Austin Lee
CS352
Final Project
Evolutionary Algorithm to Evolve Music
12/5

Problem Statement

For my final project, I created an evolutionary algorithm that generates short and easy musical compositions out of random notes. The general problem of my project was to find a way to quantifiably determine how "good" a composition sounds to assign fitness. Naturally, this is very arbitrary, as something that sounds good to one person may sound bad to another, or in this case, a composition with high fitness may sound bad to an actual listener. To solve this problem, I propose a solution that utilizes both evolution techniques based on pre-determined optima and human assisted interactions.

Related Work

There is already a large amount of research on the use of evolutionary algorithms for creating music, one of the earlier being [2]. In this paper, scientists propose a general algorithm for taking an initial note pattern and then evolving it to try and match a pre-determined final note pattern. Generally, the algorithm will first choose a thematic note pattern, then run the genetic algorithm on the pattern, check if the results are acceptable, then repeat if they are not. Finally, the algorithm returns a final string of music containing the notes from each evolution step. The algorithm models thematic bridging, which is a musical term that refers to the transformation of a music phrase or pattern over time. The algorithm makes use of multiple operations such as deleting notes, mutating notes, and rotating the pattern of notes. This work does not consider different note values, multiple octaves/clefs, or special rhythms.

[2] proposes a genetic algorithm for creating a short music composition that includes both pitch and rhythm. The genetic algorithm used in this paper utilizes genomes made of arrays of numbers that represent both pitch and duration that create short compositions of music. A scale is determined by number from 1 (middle c/c4) to 14(b). 0 is a rest. The fitness or musicalness of the composition is determined by the intervals between each tone and how different they are from a set of reference values and pre-defined "bad" tones. Algorithm structure is similar to that proposed in [1]. Overall, the algorithm can create a composition that may sound unusual but has a meaningful rhythm. While my current project does not include a feature for finding the fitness based on rhythm, it may be helpful for future endeavors.

[3] aims to create new musical phrases/compositions based on existing work. This algorithm follows a basic design- first defines a large set of patterns or motives used in musical composition, then evolve these patterns over an evolutionary algorithm, then create new patterns/motives from each bit, and finally join each part into one final composition. Each genome is represented by pitch combinations (numbered 1-12), which are then chosen to be fit based on how "good" they sound. The goodness factor is determined by a physical person, and acts as the fitness for the genome. Therefore, genomes that sound good the person will be more likely to appear in the next generation. Interactive evolution is exactly the process that I followed in this project. As determining how good a composition sounds based on music theory principles or some mathematical equation is, in essence, impossible, it will be necessary to include some form of interactive evolution with a person.

[4] provides another example of interactive evolution. In this paper, researchers create a functional scaffolding for musical composition. It explores the idea that note patterns and rhythms are functionally related. The user participates in interactive evolution by exploring possible accompaniments, both man-made and computer-generated, in different tracks of music and determines which sounds the best together. The researchers concluded that the listeners could not distinguish between these computer-generated accompaniments and ones written by people. With this in mind, it is definitely possible to generate a track of music that sounds like it could have been written by a person.

Methodology

The main aspects of my program were the fitness function (music_fitness()) and my evolutionary algorithm (evolutionary_algorithm()). My fitness function measured different aspects of each genome, awarded "fitness points" based on how accurately those aspects reflected pre-determined criteria, and returned a sum of those points as a final genome fitness. For the scope of this project, each genome was initialized as an array of size 16 (4 measures, 4 beats each) with each bit being a random integer between 0-13 (where 0 is a rest and 1-13 are C4-C5, Fig. 1).

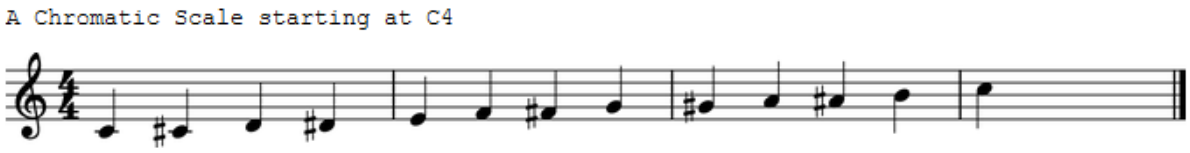

Figure 1: A Chromatic Scale starting at C4, or middle C.

