Nick Collins
Centre for Electronic Arts
Middlesex University
Cat Hill, Barnet, Herts, EN4 8HT
UNITED KINGDOM
n.collins@mdx.ac.uk
http://www.axp.mdx.ac.uk/~nicholas15/

Algorithmic Composition Methods for Breakbeat Science

Abstract

Algorithmic composition methods are applied to the generation and manipulation of beats for dance music. The goal is to create an output beat in a consistent style with continuous subtle variation rather than stale repetition. In particular two tasks are studied in depth. The first is the generation of a beat itself using algorithmic composition methods to maintain variation. Examples are given in the context of UK garage/two step. The second is an automation of breakbeat cutting, as primarily found in drum and bass, though the model and its generalisations can be applied to any music and source audio segment. A side effect of attempting to create these procedures is an enhanced modelling of the styles and therefore a contribution to analysis of the genre.

Keywords- algorithmic composition, breakbeat

1 Introduction

The techniques described herein show application of relatively simple algorithmic composition and audio processing alongside modelling of some aspects of breakbeat science. Previous exploration in the dance music field has typically involved working laboriously by hand. Algorithmic composition can help to automate certain processes, and to generate subtle variations on basic beats without breaking with the consistency of a style.

The only algorithmic composition tool with direct relevance to dance music with much current exposure is Koan [3]. It is certainly applicable to generating techno tunes that can change pattern every bar but is orientated perhaps most towards ambient background music. Koan is restricted by its interface functionality, and has none of the real versatility of music programming languages like Csound or SuperCollider. Programming languages are necessary because tinkering with the fine details of the algorithms in this paper is part of getting original usage from these ideas. Particular compositions are likely to need specific modification. The second task, of automating breakbeat cutting, is however susceptible to being packaged as a singular process with relatively few control parameters.

Our work has a side effect. Since we are forced to produce models of the rhythmic phenomena, there is a quality of analysis of these new musical styles.

There is no claim in this to replace the human's aesthetic role. Some of the drudgery can be automated, retaining the character of typical human decisions. The human composer is freed to judge from a higher level of compositional decision making. We are working towards tools that can assist production, but not to replace

the final decision about what works as a groove or riff. The producer may finetune algorithmic composition methods to be consistent with their aesthetic needs for a particular track since the models can be adapted for specific circumstances.

Where decisions are taken, the expertise of the models are hard coded in the functions and data, so any choice from the available range for a parameter should be correct. We are not trying to search for a solution amongst competing constraints, but following rules precisely. There is no need for backtracking or any of the more complex apparatus of algorithmic composition.

There are no neural nets in this study, though it would be exciting to attempt to apply them say, in imitating successful breakbeats or breakbeat cuts.

2 Task one- generation of a subtly varying beat

Dance tracks can tend to use repeating beats that do not vary at all. This can often become monotonous and irritating to a listener, but then, creating continuous but subtle variation over a six minute dance track can be very tedious for the producer. This is a situation in which algorithmic generation is ripe for exploitation in dance music. Since one would not wish to vary a beat so much as to distract from other elements present in a track, we will be dealing with subtle variation. We envisage that we have source samples for a kick drum, snare etc, which must be rhythmically placed in time. We can also use subtle signal processing of each sample to vary the audio stream somewhat. Because the brief is a general one, we hone in on a particular sub genre for a practical experiment.

2.1 A model of a UK garage beat

UK Garage might be characterised by a relatively high tempo, perhaps 140 bpm, with a definite triplet swing groove, and the renowned '2 step' feel. Usually the second and fourth beats in the 4/4 measure have a definite snare strike, whilst the first and third beats would contain a kick at some point. The two step itself is a feature of later drum and bass work, beginning around 1995 following the breakbeat origins of the field. It is characterised by the pattern (kick snare kick snare) where each strike is on its respective beat except for the second kick, which occurs on the second quaver of the third beat.

The amount of swing can vary very slightly, and is usually slightly less than a half semiquaver triplet delay for the second of two semiquavers. In proportions, a straight semiquaver triplet split of a half beat is [0.333…,0.1666…]. For this style, the straight techno feel of [0.25, 0.25] is transformed to something between [0.32, 0.18] and [0.33, 0.17]. The groove level cannot be adjusted by too much without creating something outside the style.

