Cohesive Music Generation with Genetic Algorithms

Paul Pigg

November 25, 2002

Abstract: *This paper will present an implementation of a genetic musician. The implementation will use two separate genetic algorithms to produce a piece of coherent music that is thematic, yet not overly repetitive.*

Keywords: genetic algorithms, music

1 Introduction

Over the past couple decades, the interest in using computers to perform very human functions such as the creation of music or painting has risen considerably. The reason for this is not so much practical usage as it is human curiosity. This paper will focus on music, in particular.

Much research has been done in this area of interest, with varying levels of success, as well as a broad spectrum of approaches[6]. Such approaches include: pure mathematical algorithms [3]; neural networks, which essentially *learn* a particular style of music; and genetic algorithms, which evolve according to the given *fitness* function, which has a broad coverage in approaches in its own right, including everything from an input voice to harmonize with[1], rhythm, consonance[2], and, of course, the human ear.

The main problem that most of these projects run into is that while the generated music is very interesting, it often lacks in cohesion. For example, a generated song may have very interesting movements, but there is very little that ties the different movements of the song together.

This paper will present a program which utilizes two different genetic algorithms to create a cohesive song. While the human user will still have to set up the initial layout of the song, it will be up to the genetic algorithms to

evolve populations that result in a coherent piece that is thematic, without being overly repetitive. Of course, there have been other works that have created thematic music [1], but this implementation will allow the user to set up the structure of the song, whereupon the genetic musician will create the notes to fill the song.

2 What Makes a Song Cohesive?

Songs are generally broken down into different movements. In modern music, these may be given names along the lines of *verse*, *chorus*, or *bridge*. These movements themselves will take on specific themes, with each separate version of the theme having a slight variation which is unique, yet similar enough to create a cohesive piece of music in the listener's mind. For example, the song *Eulogy* by Tool could be broken down as follows :

```
intro  A
verse  B
chorus C
verse  B
chorus C
bridge D
solo(s)E
chorus C
```

Obviously, a song requires a particular theme to hold throughout every piece of the song. However, an even closer tie must lie between the individual choruses, as well as between the individual verses. But if one were to play each chorus exactly the same way, and each verse exactly the same way, the song would be rather boring, differing only in the lyrics. To combat this, most musicians use variations of the same theme, with a basic root sound tying it all together. This is how we shall achieve cohesion in the end.

3 Two-tiered Approach

The program uses a two-tiered approach to build a song, in the vein of [1]. As mentioned before, a song can be split into separate movements, some of which are more closely tied with each other.

First, the user must give us a song layout to work with. This layout will give us the number of distinct movements, with their key, time-signature, and length in measures. The user will also input the layout of these movements, so that the program will know how to output the song, as well as the number of voices.

The program will then run a genetic algorithm for each distinct movement. This algorithm will implement the root voice, which all other voices will attempt to harmonize with. Then, a second genetic algorithm is run that creates the variation voices for each distinct movement.

Each of these genetic algorithms are explained in greater detail in the following section.

3.1 First GA: Creating the Root Voice

To begin our musical creation, we first will evolve a set of individual measures. For the sake of simplicity, certain parameters are kept constant for all measures in a population, such as key and time-signature. Since each separate piece of the song may contain it's own separate time-signature and key, we will evolve a separate population for every song element. That is, all verses will be in the same key, and choruses will all be in the same key; however, the key of the chorus is not necessarily the same as the key of the verse. For example, if we had five distinct movements, we would need five different populations of possible measures, with each population corresponding to a section of the song.

Then, phrases are built from these measures. Since these measures are all in the same population, there is enough genetic material tying them together for any of them to be used in conjunction with any other to create these phrases.

3.1.1 The Genomes

An individual in the population will contain two sets of genomes, the first relating to the notes in the measures, the second relating to the octaves of the notes. The alphabet of the first set of genomes consists of the 12 possible notes, a rest, and a *hold*. The hold is used in place of a note, to allow for different note-lengths in the song. For example, if a particular note in a measure is a quarter-note and there are eight eighth-notes in a measure, the sequence would be *note-hold*. Likewise, if it were a half-note, the sequence of

genomes would be *note-hold-hold-hold*. The alphabet of the octave genomes consists of four octave, plus a null octave, to correspond to spaces in the measure where an extend or a rest is present.

3.1.2 The Operators

Due to the fact that an individual consists of two separate pieces of "DNA", with different alphabets, genetic operators in this section will include domain-specific knowledge. Such operators will include Note Crossover & Mutation and Octave Up & Down Mutations [4]. The Note crossover and mutations will work only on the first set of genomes, while the Octave operators will work on the last set of genomes. Both crossover operators will work the same as in a typical GA, with the exception that they will work only on the appropriate sections of the individual's chromosomes.

