Eli Backer
Professor Antonio G Barata
Music 311
20 November 2012

Algorithmic Composition as an Inspiration for Drums

Abstract

Algorithmic composition is a way of writing music by formula, usually by using a computer. Compositions written by computers, however, may not always be pleasing to listen to for a variety of reasons. Algorithms that are very elegant may not produce pleasant compositions while good compositions may take complex, inflexible logic to produce. While it is hard for an algorithm to produce entirely new works, they can do very well to spark the creativity of a composer by looking at his work and returning variations on it. Some of these may be garbage but others may be close to a sound that the composer wants and can use. The code produced for this project takes in a drum loop and produces variations on it with different amounts of success in different areas. The logic behind it is clearly explained and matches the results produced. Finally, examples are presented of different drum loops created by the program.

Introduction

In modern day our culture produces hundreds of new works a day, most of which are pleasing to listen to by the average person. Why bother with algorithms if they can only produce material half the quality of a top composer? The answer lies in the always essential time. A composer may only be able to turn out a top-notch composition every month or may fall into a rut or may do the same task 100 times to produce her piece. In this field a computer excels as it can easily follow rules over and over again. By providing a computer with an algorithm to apply, it can create many variations on a theme given by a composer. It is then up to the composer, who has had many years of building her own musical preferences, to decipher which variation would be best. She can then expand on it and write another fantastic piece. The code presented at the end of this paper takes in a drum loop and creates variations on it such that it will augment a melody played over it. It is in these variations that the composer may find inspiration and expand the results of the computer.

Algorithms

What is an algorithm? According to Webster, an algorithm is broadly defined as "a step-by-step procedure for solving a problem or accomplishing some end especially by a computer." (Merriam-Webster)

In music there are often repetitive tasks that composers do based on their experience in composition. It is impossible to fit a composer's decision-making process into a computer but parts of it can be automated. Algorithms for making music are split into two camps: those that create music from scratch based on a set of rules from a style or genre, and those that, given existing music, modify it to create variation. As the focus of this paper is inspiration for drum patterns, the focus will be on the latter as the former will continually create compositions of the same style.

Suppose the computer has just been given a piece of music. How can it figure out what to play based on that? What rules should it follow? The composition in itself is already a set of rules to play one note after another. By looking at all the notes played in the composition and all the notes recently played, the computer can figure out what notes it could play next and repeat the process again. This process of looking at the whole world and the specific case is called a Markov Chain. Before the computer starts computing, it creates a weighted graph of all the notes in the piece. A good example of a two-state Markov Chain can be found in Figure 1.

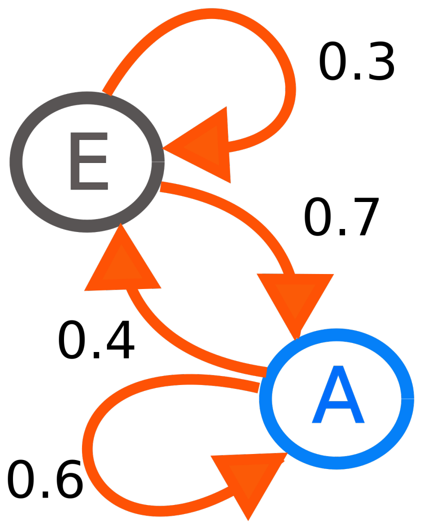
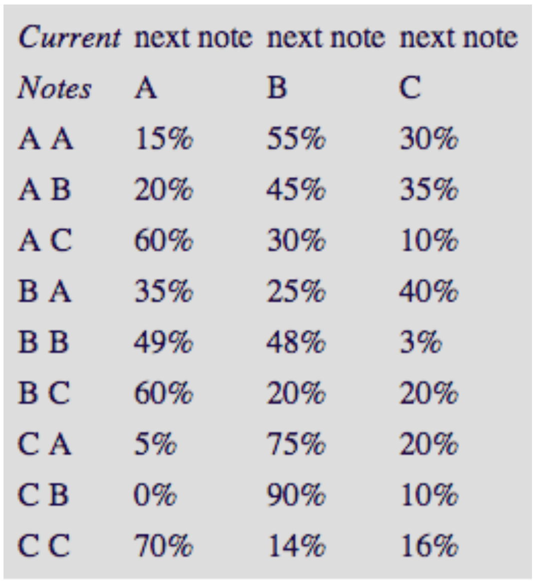