Sidestepping groove, we could model a simple dance beat that does not use rolls by a grid of 16 onset points per bar. These correspond to the semiquavers in a straight techno groove, and the time onsets of every second semiquaver are delayed for the swung grooves. Historically, famous old step sequencers like the TR-808 were set up in this idiom. Dealing with triggering of drum samples for a beat, we simply algorithmically compose patterns of onsets a bar at a time. Sixteen bits (a word in computer parlance) is sufficient to store the binary yes/no (trigger/don't trigger) decision for a single bar.

The beat is modelled here as being composed of three independent streams, for kick, snare and hihat. Below we give two methods that generate bit patterns for UK garage.

2.1.1 Version one- probability templates

In this method we have a probability that a strike will be triggered for a particular time slot. A working set of probabilities (from 0.0 to 1.0) are:

```
//kick
[0.7, 0.0, 0.4, 0.1,    0.0, 0.0, 0.0, 0.2,    0.0, 0.0, 0.5, 0.0,    0.0, 0.0, 0.0, 0.3]
//snare
[0.0, 0.0, 0.0, 0.0,    1.0, 0.0, 0.0, 0.5,    0.0, 0.2, 0.0, 0.0,    1.0, 0.0, 0.0, 0.3]
//hihat
[0.0, 0.0, 1.0, 0.0    0.0, 0.0, 1.0, 0.7,    0.0, 0.0, 1.0, 0.0,    0.0, 0.0, 1.0, 0.7]
```

The reader will note that the snares on beats two and four always occur, whilst the kick does not have to sound at all in a given bar (we could of course use the generate and test paradigm and keep rolling dice until we get a run with at least one kick). The probabilities of each strike are all independent, though we could easily set up more complex transition tables where the probability of certain events changes based on what has already occurred. It is critical that each case, and especially the extreme cases are still valid beats. Snares on two and four with offbeat hats is the minimal beat, whilst the maximal beat with all events is not so busy as to be difficult to listen to.

2.1.2 Version two- genera

Here small rhythmic cells are selected one at a time. Each cell is an exact prescription for a rhythmic figure. The cells are particular to the target line. There would be some weighted choice amongst the cells, such that some are rarer figures than others.

```
//kick
genera 1- probability 0.7- [1,0,0,0, 0,0,0,0, 0,0,1,0, 0,0,0,0]
genera 2- probability 0.2- [0,1,0,1, 0,0,0,0]
genera 3- probability 0.1- [1,0,0,1, 0,0,0,0]
//snare
genera 1- probability 0.6- [0,0,0,0, 1,0,0,0]
genera 2- probability 0.3- [0,0,0,0, 1,0,0,1]
genera 3- probability 0.1- [0,0,1,0, 1,0,0,0]
//hihat
genera 1- probability 0.7- [0,0,1,0]
genera 2- probability 0.3- [0,0,1,1]
```

Note that the genera patterns are not all one bar long. This can cause genera to appear displaced from their typical position. So the first bar long kick genera could start half way through a bar, giving an out of phase 2 step beat. It is also in the nature of a naïve choice for a genera to be repeated many times. This can be tempered by more complex choice schemes that deliberately avoid or discourage repetition (see [1]).

With three independent streams, there are problems with predicting every simultaneity that may occur. Then again, we do not want to prescribe and test every possible combination, since that is more work than doing the work by hand in the first place! A few filters (an automatic critic) can cut out any exemplars of bad style, like simultaneous kick and snare, as explored next.

It should be noted that the probability template first version can be reduced to the genera method by finding the probabilities of specific patterns of onsets. A human

would not think out a probability template in this way though, so the methods are distinct in terms of likely human planning.

2.1.3 Further variation

After a bar's beat is rendered, we can choose to discard particular simultaneities. So, we might mask the bit pattern for the kick with that of the snare. We can perform other bit operations to create variations of a given pattern.

There are also varying amplitude ranges for each possible strike position, indicating the allowable strength of a trigger at that position.

