5-2017

Cooperation Between Top-Down and Low-Level Markov Chains for Generating Rock Drumming

Chris Caulfield
University at Albany, State University of New York

Cooperation Between Top-Down and Low-Level Markov Chains for Generating Rock Drumming

An honors thesis presented to the
Department of Computer Science,
University at Albany, State University of New York
in partial fulfillment of the requirements
for graduation with Honors in Computer Science
and
graduation from The Honors College

Chris Caulfield

Research Advisor: Cristyn Magnus, Ph.D.

May 2017

Abstract

Without heavy modification, the Markov chain is insufficient to handle the task of generating rock drum parts. This paper proposes a system for generating rock drumming that involves the cooperation between a top-down Markov chain that determines the structure of created drum parts and a low-level Markov chain that determines their contents. The goal of this system is to generate verse- or chorus-length drum parts that sound reminiscent of the drumming on its input pieces.

Acknowledgements

With deep gratitude, I would like to thank Dr. Magnus for continually advising me throughout the process of this research. Without her help, the work contained in this thesis would not have been possible.

Special thanks to Giles Hall, AKA Github user “vishnubob”, for his creation of a Python library dedicated to making manipulation of MIDI files easier. MIDI is very unpleasant to manipulate algorithmically, as it is a very low-level protocol made decades ago.

Table of Contents

1. Introduction

The Markov model sees a variety of applications in algorithmic music. For example, Herremans, Weisser, Sörensen, and Conklin, constrain a first-order[1] Markov model with various quality metrics in order to create authentic-sounding music for the bagana, an Ethiopian instrument similar to the lyre [1]. Schulze and van der Merwe use a hidden Markov model in combination with predictive suffix trees in an attempt to generate harmony-based music "just as pleasurable to the average human listener as music composed by a human with moderate experience" [2]. Chordia, Sastry, and Şentürk use variable-length Markov models and variable-length hidden Markov models to model tabla, an Indian percussion instrument similar to the bongos [3].

Research in algorithmic generation of symbolic music in particular styles involves work in at least one of three categories: melody, harmony, and rhythm. There is often some overlap between these categories in research. Biles generates jazz solos over chord progressions, involving work in both melody and harmony [4]. Cope generates music in the style of specific composers, involving modeling of both melody and rhythm [5]. Eigenfeldt and Pasquier algorithmically arrange multi-instrument pieces with percussion, explicitly involving melody, harmony, and rhythm [6]. Algorithmic generation of ornamentation in music is the subject of its own separate research, such as how Puiggròs, Gómez, Ramírez, and Serra generated expressive ornaments in bassoon pieces [7]. This paper explores ornamentation in its handling of "flair" in rock drumming.

[1] An "X-order Markov model" means that the states of the model determine transition probabilities based on the X states that came before them. A Markov chain is a type of first-order Markov model.

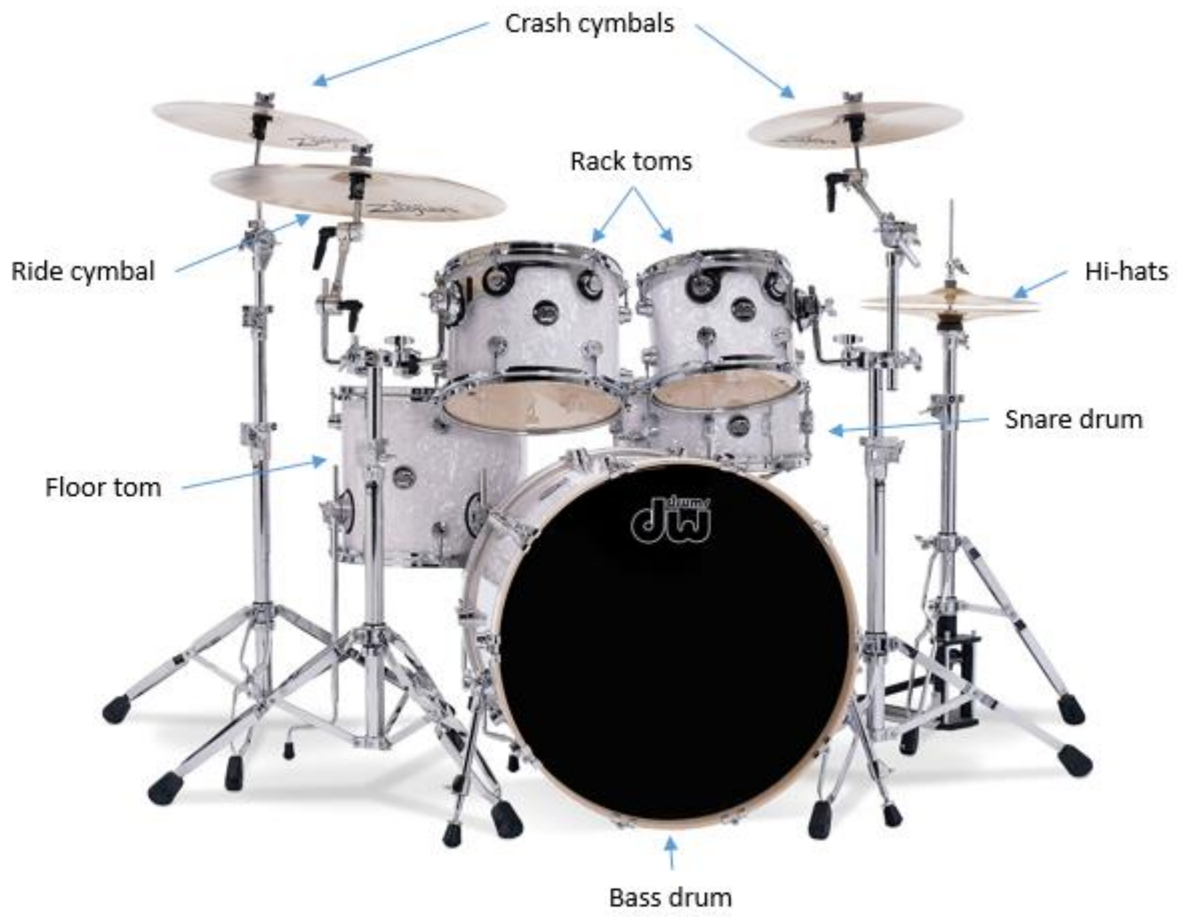
This paper gives background information on music and drumming (section 1.1), then describes its problem formulation (section 1.3) and the use of simple Markov chains for generating rock drumming (section 1.3). The methodology of our drum-generation procedure and its dual-Markov-chain system is then explained (section 2), which is followed by experimentation results (section 3) and a conclusion including suggestions for future research (section 4).

