Programming Music: The Applications of Algorithmic Composition

MU208

Music Research

Acknowledgements

We would like to thank Mr. Brian Towner for his guidance in exploring hip-hop beats as a specific implementation of algorithmic composition and his assistance in finding previous attempts to automate music. We would further like to thank Col. Steven Warr for providing a thorough background in computer science, without which we would not have been able to complete this project, and constantly encouraging creativity and curiosity among his students. We would like to express gratitude to Gena Hughey of the Texas Music Teachers Association for inspiring passion for music and proving that true musicians experiment. Finally, we would like to express love and appreciation to our parents for their constant support.

Purpose

The purpose of this study was to evaluate the efficacy of musical generative grammars as a technique for algorithmic composition. Algorithmic composition is the process of using predetermined instructions to create music with minimal intervention from humans. Generative grammar, originally a term from linguistics, is a system of rules to combine sounds in such a way as to form music. Grammars have previously been used in conjunction with algorithmic composition in order to develop algorithms which could use pre-decided rules to create music. Most previous attempts to automate music have been little more than experiments, and none have reached mainstream commercial success. Today, every song on the radio was composed by a bona fide human being rather than by a computer program.

This paper sought to explore algorithmic composition through the creation of a computer program to create hip-hop beats. Hip-hop was chosen because it is more dependent on rhythm and less dependent on melody than other genres, simplifying the program's development. In the same way that not all human composers create high-quality music, the program was never expected to necessarily be able to compose well. The quality of music created by the computer program was deemed irrelevant. Instead the benchmark of success with generative grammar was indistinguishability from music created by a human.

Hypothesis

There are a set of grammars that govern the rhythm of hip-hop beats, which when used as the basis for an algorithmic composition model, can produce works indistinguishable from work created by direct human input.

Introduction

People live in an age of increasing automation. Everything from transportation to communication to the monetary systems which allow society to function are controlled by algorithms. However, creative works such as art, music, and film have traditionally been viewed as outside the grasp of algorithms. They are seen as fundamentally human in creation and design.

Nevertheless, the groundwork for algorithmic composition has existed since 500 B.C when Pythagoras believed that music and mathematics were not separate studies (Papadopoulos and Wiggins, 1999). In the 15th century, "canonic" composition relied on simple algorithms. Singers would be given a single voice part and instructions to derive additional voices as companions. For example, the companion voice would imitate the original voice a certain number of beats later or would perform the first part backwards (Schiltz and Blackburn, 2007)

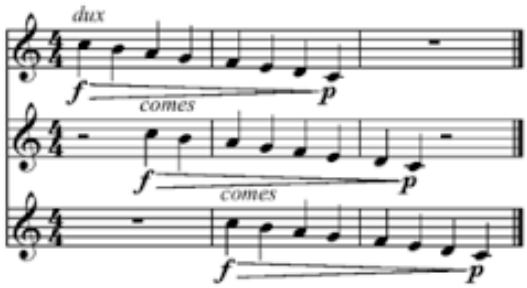


Figure 1. Original voice and variations in canonic composition

Another pre-computer experiment with algorithmic composition was Mozart's *Musikalisches Wurfelspiel* or "Dice Music," a musical game which involved assembling a new piece from a number of randomly chosen musical fragments (Maurer, 1999).

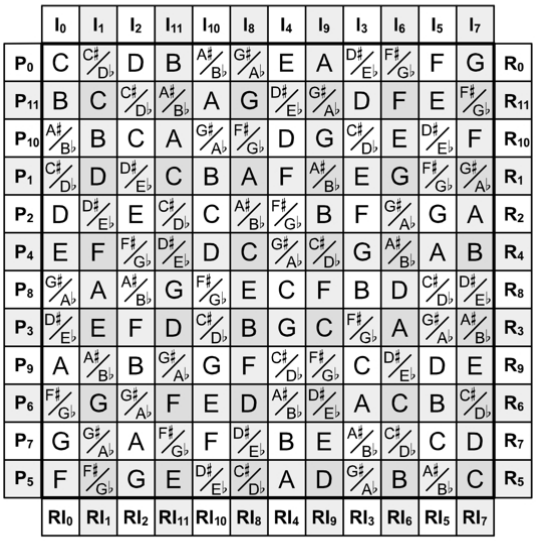
The introduction of computers created many new possibilities to tap into the vast potential of algorithmic composition. There are two main approaches to algorithmic composition: **"stochastic"** system, which involve randomness, and **"rule-based"** systems, which rely on formal grammars (Maurer, 1999).

