Composing a 12-Bar Blues Bass line using Evolutionary Methods

Kevin McMahon
Michigan State University
East Lansing, Michigan
mcmah111@msu.edu

ABSTRACT

This work introduces a system for generating walking bass lines that evolve over a given chord progression. A Genetic Algorithm is used to find a solution that satisfies a set of characteristics deemed necessary for high quality music. The results of the system for two popular Jazz tunes are evaluated and conclusions are drawn that point to a potential correlation between fitness and musical prowess for music generated with a GA.

KEYWORDS

Genetic Algorithm, Algorithmic Composition, Computational Creativity

ACM Reference Format:
Kevin McMahon. 2020. Composing a 12-Bar Blues Bass line using Evolutionary Methods. In *Proceedings of the Genetic and Evolutionary Computation Conference 2020 (GECCO '20).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Algorithmic composition has been a hot field in computer science since it's first published efforts in the 1970's. To date, there are hundreds of research projects in this field [1] that explore the broad canvas of music. Since this field is a subsection of Artificial Intelligence, the most recent advances in AI often find their way into algorithmic composition research and vice versa [1]. Some common ways to give computers the intelligence required to make music include Grammars, Machine Learning, or Evolutionary Computing [2]. Specifically within EC, Genetic Algorithms have played a huge role in shaping this field [1]. They are able to model evolution and generate results that other methods may not come up with due to their prowess on multi-modal problems [2]. The No Free Lunch theorem [3] is applicable as there is no clearly superior method of creating computer music. There is also no "best" set of parameters to use in a GA for music [4]. Music can be created that is coherent quite easily [5], but the amount of meaning it possesses is an entirely different story. Composition by humans generally requires a good deal of creativity. This is nearly impossible to replicate using a simple algorithm. [6] claims that while the creativity of a composer cannot be replicated, the "hard work" phase of composition can in fact be recreated by algorithms. Some may not view this field as useful or even possible given the relative complexity in creating music. Besides, a computer has no inherent meaning and must be programmed to gain meaning from what it encounters [6]. How can we program creativity if we don't even know how to describe it specifically enough [7] for a computer to replicate it? For what it's worth, I believe that algorithmic composition is a worthwhile field of study and has the potential to introduce new ideas to AI as a whole. In this project I aim to create machine music using as little prior information as possible to see what a computer can create when it is given no meaning and a set of parameters it must adhere to.

GECCO '20, July 8–12, 2020, Cancun, Mexico
2020. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

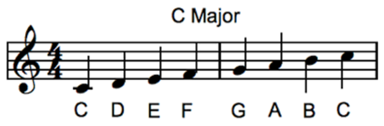


Figure 1: The pitches of the C major scale, shown on a musical staff in common time

2 BACKGROUND

2.1 Genetic Algorithms

Genetic Algorithms attempt to model the process of natural selection by evolving a population and applying operations on it's individuals. The evolutionary process takes place over a number of generations that is usually specified by the user. An overview of a general GA is shown in the blue section of Figure 2. Most of the steps of a GA can be generalized to any field they may be used. However, some steps like choosing a representation and evaluation metric require a good deal of time and domain knowledge to be successful [3]. The mutation and crossover operators seek to provide the population with some randomness and drive evolution beyond a local maximum. Parent and survivor selection will restrict the population and act as the "survival of the fittest" operators. The evaluation operator will assign each member of the population a value denoting the fitness of that individual. In general, more fit individuals will are more likely to be chosen for future generations of the GA.

2.2 Music

Western music consists of 12 notes using the letters A through G. Each note is uniquely identified by the frequency it makes when played by an instrument, also referred to as it's pitch. A note will also have a duration associated with it, which is a fancy way to say how long it will be played. The duration of notes follows a tree like structure, where whole notes have a duration of 4 beats, half notes get 2 beats, quarter notes get 1 beat, eighth notes get 1/2 of a beat, and sixteenth notes get 1/4 of a beat. For this project I will only be using quarter notes. In common time that corresponds to four equal length notes per measure. Figure 1 shows 2 measures of quarter notes in common time, where each measure consists of four quarter notes.