1.1 Music Background

Timing in music is based on subdivisions called *beats*. Beats are grouped into units called *measures*. The *downbeat* refers to the instant a measure begins. The *time signature* of a piece of music indicates what type of note counts as one beat and how many beats are contained in each measure. The most common time signature is 4/4, which specifies that a quarter note counts as one beat and that there are four beats to a measure.

Timing as it relates to rock drumming involves a concept known as the *backbeat*: the fundamental rhythmic component of a piece of rock music, which loops on a measure-by-measure basis. Backbeats are usually consistent within verses/chorus, but often change between sections of a song.

The fundamental components of a drum set are the *snare drum* and the *bass drum*, which together are used to form the backbeat of most rock songs. Drum sets also include one or more *cymbals* used to count time by striking them at a regular timing interval. The *hi-hat* and *ride* cymbals are most commonly used for this purpose in rock music. *Crash cymbals* are used to create accents; one or more of them are usually found on rock drum sets. Rock drum sets also contain one or more *toms*, which are drums most often used in fills; there are typically two or three. Drum sets also sometimes contain other accent pieces, but these are rarer and not essential to rock drumming.



(A common drum set arrangement for rock music)

The *main beat* refers to the foundational pattern that the drummer plays throughout a section of a song[2]. Main beats in rock music usually consist of playing the backbeat (i.e. the snare and bass drums) along with a cymbal used for counting time. *Flair* refers to playing the main beat with additional components that do not alter the backbeat. A *fill* refers to a short freeform pattern played in lieu of the main beat.

[2] The main beat of a piece is often simply referred to as a/the “beat”. This means that the word “beat” is ambiguous, but it should hopefully be obvious by context whether it refers to the unit of musical timing or a drum pattern.

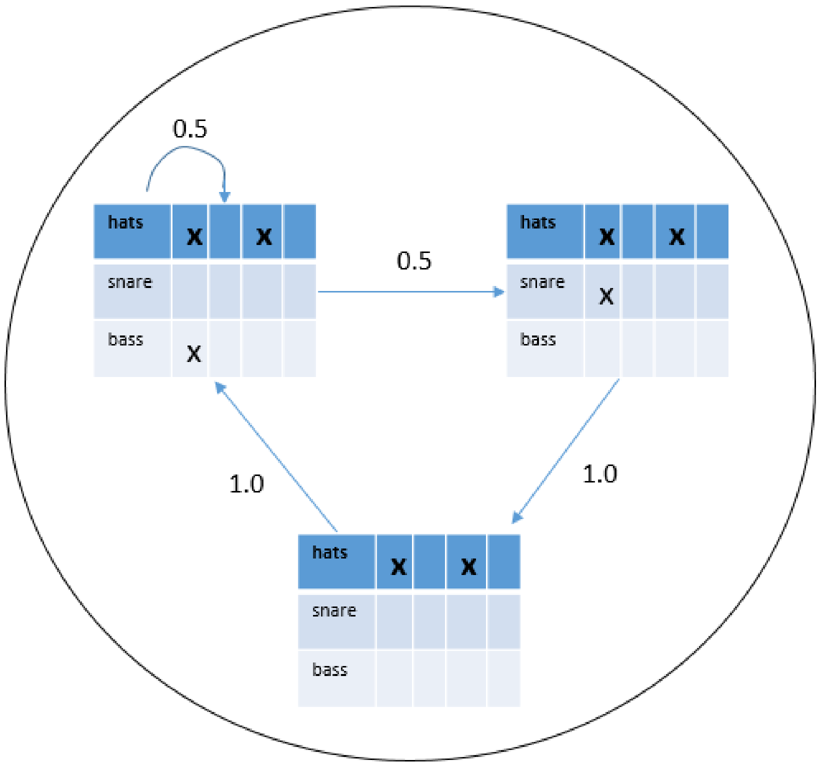
1.2 Problem Formulation

Certain types of drumming are very intricate, such as that found in jazz or very musically complex rock subgenres. Drumming in Electronic Dance Music (EDM) is far less complex, and thus simpler to model. Eigenfeldt and Pasquirer used Markov models to generate drum parts for EDM songs [6]. Due to the simplicity of EDM, they were able to quantize[3] notes to 16th note intervals and ignore expressive, subtle timing variations. Rock drumming is generally more complex than EDM drumming: rock sometimes involves subdivisions smaller than 16th notes, it may incorporate triplets in fills and flair, and there is expressivity relating to how subtly the drummer plays either ahead of or behind the beat. Our model specifically allows for these factors. It also specifically allows silence (i.e. rests) in both input and output pieces; much algorithmic music disallows rests to simplify the problem (e.g. [2]). Rests in main beats in rock music are rare, but rests are sometimes purposefully used in drum fills.

1.3 Simple Markov Chains for Modeling Rock Drumming

A plain Markov chain can be used to model rock drumming. Such a chain can be built by wrapping the contents of each beat of every measure of the input pieces into states, giving these states weighted pointers to states that follow them in the input. To see the problems with this system, let us consider a hypothetical Markov chain trained on one input: a half-time rock beat.

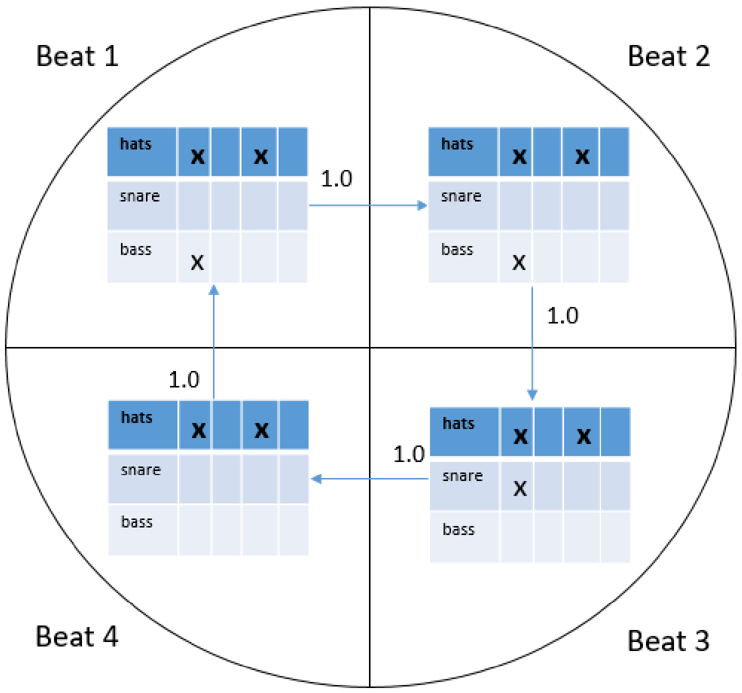
[3] "Quantization" refers to the act of taking all notes in a musical piece and changing their timing so that they line up perfectly with a specified repeating time interval (e.g. every 16th note).