Figure 1 (Joxemai4)

The figure shows that if you start at point A there is a 60% chance you will stay at point A and a 40% chance you will go to point E. Once at point E there is a 30% chance you will stay at point E and a 70% chance you will go to point A. In the context of music, this can be thought of as a piece where note A is commonly played and if an E is played, it usually resolves back to A. A chain such as this is called a First-Order Markov Chain because it only looks at the previous state for guidance to the next state. This may not provide enough information to create pleasing variations on a piece. Thus, Second-Order chains are used. A second order chain is exactly like one of the first order but it takes into account one more past state of the system. An example of a three-state Second-Order Markov Chain can be found in Figure 2.



Current	next note	next note	next note
Notes	A	B	C
A A	15%	55%	30%
A B	20%	45%	35%
A C	60%	30%	10%
B A	35%	25%	40%
B B	49%	48%	3%
B C	60%	20%	20%
C A	5%	75%	20%
C B	0%	90%	10%
C C	70%	14%	16%

Figure 2 (Sapp)

Markov Chains are also used outside of music so that systems can respond to outside changes. For example, in a car, Markov Chains are used to monitor crash sensors and other data to determine how to respond. (Tweedie)

The Code

Overview

The code on the attached CD is designed to take in a drum pattern and return that pattern a number of times with variations fitting to the original and breaks every specified measure. It should be able to take input of any time signature but was designed around the 4/4 standard of dance music.

I/O

MIDI is the standard for passing musical data between instruments and around a computer. However MIDI, unlike other standards, is not open or governed by a large standards board but instead is agreed on by the major manufacturers. This means that to the average person it is difficult to develop reliable ways of dealing with MIDI data. Therefore, instead of dealing solely with MIDI, the code below takes in a text file representing the rhythm to be varied and outputs a MIDI file of those variations. Many thanks are owed to EmergentMusics for their tools for writing MIDI files in Python.

Logic

The logic behind the code is simple and very similar to a Markov Chain. Now that the computer has the input in a useable format it can calculate how to rearrange the pattern.

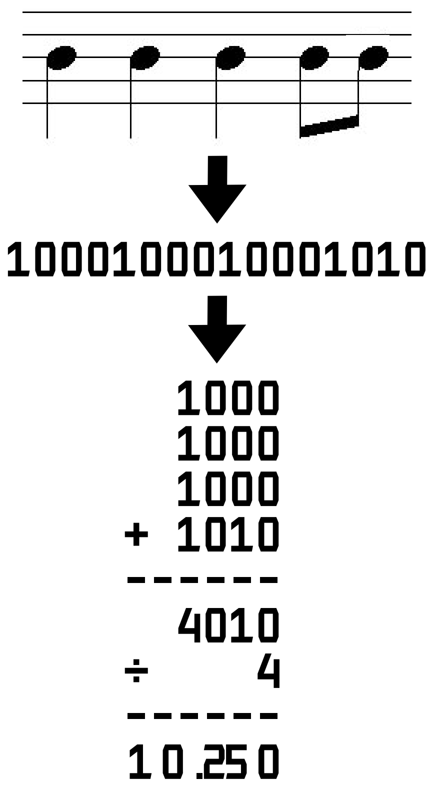


Figure 3

This is done by taking the array of ones and zeros read into the program and splitting them into equal groups based on the time signature. The groups' columns are summed then scaled so that no value is greater than one. In Figure 3, each value is divided by four so that the highest value is 1.

The resulting matrix is called the chance matrix or cm for short. The program then looks at every drum beat and its corresponding value on cm and determines whether it should be played. If it is played, this is noted in a matrix called dmc, short for drum matrix chance. When the program has finished picking which drums to play for these bars, it calculates cm's for both the original drum pattern and the one created by dmc. It adds them together and scales them so that the result is no greater than one. If only dmc was used then the beat would die out over the course of a few bars as not all the drums are played at their specified times. Thus, this "feedback loop" keeps the pattern alive and true to the original style of the input.