Note mutations consist of Note Extend & Shorten operators, and note change operators. The Extend operator extends a note's length, overwriting the note following the note that is to be mutated. The Shorten operator will do the opposite, randomly selecting a note to put in the empty space. The note change operators will change a particular note, but will result in no change if the genome to be operated on is a Hold.

Octave mutation is Octave Up and Octave Down. The following tables show examples of how each operator works.

Extend Operator																
Old	A	D	rest	C	B	**B**	E	B	1	2	0	3	1	**2**	1	3
New	A	D	rest	C	B	**hold**	E	B	1	2	0	3	1	**0**	1	3
Shorten Operator																
New	A	D	rest	C	B	**hold**	E	B	1	2	0	3	1	**0**	1	3
Old	A	D	rest	C	B	**B**	E	B	1	2	0	3	1	**2**	1	3
Note Change Operator																
Old	A	D	rest	C	B	**B**	E	B	1	2	0	3	1	2	1	3
New	A	D	rest	C	B	**D♯**	E	B	1	2	0	3	1	2	1	3
Octave Up & Down Operator																
New	A	D	rest	C	B	B	E	B	1	2	0	3	1	**1**	1	3
Old	A	D	rest	C	B	B	E	B	1	2	0	3	1	**2**	1	3

3.1.3 The Fitness Function

The fitness of a measure is a result of three distinct fitness functions, each with an associated weight, as supplied by the user. These three smaller fitness functions work on Key, Octave, and note percentage.

The first part of the fitness function looks at the actual notes. Notes that are in the key of the piece are graded more highly than notes that are accidental. The fitness returned from this part is the total fitness of all notes in the measure divided by the number of notes. The function has the effect of allowing accidentals, but not often. Notes of the Major keys appear below.

Major Key	1st	2nd	3rd	4th	5th	6th	7th
C	C	D	E	F	G	A	B
D♭	D♭	E♭	F	G♭	A♭	B♭	C
D	D	E	F♯	G	A	B	C♯
E♭	E♭	F	G	A♭	B♭	C	D
E	E	F♯	G♯	A	B	C♯	D♯
F	F	G	A	B♭	C	D	E
F♯	F♯	G♯	A♯	B	C♯	D♯	E♯
G♭	G♭	A♭	B♭	C♭	D♭	E♭	F
G	G	A	B	C	D	E	F♯
A♭	A♭	B♭	C	D♭	E♭	F	G
A	A	B	C♯	D	E	F♯	G♯
B♭	B♭	C	D	E♭	F	G	A
B	B	C♯	D♯	E	F♯	G♯	A♯

The second part of the fitness function operates on the octaves of the notes. The mean octave of the measure is first calculated. Then, any notes that are two or more octaves away are given very low scores, notes that are 1 octave up or down from the mean octave receive a medium score, and notes that are in the same octave as the mean are given a high score. The total score is divided by the number of notes and returned.

The final part of the fitness function looks at the full measure and counts the notes in the measure. This number is then divided by the total possible number of notes; e.g. if there are eight eighth-notes in a measure, then the total possible is eight.

3.2 Second GA: Creating the Variation Voices

Now, we have the root melody to create variations from. To do this, a second genetic algorithm is introduced, which will evolve a population of individuals, each of which consists of one or more voices. Once this is done, the program will select individuals from the population to use as variations. Looking at the layout of the song, as inputted by the user, we can calculate how many variations we will need, and where to place them.

3.2.1 The Genomes

The Genomes of an individual will be nearly the same as the genomes of the individuals that the first GA works on. The main difference is that instead of representing only one measure, an individual will represent a whole movement. There will also be extra voices to take care of, instead of just one.

3.2.2 The Operators

The Operators will be identical to the first section, as well.

3.2.3 The Fitness Function

Again, we will use a multi-objective fitness function.

First, for each note in each voice, we will look up a consonance value in the following table to decide how well a particular note plays with the corresponding note in the root melody. The usage and reasoning of this table is discussed in [2].

note	A	A♯	B	C	C♯	D	D♯	E	F	F♯	G	G♯
A	30	16	22	27	24	10	8	28	26	25	21	19
A♯	16	30	14	5	27	4	10	17	3	26	2	21
B	22	14	30	20	23	27	24	29	9	15	26	25
C	27	5	20	30	16	22	7	26	28	17	29	18
C♯	24	27	23	16	30	14	22	25	13	28	11	29
D	10	4	27	22	14	30	16	21	12	6	28	17
D♯	8	10	24	7	22	16	30	19	1	12	13	28
E	28	17	29	26	25	21	19	30	19	21	25	26
F	26	3	9	28	13	12	1	19	30	16	22	7
F♯	25	26	15	17	28	6	12	21	16	30	14	22
G	27	2	26	19	11	28	13	25	22	14	30	16
G♯	19	21	25	18	29	17	28	26	7	22	16	30

Second, each note in a voice will be looked up in another table of chords. For simplicity's sake, only Major and Minor chords are used in this implementation. First, we look up the root note and if the note we are considering is found in the corresponding major or minor triad, it will be rewarded.