(A Markov chain trained on a standard half-time rock beat. Each column in these states represents one 16th note.)

Note that the top-left state of this chain is allowed to transition to itself. This chain is not guaranteed to obey the measure structure of its input piece or match its backbeat. When this transition happens, the backbeat of the playback output feels like it spontaneously changes time signatures because the backbeat is delayed. Changing time signatures is virtually nonexistent outside of very musically complex rock subgenres.

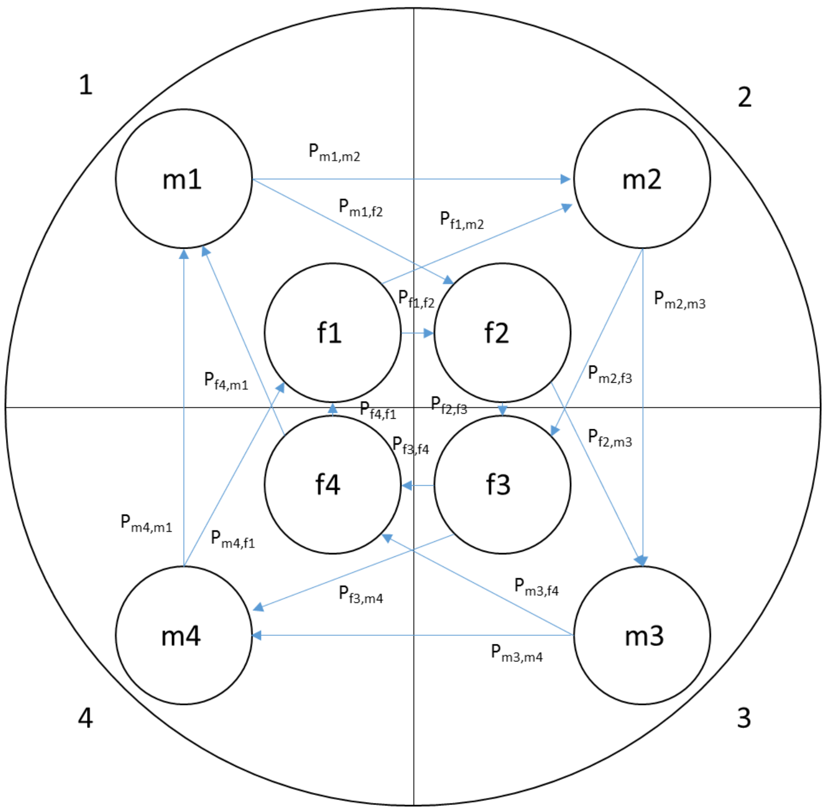
To fix this issue, the Markov chain can be given parameters allowing it to keep track of the current beat of the measure. States in this system know what beat of the measure they occurred on and are only allowed to transition to states that occurred on the following beat.



(Modified Markov chain trained on a standard half-time rock beat)

Even with parameters allowing it to keep track of the beat, this Markov chain produces peculiar-sounding drum output. It blindly transitions between states with no regard for broader context. In this system, fills appear probabilistically at random (i.e. random according to the weighted transition probabilities) without any higher-level process informing the system as to when fills should occur.

In order to fix this fill problem, a high-level Markov chain can be used that learns the transition probabilities between when input pieces play the main beat and when they play fills. This model has eight states: main beat and fill for each of the four beats of the measure. These states each maintain a list of every main beat or fill which occurred on their respective beat of the measure. Output generation with this model works by transitioning between states, at which point a pattern from that state’s list is randomly selected and played back.



(High-level main-beat-or-fill Markov chain)

By itself, a system like this is not capable of producing adequate output: it knows when to play the main beat or a fill, but it does not know what main beat or fill to play. Its method of picking a random main beat or fill state for every output state results in drumming that has no logical connection between its parts.

2. Drum Generation Procedure

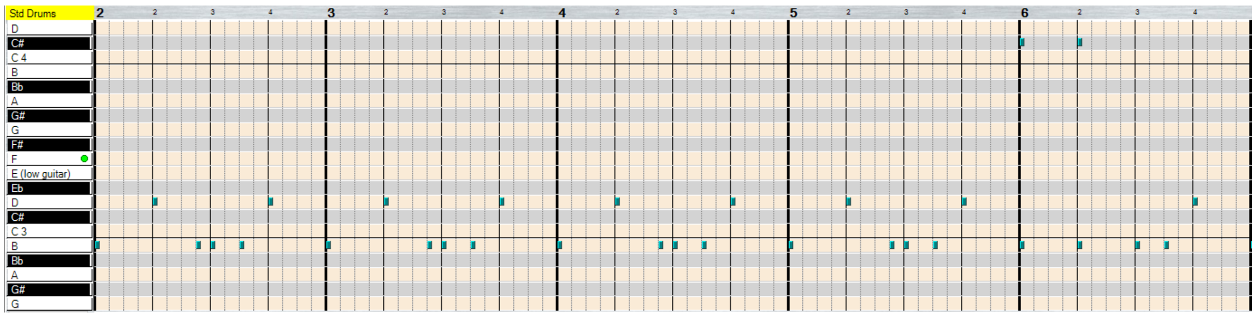
The structure of this paper's drum-generation algorithm is as follows:

1. Each input is preprocessed and its main beat is identified.
2. The input is segmented into main beat and fills.

3. This segmentation information is used to train the SegmentationChain, a modified Markov chain responsible for learning the probabilities of the transitions between main beats and fills.
4. The segmentation information for the input piece is then used in combination with the input piece itself to train the BeatChain, a modified Markov chain that learns the transitions between the contents of each beat of the input pieces.
5. The SegmentationChain determines when the output will have its main beat and when it will have fills.
6. The BeatChain generates the contents of the output.

2.1 Input Preprocessing

This project uses MIDI files as input pieces. MIDI files consist of tracks containing MIDI events. The MIDI event we most care about is the NoteOn event, which effectively specifies what piece of the drum set is hit[4], at what time, and at what volume. We are not concerned with NoteOff events, as duration is irrelevant to percussion.



(Visual representation of the contents of a MIDI file, specifically the first five measures of the drums for The Beatles' song Taxman. Screenshot taken from the program Anvil Studio.)

[4] To be more technically accurate, NoteOn events specify notes by pitch. Drums in MIDI work by mapping these pitch values to samples of percussion instruments.

The training set consists of verses and choruses from various songs by The Beatles. All input pieces are in 4/4 and start on the downbeat. The first verse/chorus of the song is used unless the drums do not come in fully or at all until a later verse/chorus, in which case the first such one is used. If necessary, verses and choruses are edited to start on the downbeat. If a verse or chorus does not meet the requirements for the training set, it is not used.

