Music Generation Project 2 Report

Arseny Savchenko

Innopolis University

18 July, 2023

I Introduction

This report aimed to provide an overview of the second project, compare it with the previous solution, analyze strong points and what can be further improved in the current state of the development. The goal of the project is to generate MIDI files with the specific algorithm, containing both melody and accompaniment.

Generated .mid files, as well as the source code, can be found on my GitHub. To build and run the project, Rust compiler is required.

II Algorithms

Genetic algorithm was chosen for this task. All 4 stages (generation of the initial population, selection with fitness function, crossover and mutation) were involved.

A. Initial population

Initial population is generated with the algorithm that was used for the first assignment, but with some changes. Algorithm was based on the idea of arpeggiation, however, it limits the randomization of tempo to produce something acceptable. Algorithm separates bar on 16 parts, creates a tonic note with a zero delay and a length equal to 1/16 of bar. Then it travels across all 16 parts of the bar and tries to put a note with the same length that is located closely to the current one (difference in pitch is not greater than 3 notes in scale).

Strict tempo management was removed: now only the delays greater than 3/16 of bar are not allowed. The number or order of them are not controlled. BPM generation is in [85;115] range now. Key generation restrictions were removed: now all possible keys are allowed to be used. All populations have size 1000. Implementation can be found here.

B. Ideal lead

Before the generation process itself, ideal lead is chosen from the list of leads. Such leads are stored in './genetic-samples' folder as .mid files. To obtain a logically better result,

file has to contain only the lead melody in Aeolian mode.

C. *Fitness function and Selection*

Algorithm uses multiple tests to check if generated lead is acceptable. First of all, all notes are compared with the ideal lead: algorithm compares both distances between pitches from the generated lead and the ideal one and tempo (delays) between notes. Partial coincidence is counted for distances in pitch, but tempo match is more strict. For pitch match it gives 1/4 of the amount that can be achieved by the single note, for tempo match it gives 3/4. Every note weights 1 / lead size. Other restrictions are also chosen (length, repetitions, delays, etc.) Implementation can be found here.

Selection is done with the roulette method described in the lecture. No additional upgrades were implemented. Implementation is here

D. *Crossover*

In the crossover method, multiple points are randomly chosen. Number of points is also randomly generated in range [1;4]. Then, two parents form the child for the next generation. Additionally, parent can still appear in the next population. Implementation can be found here.

E. *Mutation*

Mutation algorithm traverses through the lead, randomly picks some notes and randomizes their pitch. Obtained notes still lie on the scale. Implementation is here

III Analysis of the result. Further improvements

Even though algorithm produces listenable results, generated melodies are not as unique as desired. Algorithm is still limited with note's length (1/16), which can be additionally improved in the future work. Besides, it is still limited to Aeolian mode. Additionally, multiple instruments were used this time, but it would be greater to extend it with drums.