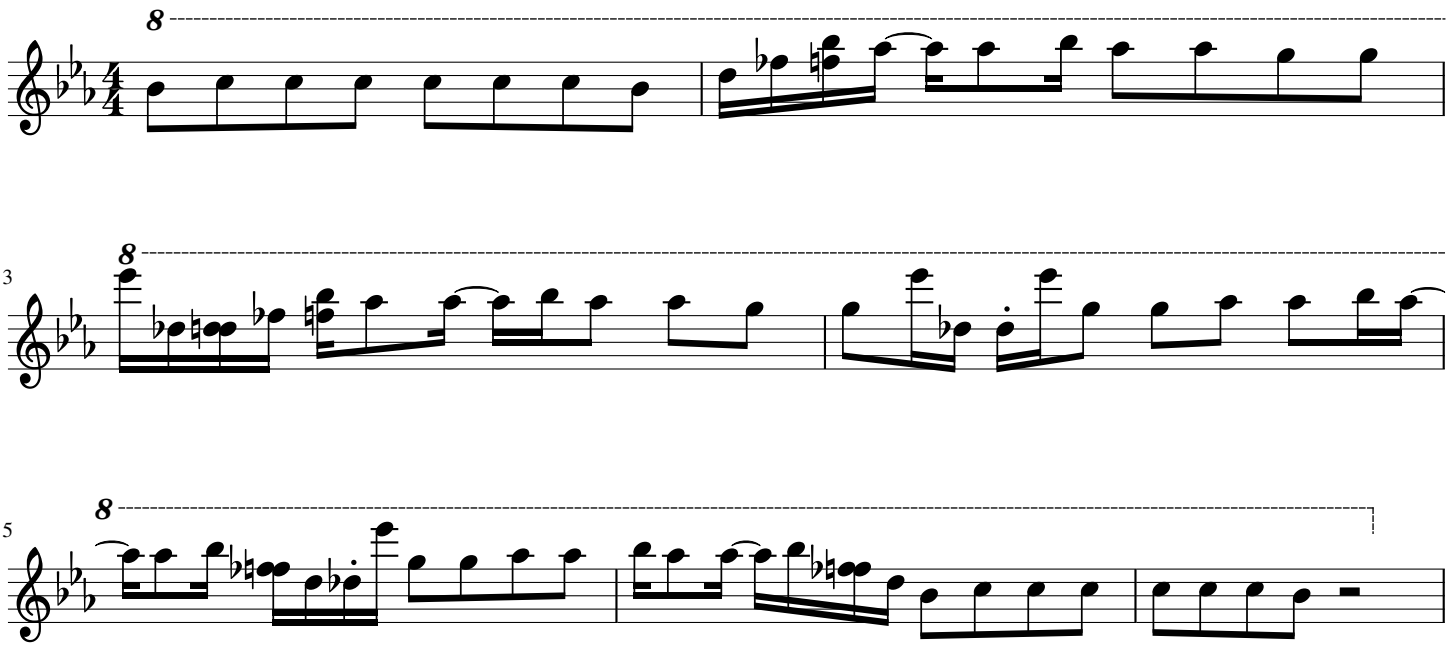
- A lot worse than expected
- Worse than expected
- As expected
- Better than expected
- A lot better than expected