Before the beat-finding algorithm is run on input, that input is first preprocessed in two ways. First, we remove events for percussion instruments not commonly found in rock music. We keep events for the following instruments: hi-hat, ride, crashes, toms, snare, side stick[5], bass drum, china cymbal, splash cymbal, and cowbell. Tambourine is commonly found in certain subgenres of rock, but we remove it because it is not usually mounted on the drum set and played by the drummer.

Second, pitches are made uniform. There are two MIDI pitches for ride, bass drum, and snare sounds. Since having two rides, bass drums, or snare drums is very rare in rock music in general, all instances of one of the ride/bass drum/snare pitches are changed to the other pitch. There are six different pitches for toms and two for crash cymbals, but these pitches are not mapped because it is common for rock drum sets to have multiple toms and/or crashes.

For the purpose of beat classification, some researchers remove all but a few pieces of the drum set during preprocessing. In their paper about drum beat classification, Ellis and Arroyo mapped all drum events of their input MIDI files to either the snare drum, bass drum, hi-hats, or null, on the grounds that "the vast majority of popular music [beats]" consist of just those three instruments [8]. While not common, it is not rare for a section of a rock song to involve a drum

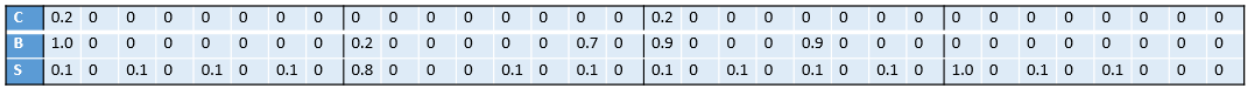
[5] "Side stick" refers to the act of holding a drum stick sideways with one end against the snare drum and quickly rotating it to strike the other side of the stick against the rim of the drum. This creates a sort of clicking sound.

beat that consists of no snare or cymbals, instead using the toms in combination with the bass drum to form the backbeat. Rock songs may also have main beats that use a tom to count time rather than a cymbal, which would be indistinguishable from a snare-and-bass-only main beat under Ellis and Arroyo's system. This project allows main beat states to consist of any combination of drum set pieces (within the set of pitches we allow). This results in more accurate main beat classification and greater versatility.

2.2 Finding the Main Beat

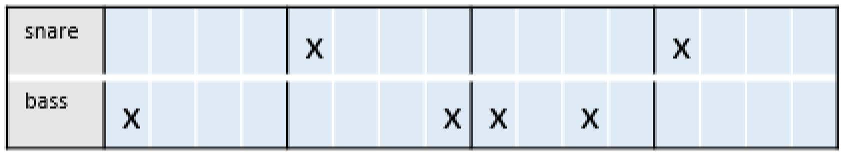
Much research has gone into the subject of musical pattern-detection [9] [10] [11] [12], but the relative simplicity of general rock drumming allows for a simple algorithm to produce accurate results. Rather than use a more complicated algorithm required for more genres with more complex drumming, we can use a naïve beat-finder (NBF) that is simple, fast, and accurate for most rock music.

The algorithm works by first finding the probability that all of the instrument pitches have of being played at each time quantum of the measure. All instrument-at-quantum instances that surpass a chosen probability threshold are considered to be part of the song's main beat. For this project, a quantization interval of one 32nd note and a probability threshold of 70% were chosen. The NBF output is best thought of as the underlying main beat of the input piece rather than the literal main drum beat being played.



C	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	1.0	0	0	0	0	0	0	0	0.2	0	0	0	0	0	0.7	0	0.9	0	0	0	0.9	0	0	0	0	0	0	0	0	0	0	0
S	0.1	0	0.1	0	0.1	0	0.1	0	0.8	0	0	0	0.1	0	0.1	0	0.1	0	0.1	0	0.1	0	0.1	0	1.0	0	0.1	0	0.1	0	0	0

(The probability-of-each-instrument-at-each-quantum data for the first verse of the song Taxman. Here "C" stands for the crash cymbal, "B" the bass drum, and "S" the snare drum. Beats of the measure are separated by vertical borders.)



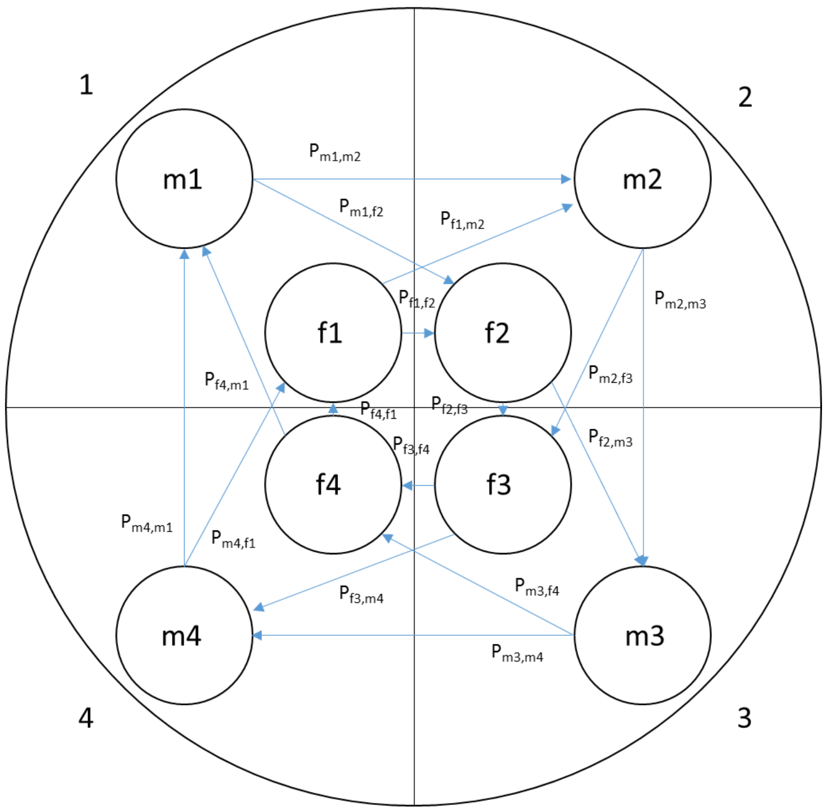
snare					X								X			
bass	X							X	X		X					

(Main beat data for the first verse of Taxman. Beats are separated by vertical lines, and each column represents a 16th note.)

The NBF quantizes its input as it parses it, meaning that triplets and subtle dragging/rushing will be ignored for the purpose of classifying the main beat of the song. Triplets are exceptionally rare in main rock drum beats, so we do not need to worry about losing them through quantization. Dragging/rushing only affects how the main beat is expressed, not what the main beat actually is, so the NBF does not need to know about this. The un-quantized input is used later for training purposes.