The fitness of each genome was based on the similarity between chord progressions and simple musical patterns. I used 3 main criteria, similarity to notes that appear in common C chords, different repetitions within single measures, repetitions between two different measures, and if the measure was generally going up in pitch or down. For similarity to common chords, I used C chords since gene 1 starts at C. These are C major, D minor, E minor, F major, G major, A minor, B diminished, C major 7^{th}, D minor 7^{th}, E minor 7^{th}, F major 7^{th}, G dominate 7^{th}, A minor 7^{th}, and B minor 7^{th}. My function found the hamming distance between the genome and these chords to quantify the differences between them. Lower hamming distances signified that the genome was close to a chord. For this function, similarity to a chord signified that the genome "sounds good". Next, my function determined repetition in single measures and two different measures. These methods were simple as the measures were short, so they were essentially a lot of if/else statements. For this function, high repetition signified that the genome "sounds good". Finally, my function determined if the genome's notes were increasing or decreasing, and by how much. In music, it is uncommon for consecutive notes to harmonize well. For instance, the note progression "C, C#, D" sounds "off", whereas "C, E, G" sounds "good". Based on this, I created my function such that if a genome's notes were increasing or decreasing by 1 half step, then it would subtract fitness points, however progressions that increase or decrease by 2 half steps, or 1 whole step, would contribute to fitness points. Therefore, if the next note in a progression was 2 steps higher/lower than the previous, then that would signify that the genome "sounds good". Finally, the function returned a tally of the fitness points awarded to the genome.

I designed my evolutionary algorithm similarly to previous assignments. First, I initialize a population. These were the randomly generated genomes of length 16 and bits between 0-13. Next, over each generation, the new population first inherited the parents' genes then one gene was mutated. My mutation was kind of different from previous assignments, as each bit had a chance of becoming and of the other bits. If the chosen gene was a 0 (a rest), it would randomly switch to any of the other values. If the chosen gene was a note (1-13), it would have a 60% chance to move up or down a half step, a 25% chance to move up or down 2 half steps, and a 15% chance to turn into a 0.

After mutation, each newly created child would be evaluated for fitness as per my fitness function. Every x generations, however, the user would be able to pick their favorite compositions. Top choices would be awarded significantly more fitness points; therefore, their general shape of notes would appear more often. Implementing interactive evolution allowed me to circumvent some of the "handwaveyness" of my original fitness function, as the user has the most say in what sounds good or not.

Finally, the population was cut in half based on basic truncation tournament selection, leaving only high performing individuals to move onto the next generation.

Results and Discussion

Overall, I think the final evolutions of the music turned out fine. I tested two main aspects, one being the fitness based solely on the fitness function and the second being the fitness based on both the fitness function and the interactive evolution element. I found that fitness based on only the fitness function was easy to test, as it was simply checking if the composition evolved into something favoring the pre-determined criteria, however, gathering data for fitness with the interactive element was kind of difficult. For each test, I only interacted with 10 compositions per 20 generations, with the number of parents/children being limited to 10. Of course, more generations and runs meant more interactions, therefore I tried to limit the total generations and number of runs as well. This means that I am missing a significant chunk of data for increased generations, increased populations, and increased parents/children. Note that while multiple runs were completed using the interactive evolution function, many were separate and not done at the same time.

For the fitness function without the interactive element, I found that the composition ended up highly resembling the patterns and keys described in the original fitness function. After many generations, many of the sharps also disappeared (Fig. 3). This is probably because of the key structure I used, as C major does have any sharps/flats in its key. As time/generations increased, fitness increased whereas diversity decreased. As I tailored my algorithm to pick the highest fitness individuals, this is as expected. Individuals that satisfy more musical criteria are awarded higher fitness. Diversity decreasing was also as expected, as the genomes converged on individuals with higher fitness.

