User-Interactive Algorithmic Composition

Kayla Gray

CSC 466 - Artificial Intelligence II

Professor Graci

State University of New York at Oswego

May 7, 2023

Abstract

This project experiments with the addition of more human collaboration in algorithmic composition. Oftentimes, the sole humans involved in the music compositions produced by algorithmic composition machines are the programmers who code the algorithms and fine-tune parameters. Music, being a medium for expression of the human experience, should involve humans in the composition process. This paper chronicles the semester-long development of a hybrid constraint-based system and interactive genetic algorithm algorithmic composition program that uses user rankings to determine the fitness of music samples in a population.

Introduction

The topic of artificial intelligence art is one of tense connotations, and it is rightfully so. Generally, the artificial intelligence artist is programmed to perform solo, undermining and perhaps even threatening the human artist in the process. In a similar sense, algorithmic music composition falls in contention with the human musician. Yet, this does not have to be the case; technology can be and should be used to optimize, as opposed to replace, the human experience. In the research I conducted on algorithmic composition for my honors thesis, I noticed a somewhat disheartening trend of minimizing human involvement – usually down to the programmers – in various algorithmic composition machines over time. I wanted to explore a

variant of the collaborative algorithmic composition machine proposed by Chip Bell to find ways to bring more humans, not just musicians and programmers, into the realm of music production.

The following section provides relevant background information pertaining to algorithmic composition with a focus on constraint-based systems and genetic algorithms. Then, a high-level description of my algorithmic composition program is presented, followed by relevant demos that depict the progress of the project throughout the semester. In the "Reflections and Conclusions" section, I consider my project experience through the lens of past, present, and future, and share my sentiments about the lessons learned during the undertaking of this programming project. The ultimate section of the paper presents a bibliography of referenced sources.

Background

Algorithmic composition is a method of composing music that relies on one or more algorithmic processes to create resultant compositions. The key component of algorithmic composition is the collaborative experience between machine and humans, of which there are varying degrees of human interaction (Maurer). The variety of approaches can be subdivided into indeterminate and deterministic approaches. Indeterminate, also known as stochastic, approaches use a degree of randomness to produce outputs that differ upon each run of the algorithm. Determinate approaches, on the other hand, result in the same outputs each time the algorithm is applied (Edwards 61).

The field of algorithmic composition has evolved from non-computer approaches to sophisticated computerized approaches over time. Mozart's *Musikalisches Würfelspiel*, or "musical dice game," was a purely non-technological approach to algorithmic composition that allowed players to create music merely through rolling some dice and choosing some playing cards. The game was the result of permutation, combination, and adherence to music theory (Maurer). *The Quadrille Melodist* is a similar variant of *Musikalisches Würfelspiel* that could create 428 million distinct quadrilles (Edwards 21). During the 1900s, composer John Cage pioneered the aleatory music movement, where probability and randomness were used to generate music compositions (Maurer). The first computer approach to algorithmic composition occurred in the 1950s: Lejaren Hiller and Leonard Isaacson created the *Illiac Suite*, which utilized Markov chains to create music (Edwards 61). Other approaches manifested in the digital era. CHORAL is a rule-based system that uses over 350 rules to generate compositions. David Cope's *Experiments in Music Intelligence* uses a synthesis of rules and grammar-related methods for creating music in the style of input compositions it trains on (Maurer). More modern AI tools include the likes of Amper, OpenAI Jukebox, and Soundraw, which create compositions featuring multiple instrumental parts using neural networks. More recently, Google released MusicLM, which generates music based on a text prompt inputted by the user (Bishop). Some other computerized methods include: constraint-based systems, genetic algorithms, and neural networks.

The two systems that are pertinent to the program that is the topic of this paper are constraint-based systems and genetic algorithms. Constraint-based systems feature a set of defined, system-wide constraints that outputs must adhere to in order to satisfy the "constraint satisfaction problem" at hand. Every element of the output must abide by the system constraints in order to be considered a solution to the constraint satisfaction problem. For instance, constraints that deal with harmony restrict the solution set of pitches that can be used based on the input melody of pitches (Anders and Miranda 30:4). To build off of the previous example, imagine the constraint satisfaction problem is as follows: given an input melody of pitches, return a solution melody of pitches that harmonizes across the octave with the input. If the input is the melody of C1D1E1, then the solution that satisfies the constraint satisfaction problem is pitches C2D2E2 , where C2D2E2 is in a different octave than C1D1E1. Constraint systems have been successful in the generation of melodies, harmonies, and counterpoint (Fernández and Vico 530).