2.3 The SegmentationChain

The SegmentationChain ("SegChain") is a modified Markov chain whose purpose is to learn the transitions between the main beat and fills. It is a Markov chain consisting of eight states, one for each combination of "main beat" and "fill" per beat of the measure. Transitions are allowed only from states representing a certain beat of the measure to states representing the following beat of the measure.



(An abstract SegChain)

In order to train a SegChain, the input must first be segmented beat-by-beat with the binary classification of “main beat” or “fill”. This is done by using the output of the beat-finding algorithm and parsing back through the input to see which beats of the song are the main beat; all others are considered to be fills.

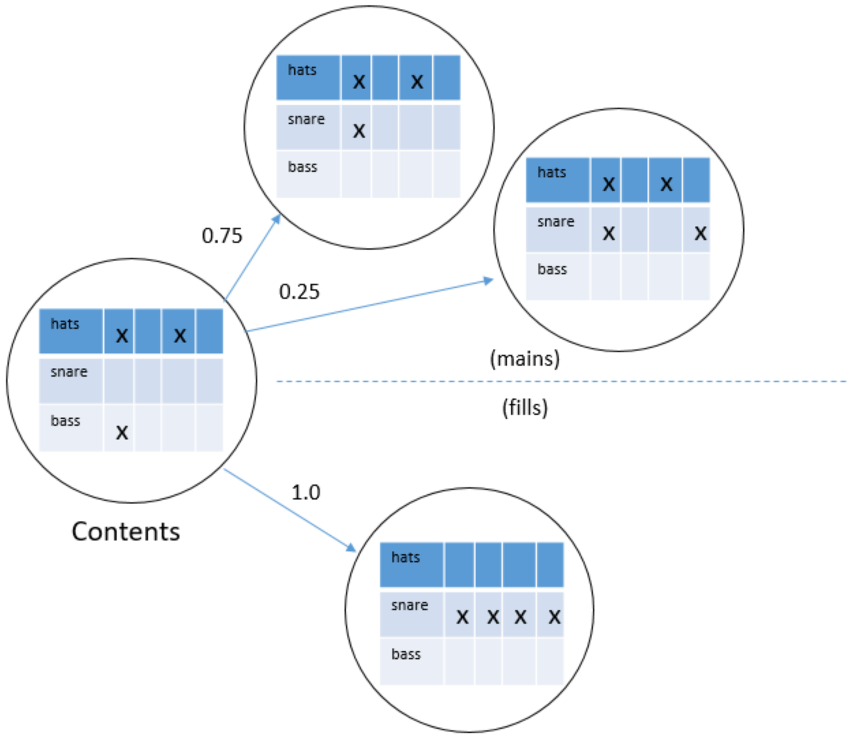
```
Main, Main, Main, Main,
Main, Main, Main, Main,
Main, Main, Main, Main,
Main, Main, Main, Main,
Fill, Fill, Main, Main,
Main, Main, Main, Main,
Main, Main, Main, Main,
Fill, Fill, Main, Main,
 Fill, Fill, Fill, Fill
```

(Segmentation data for the first verse of The Beatles’ song Taxman. Beats are indexed sequentially as they occurred in the song)

The contents of a beat of a song are considered to be part of the main beat if they contain all elements of the part of the NBF output occurring on the same beat of the measure. This is done in order to distinguish flair (which is the main beat plus ornamentation) from fills (which are unlike the main beat).

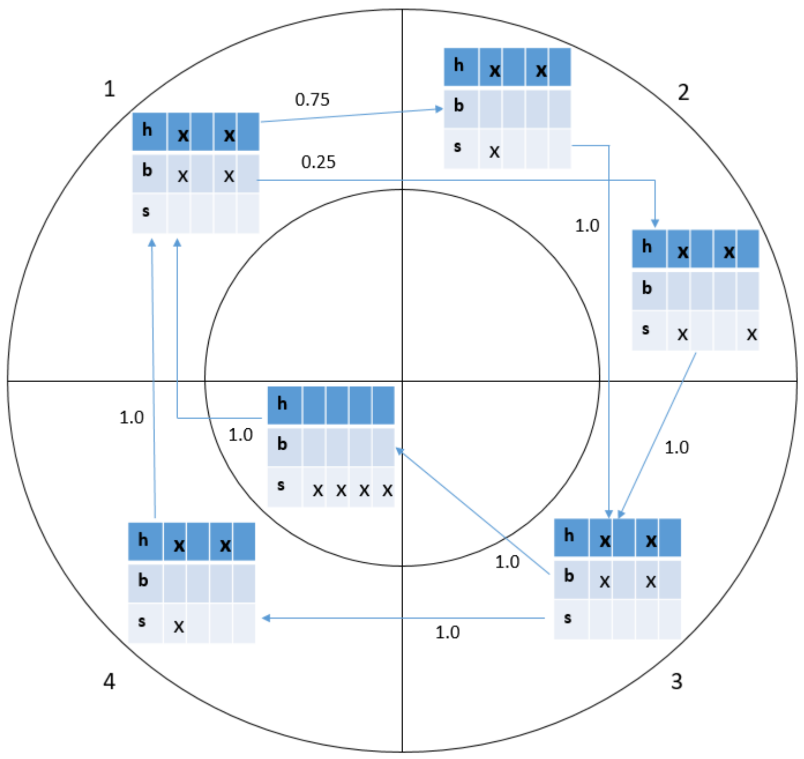
2.4 The BeatChain

The BeatChain is a modified Markov chain that learns the transitions between the contents of the input songs. BeatChain states (“BeatStates”) are one-beat patterns from the input corpus, just like the states of the first two Markov chains described in section 1.3. The data for these patterns is stored as a list of notes and, as a fraction, how far into the beat they occur. Unlike the states of the second Markov chain described in section 1.3, these states separately track transition probabilities to two different classes of states: main beat states and fill states. In other words, for each state that ever follows them sequentially in the input pieces, BeatStates keep track of the transition probabilities P(transitioning to state M_x | the state that follows is a main beat state) and P(transitioning to state F_x | the state that follows is a fill state).



(A sample BeatState. The states this one has pointers to are represented by their contents.)

To train a BeatChain, it is given the input piece in combination with that piece's segmentation information (main beat vs. fill). Training works by wrapping the contents of each beat of the input song into a BeatState and giving that state a pointer to the state that follows it if it doesn't already have one. If it does already have one, the BeatChain tells its pre-existing pointer that this state has followed it an additional time; this fact is used to calculate the transition probabilities after all inputs are parsed. The segmentation information used to train the SegChain is here used to tell each BeatState what the classification is of each state that it points to.



(An example BeatChain. The outer circle represents main beat states and the inner circle represents fill states. States in the chain are represented by their contents.)