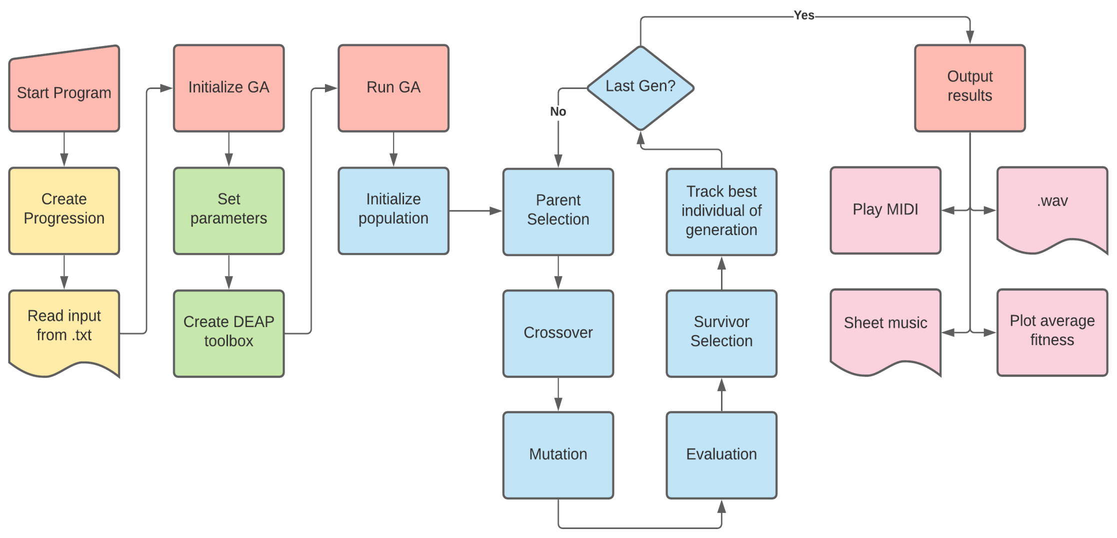


Figure 2: Architecture of the system used to test genetic operators and produce bass lines using a Genetic Algorithm

A single note on it's own is quite boring, but combinations of notes have the potential to be extremely exciting. As shown in Figure 1, a scale is a collection of notes that have something in common. In the case of the C major scale, the notes shown are all in the key of C (think of the song Do-Re-Mi from Sound of Music). The notes of a scale span an octave. Chords can be obtained by choosing notes from a particular scale: the C major chord consists of the notes C, E, and G.

A melody can be created by combining individual notes to form a musical sentence. A bass line is similar to a melody in that it is a combination of notes, but they are often much simpler and contain lower pitches than melodies. On the other hand, combining several chords will create a progression. Chord progressions are common in western music in general, but are particularly prevalent in Jazz and Blues music. These genres can be signified by their complex chord progressions and simple instrumentation. Bass is necessary in this music to keep tempo and set the tone of a song.

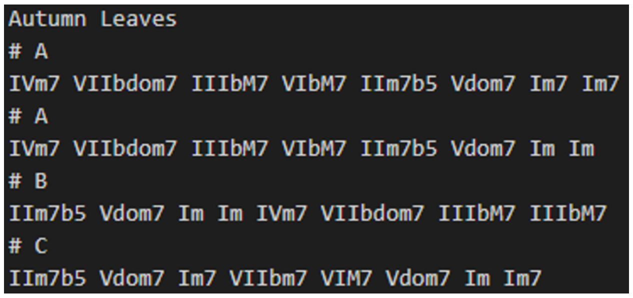
MIDI is a standard that allows musical instruments to connect and share information across interfaces [8]. In the scope of this project, MIDI is used to convert data structures in Python to audio played through the speakers of a computer. There are many pieces of software that interface between code and audio. For this there are several Python packages discussed in Section 4 that were utilized. Each MIDI note is assigned 3 values for pitch, duration, and velocity (more commonly known as volume) [8]. These values are represented as integers in computer code.