A genetic algorithm is a type of evolutionary algorithm that uses mutation, crossover, selection, and copy to evolve individuals in an initial population. The fitness metric is used to measure which individuals are more likely to "survive" in the next generation of individuals. An initial population of individuals is generated and ranked by the fitness method. Then, crossover, mutation, and copy occur depending on the probability percentages assigned to these methods. These processes repeat until the next generation is populated. Successive generations are generated until the generation number reaches the pre-defined goal number of generations

(Burton and Vladimirova 60). There have been successful applications of genetic algorithms that have created melodies and jazz solos (Fernández and Vico 552).

This project was directly inspired by Chip Bell's design for a hybrid approach to algorithmic composition using Markov chains and a genetic algorithm in "Algorithmic Music Composition Using Dynamic Markov Chains and Genetic Algorithms." In his paper, Bell argues that the subjectivity of music preference is significant and should not be ignored in algorithmic composition systems. To incorporate subjectivity in resultant music compositions, Bell relies on a variant of a genetic algorithm, known as an "interactive genetic algorithm," where the fitness metric is the user (Bell 99). The system uses Markov chains to generate compositions featuring three instrumental parts – a melody, a chord progression, and a rhythm line – which make up the initial population of music sample individuals (101). Then, the interactive genetic algorithm is used to incorporate the subjective preferences of the user into the music samples of the next generation. The user chooses the two best samples from the population. Bell uses different linear algebra computations, since Markov chains can be represented as stochastic matrices, on the two top-ranked samples to create the next generation. This process repeats until the defined number of generations is met (102). I was interested in Bell's incorporation of user preference into the generation of music samples, so I tweaked the idea to make a project I could finish in a semester.

Program Description

Unlike Bell's system, this program utilizes a hybridization of a simplistic constraint-based system and an interactive genetic algorithm to produce music samples featuring two instrumental parts. The interactive nature of the genetic algorithm stems from the fitness metric based on user ranking of the music samples. The idea is to use technology to afford an opportunity for music production to people who are less versed in music without having the user program music themselves.

The constraint-based system is used to generate the individual music samples that make up the initial population of music samples. Each music sample consists of two melodies. Melody generation consists of the following steps:

1. A list of note durations (integer values) are generated based on a defined number of beats.
2. A pitch is generated for each duration in the aforementioned list. This differs between melody 1 and melody 2.
    a. <u>For Melody 1:</u>
        i. Randomly-chosen pitches from the C major scale are selected.
    b. <u>For Melody 2:</u>
        i. Pitch generation is based on a random selection of one of following choices:
            1. Harmonization
                a. Copy melody 1's duration list.
                b. Get the position of the first pitch in melody 1's pitch list based on its location in the C major scale. Then, add 2 to the position if it does not exceed the length of the list. Otherwise, subtract 2 from the position. Use

the new position to get the harmony pitch from the corresponding location in the C major scale..

c. Add the new pitch to melody 2's list of pitches.
d. Continue this process until melody 2's list of pitches matches the length of melody 1's list of pitches.

2. Octave Harmonization
    a. Copy melody 1's duration and pitch lists.
    b. Randomly choose an octave that is different from melody 1.
3. Permutation
    a. Copy melody 1's duration list.
    b. Randomly select a pitch from melody 1's pitch list.
    c. Add that pitch to melody 2's pitch list.
    d. Remove that pitch from melody 1's pitch list.
    e. Repeat until there are no pitches left in melody 1's pitch list.
4. Bassline
    a. Bassline follows the same random creation process as melody 1, except note durations are constrained to whole notes, half notes, and quarter notes, and there is a higher probability of generating pitches that are stepwise in motion (increase by a pitch interval of one).
5. Random

3. The octave is randomly selected.

4. The results of the aforementioned steps are initialized into “note” objects for each duration-pitch pair, which are added one-by-one to a list.