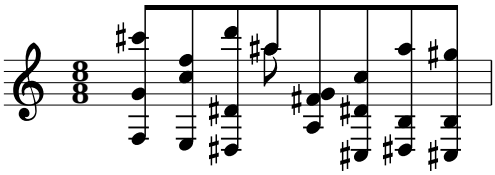
The last three parts of the fitness function are the octave, key, and note percentage fitness functions described in the first genetic algorithm will be used.

4 Results

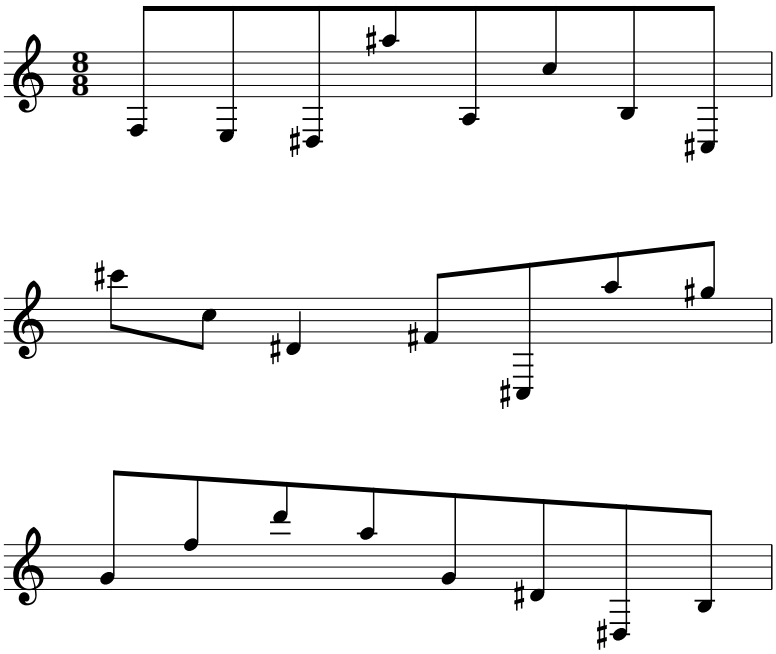
Since this implementation did not use any external input in the form of song sets, it was difficult to obtain music that differed very much in style. Most musical pieces created sounded very reasonable and, changes to the parameters were hard to notice or quantify, as the overall fitness of a piece depends on the human listening to it. However, even though the music is listenable and interesting, the voices of the pieces it creates have far stretches between the octaves which sometimes appear chaotic on sheet music. The reason this does not appear as much when listening to the piece is that the voices produced by the second genetic algorithm were meant to work together with the other voices more than its own voice.

Still, with very little musical background, the genetic musician was able to create some interesting pieces, a mere hint of which is shown in the following pieces. To really get a sense of the music the program creates, one should listen to a few examples [5].

This example piece is shown with all voices on the same staff.



This is the same piece, but with the three voices separated. Note that while there is a fair amount of chaos, there are definite patterns present, as well, such as in the third voice.



5 Conclusion

While the program is able to create reasonable pieces with very little musical background, it still requires the help of a human to set up the song for it. One possible extension would be to use a genetic algorithm or neural network to set up the layout of the song, as well as choose a key and time signature, and number of measures for each movement.

While the music created by the genetic musician is not necessarily emotional [5], it is generally easy to listen to. The aim of the project was to create thematic pieces genetically, which was accomplished.

References

[1] J. Biles. GenJam: A genetic algorithm for generating jazz solos. In ICMC Proceedings 1994. The Computer Music Association. 1994.

[2] E. Bilotta and P. Pantano. In search of Musical Fitness on Consonance. http://citeseer.nj.nec.com/416341.html .

[3] J. Greschak. Platonic Dice: Dodecahedron for 12 Woodwinds. http:// www.greschak.com/ m78mid.htm . 2002.

[4] M. Marques, V. Oliveira, S. Vieira, A.C. Rosa. Music Composition Using Genetic Evolutionary Algorithms. *Proceedings of the 2000 Congress on Evolutionary Computation*, pp.714-719, 2000.

[5] P. Pigg. Midi Pieces created by Genetic Musician. http://www.umr.edu/ pigg/musicGen/midi . 2002.

[6] G. Wiggins, G. Papadopoulos, S. Phon-Amnuaisuk, and A. Tucson. Evolutionary Methods for Musical Composition. In *International Journal of Computing Anticipatory Systems*. 1999.