For the fitness with the interactive element, I found the results to be similar to that of the function without the interactive element. Essentially, as time/generations increased, fitness increased, and diversity decreased, presumably for the same reasons (Fig. 4). One big difference, however, was the final evolution of the interactive evolution compared to the noninteractive. The interactive final evolutions contained more sharps/note variation and varied patterns (Fig. 5). This is probably due to the higher influence the user has compared to the rest of the fitness function. Fitness tended to be a

little lower in interactive evolution experiments, potentially due to the user picking “suboptimal” compositions based of the fitness criteria.

Future Work

Due to the scope of my project, I am missing quite a lot of different musical sequences, patters, and popular scales. As my function mainly utilized the C major key, adding other keys would lead to some interesting results. For example, evolving over D major sequences may cause genomes to have more F#s and C#s within their genes. Hardcoding this may be a little tedious but implementing some sort of function to emulate the Circle of 5ths, a musical diagram to show key signatures and their corresponding major and minor keys, might be feasible (Fig. 6).

In addition, mutating additional genes over each generation may lead to interesting results as well. I only mutated one bit per genome per generation in these experiments, however I suspect that it will only cause the genomes to reach higher a fitness quicker. Similarly, mutating more genes at the beginning and less at the end might also be a potential branch and vice versa, as it may lead to increased or decreased diversity.

Finally, one of the original goals of my project was to evolve for rhythm significance. As of now, my program only includes quarter notes and no other variations (such as eighth or whole notes). Originally, I wanted my genome to feature a method to account for these variations; reptations of notes would signify longer notes. However, I soon realized that determining a fitness for this would be totally arbitrary, in that it would be impossible to tell if two quarter notes were more fit than a single half note. The other fitness criteria, while rudimentary, were grounded in music theory concepts, but there is no way to scientifically deduce if one sequence of beats is better than another. Of course, an easy solution to this issue could be using a neural network of popular and famous music data and evolving the genome based on that data, similar to what I did in one of the homework assignments. This would also probably reduce the arbitrariness of the rest of fitness criteria, as the evolved music compositions would ideally begin to sound like the data set.

Figure 2: Fitness and Diversity over time using the evolutionary algorithm without the interactive element. Experiment conducted 10 runs, 50 parents/children, and 100 generations.

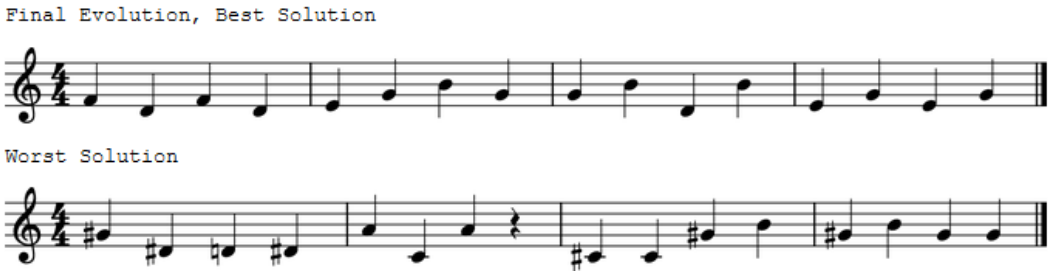

Figure 3: The best and worst solutions based on fitness function without interactive element.