3 RELATED WORK

Ever since Holland introduced the Genetic Algorithm in the 1970's, applications in music generations have been plentiful [9]. Sources [9] and [1] contain dozens of examples using many methods in AI that go beyond the scope of this paper, so I will focus on prior work that directly relates to my research.

Much relevant GA work was done in the 1990's [10–13]. These papers introduced many relevant operators specific to algorithmic composition. Horner suggested using operators to move from an initial musical pattern to a final pattern, with the resulting shift being the melody that is created [10]. The evaluation function was algorithmic, so the human had no input on what music sounded good. A similar evaluation is used in [12] with each evaluation metric assigned a value of 1 or 0 for each individual. Although this work was done using Genetic Programming, many mutation and evaluation operators like fragment a melody, transpose notes, and repeat [12] are prevalent in many evolutionary methods of algorithmic composition. The GenJam system introduced in [11] takes a drastically different approach to the evaluation by having a human act as the fitness function. Biles notes that this approach worked well but that the fitness bottleneck of human evaluation was severe [11]. He also was one of the first to use a chord progression as input for a GA, which gave the algorithm a framework to evolve around.

Many researchers responded to Biles [11] revolutionary work by digging deeper into algorithmic evaluation methods. [13] introduced the use of restrictions on the mutation methods. In this field the mutation and evaluation methods are likely to be hand crafted due to unique representations, so Wiggins added a clause that prevents poor sounding musical gestures like diminished or tritone intervals [13]. McIntrye previously explored this area by using a tiered fitness function [14]. If some fitness parameters are not met, then some of the more lucrative fitness rewards will not even be checked. [9] rewarded individuals with some standard metrics like having a note on beat one and having a good mix of long and short notes. Much of this work deals with melody creation and uses complex representations that allow for different note lengths, but bass lines on their own have not been thoroughly explored.



```
Autumn Leaves
# A
IVm7 VIIbdom7 IIIbM7 VIbM7 IIm7b5 Vdom7 Im7 Im7
# A
IVm7 VIIbdom7 IIIbM7 VIbM7 IIm7b5 Vdom7 Im Im
# B
IIm7b5 Vdom7 Im Im IVm7 VIIbdom7 IIIbM7 IIIbM7
# C
IIm7b5 Vdom7 Im7 VIIbm7 VIM7 Vdom7 Im Im7
```

Figure 3: Autumn Leaves progression file

4 GA OVERVIEW

The overarching goal of this research is to create novel bass lines without using input or training data of any kind. I opted to implement my system using a GA due to it's relative ease of use. I wanted to utilize an algorithmic evaluation method and design my own mutation operator inspired by some of the aspects of existing work. The system layout is displayed in Figure 2. The program is started by running a Python script. The system creates a progression object by reading from a text file. An example progression for the Jazz standard Autumn Leaves is shown in Figure 3. I use a Python package called mingus [15] to take the chords from the file and convert to a list of lists in Python. This data structure is what the GA measures itself against for evaluation. The parameters for the GA are then assigned and the GA is executed. There are several ways to output the results of the GA. Plotting the average fitness is used for testing genetic operators while outputting results as sheet music and as a .wav file are used to output the best individual evolved over a number of generations.

I used the Python package DEAP [16] to prototype my proposed GA. DEAP has functions that handle many steps of a GA. I decided to use crossover and selection operators provided by DEAP so I could focus on the parts of the project dealing specifically with algorithmic composition. Below I discuss some of the GA operators that I had to design and code myself.

4.1 Representation

Representation is crucial in any GA, so I took some time to think about the best representation for my problem. I started off using a list of tuples to represent each individual, where each tuple represented a note of `(pitch, duration)`. Since the duration would be the same (4) for every note, my representation was simply a list of MIDI pitches. To trim the search space I restricted the MIDI pitches to the range `[24,48]` since they represent the common playable range of a bass. The two octave range is an idea first proposed in [17]. Initialization consisted of creating a list with length 4 times the number of measures in the input progression. To initialize individuals I filled a list with integers in the range `[29,43]`.