In rendering a beat, we also subtly vary a sample each time it is used. This avoids the aural monotony of hearing exact repetitions of the component sounds of a beat. Whilst ideally we would have a very large database of source samples, say one hundred strikes of the same snare drum, we can use processes like filtering, pitch bending, distortion and reverb to create our own variations. Unless an extreme effect is desired, each would be varied very subtly with each new use. A pitch bend could remain within less than an eighthtone of the original pitch, so in the range of speed multipliers 0.986 to 1.014. This avoids any perceptible false melody of more than quartertone step in a drum part, but gives a subtle aural stimulation.

2.2 Results

The methods are simple, and stand or fall on the efficiency of the input data. Probability templates can easily sound excessively busy, whilst genera cells must be of sufficient number to give variety, without introducing any inconsistent figures. Additional information must usually be coded into the model to obtain a workable output in the style. For example, these generation methods create bit patterns that change every bar. If more organisation is required, we must add extra information to our model. Typical dance music would group bars in multiples of four, and we might generate four bar patterns as AAAB where A is a standard beat, and B is a fill pattern. Any A could have some extremely small variation of a base pattern, perhaps with a single kick displaced by one bit position, or one extra snare. B could be a different fill each time, or again subtly varied.

Incidentally, it has been observed that genera methods are fantastic for continually varying drum and bass patterns. The simplistic sixteen bit pattern model is ignored, and we work with floating point values of beats, so we can represent any fast rolls. The genera do not have to add up to 4/4 bars, though in practice, a primary kick and snare would keep a stable backdrop, to convoluted and changing rolls on a secondary snare sound.

2.3 Extensions and further work

We have already mentioned the use of bitwise operations on the 16-bit patterns, and the value of sectional thinking at a higher level than individual bars. However, whilst methods like cellular automata give processes to muddle the bit cells of bar beats, it should not be assumed that stylistically consistent beats will result from such processing. The key is still subtle variation, and the dependence on input data.

As with any form of algorithmic composition, if we keep track of our coverage of the probability space, we can guide future choices to new territories or control repetition with a slow variation. This is Charles Ames' method of statistical balance [1].

A consequence of modelling dance music, in particular beats, is that we can attempt to morph beats one into another. We require a common model to morph parameter wise to avoid abrupt changes. The reader can no doubt think up many schemes involving the bitwise interpolation of beats, the gradual change of probability templates or the substitution of genera. Initial experiments have been very promising, but a word of warning. Humans are very sensitive to the groove, and attempts to morph from a straight techno feel to the UK garage swing gave disastrous intermediates inconsistent with any normal dance style. A discontinuity would be needed in this instance.

2.4 Implementations

This work was first carried out with the author's own algorithmic composition test bed, software implementing its own input language specifically for beat generations and the rendering of subtly varying output streams. Whilst the author could carry out some interesting experiments with this code, the realisation came that the input data itself was often the most critical thing. The actual model is straightforward, and a language like SuperCollider (Mac only) is perfect for exploring combined algorithmic composition and signal processing methods in real time. Alternatively, the more primitive but better known language of Csound could be utilised alongside custom code for generating score files.

3 Task two-automatic breakbeat cutting

The second task concerns the arena of drum and bass ([2] gives a good background encyclopaedia). Sources confirm that over more than ten years of breakbeat manipulations in dance music, the dominant methodology is working by hand ([4]). Whilst software tools like Recycle or WaveSurgeon ([3]) help in the preparation of 'hits', that is, samples of kicks, snares, portions of a beat etc., these are then triggered via MIDI from a sequencer to create breakbeat cuts and beats. The MIDI events are composed by hand. If we can model the choices of event position, we can algorithmically generate similar sequences, with subtle variations to extend interest.

We do not claim to encapsulate the whole of breakbeat cutting science with this method, but the basis is there for new compositional directions based on algorithmic generation.

