Report for the Assignment 2: Accompaniment Generation

Author: Kopeikina Anna, BS20-01, a.kopeikina@innopolis.university

The goal of the assignment

We would like to build an accompaniment for monophonic input consisting of triads of notes called chords. In order to do it, we firstly analyze and get the tonality of the song. It is needed for the later selection of consonant chords, using our knowledge of music theory.

Evolutionary Algorithm

After we get the tonality of the song, we need to detect the size of the accompaniment. In our case, we would like to play a chord for each semi-tact. After it, we use the best-generated accompaniment from the last generation by our algorithm and add it to the original MIDI file.

In order to do it, we need to define some constraints and parameters of our genetic algorithm.

Chromosomes

To create the algorithm we need to define our individuals. In this case, the chromosome itself is the accompaniment consisting of chords.

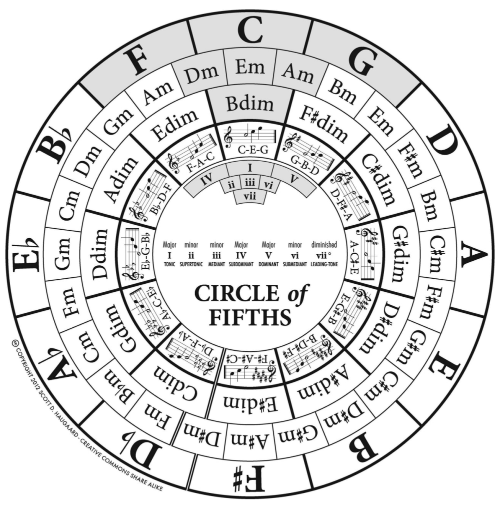
The first population's accompaniments consist of random chords. We create such a random chord by using the random integer, which will represent the main note, and give to the chord some existing defined pattern(major, minor, sus2, sus4, major-inverse, minor-inverse, and double-inverse). This pattern is defined by an array of integers, which represent the indents from the main note.

For the first generation, we use quite a small number of chromosomes: `128`. We use this number, as from tests it was noticed that while a bigger amount of chromosomes will produce the algorithm in fewer generations, the work with generations itself will be slowed down.

Fitness function

To find the best sound algorithm we need a good fitness function to rate our accompaniments. In our case to rate the accompaniments we use the circle of fifth. Firstly, we give it the worst rating by giving 1 point for each chord and afterward reward for each fitting chord by reducing this rating.

To do it, we use previously calculated consonant chords for the song and compare them to each chord of the generated accompaniment. So if the generated chord is in consonant chords of the song, we reward the chromosome by increasing its rank by 0.5.



The circle of Fifth

After it, if it is in consonant chords, we check if there is a pause in the song at this moment. If it is, it means that any chord from the consonant array will fit it, so we also reduce the rating by 0.5. However, if there is a note in the original MIDI in this semi-tact, we check if this note is in the chord, and if yes - we also reduce the rating by 0.5 points. If the note isn't in a chord but also isn't in consonant chords, it is fine and it means that any consonant chords would be fine, so we also reduce the rating by 0.5.

In other cases, we do not change the rating of the accompaniment. Afterward we sort the population by ranking. Here accompaniments with the least rating are the best fitting accompaniments.

Variation operator: Crossover

In our case we use One Point crossover - we select one point at the random position for two parents and then swap their genes for two children.

Variation operator: Mutation

To mutate chromosomes we select a specific amount of them in the population, and replace one of each accompaniment's genes with a random chord, generated the same way as in the first population. In our code we mutate 50% of chromosomes, however we always mutate only one chord of the chromosome. It allows us to achieve the best result, where the accompaniment needs the change only in 1-2 chords, as mutating more genes could make the result worse.

Selection

In our program, we select 25% of the best-rated chromosomes by sorting the population by rating and choosing the least rated ones. After we select them, we move them into the new array and then create the new population by using crossover for these survivors.

Stopping Criteria

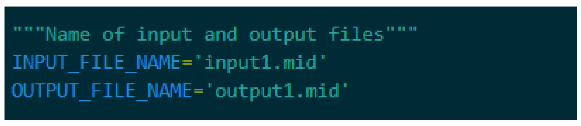
Each chromosome in our population has a rating, so we would like to stop our program in two cases:

1. The rating of the chromosome is equal to 0. Each of the fitting chords reduces the rating by 1, and the rating at the start is equal to the number of chords, so the rating, that is equal to 0, means that all chords of the accompaniment are fitting.
2. The number of iterations for the population is greater than the stopping number. In our case, this number is equal to `5000`. If the number of populations is too big, we pick at least some non ideal solution.

Setuping and starting the program

This program uses two libraries for its work: `music21` for simplifying detection of the song tonality and `mido` for parsing input, working with the midi format, and creating output song. The code was written on python, version: `Python 3.9.12`.

At the start of the program, you would see two constraints, where you can change the name of both input and generated output files.



```
"""Name of input and output files"""
INPUT_FILE_NAME='input1.mid'
OUTPUT_FILE_NAME='output1.mid'
```

The average amount of the algorithm iterations needed for generating a fitting versions is about ~400 iterations. The approximate time of completing the generation is around 3-5 seconds.