4.2 Mutation

Mutation was performed on the entire population directly after crossover. Each individual was randomly chosen for mutation with a predetermined probability. Those individuals who were chosen for mutation then had random notes transposed by a number of half-steps in the interval `[-7,7]`. This transposition operator is similar to the one discussed in [12]. To prevent the mutation from completely shattering an individual, I modified a specific note in an individual with a probability of 5%. Doing this ensured that only a few notes would be altered for each individual. To make sure that no crazy mutations occurred, I imposed bounds on the mutation that reset values to their original states if they went out of range or created intervals that were too large.

Table 1: Evaluation Characteristics and Weights

Characteristic	Weight
Non-repeated Notes	1
Note in chord	3
Note in scale	4
Note not in scale	-5
Leading tone	2
First note of bar is root	6
Walking pattern	5

4.3 Evaluation

There is a general guideline for creating bass lines that goes as follows:

(1) First note should be root of current chord
(2) Second note is a chord tone
(3) Third note is chord tone that's different from second note
(4) Last note is a leading tone to next measure

This is by no means a rule, but it forms the basis of my evaluation method. I used a method similar to [9] where good characteristics of a bass line were rewarded while bad characteristics resulted in a penalty. Each characteristic was assigned a integer for weight as seen in Table 1. These weights were hand tuned based on how important I believed each characteristic was for a good bass line. I decided to maximize fitness, thus most of the weights are positive. One thing to note about my evaluation method is that it scales with the length of an individual. In other words, longer progressions will lead to a longer representation length and a higher fitness relative to shorter progressions. I considered using normalization on a scale of 0-10 so I could compare songs, but I opted not to because it would dilute the importance of the weights for evaluation. I also deemed it unnecessary because the fitness is not a perfect reflection of the quality of a generated piece of music.

5 EXPERIMENTS

I wanted to run some experiments on my system to determine the operators and parameter values that would lead to optimal fitness values. The mutation and evaluation methods were kept constant for all experiments so only one operator or parameter changed at a time.

5.1 DEAP Operators

One of my research goals was to empirically test some crossover and selection operators in DEAP. I theorized that 2-point crossover would lead to better results than 1-point, so I used the 12-Bar Blues

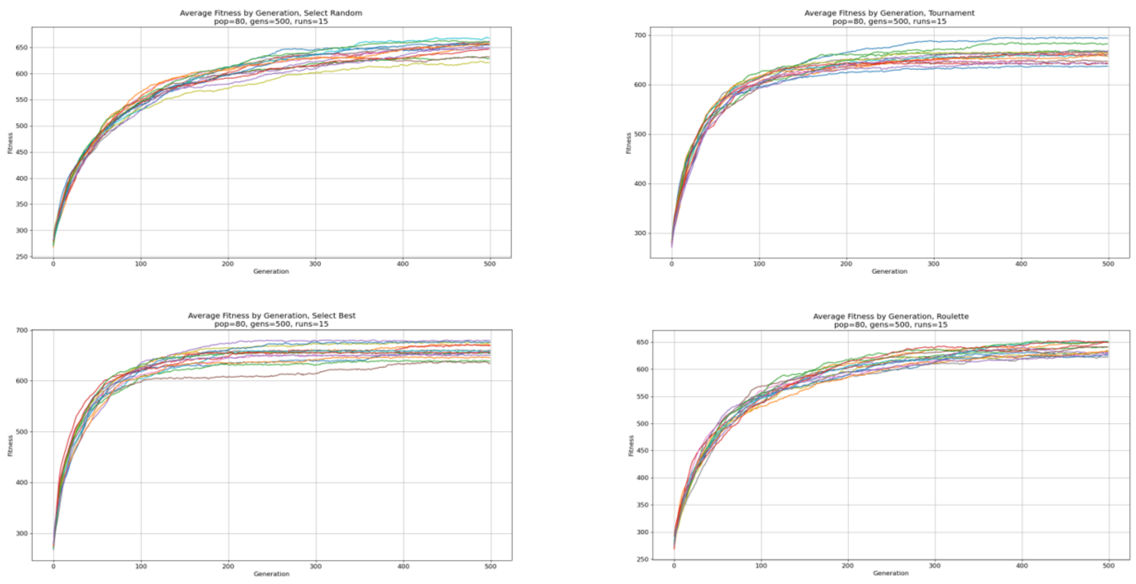


Figure 4: A comparison of selection operators from DEAP that were tested for parent selection. Clockwise from top left: Random, Tournament, Roulette, and Select Best

