Sergey Malyutkin. BS2-4

AI Assignment 2 Report.
Music Generation.

Parameters explanation

I have added some additional parameters to the standard PSO parameters, so it can execute faster: **MAX_START_ABS_VELOCITY**, **MAX_ABS_VELOCITY**, **PRECISION**. Their names are self-explaining, but it worth mention that all of them just limit PSO not to do ineffective actions.

PSO/Particle parameters

Population size – if it is too small, particles search for solution for too many iterations. If it is too big, then each iteration takes too much time. 48 +- 25% are good values.

Iterations – have to be big enough for PSO to have time to find solution if it stuck in some local optima, so I set it to 100K, but I believe that about 50K is just enough. The thick is that we actually we stop earlier due to precision constrains, so in average it takes no more than 5K iterations.

Precision – give the PSO ability to stop searching, when in reaches some precision in solution. By this I mean that best fitness becomes less than precision. Works great and does not have drawbacks. Due to usage of rounding and depending on my fitness function the best value for my algorithm has to be less than 0.5, for example 0.4 is fine.

Velocity limits – helps us to put constrains on velocity, so it will not accelerate for too much. Sample space is so small (48 values) that if the absolute value of velocity is greater that limit, then something went wrong and we have to cut it. You can think of it as a workaround, but this hack reduces average iterations count by half (at least on my realization).

Components – found out empirically but can be explained by this logic: we don't accelerate too much (**INERTIA** < 1), tend to our best (**COGNITIVE** about 2) and global best is just adds entropy (**SOCIAL** about 0.5).

How does it work

Since both **Particle1** and **Particle2** implements **IParticle** interface I was able to generify PSO, so it just takes array of particles as an input generation and executes almost standard PSO algorithm over it (difference is that I have added stop condition, when solution is found with some precision). Global best particle and global best fitness are just parameters in PSO object. On each iteration we update each particle, then find new global best values (if it exists) and then, if precision is reached we stop algorithm.

Particles Structure

Particle1 represents solution for finding chords sequence (accompaniment) and contains vector of **CHORDS_NUMBER** chords (3d vector of doubles), vector of **CHORDS_NUMBER** velocities (3d vector of doubles), **Tonality** object, current fitness (double), and best values (best chords and best fitness). On particle update we just pass global best to it and particle updates by standard PSO formula.

All values are doubles instead of integers, because we need for particles to update smoothly, otherwise we have to find really magic constants to work with (because sample space is just too small).

Particle2 stands for single notes (melody) and have small differences from **Particle1**. **CHORDS_NUMBER** is replaced by **NOTES_NUMBER**. It also stores accompaniment that melody have to match

Fitness function explanation

Consider, that 0 is the best fitness value and INT_MAX is the worst, fitness will never go negative. Then we can calculate particle fitness as the sum of distances to current appropriate value for each component of particle (for each chord or for each note).

Fitness calculations are mostly made inside **Tonality** object, outside of it we just restrict particle to fit in borders, not to repeat for too much and keep neighbors close enough to each other.

In **Tonality** object we firstly find out the closest value that we want to get and return absolute difference between current value and this closest value. For notes we just find closest note that fits tonality and chord. For chords we firstly find closest(to the first note of the current chord) first note that fits tonality, then build chord by tonality and this first note and then return distance from current chord to the just that we just got.

Misc.

All work with results of PSO work is done by the class **MidiWrapper**. It helps to easily convert notes and chords, and save and play midi values. Although, playing music by pure Java code is not the best idea, it helps to get raw view at the result. So to get the best experience it's better to use some music player (i.e. Windows Player will be enough).