Parasitic Frankensteinism: Extraction of Repetition schemas of musical features for use in an algorithm for composing music

Douglas Armstrong

27 November 2017

Abstract

Repetition is an intrinsic part of music. Schenker (1954) suggests that it is repetition that is fundamental to the shaping of form in music. Repetition of musical material is particularly prominent in electronic dance music, which stems stylistically as much from the western art music aesthetic of the twentieth century, as the influence of African music(through jazz and its more populist descendants like blues and rock and roll, as well as African World Music) and the practicalities of making music using machines. Furthermore, there is a compelling connection between music and mathematics as attested by scope of the field of Algorithmic Composing research. (Fernandez and Vico, 2013)

This research seeks to explore an automated analysis process to identify musical features that are possible to derive from a standard MIDI file. From this information we further seek to identify the repetition of these features. Through a process where these repetition schemas are recombined with new musical features, we will generate a new musical output. This algorithm will be tested qualitatively for its ability to generate appropriate musical material for three different musical situations, covering the role of a composers' assistant in the preparation of new works and as a live instrument where the manipulation of the parameters might be considered a performance.

The author would like to acknowledge the support of the AFDA School of Motion Picture Medium and Live Performance for this research, as well as family, friends and colleagues who gave the support, feedback, inspiration and collegial atmosphere that made this possible.

Contents

List of Figures

List of Tables

.

1 Introduction

Repetition is a fundamental part of music, and may be fundamental to our perception that music is cohesive. Schenker (1954) suggests that it is repetition that is fundamental to the shaping of form in music. In this study, an algorithm that analyzes and reuses easily accessible repetition data to create new musical works is proposed, and the possibility that this algorithm may create new works with a better-than-random sense of cohesion is explored. Due to a low computational overhead, the parameters of such a system could be adjusted by a human user on-the-fly in a situation such as an improvised live electronic music performance, resulting in an electronic music performer or ensemble that is better positioned to evolve their music over the course of performances, freeing themselves from the rigidity of a musical form that is in many cases entirely creatively realised in pre production.

Problem Statement

A vast palette of sonic and arrangement possibilities has become available with the use of the laptop as instrument but many aspects of a live electronic music performance have become inflexibly limited to material realized in pre production, and the use of conventional instruments in the ensemble becomes a case of performing to a backing track, with very little possibility of any musical dialogue between electronic and conventional instruments. Furthermore, several new roles (instrumentalist, conductor, improviser, sound engineer, arranger) are simultaneously thrust upon the individual performer resulting in critical points of overwork onstage which compromise the musical output. Having had wide experience of musical performance with conventional instruments (trumpet, guitar, bass, keyboards) in a variety of popular music styles (rock, funk, reggae, punk, hip hop, jazz), in 2008 I embarked on a project centred on live electronic music performance using a laptop and some conventional instruments. This ensemble exposed a set of possibilities and limitations unprecedented in my prior musical experience. I realized it was time for the laptop to become more than just a powerful playback device. It was time to find ways that the computer could be used to increase the scope of live performance interactivity and shoulder some of the workload to facilitate an easier transition of the aforementioned onstage roles for the human performer.

A software tool as outlined in the introduction would be of interest to someone who would like to prototype new musical material based on adjustable

but deterministic processes and which would not involve composing the details of such a work. A skilled user might be able to use such a tool to generate musical material which could then be successfully organised into an extended music performance which would have a sense of musical cohesion, thematic continuity and novel variation of existing material. It might have applications for composers wanting a "friendly antangonist" (Cope, 1987) in their creative process, and it may also find use amongst students of composition should the output be eloquent enough about the composing process underlying it.

Aims of Research

It is assumed that the structure in which some type of musical parameter is repeated in a musical work is significant. It is also assumed that the actual material being repeated constitutes some aspect of the signature of the work. The process of identifying musical fragments would identify material that is repeated more or less often in the work. If the repetition schema of a certain musical fragment were used with a different fragment (for example, seldom repeated material in the place of often repeated material and vice versa), a new work would result which bore some of the signature material of the original, but definitely one that maintained a similar sense of cohesion due to the duplication of the repetition structure with new material. Given these assumptions, the aim of the study is to investigate how well a non cognitively modelled musical feature repetition detection tool can generate raw material from an existing corpus, for algorithmic generation of new musical works with a better-than-random sense of cohesion. The possibility of creating an algorithm that behaves deterministically based on input material, yet may have a low computational overhead opens the door for a tool that could be used interactively by a human user, changing generation parameters on the fly to further create a sense of long form musical development.