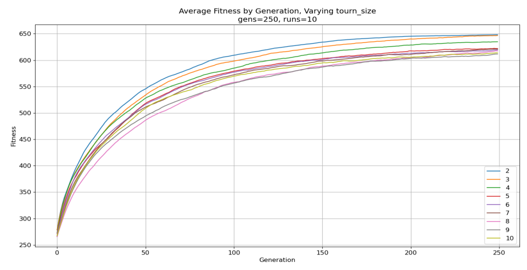


Figure 5: Testing fitness vs. tournament size for tournament selection

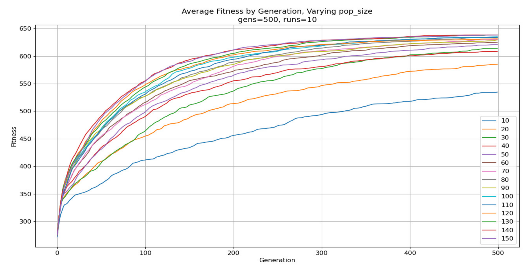


Figure 6: Testing fitness vs. population size

progression and generated 15 samples over 500 generations for each crossover method. The results show that there is a negligible difference between them. Since one is not more computationally complex, I decided to utilize 2-point crossover because I believe it may lead to more robust musical results.

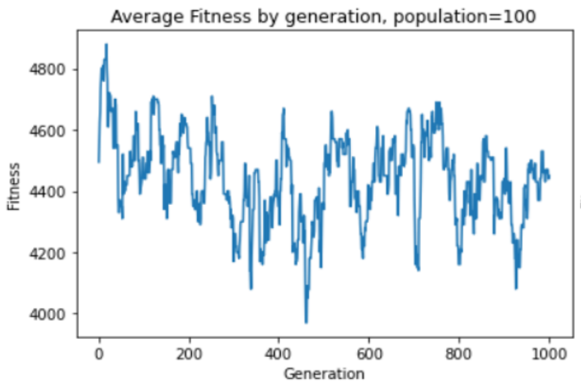
For the selection operators I settled on testing four that would work well for this problem: Random, Select Best, Roulette, and Tournament with size 2. I used the same testing setup as the crossover tests above. The results can be seen in Figure 4. Surprisingly, there was not a large discrepancy between these four selection operators. They all appeared to perform reasonably similar with an average fitness of 650. Note that the number for fitness doesn't matter much as it changes for every progression, but comparing these for one particular progression is meaningful. I opted to use tournament selection because it has the highest ceiling of the selection operators. I also wanted to test the tournament size to see if this result could be further optimized. As seen in Figure 5, the best tournament size to maximize fitness was 2 or 3. The slope of 3 was steeper as the number of generations increased, which suggests a tournament size of 3 would be optimal for more than 50 generations.

5.2 GA Parameters

The next step after choosing a set of operators was to find values for the parameters of the GA that maximize fitness.

Elitism. The first thing I tested was the effect of elitism on average fitness. I hypothesized that elitism would be necessary for this project in order to keep the strongest individuals and build future individuals off of the strongest of the whole run of the GA. I tested this hypothesis by using Select Best as my survivor selection method and comparing a μ, λ and $\mu + \lambda$ strategies. A plot comparing these two methods is shown in Figure 7. Not only does elitism result in logarithmic growth, but it also reaches an average fitness 20% higher than non-elitism. These results clearly show I should be using elitism in my system.

