Introduction to Artificial Intelligence Assignment 2: Accompaniment Generation

Report:

The task is to create a program that uses an evolutionary algorithm to generate an accompaniment for a given melody in midi format.

In my project, I used the `mido` library to work with files with the `.mid` extension.

The program I wrote has all the features of an evolutionary algorithm, below I will explain the structure of the population and a few key functions.

Population structure:

A population is a list of dictionary-type objects, each of which is an individual in the population

Each individual is a list of dictionaries containing 3 fields:

- **notes** - a list of numbers representing notes that are played during some interval `tritone_len`
- **time** - the start time of the cord that will play some constant time `tritone_len`
- **tritone** - three numbers that represent 3 notes that form a tritone which is played some constant time `tritone_len`

Time interval `tritone_len` is calculated as two quarters of a tact. The quarter value can be obtained from a midi file. If there is a note longer than or equal to 1/2 tact, then `tritone_len` will be equal to the length of the tact.

Fitness function:

Let's see how fitness value is calculated:

Fitness value is calculated as sum of multiplications of `consonant_score`, `cord_score` and `melody_score` for each time interval for some individual.

`consonant_score` for certain time interval is calculated in following way:

- For each note from the melody, check whether it is consonant with all the notes from the corresponding tritone. Consonant notes are notes whose distance in semitones is equal to 0, 3, 4, 5, 7, 8, 9 modulo 12.
- Summate number of consonant notes and then divide this value on number of conducted checks.
- The resulting ratio would be ≤ 1 and would be our `consonant_score` for certain interval.

`cord_score` for certain time interval is calculated in following way:

- Initialize `cord_score` equal to 0.
- Sort all the notes in the tritone corresponding to a certain time interval.
- Calculate the intervals between notes in semitones. Let's assume that our notes are as follows: n1, n2 n3, where n1>n2>n3. Then the intervals will be as follows n1 - n2, n2 - n3, n3+12-n1.
- Check if all the intervals are taken from the list of consonant intervals: 5, 7, 3, 4, 8, 9.
- For each consonant interval add 1 to `cord_score`.
- If any of the following conditions are satisfied, then assign 0 to `cord_score`:
    - If any of intervals is not consonant.
    - If difference between greatest and lowest note is more than octave (12 semitones).
    - If any of notes in tritone is the same.
    - If any integer values corresponding to the notes in the tritone are greater than `octave + DELTA` or less than `MIN_NOTE - DELTA`,
      where `octave = (min_note // 12 - 2) * 12` - some offset in semitones, which indicates in which octaves we can generate a melody,
      `MIN_NOTE` - the number of notes from the initial melody that has the smallest value,
      `DELTA` - number of semitones we can work with (in my program `DELTA`=24 or 2 octaves).

`melody_score` for certain time interval is calculated in following way:

- When executing the `retrieve_information()` function, all notes unique to the melody, whose values are taken modulo 12, are added to the `MELODY_NOTES` variable.
- Initialize `melody_score` equal to 0.
- For each note in the tritone taken modulo 12, we check whether it is in the list of `MELODY_NOTES`, and if it is present there, then add 1/3 to the value of `melody_score`

Crossover function:

The crossing is performed as follows: the population is randomly divided into pairs of parents, between whom the crossing will be carried out. First, the parent with the highest fitness function value is determined. Then, with a certain probability, the gene (tritone for certain time interval) of the first parent is transmitted to the child, otherwise the gene of the other parent is transmitted. In this way, the genes of the child are determined for all intervals of the melody.

Mutation function:

This mutation function adds a random number for each note in the individual's tritones.

Selection process:

Selection in the population is carried out after the crossing operation and the growth of the population size. It happens as follows: for each individual, the value of the fitness function is calculated, then the entire population is sorted by the values obtained and the best individuals are selected in the amount of the initial population size.

Change of the input monophonic midi file

If you want to generate accompaniment for some monophonic midi file you should change the `INPUT_FILE_NAME` variable in python code. You can use the absolute path to the file or just the file name if it is stored in the same folder as the code being executed. You must also specify the name of the output file in the `OUTPUT_FILE_NAME` variable.

```
INPUT_FILE_NAME= r'.../some_input_file.mid' # if you want to use absulute path
```

or

```
INPUT_FILE_NAME= 'some_input_file.mid' # if you want to use just the file name if it is stored in the same folder as the code being ex
```

```
OUTPUT_FILE_NAME = 'some_output_file.mid.mid'
```