The two melodies generated by the steps above are placed into a “Music” object to represent the full music sample. The music sample and individual melody parts are outputted as ABC notation for user playback.

Demos

This section showcases demos of significant features of the project. They are as follows: melody 2 generation, mutation, crossover, interactive selection, double-crossover, and the final genetic algorithm.

```
----------------Individual 3----------------
X:1
T:Individual 3
C:Dystopian Tuesday
M:4/4
L:1/4
Q:1/4=120
V:S clef=treble name=S
V:A clef=treble name=A
%%score [ ( S ) ( A ) ]
K:C
V:S
%%MIDI program 0
G/2 D B F/2 G/2 E D2 F2 B2 A E/2 F2 D/2 E2 F2 G B2 A/2 B2
V:A
B, A2, F, A, B2, A, B4, A2, G4, G, F4, E,
NIL
```

Figure 1: Stepwise Bassline Demo

```
[1=2]> ( demo--octave-harmonization )
----------------Individual 2----------------
X:1
T:Individual 2
C:Dystopian Tuesday
M:4/4
L:1/4
Q:1/4=120
V:S clef=treble name=S
V:A clef=treble name=A
%%score [ ( S ) ( A ) ]
K:C
V:S
%%MIDI program 0
F' C2' F/2' G' E/2' E/2' C2' D/2' G/2' E2' E2' F' F/2' D2' D' G'
C2' A' E' C' B'
V:A
F C2 F/2 G E/2 E/2 C2 D/2 G/2 E2 E2 F F/2 D2 D G C2 A E C B
NIL
```

Figure 2: Octave Harmonization Demo

Both Figure 1 and Figure 2 are examples of how the constraint system has an impact on music sample generation. In Figure 1, the highlighted portion is melody 2, which was generated using the stepwise bassline method. Bassline durations are constrained to whole notes, half-notes, and quarter notes, which can be seen in the ABC notation highlighted in Figure 1. Additionally, the bassline is in a lower octave, denoted by "," in ABC notation. The notes also show stepwise motion with respect to the C major scale. Figure 2 showcases the octave harmonization in action. When octave harmonization is chosen for melody 2, the pitches, durations, and order of notes are constrained to that of melody 1; the only change is the octave. The second melody highlighted in Figure 2 depicts this adherence to the aforementioned constraints.

