AI Assignment 2 Report

May 2022

1 Brief description of the solution

Using genetic algorithm, accompaniment generation for a monophonic MIDI file was implemented. The MIDI file is read and converted into internal melody representation, and its parameters such as key and octave are analyzed. Based on that, a pool of chords potentially compatible with that melody is produced with some report information.

Evolutionary algorithm part: we generate initial population (of accompaniments) built with these chords and start selection process with one-point crossover and swap mutation; goodness of the chord is evaluated by the fitness function. Well-fit parents are preserved in the population. After running several hundreds of evolution steps, we come up with the most fit individual that is converted back to MIDI as additional track to the copy of initial MIDI file.

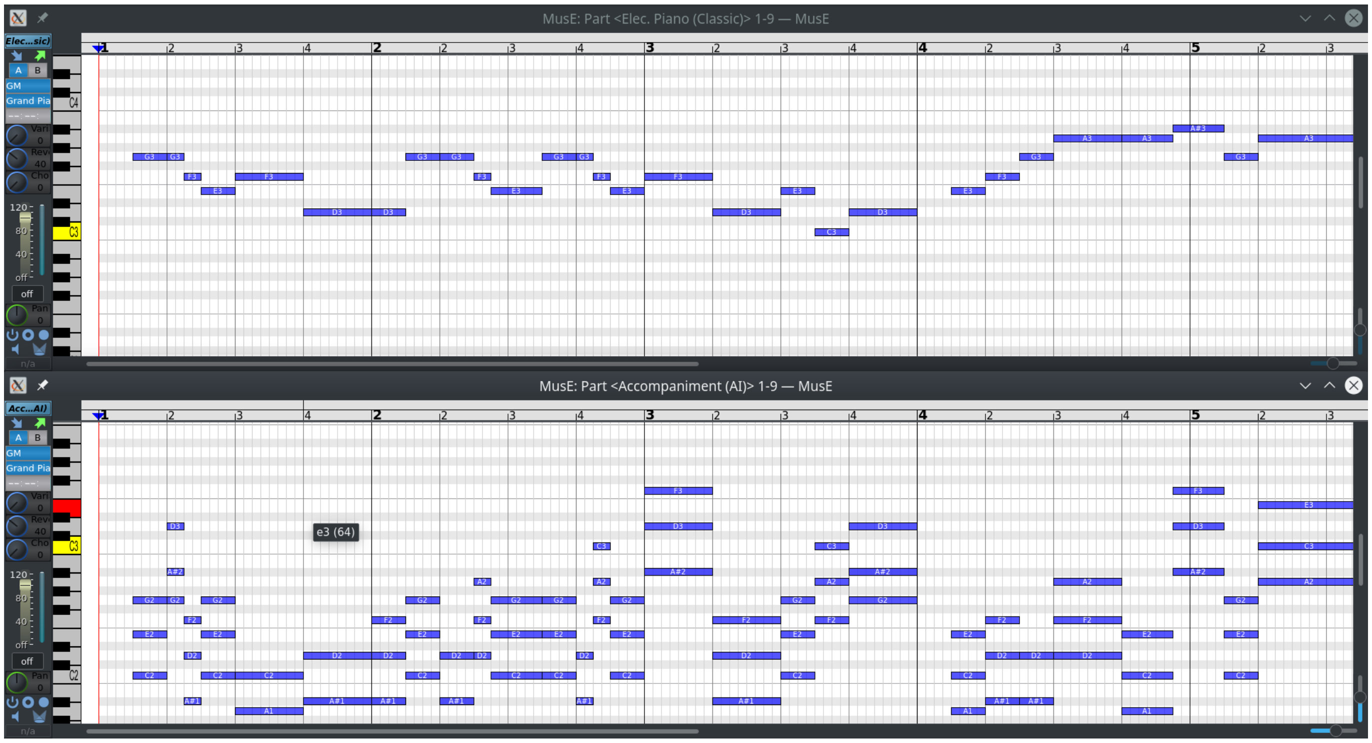

MusE: Part <Elec. Piano (Classic)> 1-9 — MusE
MusE: Part <Accompaniment (AI)> 1-9 — MusE
e3 (64)

Figure 1: Original melody (window above) and generated accompaniment (window below) in MIDI sequencer

