COMP 3710 – Applied Artificial Intelligence

Project Proposal

Applied AI - Computational Artist

Due Date: 2023/04/18

Submitted by

Name: Surkamal Singh Jhand

Number of Group Members: 1

Student Number: T00602466

Section: 01

Contents

Game Music Generation with Genetic Algorithm: *Training a Program to Create Original Melodies*

I. Introduction:

A programme called Music Composer employs genetic algorithms to create music. It is a distinctive music composing method that draws inspiration from natural selection and evolution. The software makes a starting population of people, each representing a musical composition, and then assesses their fitness using a fitness function. The fittest individuals are then chosen to produce the following generation via crossover and mutation. This procedure is repeated over several generations until the most suited person is identified. A MIDI file is then created and stored with the finished composition.

Anyone interested in discovering novel methods of music creation should enrol in this programme. It enables users to experiment with various factors and see how they affect the sound quality of the created music, such as population size, mutation rate, and generational number. In addition, users learn more about genetic algorithms and how they might be used in music creation through the program.

The Genetic Algorithm Music Composer programme, including its features, functionality, and implementation, will be thoroughly described in this study. In addition, we'll talk about the program's motivation and possible uses for genetic algorithms in music creation.

a. Brief Overview: A Music Composer program uses genetic algorithms to create unique melodies as MIDI files. A sort of optimization algorithm called a genetic algorithm imitates natural selection and evolution by choosing to reproduce and produce offspring only those individuals who have the highest fitness (in this case, the most attractive tunes). After receiving user input on many parameters, the program creates a starting population of melodies, including population size, mutation rate, and number of generations. Then, the fitness of every member of the population is assessed before the genetic algorithm is performed for a predetermined number of ages. After the run, the best participant (i.e., the melody with the highest fitness) is saved and given its MIDI file.

b. Motivation: Completing the complicated music composition work requires substantial knowledge and skill in music theory and composition. Traditionally, trained musicians and composers with the necessary abilities and expertise to produce beautiful and engrossing music have handled this responsibility. But as technology and artificial intelligence have developed, there has been an increasing interest in creating models and algorithms that can produce music autonomously.

Creating an automated music composition programme aims to investigate how creative and inventive music could be produced using artificial intelligence. Non-musicians can also learn about music creation and write songs using an automated music composition programme. Professional musicians and composers can use this programme to facilitate their creative processes and produce fresh ideas.

The capacity of the genetic algorithm to adapt and produce new tunes over time makes it a popular method for automated music composition. In this programme, we employ the genetic algorithm to create music and investigate its possibilities for producing inventive and beautiful music.

c. Genetic Algorithms and their Application in Music Composition: Search algorithms called genetic algorithms (GAs) are influenced by the principles of genetics and natural selection. In addition to engineering, economics, and medicine, they have been used lately in music creation. By repeatedly iteratively generating a population of musical phrases or "individuals" across several generations, a genetic algorithm can be used to compose music.

The use of genetic algorithms in music composition is a fascinating field of study since it enables the production of innovative and distinctive musical works that might not otherwise be conceivable. GAs can be used to create music in various styles and genres and offer a foundation for navigating the broad field of musical possibilities. Additionally, applying GAs to music creation can offer insightful understandings of the process of musical composition and the fundamental patterns and structures that give rise to musical styles.

Many people are now interested in employing machine learning methods, like GAs, to create music. Due to this, several tools and platforms enabling original music production utilizing GAs have been developed. In this study, we investigate the usage of GAs for musical composition and show how well they produce original musical works.

II. Technical Background:

Python is used to create the Genetic Algorithm Music Composer programme, using external libraries like music21, NumPy, and

Mido. The programme evolves a population of individuals representing various tunes to create new musical compositions using the genetic algorithm optimization technique. Each one is made up of a series of notes, durations, and speeds that are combined to form a musical phrase. A range of objective and subjective standards, including harmony, emotional impact, and melodic and rhythmic variation, are used to assess each person's fitness. In addition, the user can modify several factors in the programme to suit their needs, including population size, mutation rate, and number of generations.

The application is in Python and uses third-party libraries like music21, NumPy, and Mido.

New musical compositions are created using the genetic algorithm optimization method.

Each individual is an interpretation of a musical phrase made up of a series of notes, durations, and velocities.

Each person's fitness is assessed using a combination of objective and subjective standards.

To fit their demands, the user can alter several factors, including the population size, mutation rate, and number of generations.

a. Genetic Algorithm Explanation: We create music using a genetic algorithm in our implementation. A class of optimization algorithms known as a genetic algorithm searches for answers to challenging issues by simulating the natural selection process. Our "problem" is coming up with engaging and distinctive musical works.