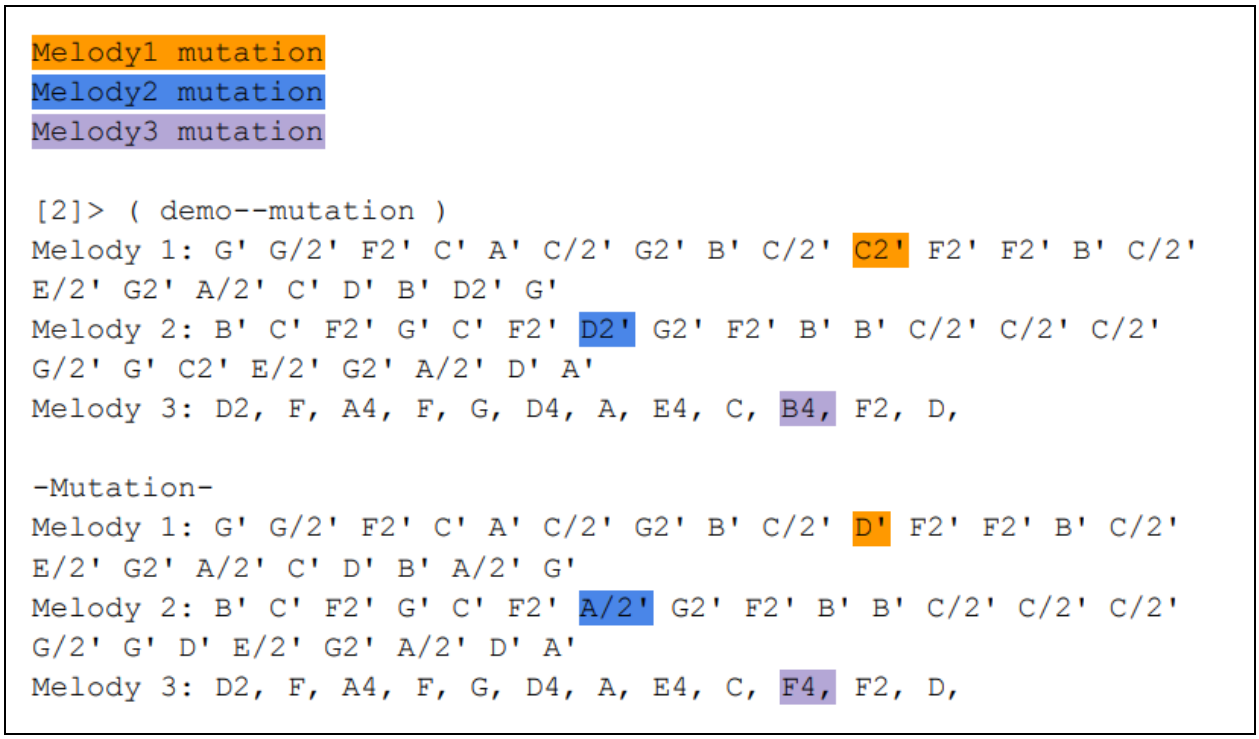

Figure 3: Mutation Demo

Figure 3 shows how a music sample is mutated in this implementation of the genetic algorithm. In an earlier implementation of this project, music samples consisted of three melodies. This was later switched to two melodies for simplicity. The highlighted notes demonstrate mutations–in each melody, the pitch and duration of a single note is changed.