1.1 Running the code

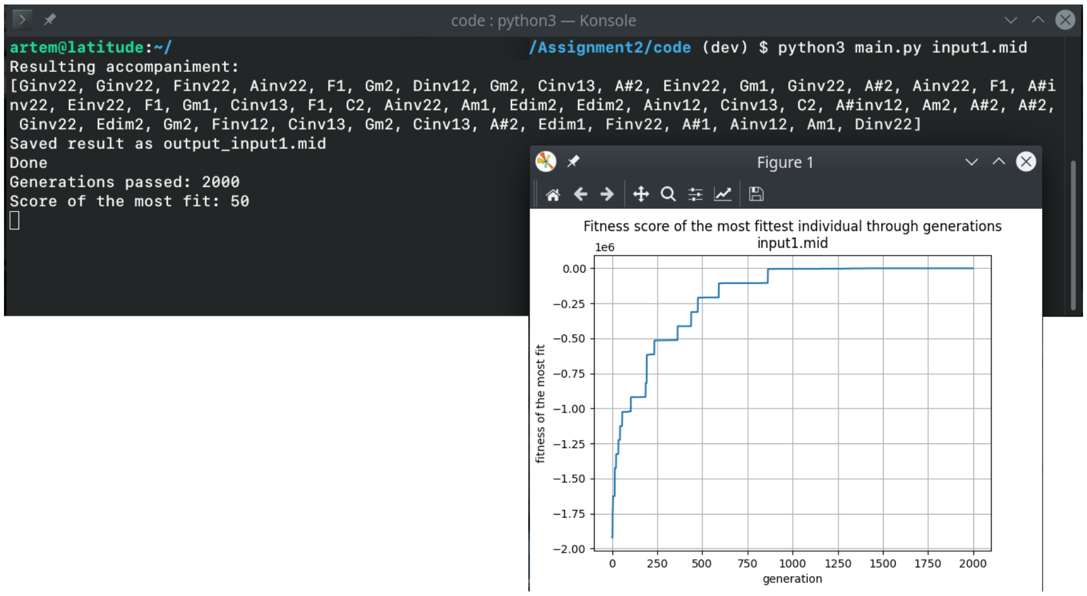
Tested on Python 3.9.

Required Python packages: `numpy, matplotlib, mido, music21`

```
$ pip install numpy matplotlib mido music21
```

Then run the Python script, specifying the input file: `python3 ArtemBulgakov.py input.mid`

The program will print the report in the terminal and plot the graph in the new window. The program terminates after the window is closed by user.



```
artem@latitude:~/[redacted]/Assignment2/code (dev) $ python3 main.py input1.mid
Resulting accompaniment:
[Ginv22, Ginv22, Finv22, Ainv22, F1, Gm2, Dinv12, Gm2, Cinv13, A#2, Einv22, Gm1, Ginv22, A#2, Ainv22, F1, A#i
nv22, Einv22, F1, Gm1, Cinv13, F1, C2, Ainv22, Am1, Edim2, Edim2, Ainv12, Cinv13, C2, A#inv12, Am2, A#2, A#2,
 Ginv22, Edim2, Gm2, Finv12, Cinv13, Gm2, Cinv13, A#2, Edim1, Finv22, A#1, Ainv12, Am1, Dinv22]
Saved result as output_input1.mid
Done
Generations passed: 2000
Score of the most fit: 50
```

Figure 2: Example of program run

2 Melody analysis

First of all, we convert the MIDI file to the internal representation and then examine it to fetch some useful information, including used octaves and the key of the melody, that we will rely on when determining good (consonant) chord for the accompaniment of the melody.

Referring to the musical theory, we can quite easily find a pair of two keys, major and minor, called **relative** or **parallel**. To do this, we just need to find a number of unique sharp keys and perform a lookup by known table.

The hard part is to tell apart which of two is exactly the **key** of a melody. Here, we use `music21` module's analyzer (that uses Krumhansl's algorithm for key detection).