17. Do you have any final comments?

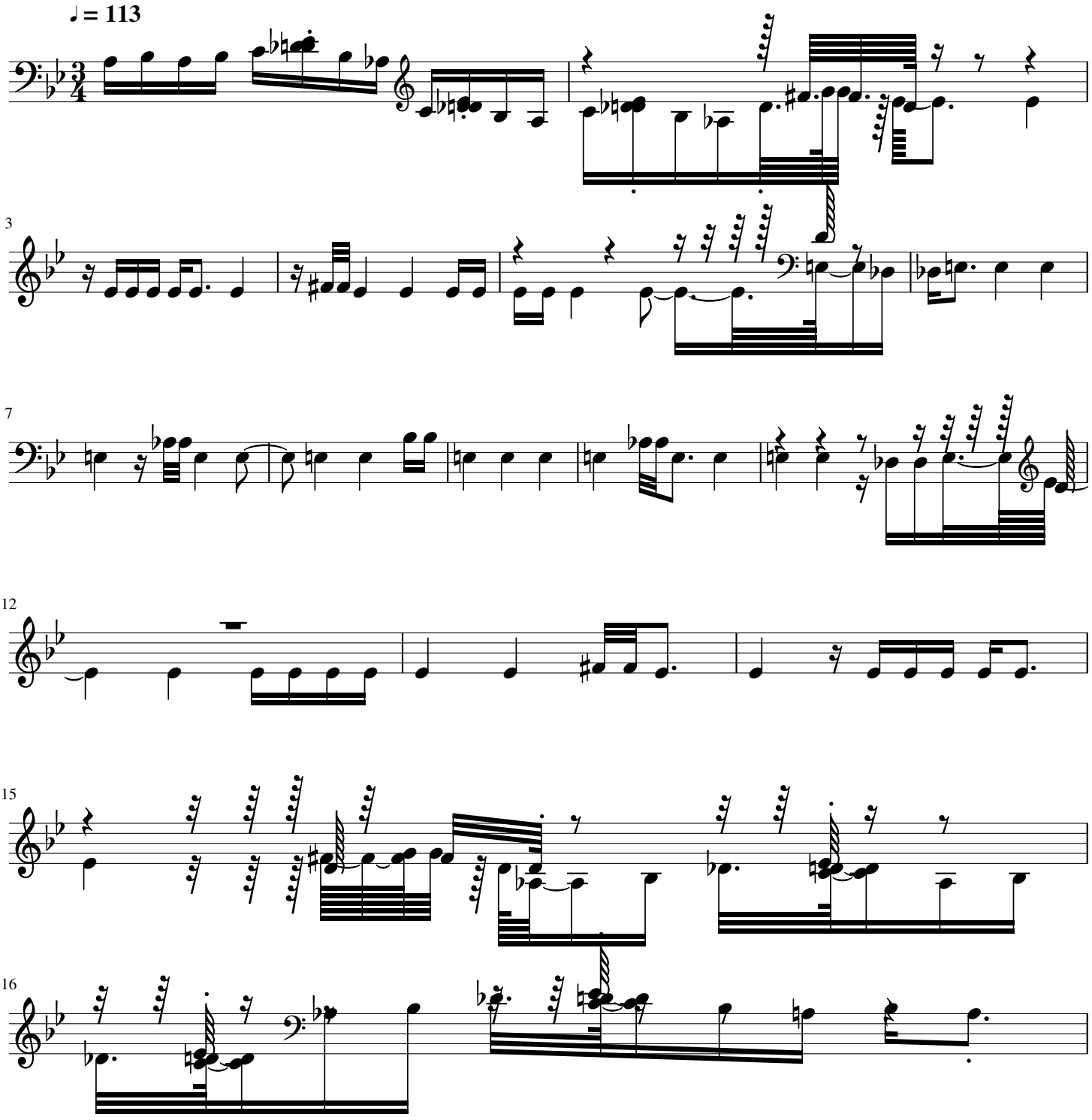
D

The scores used in the questionnaire

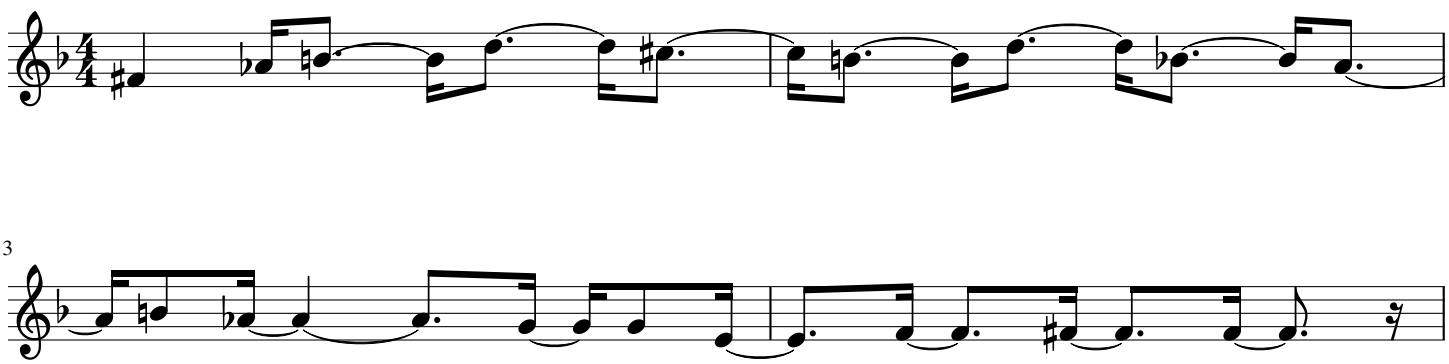