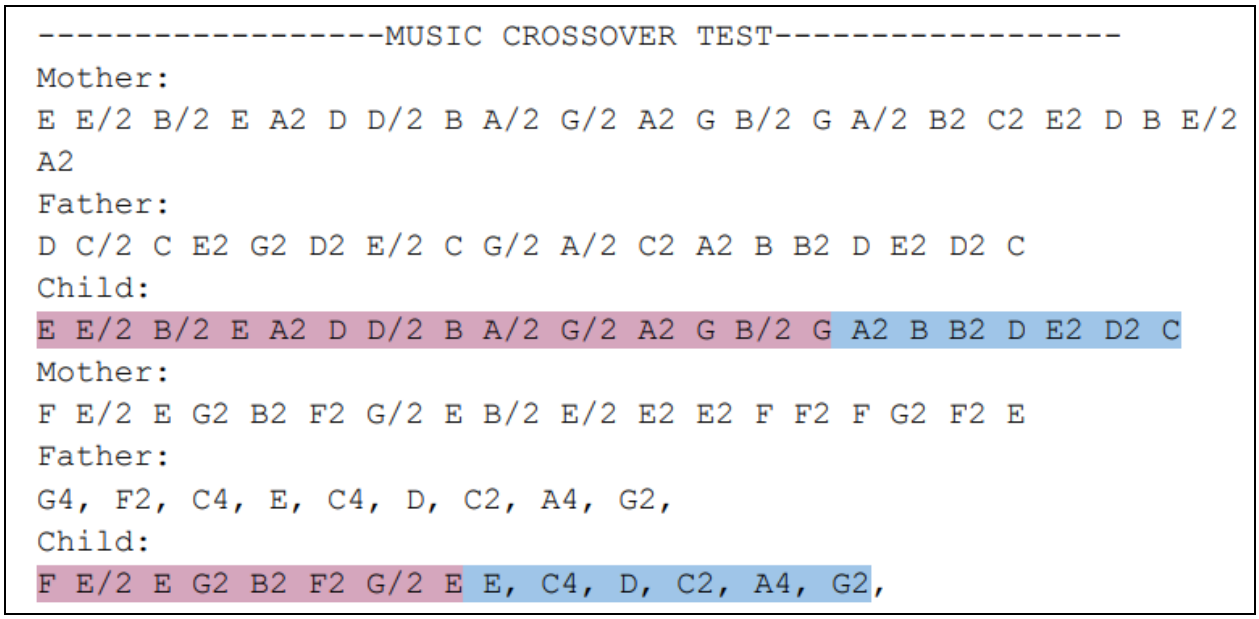

Figure 4: "Double-Crossover" Demo

Figure 4 depicts the method of "double-crossover" in the genetic algorithm implementation. The resulting child has four parents: the two top-ranked melody 1 individuals and the two top-ranked melody 2 individuals. The first *n* notes of the melody from the first parent and the remaining notes from the second parent are combined in crossover for melody 1 and melody 2, hence the name "double-crossover."

```
----------------Individual 26----------------
X:1
T:Individual 26
C:Dystopian Tuesday
M:4/4
L:1/4
Q:1/4=120
V:S clef=treble name=S
V:A clef=treble name=A
V:T clef=treble name=T
%%score [ ( S ) ( A ) ( T ) ]
K:C
V:S
%%MIDI program 0
G/2 D/2 G2 A/2 B G B/2 D2
V:A
B/2 F/2 B2 E/2 F B F/2 F2
V:T
E2, E2, D2, E2,
Melody 1 & 2 ranking (out of 10)? 2
Bassline ranking (out of 10)? 6
```

Figure 5: Interactive Selection - EasyABC Version

```
----------------Selection----------------
X:1
T:Selection
C:Dystopian Tuesday
M:4/4
L:1/4
Q:1/4=120
V:S clef=treble name=Melody1 snm=Melody1
V:A clef=treble name=Melody2 snm=Melody2
%%score [ ( S ) ( A ) ]
K:C
%%MIDI program 0
V:S
F/2 B/2 B B/2 D/2 G/2 D E2 G F C C F/2 G F B2 D2 C/2 D2 F/2 z4
|]
V:A
B2, A, B4, A, G2, F4, E4, D2, z4 |]
V:S
D F/2 A2 E2 D/2 C C D/2 E2 F2 F/2 F/2 B F E2 G F G/2 z4 |]
V:A
E C2 A E2 A G G2 A2 A/2 G/2 D/2 C/2 D2 F2 D C z4 |]
V:S
F/2 D2 C B C/2 D2 G/2 D/2 C2 F2 C B E C E D E2 z4 |]
V:A
C C G/2 F/2 D2 E C/2 C E D D2 D/2 B F2 C2 B E2 z4 |]
[Sample 0] Melody 1 ranking (out of 10)? 5
Melody 2 ranking (out of 10)? 4
```

Figure 6: Interactive Selection - MuseScore

Figures 5 and 6 show the user interaction interfaces for EasyABC and MuseScore, respectively. In the EasyABC version, samples, represented in ABC notation, are shown one at a time followed by a prompt for user ranking of the two melodies. In the MuseScore version, all of the music samples are combined into one score followed by prompting the user for every melody ranking consecutively.

A selection of music samples generated using this program can be found at http://cs.oswego.edu/~kgray3/CSC466WorkSite/dt.html. Discussion of the results of experimenting with the production of music samples can be found in the following section.

Reflections and Conclusions

There are three major issues I observed from my finished program. The first problem is the "user hostile" nature of the user interface. The program should not rely on some other third-party program, like EasyABC or MuseScore, for music playback – all of this should be handled within the program itself. This could be further improved with the implementation of a user-friendly GUI for playing back the music samples.

The next significant problem is user fatigue. Having the user rank all of the music samples in a population every generation is infeasible. I tried to mitigate some of this by alternating which generations a user ranked, but it still made for a tiresome experience. Luckily, researchers have found solutions to alleviate user fatigue resulting from interactive genetic algorithms. One solution involves using clustering and similarity metrics to determine which

music sample a user ranks; the sample that is most centric to the cluster is chosen for the user to rank each generation (Fernández and Vico 553). Neural networks can also be used to approximate some of the fitness rankings (Farooq and Siddique 48). One of the more difficult solutions to implement is to eliminate the task of users having to manually rank each sample altogether by monitoring the user's brain activity as ranking of fitness (50). This method is definitely more obscure in terms of plausibility, but it sounds incredibly fascinating to experiment with.