In this study, a software tool that uses number sequences to represent aspects of an existing musical work will be designed. Such number sequences could easily be derived from symbolic music formats such as MIDI. A simple pattern matching algorithm could easily identify repeated number sequences in a given corpus. This information would become the building blocks that would be used to construct new musical works. Thus, the final aim of the study is to design a user interface that would allow easy access to the parameters of the system, and allow rapid reviewing of new material generated by new arrangements of the system parameters.

Methodological Trajectory

- Design the architecture -
  - Criteria for corpus
  - Data objects
  - Output formats
- Design the analysis tool
- Design the selection tool - Human user interface(HUI).
- Design the generation tool
- Evaluate the tool in terms of its ability to sustain an engaging interaction with a human user. Factors to consider would be the quality of the musical output for a given genre, the speed at which new arrangements of parameters could be tested.

2 Literature Review

Musical features

The elements of music are described as follows (Schmidt-Jones, 2008):

- Rhythm - rhythmic pattern, meter, syncopation,
- Tempo - speed of the music
- Articulation - the way a note is played, instrument specific but could generally include such terms as staccato or legato.
- Dynamics - overall volume of the music
- Pitch - sequential series of pitches - makes melody when combined with rhythmic pattern
- Harmony - simultaneous sounding of pitches
- Timbre - the sound of the instrument or combination of instruments, as separated from melody or rhythm
- Texture - the combination of rhythm, melody, timbre etc that makes up the overall sound of a piece of music. E.g monophonic, homophonic, polyphonic, contrapuntal
- Form - overall structure of a piece, or, on a smaller scale, phrasing.

(Eurola, 2003:26) categorizes musical features as melodic, temporal, harmonic, structural and timbral. Therefore we could say that a musical feature may be described in terms of the elements of music.

Huron (2001) argues that at a most basic level a music feature is something that is at least evident in the work in question, although he does develop this by stating that it is entirely possible for the absence of some musical element to be stylistically significant to a work existing in a broader context of cultural product. Furthermore, a variety of musical features (he uses the example of a melodic interval) require some kind of interpretation to differentiate it from other material. In the case of polyphonic music this could be quite a sophisticated cognitive process. In terms of isolating musical features in a work to be analysed, he also brings up the issue of salience, with the implication that just because something exists within the piece, it is not necessarily a significant feature of the music. Table 1 represents a group of descriptive terms around salience:

Table 1: Some terms around salience (Huron, 2001)

General terms:	
presence	existent within an artifact
negative presence	absent from an artifact (although expected)
salience	noticeability of an event
distinctiveness	greater salience compared to occurrences in other artifacts
significance	notable, important, worthy of attention
Some intratextual factors contributing to salience:	
prevalence	noticeable because it recurs frequently
accent	noticeable because it is stressed (e.g., dynamic, agogic ...)
recency	noticeable because it is last
primacy	noticeable because it is first
mnemonic	noticeable because it is easily remembered
Some intertextual factors contributing to salience:	
evocation	unintended reminder of similar passage in another artifact
quotation	intended exact quotation from another artifact
allusion	intended indirect reference to another artifact
parody	intended exact or indirect reference, intended to spurn
model	intended or unintended borrowing of a structural framework

Repetition

> ...humans are similarity automatons regardless of domain, level, or operation. Such cognitive dependence of so much on so little stems directly from evolution, where the ubiquitous complexity found in life is comprehensible only because the high-level abstract rules governing our unconscious processing are so very simple – and for that reason so very powerful. (Narmour, 2000:218)

Narmour (2000) details many musical examples where listeners instinctively create rules of continuation around identifying repeated musical fragments. He argues that expectations are governed by a simple rule of continuation: If a musical item is repeated it leads to the expectation that this will be repeated

again. The musical item could be an actual series of pitches, or rhythmic placements or a rule as to how pitches may deviate from a simple repetition of the pitches or rhythmic placements.

Listeners are able to extrapolate new rules for cases where expectations are not fulfilled and project those cases into more complicated expectations about future musical events. He starts with the case of the musical sequence, where a musical motif is repeated but transposed up or down a short distance (generally less than or equal to a perfect 4th), which generates the expectation of continuation. He continues with various extensions of this rule making hypothesis:

- Melodic expectations based on incomplete repetitions or non obvious expectations (requiring a readjustment of the repetition rule formulated).
- Temporal events such as repeated rhythms, augmentation and diminution, rhythmic and durational additivity, tempo and pace.
- Harmonic expectations around closure.
- Structural symmetries such as melodic inversion and retrograde, additivity and subtractivity on a formal level.