Stycke 1



Stycke 2

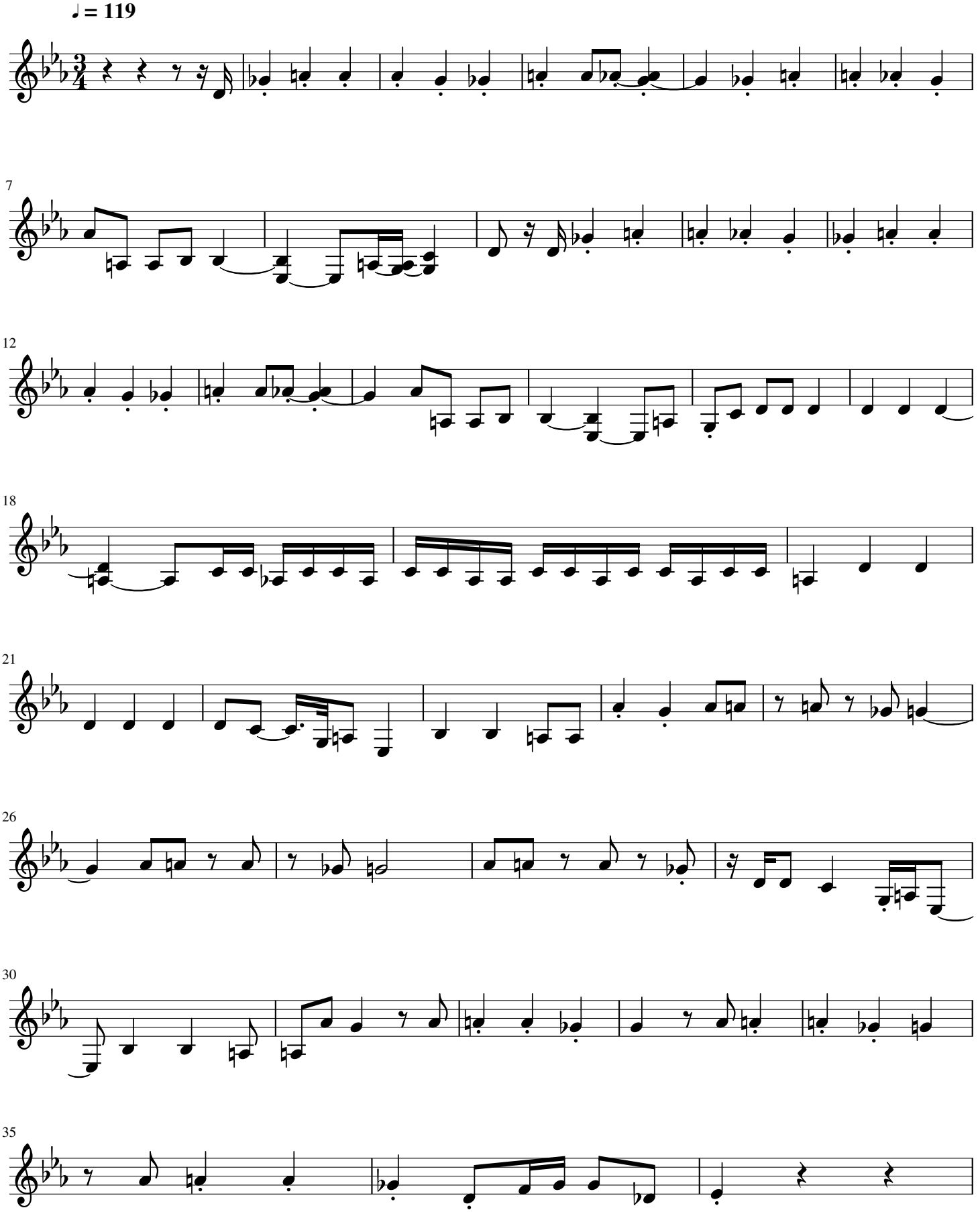


Stycke 3

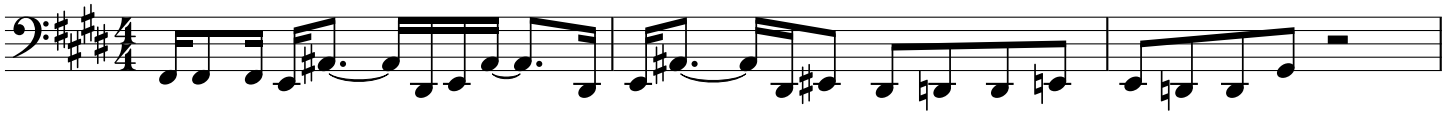


3