Population Size. I also tested the population size vs. number of generations. I decided to run the GA with population size ranging



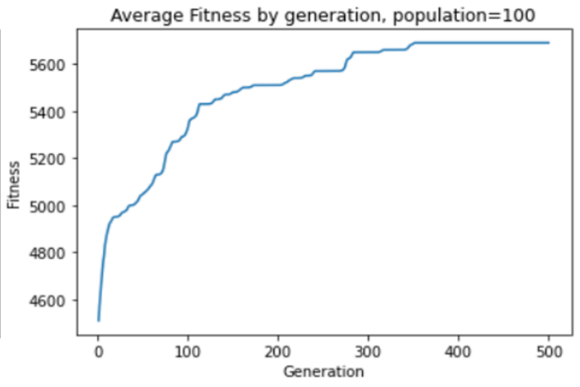


Figure 7: Comparing μ, λ *(left)* and $\mu+\lambda$ *(right)* strategies.

Table 2: Optimal Parameters for GA

Parameter	Value
Strategy	$\mu + \lambda$
Population Size	80
Parent Selection	Tournament
Tournament Size	3
Survivor Selection	Select Best
Crossover	2-point crossover
Hall of Fame Size	1
$P_{crossover}$	0.3
$P_{mutation}$	0.65

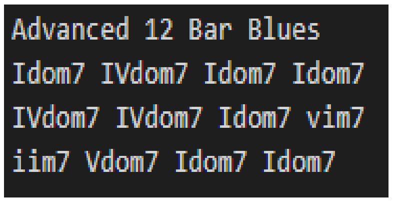
from 10 to 150 for 500 generations. I ran the GA 10 times for each population size and averaged them out to produce the plot shown in Figure 6. This plot took almost 4 hours to generate since it was a simple Python script that ran many times. Population size of 10 to 30 did not work well (as shown by the blue and orange lines in the plot). However, there appears to be a threshold to the fitness that can be obtained. The group of lines with the highest fitness represent contains the averages for all population sizes greater than 60. I decided that a population size of 80 would be appropriate since it provides a good relative fitness without wasting many CPU cycles for small fitness gains.

6 MAKING MUSIC

Now it is time to explore some of the musical results generated by my system. The final parameters I chose are shown in Table 2. I used the simple 12-bar Blues and Autumn Leaves progressions, shown in Figures 8 and 3 respectively, to showcase my system. I describe the methods and output for each progression in this section. Any overarching discussion points will be addressed in the Results section.

6.1 12-bar Blues

I ran several samples on the blues progression with the only variant being the number of generations the GA was ran. Some music results are displayed in Figure 9. The top sample has a fitness of



```
Advanced 12 Bar Blues
Idom7 IVdom7 Idom7 Idom7
IVdom7 IVdom7 Idom7 vim7
iim7 Vdom7 Idom7 Idom7
```

Figure 8: 12-bar Blues progression file

563 while the bottom sample has a fitness of 687. Both lines sound okay when played with the progression, but the 500 generation sample is clearly musically superior. Since this blues is in the key of C to reduce the search space, most sharp notes (denoted by the # symbol) will sound bad. The top sample is littered with sharps that don't sound good at all. However, the bottom sample has a few sharps in measure 8. This does actually sound good and can be explained by the chord chosen for measure 8 in the progression, which includes sharps. There is also a more definitive "walking" pattern seen in the bottom sample. Generating a walking bass line pattern was a core goal of my project, so I was very satisfied to see it prominently displayed after just 500 generations.

6.2 Autumn Leaves

The GA performed well on a simple progression, but how will it perform on a realistic progression that is more indicative of real-world usability? Autumn Leaves is a famous song in the Jazz community due to it's circle progression and smooth sound that reminds listeners of the peace of falling leaves. Since the previous progression produced quality results after 500 generations, I decided to expand and see if more generations would benefit the musical quality at all. Some sample lines produced using 500 and 1000 generations are shown in Figures 10 and 11 respectively. The number of sharps is not a valid metric for this song since it uses many fancier chords which contain many sharps. The saw tooth pattern is still prevalent in both samples, although it consists of some longer runs for the 1000 generation case. I believe this is a potential artifact of overfitting since it will result in a higher fitness value. But this is not the case! More generations actually does nothing to the fitness. The fitness for these two individuals are 1611 and 1590, but the time