After finding out the key, we can apply some methods to find good **progressions of chords** as a part of the **fitness function** (details below)

3 Evolution

The solution implements evolutionary algorithm for finding the accompaniment. Here, individuals are possible accompaniments, with chromosomes consisting of genes that are are single chords. Running evolutionary procedure for a `generations` times, we find the most fit individual (accompaniment), that is translated to the MIDI file as a new track added in parallel to the existing melody (details in the corresponding section). To make the results better, we run **several independent evolution runs with different initial populations**, then **pick the most fittest of the most fittest individuals** for every run of the evolutionary algorithm. The default number of generations is 2000, that is enough to get a good result. There are no premature stop conditions.

3.1 Initial population

We could generate a pool of all possible chords that can be taken as genes in principle. They are generated as the following. Knowing the key (tonic chord), we can generate all chords by the following interval signature:

$$Tone - Tone - Semitone - Tone - Tone - Tone - Semitone$$

(major key)

$$Tone - Semitone - Tone - Tone - Semitone - Tone - Tone$$

(minor key)

However, we use one small optimization: after finding out the key of the melody, **we know what chords are compatible with the melody by themselves** (i.e. those that are in its scale), and **form our chord pool from the scale chords**. This doesn't tells us much about the chord layout, used octaves, etc. but **drastically reduces the search space** and thus the **number of generations needed** to obtain a good result. Inverted and diminished chords are also included in the pool. By default, the population size is 100. Each generation (except the final one) has 80 offspring.

3.2 Crossover

Is performed as **one-point crossover**. The point is taken at random, then genes up to this point are swapped.

3.3 Mutation

Consists of two steps:

1. Two genes are swapped at random;
2. One gene (not necessary that was affected in previous step) is replaced with the random one.

3.4 Selection

Generated offspring is added to the current population; then all individuals get sorted by the fitness score. Then non-fit part of population gets cut off (to make the population size exactly `generations`). Thus, the selection preserves well-fit parents, i.e. has **elitism**.

3.5 Fitness function

3.5.1 Musical considerations

For each chord, we have a corresponding note (that the chord will accompany). We check **how good they sound together** and **overall harmony** of a given accompaniment (i.e. overall fitness of an individual).

What we penalize accompaniments for?

1. The chord is very bad if it **doesn't contain a note of the melody it is played simultaneously with**;
2. The chord is very bad if it contains **notes out of the melody's tonality** (scale) (i.e., contains the chords not mentioned in the table). **This case is eliminated in advance since such chords do not get into chords pool.**
3. The chord gets mixed with the melody. It happens when the chord has the same octave as the melody (or higher). This is also eliminated in advance: **octaves of the chord pool also do not exceed the octave of the melody**;
4. At the same time, it is undesirable to have **large span of notes' octaves** in the accompaniment;
5. Also it is undesirable to have **accompaniment lower than two octaves compared to the melody**;
6. It is undesirable for two adjacent chord to **have root notes too far from each other**
7. **1, 2, 10, 11 halftone intervals** between notes in a chord+ melody 4-note sequence are bad as they lead to dissonances.

What we reward accompaniments for?

1. It is good for a sequence of adjacent chords to produce **progressions** (S-D-T, S-T, D-T)
2. It is good for a sequence to have **some inverted chords** (they sound higher and make the accompaniment overall nicer). However, having too much inverted chords (more than a threshold `rate*len(chords)` is also undesirable

These considerations are implemented into the fitness functions. As we see on the graph, eventually population (to be precise, the most fit individual) finds a way to almost fully comply with the fitness function requirements.

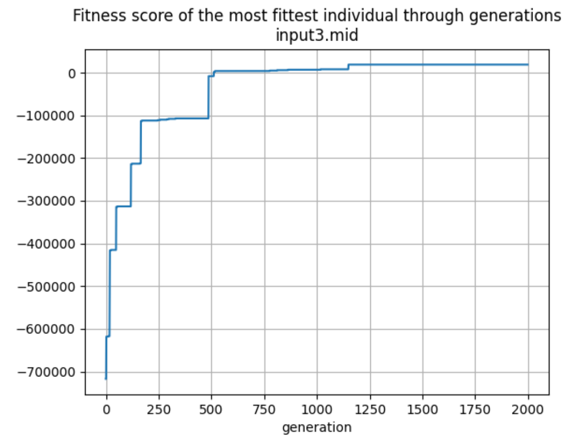


Figure 3: Fitness value of the most fittest through generations