The BeatChain is designed to handle unquantized input. Whenever a state is created from the contents of a beat of an input song, the BeatChain checks whether it has previously encountered a similar state of the same class and on the same beat of the measure. “Similar” here means that the MIDI events of which the states consist correspond to the same pitches happening within a specified margin of timing error. The two states correspond to functionally the same state, but have subtle differences in timing between their components. A timing error margin of a 64th note, or 1/16th of a beat, was chosen for this project.

BeatChain training also checks state similarity when it checks if a state already has a pointer to a specific state; if the new state is similar to an older one that it already has a pointer to, the training algorithm acts as though that older state has followed this state an additional time.

2.5 Playback

Playback works by first specifying how many measures of output is desired. The SegChain is then run for that duration, creating a sequential segmentation of when the output is to play its main beat and when it is to play fills. This segmentation information, along with the output duration, is then used as input for the BeatChain's playback algorithm. Two different playback algorithms were devised for the BeatChain, each with its own strengths and weaknesses.

The "normal" playback algorithm starts by selecting its first output state at random from the list of main beat states that occurred on beat one of the measure in the input corpus. We ensure this state has pointers to main beat states; if it doesn't, we fetch another at random from the list. From there, most output states are generated by having the previous output state transition according to the segmentation information created by the SegChain. This method of getting the next state by having the previous state transition is done in all circumstances except when a fill state transitions to a main beat state.

A special case for output state generation occurs when a main beat state is followed by one or more sequential fill states. In this case, rather than have the last fill state in the sequence transition to a main beat state, the output state is generated by having the main beat state preceding the fill(s) undergo several sequential transitions to main beat states. Specifically, it undergoes a number of sequential transitions equal to the number of fills that followed it plus one. To guarantee that this transition sequence does not get trapped in a dead end (i.e. doesn't unluckily transition to a state somewhere that does not have any pointers to main beat states), each temporary state in this sequence first determines which of its pointers to main beat states won’t necessarily result in a dead end if transitioned to. The probabilities for these states are then normalized, and the state transitions to one of them based on the normalized probability distribution.

Fills are chosen contextually if possible (i.e. if the current output state is told to transition to a fill state and it has pointers to fill states, it will transition to one of them). However, it is often the case, even on larger data sets, that the output segmentation will dictate that a BeatState with no pointers to fill states transition to a fill state. In this case, a fill state is chosen randomly from the list of ones encountered on the current beat of the measure. This is generally permissible due to the relative interchangeability of most drum fills; their purpose is to deviate from the backbeat and play an alternate rhythm, so a fill in one song can often fit perfectly fine in another.

An interesting behavior of this playback method is that it is capable of transitioning between different main beats that share common states, which we refer to as “waffling” between different beats. Further explanation of how waffling manifests can be found in section 3.3 of this paper.

The “beat waffling” behavior in the normal playback method led to the creation of the “anti-waffle” playback method, which was explicitly designed to not have this behavior. The anti-waffle playback method differs from the normal playback algorithm in that it pre-chooses its main beat output states. It does this by choosing a random main beat state for beat one of the measure and then storing it along with the results of three sequential transitions to main beat states. Whenever the output segmentation specifies that there is to be a main beat state in the output, the associated state from the list of pre-chosen states is used. Fills are chosen the same way as in the normal playback method: by context if possible or by random choice if not.

3. Testing

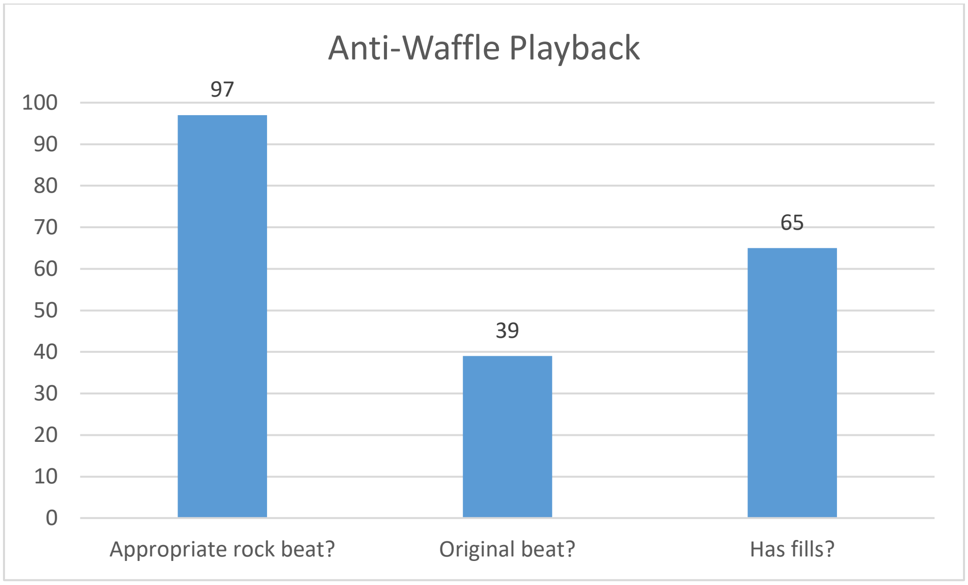
To test the efficacy of this paper’s proposed drum generation system, experiments were run using a total of 50 MIDI versions of verses and choruses from various Beatles songs as input. While all songs in the training set are by the same artist, the data set has a fair degree of variety. Compositions by The Beatles became far more musically intricate in the later part of their catalogue, and as such, the drumming on these songs tends to be different than the drumming on early Beatles songs. Beats found in the training set include a standard rock beat, variations on a standard rock beat, a surfer beat, a half-time rock beat, a beat with toms in it, and a beat with no cymbals. This variety in style of input pieces serves to show how this paper’s drum-generation system can generalize on larger, more varied input corpora.

Many of the input pieces’ main beats share common states, but there are also pieces whose main beats share no states in common with any others. This serves to demonstrate how the drum-generation system can combine parts of similar main beats from the training set to produce original patterns. It also shows that the system knows which beats from the training set are unrelated to the others and should not be combined.

For testing, 100 output pieces were created using each of the two playback methods (for a total of 200 outputs). For consistency, all output pieces are played at a tempo of 120bpm and are eight measures in duration (a standard verse length).

3.1 Anti-Waffle Playback

To gauge the effectiveness of the anti-waffle playback method, we judged each output piece as to whether it is an appropriate rock beat, whether the main beat is original (i.e. it is not the main beat of any of the songs in the input corpus), and whether the piece has at least one fill. "Appropriate rock beat" means that it has a backbeat that is suitable to rock music. Non-half-time rock backbeats usually accent the second and fourth beats of the measure with the snare drum and the first and third beats of the measure with one or more notes on the bass drum. Half-time rock backbeats use the same pattern but spaced out over twice as much time.