Breaks

If a drum beat were to be played over and over again with no major pause or change it would not be interesting and could stifle the melody that it is there to support. By adding breaks, the rest of the track is highlighted at that point. For the purpose of this paper, a break is defined as follows: a portion of drums where there is more silence and/or different drum patterns.

In electronic music breaks small or large come before a new element is introduced or removed. These provide a heads up to the listener that, "This is going to be new and different, pay attention." All discussion of breaks will be done in the context of 4/4. A loop refers to a pattern 4 bars long.

The code creates breaks as specified by a constant at the top of the code. By default this is set to 8 which means on the 8th loop through the pattern there will be a break. This also produces a less dramatic break at the 4th loop. The code creates a break in the 8th loops by changing the ratio of the original pattern "feedback" while keeping the contribution of dmc constant. This lowers the overall probability of drums being played during the break. In addition the cm created has its first and last elements swapped as shown in Figure 4.

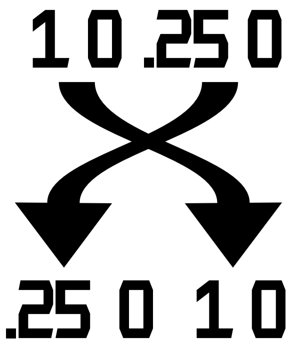


Figure 4

This means that there will be almost no drums at the start of the break and more towards the end. Because of this the listener notices the lack of drums at the start of loop 8 and is pulled into loop 1 by the drums starting again, a little before the downbeat.

However, this creates an unintended side-effect. Because we are drastically changing cm in the 8th loop, we are changing the dmc it produces for bar one. The effect is that the clean break we started in loop 8 bleeds over into loop 1 creating an undesired effect. This is fixed by taking the dmc produced by loop 7 and giving it to loop 1. This creates a clean restart of the pattern.

To add even more movement to the pattern we can add a slight break at loop 4. This makes the track more interesting to the listener without grabbing their attention. To create this, the original dmc produced by loop 8 is inserted for the cm calculation of loop 4. This means that the once awkward continuation of the break from loop 8 is moved to loop 4 where it sounds good.

The secondary break is automatically created based on how often the user wants a break to occur. There is no simple setting to prescribe more breaks more often, nor would it work well as it is driven entirely from the break at the 8th loop.

Results

The results of this program are surprisingly listenable and useful. It is important to remember that one cannot expect to put bad patterns in and get great ones out. One also cannot expect it to work without some tweaking as is true of any program.

For this algorithm to work well with 4/4 dance beats (the one most used in testing) the cm constants must be adjusted. The best results were found when the original values were multiplied by two, then 11 was added, and then everything was divided by 20. This ensures that there is never a drum played 100% of the time while giving notes not formally to be played a greater chance.

The text files drumPattern.txt, drumPattern2.txt and drumPattern4.txt all produce good results when run through the program but, because of how the program builds the variations, others (such as drumPattern3.txt) will not. If the user feeds it an already complex and varied beat, it will take all of the variations and combine them into a cm to make the new beat. This is like taking your favorite Italian, Mexican, and Asian dishes and putting them on top of one another for dinner. It will not taste as good as any one of them on its own. What the program is good at is taking a pile of ingredients and turning them into something closer to a finished product. The breaks in particular are useful for inspiration.

A quick mention should be made to the swing setting. The swing is controlled by a constant at the top of the code and makes typical dance beats sound more interesting and slightly less programmed by making notes on the 1, and slightly longer and those on the ee, uh start slightly later.

See Appendix A and B for more information on the examples included on the CD and the conventions used in the plaintext and MIDI files.

Bibliography

EmergentMusics. MIDIUtil. 1 2010. 19 11 2012 <https://code.google.com/p/midiutil/#MIDIUtil>.

Francis, Dr. John R. Computer Music Algorithms. PDF. 5 5 2012.

Joxemai4. Wikipedia. 9 5 2010. 19 11 2012 <https://en.wikipedia.org/wiki/File:Markovkate_01.svg>.

Merriam-Webster. 18 11 2012 <http://www.merriam-webster.com/dictionary/algorithm>.