Similarity

Urbano et al. (2011), in dealing with a method of evaluating melodic similarity using a geometric model which represents melody as splines[1] in a 2 dimensional pitch time frame, present a list of melodic similarity requirements. These are:

- Vertical requirements
  - Note equivalence – where the notes of two melodies are exactly the same.
  - Octave equivalence – melodies which would have note equivalence if one were transposed to a new octave.
  - Degree equivalence – melodies which would have note equivalence if one were transposed to the same key and octave.
  - Pitch variation – two melodies with note, octave or degree equivalence except for a few notes which are different.

[1] Splines are used in mathematics and computer graphics and can generally be "considered as a mathematical model that associate a continuous representation of a curve or surface with a discrete set of points in a given space" (Weston, 2002)

  - Harmonic similarity – a harmonised melody where the top voice has equivalence to an unharmonized melody.
  - Voice separation – where a melody extracted from a multipart texture would be equivalent to a monophonic melody.

- Horizontal requirements
  - Time signature equivalence – where two equivalent melodies might be written in different time signatures, ultimately sounding the same.
  - Tempo equivalence – where a sequence of equivalent notes might in one case might be written with double the durations, but played at twice the speed, ultimately sounding the same.
  - Duration equality – an equivalent melody played at different tempos would be regarded as highly similar.
  - Duration variation – where two melodies with an equivalent series of notes was played with some different durations.

Furthermore, the writers suggest some ways of taking these similarity requirements into account. The vertical equivalence issues could be dealt with by using a relative distance between successive pitches, or a notes distance from the tonic, if the key is known. Dealing with pitch variation would be a case of in some way measuring the degree of similarity rather than using an all or nothing approach. Harmonic similarity and pitch separation are assumed by these writers to have been dealt with in the preparation of the material for analysis, as they deal with monophonic melody. In terms of the horizontal considerations, the writers use the concept of score time and actual time. Time signature and tempo equivalence would consider the notes in terms of actual time, because it is in this domain that they are most similar. Duration equality and variation would need a score time representation as this is where they are more similar.

Algorithmic Composition

> A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer. [2]

Although algorithmic composition of music has found a distinct place for the use of 20th and 21st century computers, Maurer (1999) identifies several pre information age instances of the relationship between music and numbers.

[2] https://en.oxforddictionaries.com/definition/algorithm

Pythogoras devised a musical system which considered music and and numbers inseperable, and indeed a "key to the whole spiritual and physical universe" (Grout and Palisca (1996:843) quoted in Maurer (1999)) While it is debatable whether this was actually a fully fledged composition and performance system, it is a documented instance of a compelling sense of connection between music and mathematics, which is a theme I will explore further in this section.

We all probably know of the canon in terms of the children's song '*Frere Jacques*' or one of its translations ('*Are you sleeping*' (English), '*Bruder Jakob*' (German), '*Fra Martino*' (Italian), '*Martinillo*' (Spanish) or '*Vader Jakob*' (Dutch)) and readily understand the idea of a canon as being a melody which is can be sung by several voices each with a different starting point. What is less well known is that this is just one technique from the 15th century practice of singing and writing canons. According to Grout and Palisca (1996:843), these compositions consisted of a simple melody with instructions as to how further voices could be added. In the case of '*Frere Jacques*' that would involve a new voice starting the melody every four beats, but in other instances new voices could be inversions or retrogrades of the original. This 'canon' ('rule' or 'law') was in essence an algorithm to derive new musical material based on an existing work.

Mozart's dice game' was another example of using chance to assemble musical fragments into new compositions. John Cage embraced a variety of techniques to include indeterminacy into his compositions. Early 20th century serialism was a rigidly formalized system aimed at abstracting the compositional process in mathematical terms.

With the arrival of computers, with their almost effortless ability to handle hugely repetitive tasks, it was possible to develop algorithmic processes which were more involved than those including humans. Early efforts, possibly due to the emergence of computer access in the 1950s and the issues of art music composition at the time, seemed involved with the exploration of the use of randomness, but with the increased complexity of algorithmic composing solutions came a new understanding of the complexity of the composing task itself.

One of the problems encountered when tackling this extremely diverse field has been the issue surrounding classification of the material. For example, Papadopoulus and Wiggins (1999) use these classifications:

- Mathematical models - divided this section into stochastic, statistical and chaotic approaches.