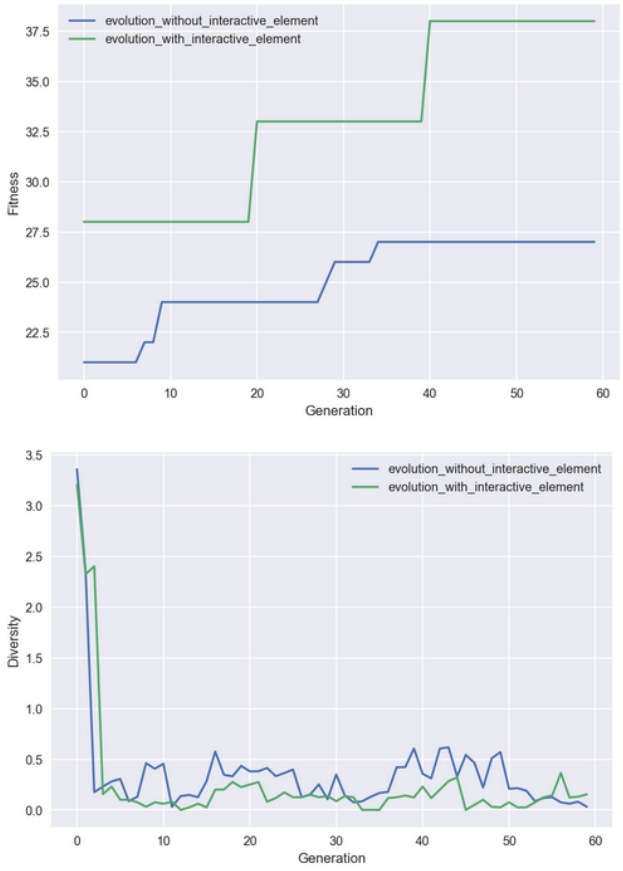

Figure 4: Fitness and Diversity over time using fitness functions with and without the interactive element. Experiment conducted over 1 run, 5 parents/children, and 60 generations.

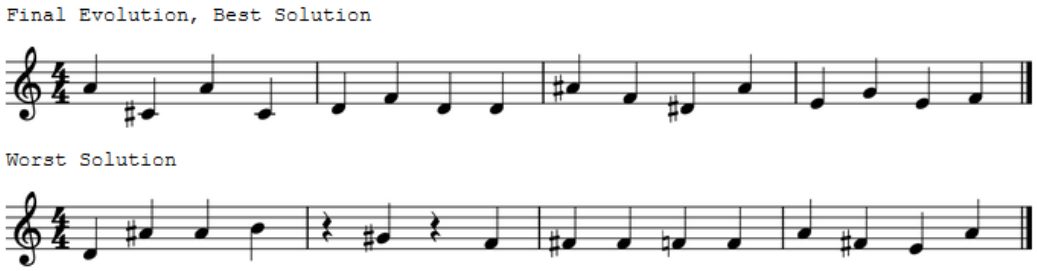

Figure 5: The best and worst solutions based on fitness function with interactive element.

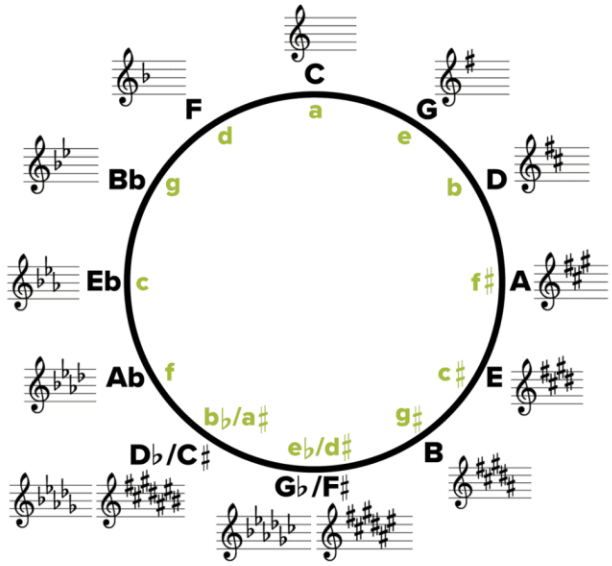

Figure 6: The Circle of Fifths.
Sourced from https://www.musicnotes.com/now/tips/circle-of-fifths-guide/

References

[1] D. Matic, "A genetic algorithm for composing music," *Yugosl J Oper Rres*, vol. 20, no. 1, pp. 157–177, 2010, doi: 10.2298/YJOR1001157M.

[2] A. Horner and D. Goldberg, "Genetic Algorithms and Computer-Assisted Music Composition," *Urbana*, vol. 51, pp. 437–441, Jan. 1991.

[3] B. Jacob, "Composing With Genetic Algorithms," Sep. 1995, Accessed: Oct. 24, 2021. [Online]. Available: https://drum.lib.umd.edu/handle/1903/7474

[4] A. K. Hoover, P. A. Szerlip, and K. O. Stanley, "GENERATING MUSICAL ACCOMPANIMENT THROUGH FUNCTIONAL SCAFFOLDING," p. 8.