3.1 The model- partitions and repetitions

Given a block of time, we may subdivide this block into as many equal units as we please. From this come the duration names, quarter note, eighth note, sixteenth, being integer subdivisions of a whole note. If our block to be subdivided is a 4/4 bar, and we cut into eighth notes, then we could typically set up rhythms based on partitions by concatenating adjacent eighths, for example:

3+3+2 over the 8 quavers of one bar, or 3+3+3+3+2+2 over the 16 of two

where 2 means an event of two eighth notes (equals one quarter notes) duration. These sorts of partitions are prevalent in music that seeks excitement. They allow one to syncopate against the beats of a 4/4 measure, whilst recovering in time to meet a downbeat. This is the basis of our idea for automating cuts. We compose rhythmic figures based on syncopated 'odd' integer durations, which eventually even out using

the remainder of the space available before a downbeat. We do not have to restrict ourselves to cuts resolving within a single bar, as the 3+3+3+3+2+2 example shows. Mathematically, the algorithm will come down to choosing particular partitions that favour syncopation based on odd durations.

There is one slight complication for the application of the algorithm to the cutting of breakbeats. In practise, portions of a breakbeat are often repeated, as if the beat is being restarted on an offbeat. When 3+3+2 gives the durations in eighth notes of cuts, the second three quaver duration event could represent the retriggering of the same portion of a breakbeat as the first three. The algorithm presented will deal with this by taking a number of repeats. The algorithm below will also allow a 'stutter' effect caused by a small portion of the source being repeated many times.

The input parameters of the model are:

SourceSample- a (breakbeat) sample. It is assumed to be an integer number of bars long at the given tempo.
Tempo- the tempo of the source sample
SubDiv- the subdivisions of a 4/4 bar for the purposes of cutting. Take 8 for really standard drum and bass cuts. It is assumed that SubDiv >4.
MaxPhraseLength- the maximum number of bars in a phrase, that is, a partition.
RepeatChance- a probability from 0 to 1 that a repeating one subdiv unit duration sample will complete a phrase (in some circumstances). This is a stutter effect of cliché potential in the drum and bass idiom. 0.2 is a good value to avoid overuse.
MaxRepeats- the maximum number of times a portion of the breakbeat may be repeated within a phrase. The default is 2.

The appendix gives a simplified version of the algorithm in pseudo C code. We try to give some essense of the calculation here.

The algorithm calculates through one phrase at a time. We talk of a unit as being the duration of a single subdivision of the 4/4 bar, that is a SubDiv-th note. Each phrase corresponds to N bars, and hence N*SubDiv total units. The partition is created proceeding by one cut at a time, choosing (randomly) a length for the cut and some number of repeats from 1 to MaxRepeats that will keep us within the number of units left to fill. There may be a 'one unit repeater' to end a phrase, which will still be represented in the same way, using one unit cut and as many repeats as units remain to fill.

Generating the cuts information, we can then render the appropriate audio block at the necessary point(s) (for repeats) in time, with a slight envelope to avoid clicks. We access the source audio by the position within a phrase modulo the unit length of the source. So a one bar source can supply data for an N bar phrase. Repeats will utilise a previous access position.

The critical part of the algorithm is in choosing the cut length. Choose too long, and over time, too much of the original breakbeat tends to be revealed, diminishing the rhythmic virtuosity characteristic of this style. Further, the cuts can start to sound a little unnatural and forced. This algorithm then tends to favour compact short cuts from the source sample. We choose odd numbers of units up to half of a single bar. The phrase length in units is irrelevant to this, for if we take cuts over a bar, we tend to just sound like we're repeating the original breakbeat too much.

Given SubDiv, let

ODDNUMBERS= {n integer: n is odd, and n<= SubDiv/2}

These give us our possible cut lengths. We can generate a random member of the set by this procedure:

Let TEMP= SubDiv/2 (integer division).
If TEMP is even, take MAX as (TEMP-2)/2, else MAX= (TEMP-1)/2
CHOICE = randominteger(0, MAX).
Convert CHOICE to an integer as CHOICE= 2*CHOICE +1.

Now that CHOICE has been created, we must make sure it fits the available number of units left in the phrase, Reduce CHOICE by two units a time until it fits into the remaining space.

Finally choose a number of repeats from 1 to MaxRepeats. If we overshoot the remaining space (by having too many repeats) then take a single block of size the remaining number of units. This gives the chance of a terminating segment, like the 2 in 3+3+2.

3.2 Results

Figure 1 demonstrates three partitions created from running the algorithm on three bars worth of units with SubDiv = 8. The filled blocks represent repeats, with a stutter effect finishing the first example.