- Knowledge based systems
- Grammars
- Evolutionary systems
- Systems which learn
- Hybrid systems

Nierhaus (2009:270) however, divides all algorithmic composing processes into either knowledge based or non knowledge based systems, and then subdivides them further, including, but not limiting himself to other items from the above list. It therefore seemed appropriate for me to exclude knowledge based systems from the above list as it appears to be a classification that is hierarchically superior to the other list items. The chapters of Nierhaus text on algorithmic composition are set out below, and show a strong bias towards functional classification, as would be appropriate for a work that is intended as a textbook for a course in the subject:

- Markov models
- Generative grammars
- Transition networks
- Chaos and self similarity
- Genetic algorithms
- Cellular automata
- Artificial neural networks
- Artificial intelligence

From these classifications, various problems arise. For instance: given that an Artificial Neural Network seeks to emulate the structure of the brain, how does one differentiate it from Artificial Intelligence? In the same way, given that generative grammars seek underlying structural similarities across languages (a product of human intelligence), is this not also a model of AI? If Nierhaus AI could be equated with Papadopoulos and Wiggins hybrid systems, why then do the latter include Systems that Learn which can be seen as an unavoidable aspect of the modeling of intelligence?

To add to this complexity, as I delved into the details of many of the systems related further on in this review, it emerged that many of these systems began to acquire elements of different computational models as the limits of their initial approaches became apparent to the researcher. Furthermore, it seems that all of these approaches focus explicitly only on the functional nature of the algorithm, leaving the compositional goals of the composer/designer unstated, and failing to factor user interaction as an integral part of the algorithmic process.

Fernandez and Vico (2013) divides the general field of computers and music into two general areas: the first is named Computer Aided Algorithmic Composition and is characterized by automation of the composing task by computer in constant collaboration with a composer. The second, termed simply Algorithmic Composition, involves far less human interaction and has much more to do with the discourse around computational creativity, a debate which is far beyond the scope of this thesis. This gives us at least the beginnings of a classification for the level of interactivity required of each system, and can open an explicit examination of the compositional goals of each system. Furthermore the authors identify several scopes for the output of any algorithmic composing tool. Some tools focus their attention on larger scale aspects of music (form, phrase, motif and note (which encompasses pitch, dynamic and duration)). The authors differentiate this from systems which deal with the nuances of expressive performance, for instance small deviations in pitch and timing which make up an expressive performance. Still other systems concern themselves with the actual synthesis of the output waveform where detailed attention can be paid to the final timbre of the output. The authors also separate out the field of procedural audio which specifically relates to underscore music for video games which needs to adapt itself in real time to the changing state of the game. Although this seems to be a special case of computer aided algorithmic composition the user does not bring any creativity to bear directly on the music, so we accept this as a specialized sub genre of algorithmic music.

Thus I examined several algorithmic music systems in detail, and evaluate each equally in terms of the following:

- Computational workings
- Compositional goals
- Interactivity

The Illiac Suite (1958) is cited as the first instance of the use of computerized processes in the composition of a musical piece by using a random number

generator and probability tables to generate musical information. Sandred et al. (2009) describe Hiller and Isaacsons process for the 4th movement of this string Quartet as employing these probability tables to filter note intervals proposed by the random number generator. This is an early example of the generator/critic structural model employed in many algorithmic composing models. The random number generator delivers interval options for every quaver of a 6/8 time signature. The probability table for the first two bars indicates that it will allow the interval of a unison to pass with a probability of 100%. The net result of this algorithm is two bars of sustained notes. (in Hiller and Isaacsons algorithm, a subsequent note of the same pitch results in the note being tied to its predecessor) By adjusting the probability tables over the course of the work, new intervals arise as possibilities. The composers further critical evaluation of material delivered by this process was used to fine tune the algorithm to generate a musical result that satisfied the goals of the project. Elsewhere in the suite there are examples of the use of Markov chains, where the choice of note is dependant on the previous note. Sandred et al. (2009) revisit the process used by Hiller and Isaacson in the 4th movement and further develop it to include constraints for the resulting harmonic possibilities, and generate valid alternative versions of this movement of the suite.

Hillers subsequent work took this model of constraint based stochastic composition, with final validation by the composer, and developed it further in such computer programs as MUSICOMP and PHRASE, as well as incorporating electronic sound synthesis in the realization of the final audio material (The Illiac Suite was performed by a standard string quartet). In their exhaustive methodological description of the composing of Computer Cantata Hiller & Baker declare MUSICOMP to be the outgrowth of several years experimentation in developing a logic of musical composition that is not bound to specific style parameters, historical or otherwise (Hiller and Baker, 1964:62). This puts into context the largely mathematical approach to the subject of composition (indeed, Hiller considered the Illiac Suite to be a series of experiments (Westergaard, 1959).

