Q320 Final Project

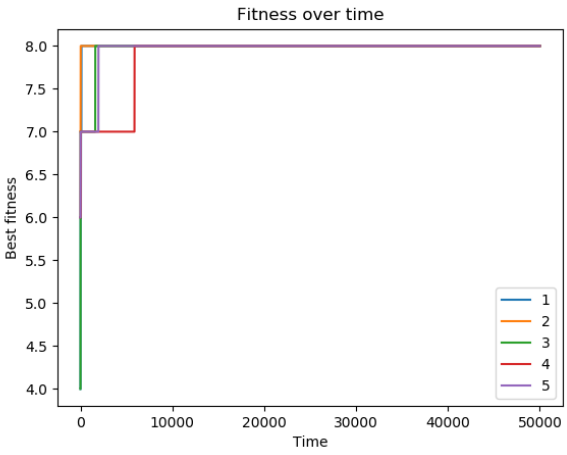
Robert Kellems

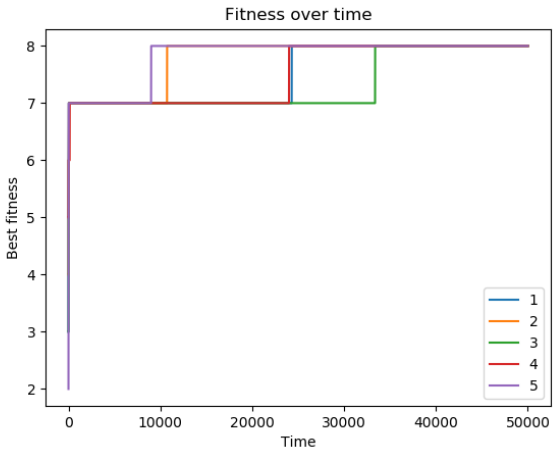
For my final project, I created a genetic algorithm that is capable of generating melodies based on certain parameters. In order to provide a way to better understand the melodies this GA produces, I also made sure that the melodies produced by the GA could be translated into MIDI and saved as ".mid" files, thus allowing one to both look at the melodies as actual musical notation and listen to the melodies. Music is probably my greatest passion, so I knew that I wanted my project to be related to it in some way; I settled on this model in particular because of the large room for experimentation it provides. This project is based heavily on this paper, written by Dragan Matić: http://scindeks-clanci.ceon.rs/data/pdf/0354-0243/2010/0354-02431001157M.pdf. Besides the general concept, I also utilized his method for constructing genomes.

In order to create a population of melodies, the Microbial class has a parameter (among other more standard GA parameters like mutation probability and population size) called "noteValue", which serves as the minimum note value (or duration of a note) for a note within each melody (e.g. if noteValue == ½, then the shortest note within the melodies of that population is a half note). Because each melody is supposed to be four bars in length, the length of each genome is 4/noteValue. The genes within these genomes are all integers within the range 0-14. 0 indicates a rest, or a moment of silence, 1-12 indicate each of the 12 notes in Western musical notation (starting with C), 13 indicates C at a higher octave, and 14 extends the duration of the note preceding it by noteValue with each successive 14 (this is done in lines 79-85 of "main.py" using a while loop and try/except to avoid indexing errors).

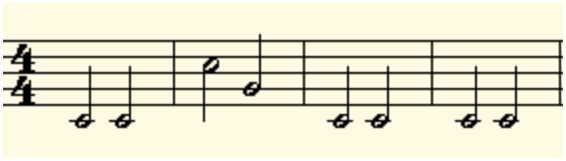
The process of evolving a melody that best fits the provided parameters is largely the same process as seen in previous projects featuring a GA: a population goes through a set number of tournaments in which two individuals are compared using a fitness function, with the losing genome undergoing mutation afterwards. I've created three different fitness functions that can be used: "noteFitness", which determines fitness based on which notes/integers appear in the melody/genome, "intervalFitness", which is based on the intervals between each note in the melody (the difference of two consecutive genes), and "bothFitness", which just combines the previous two. All three of these refer to a list (or in "bothFitness"'s case, a list of two lists) containing 15 binary values that correspond with each possible gene; for example, if I wanted to create a melody that would only include rests and C, the first two items in the list would be "1", while the rest would be "0". In regards to mutation, each gene has a chance of having 1 either added to or subtracted from it; this is done instead of generating a random integer because moving a note a half-step up or down tends to be a more effective and "human-like" way of eliminating dissonance in music.

Upon generating some melodies using "noteFitness" as a fitness function, ½ as noteValue, and value lists that would encourage melodies containing either only some of the notes in a major scale or some of the notes in the minor scale, I found that the GA was largely





successful. Almost every run created melodies with the maximum fitness (8 in this case) within 40000 tournaments (as shown in the graphs above; minor on the left side, major on the right side). The melodies based on the minor key seemed to reach maximum fitness at a much quicker pace; I believe this can be attributed to having more allowed notes for the minor melodies (more 1s in the value list). Although experiments with "intervalFitness" did not lead to maximizing fitness in the same way as "noteFitness", it still achieved its intended goal to a degree; when encouraged to produce melodies with notes clustered together, melodies with close intervals were produced. Similarly, "bothFitness" was mostly successful in generating melodies with both certain notes and intervals. Take this melody, created when directed to produce a melody within the major scale and with close intervals, for example: it mostly consists of a single repeated note, with the two outliers both being within the major scale, thus accomplishing both goals.



This shows that a GA can successfully create melodies according to set standards on which notes and/or intervals should be included. However, there are some challenges I have noticed: with smaller values for noteValue (e.g. ⅛), the increased length of the genomes leads to decreased effectiveness, since it's much more difficult to create a melody with no "wrong" notes when there are more notes to work with. The randomness inherent in a GA also leaves many of these melodies sounding somewhat aimless and unimpressive, although this problem could certainly be fixed by providing the GA with chords to a create melody within (thus possibly lending some harmonic direction) or by creating a more advanced version of "intervalFitness". Regardless, I view the project as largely successful in fulfilling its intended purpose.