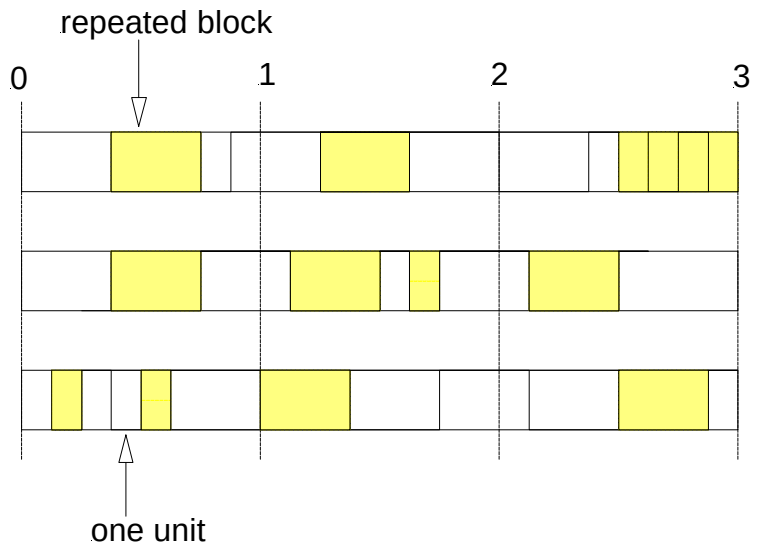

Figure 1 three bar, 24 unit partition examples

The model works really well on a SubDiv of 8. The syncopations are dynamic enough to be interesting, but not so complex as to lose the listener. The sense of time is not compromised by blocks that cut across bar lines, since the source sample is accessed modulo its own length. Since the model assumes the source sample is an integer number of bars long at the given tempo, then access beginning of the seventh eighth note of the first measure will sound relevant and correct. Because cuts in eighth notes are the stock in trade of style, the model has been optimised with this situation in mind.

Cutups which involved 5 unit blocks tended to sound more awkward than three unit blocks. This is the motivation for keeping possible block sizes less than or equal to half a bar. Yet to avoid the clichés of the style, allowing 5 unit blocks gave a greater originality. Still, 7 unit blocks would reveal too much of the original breakbeat. So for this modification to the algorithm, use the set:

ODDNUMBERS2= {n integer: n is odd, and n<= (SubDiv/2+1) }

A cutup using 16ths is more easily interpreted if played against a solid backing beat. Then it becomes a very dextrous cutup indeed. This is only if the groove of the breakbeat original is in straight 16s. Otherwise the algorithm must be adapted with groove information, as below.

As for subdivisions which have prime numbers other than two in their decompositions, these play against the source breakbeat, which is usually quantised in rigid 16s. Used to cut up source audio that is not itself rhythmic, they can lend their own rhythmic flavour without any clash. Or the clash is itself an interesting musical effect.

3.3 Extending the Model

The design of phrases could show a lot more care. We might want to favour the choice of larger blocks to begin, with smaller ones to end. We can keep track of all choices of partitions made so far, and always guide the choice towards examples so far ignored in the probability space (this might use Charles Ames' method of statistical balance [1]). We can vary the Subdivision level based upon position in a phrase. A stuttering completion of the phrase might use 16s or 32s for those manic roll effects, while the main cuts are in 8s.

There is an essential issue of groove. Levels of swing or even more complex groove templates would have to be built into the algorithm to cope with crooked 16^{th} note cuts that preserve a 'swing' in the original source sample.

At base we assume 4/4 measures, without change of time signature or tempo. The algorithm can be adapted to work from more complex changing time signatures by taking lowest common multiples of the time signature denominator. Tempo changes are easily dealt with, since we just change the time positions of cuts after the partitions have been calculated.

The ability to work from a changing set of time signatures would have the helpful side effect of dealing with cases where we wished phrases to be subdivided to favour particular beats. So, for instance, a phrase of three bars of 4/4 could be rewritten as three phrases of 3/4+4/4+5/4 to force a strike on the first, fourth, seventh beats.

There is a quick fudge to enable the algorithm as is to work with a n/d time signature. Multiply the subdivisions by n/d, and the tempo by d/n. Then to the algorithm, it is dealing with a 4/4 bar, but the output will be cut in the requisite way.