In the original preface to *A Generative Theory of Tonal Music*, Lehrdahl and Jakendoff (1983:ix) declare that the possibility of a correlation between the transformational generative grammar of Chomsky, in the field of linguistics, and a grammatical structure for music, was proposed by Leonard Bernstein in a lecture series at Harvard University in 1973. As a result of the interest generated by these lectures, Lerdahl and Jackendoff, as experts in the fields of

composing/music theory and linguistics respectively found common ground to collaborate on the aforementioned work, which was a major step forwards in tying together the fields of music theory, a grammatical basis for music structure, music cognition and psychology. This work formed the starting point of much research into further applications of generative grammar to music analysis and algorithmic composition. Nierhaus classifies Copes Experiments in Musical Intelligence (EMI) as an example of an Augmented Transition Network (ATN), but the system has a major functional module that uses a grammatical approach, and indeed Cope declares: "In the EMI program, it is assumed that musical intervals can be defined linguistically and that style emanates from the manner in which composers categorize these definitions (Cope, 1987)". This appears to best summarize the compositional intent, even though the use of statistical and Markov models abound in Copes system.

EMI can either generate an analysis of a complete work, or a fragment thereof, as a basis for a new composition, or the operator can directly enter material according to the classification system that the program requires for operation. In an analysis scenario, source material is analyzed for statistical appearance of various features. In the case of melody, this would include such tests as repeated notes, outlines of triads and leaps followed by steps (Cope, 1992:71). There are also analysis guidelines for durations and harmony. These musical materials are classified in terms of their function as antecedent, consequent, ornament and statement (ACOS). These ACOS classifications operate at 3 different levels (fore-, middle- and background), with the foreground being the least important surface feature of the music and the background being the most important. Rules are established which define which of these elements (in Copes terminology this is called an identifier (Cope, 1987)) can follow which other identifier. From these rules are derived a dictionary of possibilities which are combined, using a pattern matching algorithm with three variables (motive size, variance and interpolation). This material is used in the ATN to generate new material which then undergoes the same statistical analysis as the source material. If this analysis is within an acceptable minimum deviation from the original, the new material is stored as a file, otherwise the variables are adjusted, and the process proceeds again from the beginning. The program can back track if it reaches an unsatisfactory cul-de-sac.

Cope admits (Cope, 1992:71) that the type of analysis used depends on the source material. Processes which are successful in analysing short musical fragments do not work when a complete work is being used, so ultimately there

is both an automated and human critical process at work. EMI has been successful at replicating the style of various composers, and has (pleasingly, in my opinion) generated a degree of controversy with this output. Cope terminated development of EMI, which he began in 1984, in 2006. Recently he has been releasing the works of a new composing program, named Emily Howell, focusing on original material instead of stylistic imitation (Ahmed (2009); Blitstein (2010) & Christensen (2009)). Needless to say, the controversy continues.

Cope describes the initial motivation to develop the system as an exercise in applying musical logic on a computer, to be used as a "friendly antagonist" (Cope, 1987:30) during his own composing process, largely as a result of a bad case of writers block. The composer would supply seed ideas and the computer deliver variations thereof, to be accepted or ignored by the composer and fed back into the system to further explore the possibilities of the material. Furthermore, the scope of the output was always only note values, durations and dynamics. From these beginnings, Cope discovered the possibility of developing automated compositional models of musical styles which were capable of outputting complete works.

Todd (1989) describes a neural network system designed to learn the pitch and duration of a melody and then reproduce variations based on the original. An Artificial Neural Network (ANN) models (simplistically) the structure of the brain. Artificial neurons, with a variety of possible transfer functions, are connected in layers with artificial synapses which have either an excitatory or inhibitory effect on the incoming signal. The ANNs simplest form is the feedforward system, where signals proceeds from input layer to hidden layer(s) to output layer without any connection of one layer to a previous layer. The ANN is trained using examples which relate inputs to expected outputs. The network error is the sum of the networks output compared to the expected output for a given set of inputs. This error is then processed using either the gradient descent[3] or error backpropagation[4] method, which adjusts the weightings of the

[3]Gradient descent, in its simplest form, involves testing the output of the system when one of the connections (synapses) is adjusted positively and negatively by a given amount. Whichever value delivers a lower overall error is taken as the new value for that connection. Each connection in the system is tested in this way, and once all the connections are tested, a single training epoch is said to be comlete. Training an ANN takes many epochs.