Of the two, stochastic systems are far simpler. Many stochastic approaches are little more than generating a random series of notes, such as Mozart's "Dice Music." Others use more complicated probability when creating music. For example, in the early 1950s, composer and music theorist Iannis Xenakis began experimenting with stochastic processes in order to protest against what he perceived as the strict control of individual sounds in music. He used probability distribution based on random numbers in order to compose continuously changing sound; for example, in his 1955 work Pithoprakta, the duration of the sounds deviate from a given mean based off probability (Järveläinen, 2000). In 1955, Hiller and Isaacson used the ILLIAC, a computer at the University of Illinois, to create music using Markov chains, discrete systems in which the present outcome is randomized based on previous outcomes. This was the first attempt to attempt to use computer language for algorithmic composition. (Farbood and Schoner, 2001)

Rule-based systems find their roots in Arnold Schoenberg's highly structured twelve-tone technique. The pitch structure of Schoenberg's system was derived from a specific ordering of the twelve tones of the chromatic scale, providing the basic organization of a work. All twelve

tones were to be used before any one could be repeated. The constraints of the twelve-tone technique produced music that was atonal and formless (Raffman, 2003).

Figure 2. A twelve- tone technique matrix



	I_0	I_1	I_2	I_{11}	I_{10}	I_8	I_4	I_9	I_3	I_6	I_5	I_7	
P_0	C	C♯/D♭	D	B	A♯/B♭	G♯/A♭	E	A	D♯/E♭	F♯/G♭	F	G	R_0
P_{11}	B	C	C♯/D♭	A♯/B♭	A	G	D♯/E♭	G♯/A♭	D	F	E	F♯/G♭	R_{11}
P_{10}	A♯/B♭	B	C	A	G♯/A♭	F♯/G♭	D	G	C♯/D♭	E	D♯/E♭	F	R_{10}
P_1	C♯/D♭	D	D♯/E♭	C	B	A	F	A♯/B♭	E	G	F♯/G♭	G♯/A♭	R_1
P_2	D	D♯/E♭	E	C♯/D♭	C	A♯/B♭	F♯/G♭	B	F	G♯/A♭	G	A	R_2
P_4	E	F	F♯/G♭	D♯/E♭	D	C	G♯/A♭	C♯/D♭	G	A♯/B♭	A	B	R_4
P_8	G♯/A♭	A	A♯/B♭	G	F♯/G♭	E	C	F	B	D	C♯/D♭	D♯/E♭	R_8
P_3	D♯/E♭	E	F	D	C♯/D♭	B	G	C	F♯/G♭	A	G♯/A♭	A♯/B♭	R_3
P_9	A	A♯/B♭	B	G♯/A♭	G	F	C♯/D♭	F♯/G♭	C	D♯/E♭	D	E	R_9
P_6	F♯/G♭	G	G♯/A♭	F	E	D	A♯/B♭	D♯/E♭	A	C	B	C♯/D♭	R_6
P_7	G	G♯/A♭	A	F♯/G♭	F	D♯/E♭	B	E	A♯/B♭	C♯/D♭	C	D	R_7
P_5	F	F♯/G♭	G	E	D♯/E♭	C♯/D♭	A	D	G♯/A♭	B	A♯/B♭	C	R_5
	RI_0	RI_1	RI_2	RI_{11}	RI_{10}	RI_8	RI_4	RI_9	RI_3	RI_6	RI_5	RI_7	

This paper will be studying rule-based systems by developing an algorithmic composition program using generative grammars. Rule-based systems cannot be properly discussed without mentioning artificial intelligence systems which can learn and adapt to autonomously create their own grammars. Essentially, these A.I. systems derive mathematical models from a set of musical examples that can be used for inference and prediction (Dubnov, 2003). Systems have even been developed that seek to analyze and imitate the styles great composers (Murphy, 2015).