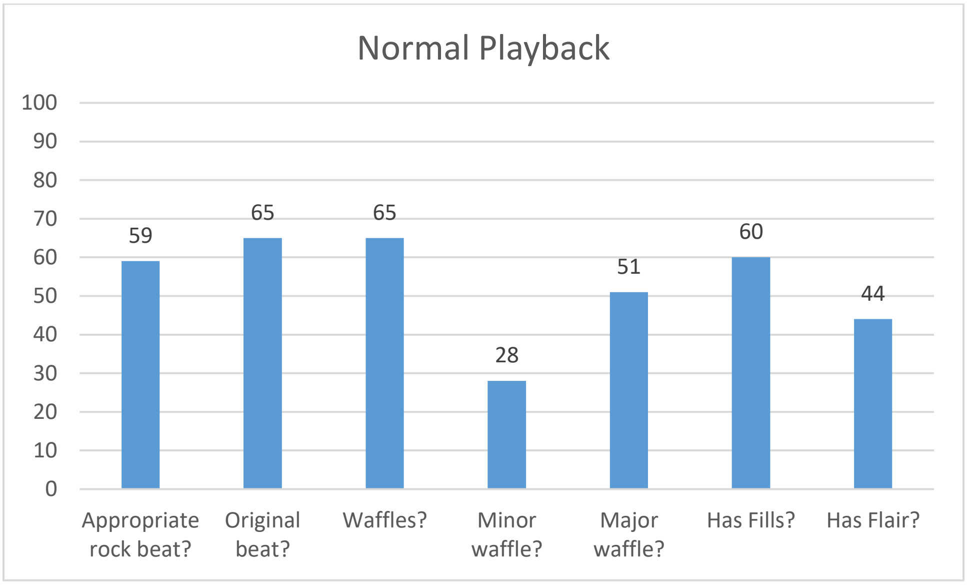
This playback method generated appropriate rock beats an impressive 97% of the time and generated original beats 39% of the time. Even when this method creates original beats, they are still reminiscent of the input corpus because they contain backbeats found in the input corpus. The most common way of generating original beats was by incorporating flair from an input piece as a part of its main beat. Of note is the fact that this playback method often created original main beats by playing a crash cymbal on the first beat of the measure. Crash cymbals are often used to accent the end of fills, and since fills often come at the end of sections of a song, the first beat of sections of songs often have a crash cymbal. If this playback method happens to pick one such state to play for the first beat of its output measures, the main beat will have a crash cymbal in it. This was not an intended feature, but since the main beat is still consistent when this happens, it ends up being an appropriate rock beat.

3.2 Normal Playback

Analysis of the normal playback algorithm's outputs proves to be more complex due to its beat-waffling behavior. This method's output pieces were judged as to whether they are an appropriate rock beat, whether the beat is original, whether the piece waffles at all, whether it has minor waffling, whether it has major waffling, whether it has at least one fill, and whether it has flair. We define "minor waffling" as waffling between beats that have similar backbeats, resulting in a section that feels like there is variation on a simple underlying backbeat. We define "major waffling" as waffling between beats that have notably dissimilar backbeats, which results in a section that feels like it completely changes main beats. We consider outputs that waffle to have original main beats.

In general, outputs that have major waffling are not considered to have an appropriate rock beat, as this behavior is not appropriate for most subgenres of rock. The exception to this is when one of the beats being waffled between is one measure in length and occurs immediately at the beginning or end of a piece; this has the characteristics of an intro/outro of a section of a song, which is a somewhat common practice in rock music..

Output that waffles can sound like flair in some circumstances. Minor waffling whose duration is two beats or less sounds indistinguishable from flair to a listener, so we treat it as flair for classification purposes.



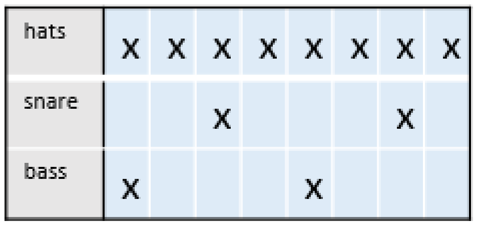
This playback method generated appropriate rock beats a modest 59% of the time and generated original beats 65% of the time, which always coincided with waffling. Outputs contained waffling 65% of the time, which manifested in various forms. Some outputs waffled to a new beat and never returned to the old one, while others waffled back and forth between the same two beats. Some outputs waffled between more than two beats, sometimes returning to an old beat and other times not. Some outputs exhibited both major and minor waffling in different sections of the piece. Furthermore, some outputs even exhibited both forms of waffling and had flair. Major waffling occurred in 51% of outputs and minor waffling occurred in 28% of outputs. 13% of outputs exhibited both forms of waffling.

Fills occurred in a slightly lower percentage of output pieces compared to the anti-waffle playback (60% versus 65%), but a minor difference like this is to be expected due to the randomness inherent in Markov models. Flair occurred in 44% of output pieces, and due to how the model stores flair along with main beat states, flair always occurred in its proper context. Flair often occurred alongside minor waffling, since the two phenomena are closely related as explained above.

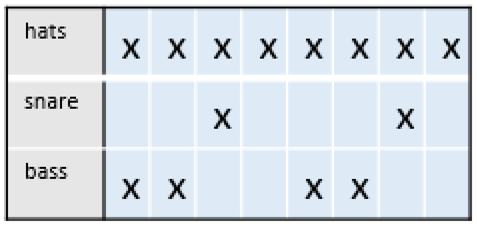
3.3 Analysis of Playback Methods

The normal playback method's main strength is that it can spontaneously generate flair, which is desirable in situations where a composer wants a drum pattern that is not rigidly regular throughout an entire section. The "beat waffling" behavior of this method is noteworthy, as it results in the creation of pieces that are far more musically complex than the pieces of the input corpus. If the beats being waffled between are very similar, the output tends to have a loose feel, mimicking an expressive drummer applying constant variation on a simple underlying beat. If the beats being waffled between are very different except for their shared state, the output can end up being very musically complex, sometimes with a "jam"-like[6] feel to it. Drums of this style could be desirable for composers working in rock subgenres such as progressive, art, or experimental rock.