The third major problem I encountered is the "variation versus convergence conundrum." In this case, convergence represents a lack of variation in the individuals of a genetic algorithm, not convergence to the most fit sample. While conducting experiments in which I changed different variables of the genetic algorithm, I ran into issues where either too much variance or not enough variance was occurring. Too much variance occurred when I gave the user a break from ranking individuals in a population every one or two generations. Too little variance occurred in smaller populations – I ended up having to rank copies of the same individual repeatedly in experiments of that nature. Both of these problems share one great detriment: they negatively impact the user's collaborative impact on the final music sample. There is also an element of psychology that comes into play with this conundrum; I had a couple trials where I mistakenly thought the samples had converged, when, in reality, the samples were long enough where I did not notice mutations.

The aforementioned fixes to the user fatigue problems could help immensely with mitigating the variance versus convergence conundrum. It may be worthwhile to use some other algorithmic composition method to generate individuals in the style of a given individual or individuals – perhaps, by using a hidden Markov model – for mutation and crossover. Bell proposes a similar idea, instead using linear algebra to perform transformations on the matrices of the Markov chains of the two best-ranked individuals to build the next generation of individuals in the style of the parent music samples (Bell 102).

Overall, I am very satisfied with my work on this project. I went into it unsure of what would happen and came out of it with a finished program that enhanced my understanding of what it takes to create a collaborative algorithmic composition program. It is daunting to take on an experimental project that may not work out in the end. Ultimately, I found the experimentation to be incredibly rewarding, especially since I believe there is a need to create algorithmic composition machines that offer a collaborative experience for humans, rather than subverting their musicianship completely. Although I am hesitant to declare that this system offers an optimal collaborative experience, this project inspired me with some other ideas in designing collaborative algorithmic composition systems. I would love to look into more collaborative machines in the future.

Bibliography

Works Cited

Anders, Torsten, and Eduardo R. Miranda. “Constraint Programming Systems for Modeling Music Theories and Composition.” *ACM Computing Surveys*, vol. 43, no. 4, Oct. 2011, pp. 1–38. *ACM Digital Library*, https://dl.acm.org/doi/10.1145/1978802.1978809. Accessed 3 May 2023.

Bell, Chip. “Algorithmic Music Composition Using Dynamic Markov Chains and Genetic Algorithms.” *Journal of Computing Sciences in Colleges*, vol. 27, no. 2, Dec. 2011, pp. 99–107. *ACM Digital Library*, https://dl.acm.org/doi/10.5555/2038836.2038850. Accessed 3 May 2023.

Bishop, Kelly. “Is AI Music a Genuine Threat to Real Artists?” *VICE*, 16 Feb. 2023, https://www.vice.com/en/article/88qzpa/artificial-intelligence-music-industry-future.

Burton, Anthony R., and Tanya Vladimirova. "Generation of musical sequences with genetic techniques." *Computer Music Journal*, vol. 23, no. 4, winter 1999, pp. 59+. *Gale Academic OneFile*, link.gale.com/apps/doc/A168282960/AONE?u=oswego&sid=bookmark-AONE&xid=e7ebe010. Accessed 3 May 2023.

Edwards, Michael. “Algorithmic Composition: Computational Thinking in Music.” *Communications of the ACM*, vol. 54, no. 7, 2011, pp. 58–67. *ACM Digital Library*, https://dl.acm.org/doi/pdf/10.1145/1965724.1965742. Accessed 3 May 2023.

Farooq, Humera, and Muhummad Tariq Siddique. “A Comparative Study on User Interfaces of Interactive Genetic Algorithm.” *Procedia Computer Science*, vol. 32, 2014, pp. 45–52., https://doi.org/https://doi.org/10.1016/j.procs.2014.05.396.

Fernández, Jose David, and Francisco Vico. “AI Methods in Algorithmic Composition: A Comprehensive Survey.” *AI Methods in Algorithmic Composition: A Comprehensive Survey*, vol. 48, no. 1, 1 Oct. 2013, pp. 513–582. *ACM Digital Library*, https://dl.acm.org/doi/10.5555/2591248.2591260. Accessed 3 May 2023.

Maurer, John A. *A Brief History of Algorithmic Composition*, Stanford, Mar. 1999, https://ccrma.stanford.edu/~blackrse/algorithm.html.