Research Methodology

Instead of relying on existing generative grammars or formal music conventions, this study created its own set of grammars. This presents two unique benefits:

1. Generative grammars are intended to be intuitive and inherent to music itself.
2. Flaws in the program could be adjusted by ear.

In order to develop a set of generative grammars, which would lay the foundation for the algorithmic composition model, a series of forty rudimentary hip-hop beats were created and analyzed in order to find similar patterns. Each beat was created with a single instrument common to hip-hop instrumentals. Ten were solely composed solely of the kick, ten were composed solely of the clap, ten were composed solely of snares, and ten were composed solely of hi-hats.

Each group of ten was split into two groups, one for loops deemed normal sounding and one for loops which sounded awkward or "off". A rhythm was jarring and awkward-sounding if two notes were placed too closely together or if there were too many notes in too brief an amount of time.

1. Kick Patterns

The logic of the algorithmic composition program was based around the assumption that to achieve "good" sounding rhythms, one only has to eliminate awkward-sounding patterns. Through experimentation, the kick pattern was deemed most important in the overall beat as it lays the foundation. Therefore, the kick should be the first pattern generated. The following observations were noted for the kick while searching for "off" patterns:

- The kick and snare seem to be inseparable; one cannot listen to the kick without imaging where the snare could be placed. Upon further experimentation, this snare was found to be replaceable with another kick.

- Kicks of sixteenth duration should usually have at least one rest of sixteenth duration between them.
- The first kick should be placed on the first beat because a pattern with a kick on a later beat is the same as a kick on the first beat shifted over how ever many.
- There should not be more than two kicks spaced with a rest in between in a row.
- The kick pattern should have at least one kick in the first four steps and one kick in the last four steps.
- The kick pattern's second half should not be the same as the first half in order to avoid being repetitious.
- There should not be more than 5 kicks per 4 sixteenth duration notes or fewer than 2.

2. Snare Patterns

Once the computer program was adjusted to reflect new findings about the kick, the other layers could build off the new foundation and create a nice sounding rhythm. The snare was deemed the next most important pattern for the beat. By combing through existing hip-hop beats, the snare was found to usually be used sparingly. Many of the songs examined interestingly demonstrated the exact same snare pattern, with one snare on the first bar of the third beat. This was evidently a very common pattern seen in new types of hip-hop beats, especially trap (Frank Jev Cee). It was also observed that hip-hop beats that use the snare more frequently typically do not have a clap. As mentioned earlier, the snare's placement is highly dependent on the kick. Using these observations, the program was designed to place a snare on the first beat of the third bar.

3. Clap Patterns

While developing grammars for the clap, it was discovered that the clap is completely dependent on the kick. If a clap pattern were generated independently from the kick, the combination of the two patterns consistently sounded off. After analyzing the relationship between the kick and clap, the following observations were made:

- Claps do not necessarily contribute anything when they are placed on the same beat as a kick.
- Claps always sound correctly placed when they are before or after a kick with a sixteenth duration rest in between.
- Claps placed an odd number of beats away from a kick always sound correct; however, when placed an even number of beats notes away, they always sound off.

4. Hi-Hat Patterns

Hi-hat patterns were very tricky to write grammars for because unlike the kick, snare, or clap, a slight mistake in a hi-hat pattern is extremely noticeable and always sounds off. Mutes, pauses in the pattern, are very common in hi-hat patterns. The amount of mutes must work well with the hi-hat rolls, which consist of either triplet rolls or quarter rolls. The following observations were made regarding hi-hats:

- The pattern should not start on a rest.
- Hi-hat rolls should be either split into triplet rolls or quarter rolls.
- Roles should be on the first beat of a bar or the last beat.
- Two roles shouldn’t be placed adjacent to each other.