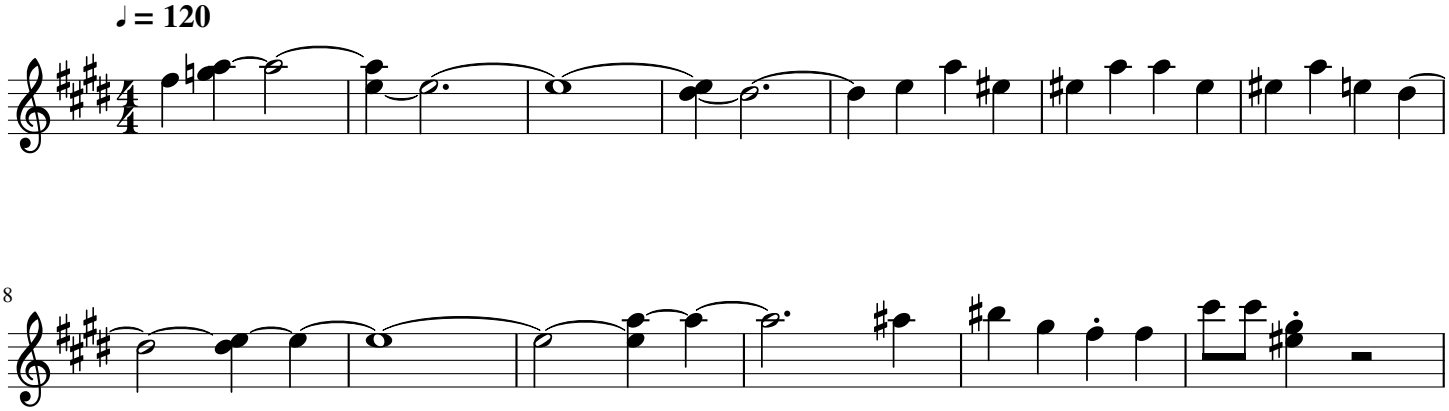
Stycke 4



Stycke 5



Stycke 6



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

UNIVERSITY OF
GOTHENBURG

CHALMERS