A series of notes, durations, and velocities in our genetic algorithm represent each member of the population. The "fitness" of these sequences is then assessed using a variety of factors, including how well they adhere to musical conventions and how pleasing they are to the ear. The highest-scoring people are then chosen as parents for the following generation, and their sequences are merged to produce new "children" who may have higher fitness ratings.

We introduce a mutation into the population by randomly changing the children's sequences' notes, durations, and velocities. This allows for exploring fresh musical concepts and allows the algorithm to settle on a less-than-ideal answer later.

Our application of the genetic algorithm for music composition is a novel and inventive use of this powerful optimization method. We are eager to hear the musical works it generates.

b. Overview of the MIDI file format: Digital representations of music are made using the MIDI file format. The data concerning notes, durations, velocities, and other musical characteristics are stored in a binary form. In our implementation, the music produced by the genetic algorithm is represented through MIDI files. First, we create MIDI files from the music produced by the genetic algorithm using the pretty_midi module. The library offers a simple interface that makes it easy to create MIDI files by letting us customize the notes' pitch, length, and velocity. Then, to hear the music and assess the quality of the music produced by the algorithm, we use the created MIDI file. The MIDI file format is helpful for our needs because it is extensively used in the music industry and is supported by most digital audio workstations and music software.

c. Description of Python Libraries Used: The program is written in Python and uses several libraries to implement the genetic algorithm for music composition. The libraries used include:

- random: This library generates random numbers used in several program parts, such as developing the initial population and mutation.

- numpy: This library performs numerical computations and generates random numbers from a specified distribution.

- typing: This library defines types and function signatures to improve code clarity and readability.

- midiutil: This library is used to generate MIDI files from the output of the genetic algorithm.

- sys: This library provides access to some variables used or maintained by the Python interpreter and functions that interact strongly with the interpreter. This program is used to exit the program when an error occurs.

- fluidsynth: Using a software synthesizer, this library plays the generated MIDI files.

- logging: This library is used to configure the logging system for the program.

These libraries were chosen for their ease of use, reliability, and suitability for the specific tasks used in the program.

III. Design and Implementation:

Python has been used to build and construct the genetic algorithm music composer. The programme uses a genetic algorithm to generate a population of individuals, where each individual represents a musical composition, and the population is then used to make music. The programme accepts user input for several factors, including population size, mutation rate, and generations. The user can also change the length of each component individually.

The program runs the genetic algorithm for a predetermined number of generations after creating an initial population and assessing each person's fitness according to the standard procedures of a genetic algorithm. Next, the top person from each generation is printed, and the complete person from the entire run is kept. Finally, a MIDI file is created for the leading candidate, with the filename containing the candidate's fitness score.

a. Project Structure:

The project is structured in a single file, "genetic_music.py," which contains all the code for the genetic algorithm music composer. The file starts with importing the necessary Python libraries, followed by global variables used throughout the program. Next, the primary function, "run_genetic_algorithm," is defined and is responsible for prompting the user for input regarding various parameters, generating an initial population, and running the genetic algorithm for the specified number of generations. The program also includes several helper functions for generating individuals, evaluating fitness, selecting parents for crossover, and generating MIDI files.

Finally, the program uses the "if name == 'main'" construct to call the "run_genetic_algorithm" function and run the entire genetic algorithm music composer program.

b. Description of Main Components:

- generate_individual: The generate_individual function generates a random individual, representing a musical composition. The function creates a dictionary with three keys - notes, durations, and velocities. The critical value of the note is a list of tuples, where each tuple represents a musical note and its octave. The value of the durations key is a list of integers representing each note's duration in ticks. The value of the velocities key is a list of integers representing each note's velocity or volume.

- Fitness: The fitness function is used to evaluate an individual's fitness, which represents how good the musical composition is. The function computes the total length of the composition in ticks and the number of unique notes in the composition. The fitness score is calculated as the sum of the size of the composition and the square root of the number of individual notes.

- Selection: The selection function selects parents for crossover using tournament selection. Tournament selection is a technique where a group of individuals are randomly chosen from the population, and the fittest individual from the group is selected as a parent. First, the function considers the current population, their fitness scores, and the tournament size as inputs. Then, it creates a new list of parents by repeatedly selecting the fittest individual from random groups of individuals.

- Crossover: Using a one-point crossover technique, the crossover function creates children from parents. In this technique, a random point is chosen in the composition, and the parent individuals' notes, durations, and velocities are swapped after the crossover point to create the children. The function takes in two-parent individuals as inputs and returns two child individuals.