[4]Error backpropagation uses a mathematical algorithm to calculate the error of a given neuron as a function of its connection weights. This, by exptension, allows the weights to be adjusted to minimize the error. The potential advantage of this is relatively less computational time is taken during training than with gradient descent, as the entire network does not have to be recalculated twice for each possible value of each connection.

connections so as to minimize the total network error. A given number of training examples is iterated through the training procedure a number of times (either until and certain minimum error is reached or a number of training cycles is exceeded)

The field of artificial neural networks is extremely diverse, but there are some variations that are important to mention. The Recurrent Neural Network differs from a feedforward network in that the output of the hidden layer is fed back into the inputs of the hidden layer, thus effectively taking into consideration the state of the network prior to the current input. This is broadly similar to the Markov process where prior events affect the outcome of the next event. An Echo State Network is a particular type of Recurrent Neural Network with a large, sparsely connected hidden layer. The weights of the input layer, hidden layer and hidden feedback layer are generated randomly, and only the output weightings are adjusted during training. This results in a low computational overhead for training (i.e. very fast) but does mean that the ultimate success of any version of the network depends on the initial randomized weighting structure. However, because of the rapid training time it is possible to compare several versions for best results.

Todd speculates on a system design which could deal with input and output either on a note by note basis, or as a time window of notes (e.g. a bar). He decides on a process of outputting single notes and terms this a sequential network. The advantage of the time windowed system is the built in memory of previous notes, however Todd's sequential system includes a process which implements a memory of previous notes but in a more dynamic way. In the sequential network, all new notes generated by the network are fed back into memory inputs of the system, and thus have an effect on the generation of succeeding notes. This is a structural variation of the Recurrent Neural Network in that, in this case, the output is returned to the input, rather than the hidden layer feeding back into itself. Todd concedes that these approaches (time window versus single note input) are not contradictory, and that combinations of them (for instance a sequential network producing time windows) could yield musically relevant results. Aside from the fact that some of the inputs of this network are taken from the previous cycle's output, it is effectively a 3 layer feedforward network. Input units are divided into plan units, which indicate the body of source material to be used for training or as a stylistic constraint for output, and context or state units which hold the memory of previous cycles output. Todd debates whether the output units as fixed pitches, or as relative

intervals, are more appropriate and selects fixed pitches for this experimental setup due to the problematic potential for unrestrained modulation in a relative interval setup, which would impact negatively on a human listener in terms of recognizing the material as being drawn from previously known material. He mentions that combining these two systems in a meaningful way could result in a system which could employ modulation as a further possibility in its generation of variations.

The number of hidden units is dependant on the amount of information required to be assimilated. Too few can result in the network being unable to represent all the training information and too many has a negative effect on the ability of the network to generalize the most important aspects of the input data. The number of output units represents the number of possible pitches that the network can generate. Only one unit can be on at a time, or they can all be off to indicate a rest. To represent duration, the network generates a new note at a fixed time subdivision. This is combined with an extra output unit indicating the onset of a new note. Thus, a succession of the same notes with a single note onset on the first represents a single note with time duration that is the sum of the number of repeated subdivisions. A similar note sequence with a note onset event for each subsequent subdivision would then indicate a repeated note of the same pitch. This limits the possibility of using the system to represent polyphonic material, and is one of the reasons the scope of the experiment is limited to a single melody line.

Generally speaking, large number of training cycles was required to train the network, depending on the amount of training data required. This amounted to anything from 2000 up to 50000 epochs, which on 1989 equipment was counted in hours of processing time. My personal experience with ANNs (on an average 2010 computer) has this time being reduced to tens of minutes rather than hours, which still represents a potential hindrance if the training is to be included in a real time performance situation. Once trained, however, the output was effectively instantaneous, which means a trained Recurrent Neural Network it a viable possiblity for real time performance.

Besides successfully generating new melodies similar to given material, there is the added possibility of plan interpolation. By selecting a plan unit input value that lies between input values represented in the original training data, it was possible to successfully generate melodies that were a complex, but recognizable, hybrid of different training sets. The system is designed to represent musical structures no larger than about 4 bars, and so does not address

or succeed at the issue of representing larger forms.

Todd's concern with this system appears to be much the same as Cope's in terms of emulating musical input. Indeed, at the time of writing in 1989, Cope's work would have been much publicized, focusing a great deal of attention on the processes of Markov models and generative grammars as ameans to a musical end. Todd finds that new computational models can be applied to similar compositional goals and achieve results which are successful, yet differ in detail.