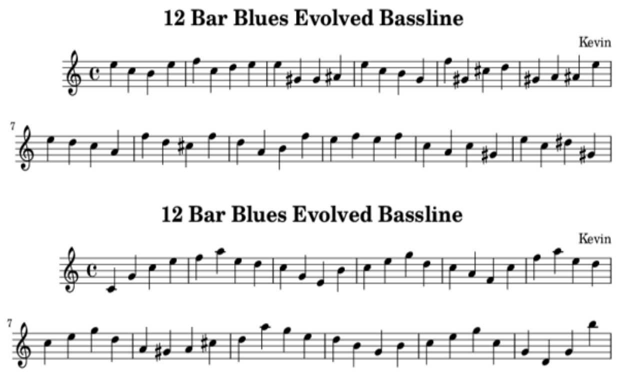


Figure 9: Bass lines generated for the 12-bar Blues progression. Top sample is evolved over 50 generations and the bottom over 500 generations

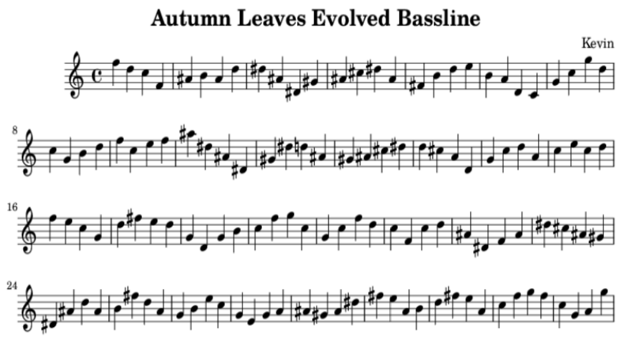


Figure 10: Autumn Leaves sample bass line, 500 generations

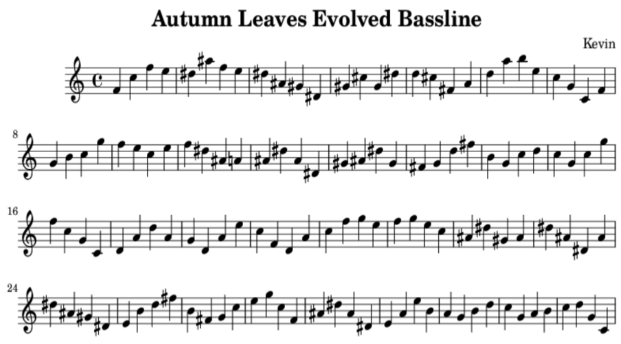


Figure 11: Autumn Leaves sample bass line, 1000 generations

required for 100 generations is 2.2 times more than 500 generations! In a blind test between the two samples, I actually preferred listening to the 500 generation sample.

7 RESULTS

The music samples show that there is a relationship between fitness and musical prowess. 10 generations is simply not enough to create

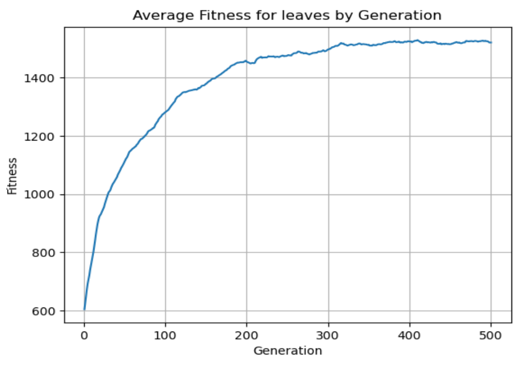


Figure 12: Plotting average fitness for a sample bass line with a population size of 80

any music worth listening to. However, the samples also suggest that there is a threshold for the musical results in addition to the fitness. I wanted to avoid sitting down and listening to hundreds of bad bass lines, so instead I generated a few samples for autumn leaves using the same GA ran for 100, 200, 300, and 400 generations to compare to the previous samples. I was astonished to hear that these low generation samples did not sound terrible compared to 1000 generations. This can likely be attributed to the random nature of the GA. Since there is no way to keep track of good sounding musical phrases, the bass lines for longer progressions tended to sound very random and lacked musical direction. This explains why the lines for the 12-bar Blues sound pretty good relative to Autumn Leaves, which is 2.3 times longer. I show the fitness plateau in Figure 12 for Autumn Leaves. Around 300 generations may be viewed as optimal since it provides a maximum fitness for the smallest number of generations.