Sapp, Professor Craig Stuart. Digital Music Programming II: Markov Chains. Peabody Conservatory of Music, Johns Hopkins University. 19 11 2012 <http://peabody.sapp.org/class/dmp2/lab/markov1/>.

Tweedie, S.P. Meyn and R.L. Markov Chains and Stochastic Stability. London: Springer-Verlag, 2005.

Appendix A

The Examples Provided

This appendix provides a brief overview of the sound examples included on the CD. The "faves" folder contains the author's favorite MIDI files produced which may or may not be in these examples. All the examples listed can be found in the "Examples" folder. They are presented in order of creation by the program.

1. This is the bass/melody that is presented with many of the other examples. The piano is from the Philharmonik VST from IK and the bass is from a Shruthi-1 by Mutable Instruments.
2. Loops two thru
3. …
4. …
5. five are the unaltered loops used throughout the examples. They are labeled drumPattern.txt, drumPattern2.txt, drumPattern3.txt, and drumPattern4.txt respectively on the CD. Pattern 3 later proves that not all drum beats sound better when varied by this program.
6. This is the first pattern that was varied by the program. Unfortunately, the inequality for cm to the random number was backwards.
7. One of the first successful variations produced by the program. Still very random but not unpleasantly so.
8. Another of the first variations, this time with the cm settings such that fewer beats are triggered.
9. This demonstrates that the effect of 8 could be pleasing at a lower bpm.
10. What a difference swing makes! This is especially noticeable in the closed hi-hat.
11. An example of what happens when there is no "feedback" from dm and the beat is allowed to die out.
12. The first successful example of a break at loop 8. Notice how the break bleeds into the loop 1.
13. The break created here is a good example of one that might spark a composer's imagination.
14. An example, using pattern 3, of how more complex beats do not work well with the program as this sounds a little too random.
15. One of the first examples of a minor break, generated by the leftover dmc of the major break.
16. The "amen break" played first by its self, then over the melody. The Amen was popular among Jungle artists in the 80s and has become the backbone of the genre due to its easy sampling and style.
17. The Amen does not work well when plugged straight into the program, thus the middle section was extracted and run through. This example preserves some of the feel of the Amen while creating new variation as well.
18. This is an example of pattern 2 playing for quite some time.

Appendix B

Conventions Used and Using the Program

These are presented in no particular order. They should be known by anyone trying to run the program.

- Files input to the program will consist of 0s and 1s. A 1 represents that the drum should be played, a 0 is used to fill the space where drums are not played.
- No new-line characters should follow the last line of drums. This will cause an out of bounds error.
- If there are no new-lines at the end of the input file and a out of bounds error still occurs it is possible that the time signature does not match the file given or that the time signature is not supported. Common time signatures were tested, uncommon were not. If the user is trying to break the code, they will.
- MIDI files produced are meant to work well with Ableton Live's Drum Rack. This means that the top row of the text file corresponds to the note C1. The next row is C#1 and so on.
- The drums intended for use with the included drumPattern files are as follows (from the top of the text file going down or from C1 going up):
    - Kick
    - Snare
    - Open Hi-Hat
    - Closed Hi-Hat
    - Bass Tom
    - Clave
    - Mid-range Tom
- The code will default to using drumPattern.txt as the input file if another is not specified. A different file can be used if entered after the evocation of the program itself. See Appendix C.
- The MIDI file output by the program is titled output.mid and can be found in the folder from which the python file was executed.
- A minor break will not be created in the first 8 loops because there has not yet been a loop 8 to feed into it.
- The python file that produces the variations is meant to be read and is well commented. Please use a IDE that supports over 80 characters per line and has syntax highlighting. For Mac OSX, Xcode is free and recommended.

Appendix C

Running the code in Mac OSX

This is intended to help the user run the code on a computer running Mac OSX, such as the lab machine.

1. Open Terminal by typing command-space and Terminal and return.
2. Type "cd" and a space. Then click and drag the file containing the assignment to the line on which text was just entered on. Press return.
3. Type "python algCompDrums.py" and press return. A new output.mid file should have been created.
4. To run with a different text file use the following command: "python algCompDrums.py your_filename_here"

Please direct all problems to the internet or ebacker@calpoly.edu.