The reader will of course see their own generalisations of the algorithm. The exposition here gives a model helpful for automating the cutting of breakbeats in a consistent style. The same algorithm can be adapted for any sort of source audio, and any partition effect on macro down to micro level sample triggering. The assumptions of the model can also be ignored for some strange rhythmic effects. This may be used subtly, as in telling the model a tempo slightly above that of the source, for a catch up effect, or an underestimated tempo, for an overloading effect.

3.4 Implementations

Implementations of the model have been made in SuperCollider for realtime use, and in a stand alone Win32 application for non-realtime (much faster than realtime) rendering. It would be relatively easy to set up implementations in Csound, or any other desired rendering language.

4 Conclusions

In this paper solutions were described to two tasks using algorithmic composition techniques, with application to, though by no means limited to, dance music. The construction of models gave analytical insight into dance music styles. Using music programming languages like SuperCollider allows the ready adaptation of this work to specific compositions, but specific processes could be packaged in a limited interface tool. The use of algorithmic composition systems for specific higher level composition tasks in home virtual studios will surely gain precedence in the next few years. The tools themselves will have their own distinctive features, making their use recognisable much like synthesis systems or sound processing tools can be distinguished. Hopefully, the tools will inspire new creative uses, and not just blind imitation of past masters.

5 Bibliography

5.1 Publications

[1] Ames, Charles. 1990. Statistics and compositional balance. *Perspectives of New Music*. 28:1.

[2] Shapiro, P. 1999. *Drum 'n' bass, the Rough Guide*. Rough Guides Ltd.

5.2 Web Sites

[3] software
Recycle- http://www.propellerheads.se
Wave Surgeon/Mobius- http://www.squarecircle.co.uk
SuperCollider- http://www.audiosynth.com/
Koan- http://www.sseyo.com/

[4] techniques
breakbeat cutting
good FAQ- http://www.dnbproduction.com/index.asp
very good techniques area- http://spinwarp.com/

general (some tips and tricks, discussions of groove)
http://www.dancetech.com/

6 Appendix- Pseudo C code for automatic breakbeat cutting

Local variables and variables persisting between calls to this function are not declared here.

```
//while loop to desired length output (non realtime),
//or scheduling itself for next decision(realtime)
{
if (unitsdone==totalunits)    //NEW PHRASE
{
//How many bars?
barsnow= randominteger(1, MaxPhraseLength);
//prepare units counter
totalunits= barsnow*SubDiv;
unitsdone=0;
repeatsdone=0; repeats=0;
}

unitsleft= totalunits- unitsdone;     //indirectly gives current time position

if(repeatsdone==repeats)   //NEW CUT
{
repeatsdone=0;
//How large a cut?

//ending a phrase with a repeating unit block? can only happen if less than a
bar left
if(unitsleft<SubDiv && (randomfloat(0.0, 1.0)< RepeatChance))
{
unitsincut=1;
repeats=unitsleft;
}
else
{
temp= SubDiv/2;

if(temp % 2==0) temp= (temp-2)/2;
else temp= (temp-1)/2;

unitsincut = randominteger(0, temp);

//turn unitsincut into an odd number less than SubDiv/2
unitsincut = 2* unitsincut +1;

//unitsleft is at least 1, and unitsincut is odd, so will terminate
while(unitsincut >unitsleft) unitsincut -=2;

repeats=randominteger(1, MaxRepeats);

//if overshoot on repeats, allow a terminating segment
if(repeats*unitsincut>unitsleft)
{
unitsincut= unitsleft;
repeats=1;
}
```

```
}

//prepare indexing info into the SourceSample.
//take current position within phrase modulo the SourceSample length.
///(for accurate breakbeat cutting, assumes that the SourceSample is
//in 4/4, and an exact number of bars long at the given tempo)
//store the sample indexing data so that each repeat can access the same
block
//CODE NOT REPRODUCED HERE
}

{       //CARRY OUT AN EVENT
//have source indexing data for this repeated block

//use unitsleft to get start time position, unitsincut gives duration

//RENDER BLOCK- CODE NOT REPRODUCED HERE

unitsdone += unitsincut;
repeatsdone += 1;
}

}
```