Collins (2001) describes a system for the creation and manipulation of drumbeats for dance music. On account of the narrow stylistic range he is attempting to address, UK Garage, his system is quite simple, though he describes several approaches, each with a different variation of results. Version one of his method uses probability templates to describe the possibility of the kick drum, snare drum and hihat playing on any particular 16th note in a one bar cycle, and generates MIDI output to trigger sample playback. A second version uses bar fragments of the three drum parts (in this case 4, 8 or 16 sixteenth notes in length), and then assigns probabilities to the occurrence of the fragment. Because of the increase in the number of possible combinations using this method, a critic unit is necessary to filter out stylistically inappropriate exemplars, in particular the simultaneous occurrence of the kick and snare drum (Another example of the generator/critic structural paradigm). Further variation is included after the basic pattern has been realized, which involve a probability template for the velocity on a particular 16th note, as well as introducing degrees of pitch bend, distortion, filtering and reverb to the playback sample. In small amounts, this has the effect of reducing the monotony of repetitive playback of drum samples. Collins states that in his testing it became apparent that the critical factor in successful use of this algorithm was the input data. A further development suggested by the creation of these statistical models for a drumbeat was the possibility of morphing from one beat to another. Collins reported mixed findings in this endeavor, as certain approaches worked plausibly while changes in rhythmic feel (straight time to shuffle time) produced intermediate results stylistically dissociated from any dance music genre.

Collins second algorithmic system involves a system for automatic breakbeat cutting in a Drum and Bass style. This system works with audio drum loops rather than MIDI sequences. A given drum loop, generally of one or two bar length in 4/4 time, is subdivided into sections of a certain beat length. Then a probability is assigned to the chance that a certain subdivision may be repeated after its regular occurrence in the bar, rather than playing the subsequent

subdivision from that part of the bar. A repeat is not limited to the length of the subdivision, but can be an integer multiple of the subdivision, creating a sense of beat displacement. A limit is set to the number of times a subdivision may repeat. Finally a total length for the output is decided upon. Because the system has the effect of displacing the sense of the downbeat, this total length is the maximum number of bars after which a definite downbeat will be forthcoming. Collins' results for best success in this experiment is as follows: a one bar loop, subdivided into 8th note subdivisions, a 0.2 (20%) chance of a repeat happening, and generally an odd multiple of the subdivision for repeat playback (1 and 3 eighth notes sounding most stylistically appropriate, 5 increasing the potential for a listener to lose place in the bar without a consistent supporting rhythm, but creating a greater sense of originality when this support was present, and 7 being too long and showing too much of the original beat to be interesting).

Keller and Morrison use a probabilistic context free grammar to construct jazz improvisations in their Impro-Visor software. A context free grammar generates strings of symbols using a terminal alphabet, an auxiliary alphabet, a starting string and production rules. Auxiliary alphabet symbols in the starting string are replaced according to the production rules with either terminal alphabet symbols or other auxiliary alphabet symbols until no auxiliary symbols remain. Probabilities are applied as to which production rule may be used. Furthermore, in Keller and Morrison's system, an argument is added to each member of the auxiliary alphabet to control the length of the output.

The Impro-Visor software generates sequences of classes of notes using this method. For any given chord type, a note can be tagged as a chord tone, a colour tone, an approach tone, a scale tone or an arbitrary tone (Keller and Morrison, 2007:333). The generative grammar generates sequences of these types of tones, but includes the possibility of a helpful tone (which could be either a chord tone , colour tone or approach tone) or a rest. To these melodic possibilities are attached possible durations from a whole note to a triplet sixteenth, but excluding dotted eighths and sixteenths as they are were deemed unimportant to the jazz style being represented.

Prior to the generation process, a terminal alphabet, auxiliary alphabet, starting symbol and production rules need to be defined. Members of the terminal alphabet are combinations of the above mentioned classes of notes and durations. E.g. an approach tone of one quarter note duration. The auxiliary alphabet is a set of symbols that get replaced during the generation process either by items from the terminal alphabet or other auxiliary symbols, according to the production rules.

