Rule-Based Algorithms for Music Generation

Carla Llewellyn
Music and Computer Science
Simpson College
701 North C Street Indianola, Iowa 50125
riherd@simpson.edu

Abstract

The paper presents work done under a project assignment in a "Fundamentals of Computing III" class. It discusses a rule-based approach for music generation. In this approach, a melody is viewed as a sentence generated by some grammar. The grammar rules describe the internal structure of the melody to be generated. It is the task of the composer/programmer to choose how the constituents of a melody will be related and to describe this in the grammar rules. In the presented work, a melody is generated as a string of notes each with its duration by a program that acts as a language generator. These strings can be plugged into a web based program that plays the melody. The paper describes experiments with different grammar rules and presents an analysis of how music theory is used to create music that fits well together.

Introduction

This paper presents work done for a "Fundamentals of Computing III" class. For this assignment, the professor let the students decide what project they would like to do. As a music major and computer science minor, I decided to put my two areas of education together. With my professor's guidance, I picked the topic: Algorithms for Music Generation. I found that there are two broad approaches to algorithms: rule-based algorithms and artificial intelligence (AI) algorithms. Rule-based algorithms use a series of steps to come to a logical conclusion while AI algorithms use logic by "learning" which solutions are attainable by comparing previously stated data. Both of these can be used when generating music. If the program is made from a rule-based algorithm, it would use rules and basic melody writing regulations to compose the music. AI programs would "learn" what worked and what didn't by using what sounds good.

I was impressed by the work done at the Music Research Laboratory, University of Helsinki where a rule-based approach was used to generate music similar to the music of a well-known composer. So I decided to design and make experiments with a grammar that generates music using my own compositions.

I found two programs online that generated music from a series of input by the user. One used fractals and the other used mathematical algorithms like the Fibonacci sequence, powers, and constants to generate music. When I used the fractal program I found that the music was more polyphonic than the music made with the Fibonacci sequence, for example. The fractal program was more complex in nature; it allowed more input by the user, so the music generated was more my own work, while the music generated with the Fibonacci sequence could have been made by anyone because there weren't any choices to be made. In the latter program, however, there is a place where one can input their own series of numbers. This is where the generated string will be placed to listen to the ending result.

Music Theory

For this project, I composed several little, simple melodies in the key of C and in common time as the time signature. They do not exceed two octaves. The melodies as a whole consist of eighth notes, quarter notes, half notes, dotted half notes, and whole notes.

First I will explain what it means to be in the key of C. A melody in the key of C uses seven notes labeled A, B, C, D, E, F, and G. If you start on C and go up to B and repeat C again on the top, that is a scale in the key of C and it is also called an octave in the key of C. It will look like this: C, D, E, F, G, A, B, C. If you do this twice in a row, it creates two octaves: C, D, E, F, G, A, B, C, D, E, F, G, A, B, C and so on. These notes are the building blocks to the melodies.

Next I will explain common time or 4/4 time (read as “four four time”). The top number means how many beats per measure and the bottom number means what kind of note gets one beat. In 4/4 time, there are 4 beats per measure and the quarter note gets one beat.

The difference between eighth notes, quarter notes, half notes, dotted half notes, and whole notes will be explained here. An eighth note gets a half of a beat. A quarter note gets one beat. A half note gets two beats. A dot on any note means you add half the value of the note to the note itself, so a dotted half note gets three beats. A whole note gets four beats.

As a rule of traditional music theory, each melody should end on the note C, whether it is the top C in the scale, or the bottom one, it does not matter. Also as a rule, a melody should either start on C or the fifth note in the scale, in this case G. To make a melody, the notes are strategically placed in order to form a line that is pleasing to the ear. If the notes are just randomly placed, there would be no coherence to the music and wouldn’t sound “good” according to classical music theory.

Grammars

The program that outputs the final generated string is a language generator. It was given to me by my professor. It uses a context-free grammar. A context-free grammar is a formal system that uses recursive, rewriting rules to define languages and generate patterns of strings. It consists of terminal and nonterminal symbols, production rules, and a starting symbol. The formal definition describes a context-free grammar as an alphabet with a set of terminal symbols in which these terminal symbols are a subset of the alphabet. When a string is being generated, nonterminal symbols are replaced with strings of terminal and nonterminal symbols according to the grammar rules. Thus, each grammar rule can be viewed as describing the structure of the nonterminal symbol on its left side.