- Two notes should be muted at a time for the best results, and these mutes should not continue over a bar.

Through observation, the following steps were developed to compose a hi-hat component:

- First, a pattern should be created consisting of 16 sixteenth duration pedal hats with no rests in between.
- Next, some of the hi-hat notes would be chopped up into either 1/32 rolls or 1/64 rolls.
- Also, some of these hi-hat notes would be muted.

5. Algorithmic Composition Model

After a finished set of generative grammars was documented, development of the algorithmic composition program, codenamed "BeatOven," began in proper. Java was ultimately decided as the programming language for this program, as Java is one of the most logical and easiest programming languages to read and to translate into pseudocode, a notation representing a simplified programming language. Furthermore, Java has a massive amount of already finished libraries that expedited the completion of the program. JFugue is an open source programming library that facilitates programming music in Java by circumventing MIDI, a format of music for computers that is very difficult to produce for a program like "BeatOven." JFugue also provides many methods for working with chords, based off music theory, which would assist the development of an algorithmic composition program; however, this paper did not take advantage of these options. First, aspects of every program were implemented such as GUI and user input. After experimenting with the note durations, a program was created resembling other beat producing software such as Image Line's FL Studio.

```
method to generate kick string
{
        kickPattern = "k..............." // a '.' is a rest and k is a kick
        for every other note
        {
                if A random number 0-100 is greater than .65
                {
                        add a kick note on this SQUARE
                }
        }
        return kickPattern;
}
method generate kick pattern
{
        String kickPattern
        number kickCount

        do the following
        {
                kickPattern = get kick pattern from method above

                kickCount = 0
                for every kick in the pattern above
                                kickCount + 1
        }
        until (the first half and second half of the pattern are different  and
                        the second half of the pattern has at least 1 kick and
                        there is less than 5 kicks and
                        there is at least 2 kicks);

        return kickPattern;
}
```

Figure 3. Pseudocode for the kick pattern generation

The pseudocode above basically outlines how the program functioned. The pseudocode for the other instruments is very similar, with the main difference being the grammars controlling whether the pattern is allowed to proceed.

Figure 4. Example of a beat produced by "BeatOven"

In this figure, each dark gray square has a $1/16^{th}$ duration. The squares with colors represent an instrument being played for that $1/16^{th}$ duration. Red corresponds to the kick, blue corresponds to the snare, yellow corresponds to the clap, and green corresponds to the hi-hat. The green squares broken into four sections represent hi hat rolls. Each four squares is called a bar, and each square is called a beat.

After the algorithm was developed, it was extensively tested, to confirm that the grammars were effective but not too restrictive. Now that the program was successfully generating beats, a metric was developed to evaluate the success or failure of "BeatOven" and its

grammars. The best way to evaluate the quality of music has been an ongoing debate for centuries. Instead, this study sought to test whether or not "BeatOven" could create beats indistinguishable from those created by humans. The success of the algorithm would be based on people's ability or disability to identify which of two beats was made by the algorithm and which was made by a human. To prepare the algorithm sample, the first beat produced by the algorithm was used. A human-made beat of similar structural complexity was found on the internet. Both the algorithm-generated beat and the human-generated beat were copied and reproduced exactly in FL Studio with the default instruments for kick, snare, clap, and hi-hat, so those surveyed would not based their decision based on instruments, velocity, or sound quality. The survey involved first playing the two samples for the participants. Respondents were then informed that one sample was created by a human and one was created by a computer relying on pre-decided algorithms. The participants were then asked whether they could confidently choose which sample was made by the algorithm and which was made by the human.

Results

The graph below shows the results from the survey. The first column lists the three predominant trends: respondents who believed the first sample was a computer and the second sample was a human, respondents who believed the first sample was human and the second sample was a computer, and respondents who responded with some variation of "I don't know." It should be noted that while three respondents who stated that they did not know ultimately had no leaning one way or another, the rest of those respondents ultimately settled on one of the options.

