Damir Nabiullin B20-03. Assignment 2.

Chord:

For my accompaniment I chose the length of chords is equal to half of the bar.

Code Structure:

- Class ChordGenerator – class that helps to generate chords according to tonic and mode.
- Class MidiGenerator – class that works with midi files (read input file/create output file).
- Class Chord – represents chord, makes inversions, and checks dissonance.
- Class Chromosome – represents chromosome in genetic algorithm. It has it's own genes (in my case genes = chords) which can be mutated. Moreover, two chromosome can be crossover (it will generate two new chromosomes).
- Class GeneticAlgorithm – class that represents the whole genetic algorithm.

How Code Works:

When you print the correct file name, this file will be opened by MidiGenerator. MidiGenerator will get all data about tempo, tonic, mode and etc. This data we put into the ChordGenerator to generate good chords for our accompaniment. Moreover, ChordGenerator will generate chords only one or two tones below from the initial melody. When all generators were created, GeneticAlgorithm is created in the code. We put all needed data about melody such as tonic, mode, ticks and etc., and put ChordGenerator make manipulations with chords in mutation. When all needed classes are created, algorithm starts work.

Algorithm flow:

When algorithm got/generated initial population, it starts to work. Genetic algorithm has some number of iterations and probability of mutation. On each iteration algorithm makes random number of crossovers between random chromosomes in the population and make random number of mutations according to probability. When all iterations are done – algorithm calculates fitness for all chromosomes and returns sorted chromosomes (in the end we have the best one).

In the main file I run the GeneticAlgorithm several times (I called this value as epoch). At the first time, algorithm generates random initial population. At the next time, I put to algorithm the best results from previous run and some number of random initial population.

After all epochs, we get the best chromosome from all epochs and iterations. From this chromosome we receive chords. This chords are used to create accompaniment in the output file.

Mutation:

In mutation I randomly select (initial population length / 2) chromosomes from population. For each chosen chromosome:

- I change the whole cromosome with probability 10%
- I make up inversion with probability 25%
- I make down inversion with probability 25%
- I make sus2 with probability 20%
- I make sus4 with probability 20%

Crossover:

In crossover I randomly select (initial population length / 2) pairs of chromosomes from population. For each chosen pair:

- Split each chromosome in half
- Generate 2 new chromosomes: one with the first half of the first chromosome and the second half of the second chromosome. The other with the first half of the second chromosome and the second half with the second half of first chromosome.

Selection:

In selection I sort all population via fitness function and return (initial population length / 2) the best chromosomes. From this list of the chromosomes I will generate new initial population or take the best chromosome to output.

Fitness function:

In fitness function I calculate how good is this chromosome. I check several conditions for each chord in chromosome:

1. If there is no empty space in a melody where chord occurs:
    - If absolute value of difference between the bottom note of the previous chord and the bottom note of this chord is more than 4 - decrease total cost by (80 * absolute value of difference).
    - If we have progression: ST – increase total cost by 50, DT - increase total cost by 50, SDT - increase total cost by 100.
    - If previous chord has the same notes with this chord - increase total cost by (30 * intersection count). Otherwise – decrease total cost by 70.
    - If chord have the same note with melody - increase total cost by 100/50 (different conditions).
    - If chord has dissonance with note in melody - decrease total cost by 300/200/150/100 (different conditions).
    - If chord is so close or so far from melody - decrease total cost by 200.
2. If there is empty space in a melody where chord occurs:
    - If chord is SUS - decrease total cost by 400.

Parameters:

1. Initial population = 10 (should be more than 1)
2. Mutation probability = 80%
3. Number of iterations per run = 200

4. Epochs (or number of runs of the algorithm) = 25

How to run code:

1. Install mido / nusic21 / numpy libraries
2. Run DamirNabiullin.py
3. Print the file name of the midi file into console (e.g. input1.mid)