I use context-free grammar rules to describe the internal structure of the melody to be generated. In my particular grammar, there are 48 production rules and also a starting symbol. The starting symbol is defined as, S $\rightarrow$ Beg Mid End. Because of this, the rest of the production rules are split up into three categories, Beg (beginning), Mid (middle), and End. They are split into these categories because of how they would best fit into a melody. If a particular terminal symbol would be a good choice to start a melody, then it is placed in the Beg category. If it is a good connector piece or not fit to begin or end a melody, then it is placed in the Mid category. If it would end a melody well, then it is placed in the End category. Here is a sample list of the production rules:

S $\rightarrow$ Beg Mid End
Beg $\rightarrow$ 47'1 49'1 51'1 52'1
Beg $\rightarrow$ 52'2 56'2 59'2 57'2
Beg $\rightarrow$ Beg 52'2 54'2 56'2 54'2 Mid
Beg $\rightarrow$ 47'1 49'1 51'1 52'1 54'3 End

Mid → Beg 52'2 56'2 End
Mid → 54'2 56'1 54'1 52'2 47'2
Mid → Mid 47'2 52'2 56'1 57'1 56'1 52'1 Beg
Mid → 56'2 52'1 54'1 56'2
End → Mid Beg 54'2 51'2 52'3
End → 52'2 52'2 51'2 52'3
End → Beg 52'2 54'2 52'3 End
End → 52'4
End → 42'2 40'3
End → 49'1 47'1 49'1 51'1 52'3 End

I broke my melodies up into parts that I thought should stay together. Their size ranges between a half of a measure to a full measure. Each piece of the melodies I composed is defined in the grammar as a sequence of notes represented as pairs of numbers. The first number in a pair corresponds to a pitch on a piano keyboard (0-87). The second number corresponds to a duration. A 1 represents an eighth note, a 2 represents a quarter note and so on. The pitch and duration are separated by a single quotation mark.

Once the melody is generated as a string of these pairs of numbers, the program outputs the left sides of the pairs and the right sides of the pairs as two separate sequences. The first sequence is the string that is to be copied and pasted into the online program shown below. When I made my first experiment with this, I found that I could not enter the second string of durations into the program. The durations are not allowed as input by the user, but are randomly assigned by the program.

Experiments with an Online Music Algorithm Program

After the string is generated, it is then put into an online music algorithm program that will play the melody (http://musicalgorithms.ewu.edu/algorithms/import.html). Figure 1 shows a snapshot of what the program looks like.

To test the grammar program, initially I used a small set of production rules without specific nonterminal symbols for the beginning, middle, or end part of the string of notes. Once I found that the program worked, I added the rest of the rules and split them up according to whether they should begin, be in the middle, or end as described earlier. Then came the stage in which I just kept running the program, copied and pasted the numbers into the above online program, and listened to the results. They weren't bad; my rules didn't need much change. But sometimes I came across a couple of notes that didn't sound good played one right after the other. I had to find the first terminal symbol in the production rules and either change it, or add something after it that I knew would sound fine with it.

To add another element to the program, I tried, with the help of my professor, to add probability to the rules so that the terminal symbols that I liked best would be played more often. In the majority of experiments, however, the music generated by the

probabilistic grammar sounded monotonous with my favorite passages, while the non-probabilistic grammar gave better results.

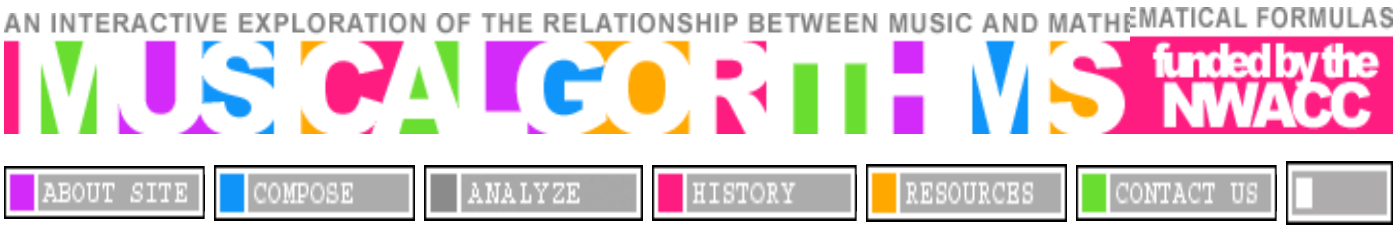

Figure 1: Online Music Algorithm Program

Conclusion

This program that generates a variety of different melodies from a set of production rules can be very useful to a composer who needs to add variety to a piece and is experiencing a sort of "writer's block". While listening to the generated music, I found many combinations of tunes that were pleasing and that I had not thought of. I intend to do many more experiments to fine-tune the grammar rules so that "good" melodies are always the result.

References

Music Research Laboratory at the University of Helsinki
(http://www.music.helsinki.fi/studio.html)

Context-Free Grammars, Lecture Notes on Theory of Computation as prepared by Dr. Lydia Sinapova
(http://faculty.simpson.edu/lydia.sinapova/www/cmsc365/LN365_Lewis/L06-CFGs.htm)