- Mutation: The mutation function introduces random changes in the children's compositions to increase genetic diversity. The function iterates through each note in the child's composition and randomly changes its note, duration, or velocity with a probability equal to the mutation rate. The mutation rate is a user-defined parameter that controls the rate at which mutations occur.

- Genetic Algorithm Implementation: In our project, we used a fixed-length array of integers as the chromosomal representation and a Genetic Algorithm (GA) to optimize the provided problem—a population of 100 chromosomes that were randomly created at the beginning of the GA. The objective function of the problem served as the foundation for the fitness function, which evaluated each chromosome's quality.

  Necessary GA operations included one-point crossover, integer-based mutation with a chance of 1/n, and tournament-based selection, where n is the length of the chromosome. We used an elitism method during the replacement process to ensure the best solutions were kept. The GA was programmed to operate for up to 500 generations or until a certain level of fitness development was reached. Our project successfully and effectively found near-optimal solutions because of this deployment.

c. Parameter Selection Process: In our project, the method for choosing the parameters was vital to enhancing the effectiveness of the Genetic Algorithm (GA). Population size, crossover rate, mutation rate, and the number of generations were the main factors considered. To choose the best settings, we experimented with several combinations and assessed how they affected the performance and accuracy of the GA.

   We experimented with population sizes ranging from 50 to 200 before settling on 100 since it struck a nice balance between computational effectiveness and the diversity needed for exploration.

IV. Results and Evaluation:

a. Best Individual: Throughout our experiments, we isolated a small subset of the population, producing comparatively more pleasing tunes. These people had more melodic and well-structured music thanks to their optimized parameter values and use of weights and consonants. Even though most of the melodies fell short of the desired standard, the greatest ones served as a foundation for future development.

b. Analysis of Generated Music: In its current form, the generated music needs to improve the quality of human creations. The algorithm does produce some tunes with structure, but it needs to include the intricacy, harmony, and diversity required in expert compositions. This suggests that further testing and fine-tuning with the program's parameters, weights, and consonants are needed to produce more aesthetically pleasing music.

c. Other Music Generation Techniques: Markov chains, LSTM-based neural networks, and Transformer-based models like OpenAI's MuseNet are just a few examples of additional music creation methods. Exploring these options may offer ideas for improving our GA-based strategy because each method has advantages and disadvantages.

d. Limitations and Possible Improvements: The quality of the created music is our current implementation's main drawback. Further tuning of the GA parameters, weights, and consonants is required to improve this. In addition, compositions might be more appealing if they incorporate more sophisticated fitness functions and restrictions to follow music theory norms. Furthermore, more diversified and original melodies may result from broadening the search area by raising the population and generational numbers. Finally, combining our GA-based approach with other methods for creating music, such as deep learning models, may produce more sophisticated and musically varied outputs.

V. Conclusion:

This study has shown the potential of genetic algorithms in creating music. Although there is potential for improvement in the current implementation, the groundwork has already been done to strengthen and improve the algorithm. We intend to build a more robust and flexible music production system with plans to fine-tune the programme, add a GUI, incorporate a notes dataset, and investigate other fitness functions. As we develop, disseminating our expertise to the public will advance our understanding of algorithmic music composition and open the door to fresh and exciting musical experiences.

a. Project Summary: We used a genetic algorithm-based strategy to create music for this project. Our main objective was to develop an algorithm to generate catchy melodies with a specific amount of harmony and structure. Even though the current implementation has resulted in poor tunes, it provides a foundation for further development and improvement.

b. Achievements and Challenges Faced: The successful use of genetic algorithms for music generation has been this project's major accomplishment. We've established a fundamental framework for producing tunes, which can still be enhanced. But the main issue was the calibre of the music that was generated. The complexity and musical depth necessary to produce genuinely captivating tunes are outside the algorithm in its current version.

c. Future Works: To enhance this project following is planned:

- Fine-tuning the program: We will work on optimizing the GA parameters, weights, and consonants to improve the quality of the generated melodies.

- Implementing a GUI: By developing a user-friendly interface, we aim to make the program more accessible and easier to use, allowing users to perform experiments more efficiently.

- Incorporating a notes dataset: Integrating a dataset of musical notes into the program will help create more diverse and creative melodies while adhering to music theory rules.

- Exploring various fitness functions: We plan to investigate different fitness functions and compare their performance in generating music using standardized tests. This will help identify the most effective fitness functions for our GA-based approach.

- Reporting findings: After completing the planned improvements, we will document and share our results with the community, potentially contributing to developing more advanced music generation techniques.