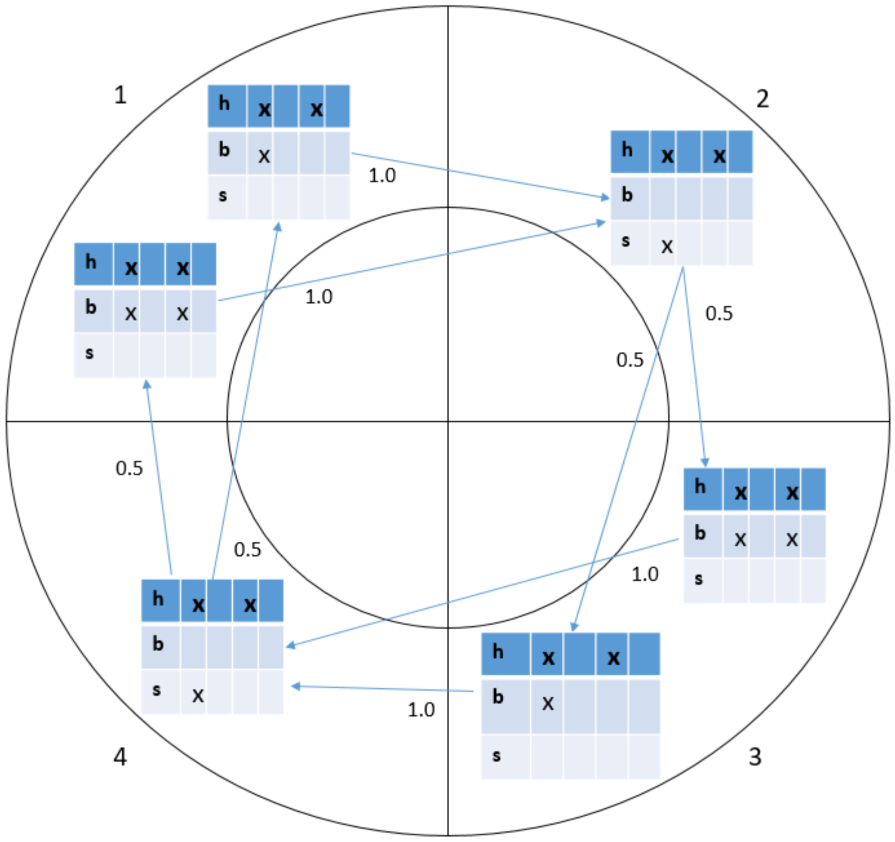
[6] "Jamming" is the act of improvising music with little to no planning for the overlying structure of the music being created. This can result in music that spontaneously changes style, tone, or rhythm.



hats	X	X	X	X	X	X	X	X
snare			X				X	
bass	X				X			



hats	X	X	X	X	X	X	X	X
snare			X				X	
bass	X	X			X	X		



(Example of a BeatChain, created from the two shown main beats, capable of “waffling”. Notice how the different backbeats are connected by shared states for the second and fourth beats of the measure.)

The anti-waffle playback method, as one would expect from its name, succeeds in preventing beat waffling. However, it is not capable of generating spontaneous flair due to the fact that all its main beat output states are guaranteed to be the same every measure. This playback method has applications where a composer wants a drum beat to be consistent throughout a section of a song, as a lack of flair in this circumstance is desirable. This is common in many rock subgenres, such as hard rock, pop-rock, and punk rock.

Both playback methods generate fills in appropriate ways. Placement of fills in output pieces was consistent with placement of fills in the input corpus. Specifically, fills in the input corpus usually ended on the fourth beat of the measure, returning to the main beat on the first beat of the following measure. Even in outputs where this pattern was not followed, placement of fills always sounded interesting rather than inappropriate.

Our system sometimes used fills from the input corpus in their entirety, but at other times it generated semi-original fills. One of the two main ways of doing this was by generating short fills using a section of a fill from the corpus that spanned multiple beats. The other method was by generating long fills by stringing together multiple fills from the corpus; sometimes the same fill was repeated sequentially, creating the feeling of deliberate repetition for effect.

It is this author's opinion that the anti-waffle method is more suitable for generation of general rock drum parts because of the consistency of its outputs' beats and the general unwieldiness of the normal playback due to its waffling phenomenon.

4. Conclusion

We begin our conclusion by suggesting ways that the research contained in this paper could be extended and/or improved. A good first step would be devising a playback algorithm that combines the spontaneous flair creation of the normal playback method and the lack of "beat waffling" in the anti-waffle playback method. Another way would be to use a more complex main-beat-finding algorithm, allowing our system to be applied to more musically complex subgenres of rock music and potentially to entirely different genres of music. Yet another way would be to apply the system to audio recordings by utilizing existing research in extraction of musical information, such as downbeat detection and polyphonic pitch detection. Lastly, we suggest altering the dual-Markov system by experimenting with higher-order or variable-order Markov models, as this may serve to prevent beat waffling.

This paper’s combined Markov-chain-approach to drum generation successfully creates rock drum parts reminiscent of the input corpus, which consists of verses and choruses from various Beatles songs. One of the playback methods creates drum parts with a consistent main beat; the other method sometimes does this, but often creates more musically complex drum parts with a variety of beat variations. The expressive differences between these playback methods means this paper’s dual-Markov-chain-system can have applications for composers in a variety of rock subgenres. Both playback methods also have the capability of combining related drum beats from the input corpus into original beats not found in the corpus.

References

[1] D. Herremans, S. Weisser, K. Sörensen and D. Conklin, "Generating structured music for bagana using quality metrics based on Markov models," *Expert Systems with Applications,* pp. 7424-7435, 2015.

[2] W. Schulze and B. van der Merwe, "Music Generation with Markov Models," *IEEE Multimedia,* pp. 78-85, 2011.

[3] P. Chordia, A. Sastry and S. Şentürk, "Predictive Tabla Modelling Using Variable-length Markov and Hidden Markov Models," *Journal of New Music Research,* vol. 40, no. 2, pp. 105-118, 2011.

[4] J. A. Biles, "Genjam: A genetic algorithm for generating jazz solos," *ICMC,* vol. 94, pp. 131-137, 1994.

[5] D. Cope, "Computer Modelling of Musical Intelligence in EMI," *Computer Music Journal,* vol. 16, no. 2, pp. 69-83, 192.

[6] A. Eigenfeldt and P. Pasquier, "Considering Vertical and Horizontal Context in Corpus-based Generative Electronic Dance Music," in *Proceedings of the Fourth International Conference on Computational Creativity*, Sydney, 2013.

[7] M. Puiggròs, E. Gómez, R. Ramírez and X. Serra, "Automatic characterization of ornamentation from bassoon recordings for expressive synthesis," in *9th International Conference on Music Perception and Cognition*, Bologna, 2006.

[8] D. P. Ellis and J. Arroyo, "Eigenrhythms: Drum Pattern Basis Sets for Classification and Generation," in *ISMIR 2004: 5th International Conference on Music Information Retrieval*, 2004.

[9] H.-H. Shih, S. S. Narayanan and J. Kuo, "Automatic Main Melody Extraction from MIDI Files With a Modified Lempel-Ziv Algorithm," in *International Symposium on Intelligent Multimedia, Video and Speech Processing*, Hong Kong, 2001.

[10] M. Goto, "A Chorus-Section Detecting Method for Musical Audio Signals," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Hong Kong, 2003.

[11] N. C. Maddage, "Automatic Structure Detection for Popular Music," *IEEE Multimedia,* pp. 65-77, 2006.

[12] D. Meredith, K. Lemström and G. A. Wiggins, "Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music," 2003.