On it's own this plot does not carry any additional meaning. When combined with the musical samples generated in Section 7, a different story may be told. It is undeniable that there is a fitness plateau for the GA, but I suggest that there may also be a plateau for the musical prowess of a bass line. The samples I generated indicate that this point occurs when the fitness begins to plateau. In Figure 12 this occurs around 300 generations. The specific inflection point will clearly change for different progressions or representations. However, I believe that this will hold true for many algorithmic composition tasks beyond Genetic Algorithms.

8 CONCLUSIONS

In this project I have explored algorithmic composition through the lens of a Genetic Algorithm. I created a software system that can take a chord progression and evolve a bass line that fits in the context of that progression. Some of the musical results are broken down and compared for varying number of generations. By looking into the relationship between fitness and musical prowess I deduced that there may in fact be a musical plateau in addition to the well studied fitness plateau.

There are many future extensions to this work, as algorithmic composition will likely remain largely unsolved for many years. Using multi-objective optimization is something I wanted to explore more but ran out of time as the semester came to a close. Instead of assigning weights to each evaluation characteristic from Figure 1, an algorithm like NSGAII can attempt to find a Pareto-optimal front that balances out these parameters. This would be tricky to do since there are more than 2 dimensions to take into account but would likely lead to improved results.

Another possible avenue to explore is altering the representation to allow different note lengths. This would make each individual a list of tuples and require the design of some enhanced mutation methods to split a note into smaller notes and combine smaller notes into larger ones. The evaluation would also have to be altered. This work should be done because in real life bass players regularly go beyond quarter notes.

Yet another extension is using a population of 'licks' that may be randomly plugged into an individual during the mutation process. A similar method has bee explored in [11], but I think adding in licks during mutation may lead to more lucrative musical results without a large amount of training data or human evaluation.

A APPENDIX

The GitHub repository evolutionary-music contains code, documentation, and additional plots for this project

REFERENCES

[1] Jose David Fernández and Francisco Vico. Ai methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research*, 48, 2013.

[2] George Papadopoulos and Geraint Wiggins. Ai methods for algorithmic composition : A survey, a critical view and future prospects. *AISB Symposium on Musical Creativity*, 1999.

[3] David H. Wolpert. No free lunch theorems for search, 1995.

[4] Róisín Loughran and Michael O'Neill. Evolutionary music: applying evolutionary computation to the art of creating music. *Genetic Programming and Evolvable Machines*, 21, 2020.

[5] Jon Gillick, Kevin Tang, and Robert M. Keller. Machine learning of jazz grammars. volume 34, 2010.

[6] Bruce L. Jacob. Algorithmic composition as a model of creativity. *Organised Sound*, 1, 1996. fdslflkfj<br/><br/><br/><br/>.

[7] Margaret A. Boden. What is creativity?, 2005.

[8] The midi standard, 1982.

[9] George Papadopoulos and Geraint Wiggins. A genetic algorithm for the generation of jazz melodies. *Knowledge Creation Diffusion Utilization*, 98, 1998.

[10] Andrew Horner and David Goldberg. Genetic algorithms and computer-assisted music composition. *Urbana*, 51, 1991.

[11] John a. Biles. Genjam: A genetic algorithm for generating jazz solos. 1994.

[12] Lee Spector and Adam Alpern. Criticism, culture, and the automatic generation of artworks. volume 1, 1994.

[13] George Geraint Wiggins, George Papadopoulos, Somnuk Phon-amnuaisuk, and Andrew Tuson. Evolutionary methods for musical composition. *In International Journal of Computing Anticipatory Systems*, 1998.

[14] Ryan A. Mc Intyre. Bach in a box: The evolution of four part baroque harmony using the genetic algorithm. 1994.

[15] Python mingus, 2015.

[16] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[17] Antonino Santos, Bernardino Arcay, Julián Dorado, Juan Romero, and Jose Rodriguez. Evolutionary computation systems for musical composition. *Mathematics and Computers in Modern Science - Acoustics and Music, Biology and Chemistry, Business and Economics*, 2000.