Trend	Respondents	Frequency
Sample 1- Computer Sample 2- Human	18	36%
Sample 1- Human Sample 2- Computer	29	58%
Do not know	24	48%

Figure 5. Results from the survey

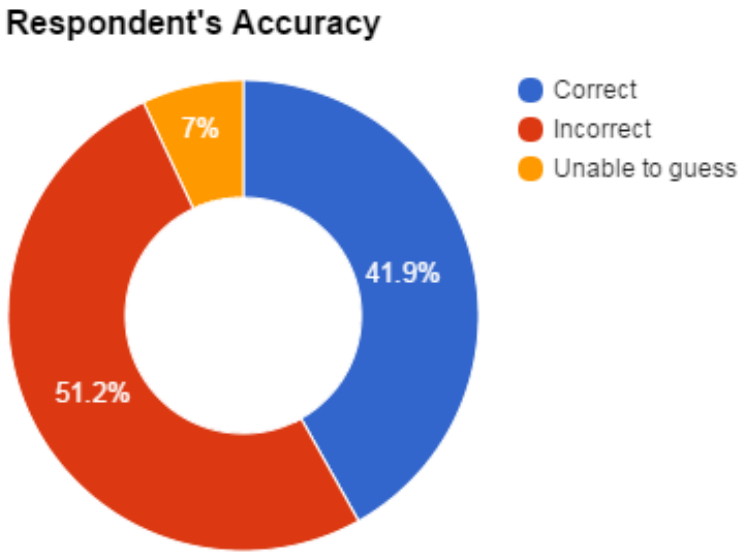


Figure 6. Pie chart of respondents' accuracy

Conclusion

The results above prove part of the hypothesis correct: grammars used for the basis of an algorithmic composition model can produce works almost indistinguishable from works created by direct human input. As one can conclude from the datatables above, the structures of the

human-made hip-hop beat and the computer-generated hip-hop beat were so similar that the majority of survey participants were unable to confidently and correctly identify the artist of each beat. Furthermore, those who did confidently answer correctly were unable to provide firm backing to their decision. Although the program demonstrated that grammars do exist in rhythmic based hip-hop tracks, further experimentation is needed in order to produce accurate grammers. The grammars used and discovered in this paper were in no way perfect. However, this paper successfully applied grammars in algorithmic composition.

Bibliography

Maurer, John A., IV. "The History of Algorithmic Composition." Center for Computer Research in Music and Acoustics. Stanford, Mar. 1999. Web.

Papadopoulos, George, and Geraint Wiggins. "AI methods for algorithmic composition: A survey, a critical view and future prospects." *AISB Symposium on Musical Creativity*. Edinburgh, UK, 1999.

Grout, Donald Jay and Claude V. Palisca (1996), *A History of Western Music*. 5th ed. W. W. Norton & Company: New York. 843 pp.

Schiltz, Katelijne, and Bonnie J. Blackburn. *Canons and Canonic Techniques, 14th-16th Centuries: Theory, Practice, and Reception History; Proceedings of the International Conference, Leuven, 4-6 October 2005*. Vol. 1. Peeters Publishers, 2007.

Farbood, Mary, and Bernd Schoner. "Analysis and synthesis of Palestrina-style counterpoint using Markov chains." *Proceedings of the International Computer Music Conference*. 2001.

Järveläinen, Hanna. "Algorithmic musical composition." *Seminar on content creation Art@ Science*. 2000.

Raffman, Diana. "Is twelve-tone music artistically defective?." *Midwest studies in philosophy* 27.1 (2003): 69-87.

Xenakis, Iannis. *Formalized Music: Thought and Mathematics in Composition*. Indiana University Press. 387 pp.

Dubnov, S., G. Assayag, O. Lartillot, and G. Bejerano. "Using Machine-learning Methods for Musical Style Modeling." Computer 36.10 (2003): 73-80. Web.

Murphy, Mike. "People Are Confusing Computer-generated Music with the Works of J.S.

Bach." Quartz. N.p., 26 Aug. 2015. Web.