The auxiliary alphabet consists of an abstract symbol plus an argument specifying the length of the output. For example, an auxiliary symbol P of duration 8 would be represented as P(8). The production rule would add terminal symbols, for example X4 (with 4 being the duration, and X representing one of the note classes mentioned above, in this case an approach tone) plus further auxiliary symbol with the required length from the auxiliary symbol being reduced by the duration of the terminal symbol added in that iteration. So:

Auxiliary symbol:	P(8)
Production rule:	P(n) = X4 P(n-4)
Result after one iteration:	X4 P(4)

The net result after one iteration would be a melodic line playing a note defined as an approach note for the given chord in that bar, for a duration of 4 eighth notes, plus the further notes which would be delivered by the P(4) auxiliary symbol in subsequent iterations of the grammar. Once the terminal alphabet string has been generated, they would be applied to successive chords and note assignments, allowing a single terminal string to generate a large number of melodies, albeit with the same rhythmic pattern. Production rules can then be fine tuned, either by adding more, removing inappropriate rules or adjusting the probability that a production rule would be used. This grammatical process can also be applied to the generation of accompaniment tracks.

As a possible feature to be included in a future version of Impro-Visor, Keller and Morrison propose a procedure that allows the user to rate members of the corpus of melodic fragments, and have the program use this information to train an automated critical process tuned to the tastes of the user. They also propose expanding the terminal alphabet to include, for instance, particular scalar sequences. Subjectively, Keller and Morrison evaluate the output of their system as being equivalent to an intermediate level student of jazz in a tertiary education situation, which suggests that the process is making a degree of musical sense. Future plans proposed by the designers are intended to extend the quality of the musical output.

In the preamble to the description of several of his algorithmic composing systems, Alsop (1999) highlights the importance of the use of computer based algorithmic processes in both the realization of his works and as Cope's "friendly antagonist" in the exploration of his (Alsops) relationship as an artist, to his musical material. Whereas many of the aforementioned systems have as their goal the emulation of a style, the systems he describes have each been built to

generate variations of a single composition, with each having customized algorithms, source material and user interfaces. This could be seen to be similar to Sandred et al. (2009) reworking of the composing procedure for the 4th movement of the Illiac Suite to generate variations of the movement that still retain the validity of the original composers intent. These works ('Someone', 'Compose/Canon', 'Guitar21' and 'Selectnotes') use some or all of the following:

- Granular synthesis
- Comb filtering
- Mathematical transformations of midi data
- Confounding the gestural domain of user interface
- Automatic harmonization

For his works 'Someone', and 'Compose/Canon', Alsop describes the use of **granular synthesis** to explore a source audio file. In each case this is a recording of a poem. Granular synthesis is a process whereby an audio file is deconstructed into small, overlapping segments, called grains[5]. These grains can then be pitch shifted (by a fixed or time varying amount), time stretched (or contracted) and repeated. The result is that sections of an audio file can become extended, while retaining a sonic coherence with the original material, and not limiting the results to the lethargic slurring or chipmunk effect that was the only possibility in the analogue era when tape playback was slowed down or sped up.

Comb filtering is a spectral perturbation which occurs when two identical sounds are played with a slight delay between them, generally in the range of 0 to 5 ms. The sonic result is an attenuation of certain frequencies due to phase cancellation. The spectral analysis graph of amplitude versus frequency looks like a comb, hence the name. In the composition 'Someone', the source file output is split: one audio stream is processed by the abovementioned granular synthesis and the other by 10 comb filters. Analysis of the incoming audio file is used to modulate the feedback of these filters, resulting in changing emphasis on different harmonics.

Mathematical transformations of midi data are used in several of Alsops algorithms. Compose/Canon contains two serial processes, 'Compose' and

[5]In this particular case, the grains are 60 milliseconds in length. Practically, a grain can be no shorter than a single sample and no longer than the length of the audio file, although both of these extremes would defeat the aesthetic intent of the use of this technique.

'Canon'. The first takes an incoming midi stream and applies simple mathematical processes such as multiplication, modulus and addition to the midi pitch value to generate an altered melodic line. 'Canon' is an automatic harmonization process which will be dealt with under a later heading. 'Guitar21' uses a midi guitar controller to trigger cyclical procedures (e.g. sine or cosine functions) that either generate notes or adjust other musical parameters within boundaries set in the design of the system. In 'SelectNotes', the live midi input of the user is stored and transformed by a process that randomly selects between a set of possible variation procedures. This includes retrograde, inversion, transposition, playback of a selection of the input, use of input material to create an ostinato, displacement of duration and/or velocity with respect to pitch, and duplicating or contrasting broader qualities in the music, such as rhythmic density. These processes act and interact to create the final output.

'Guitar21', as mentioned before, makes use of a midi guitar to interact with the software. The processes being triggered however have no bearing on the conventional way in which a guitar is used to trigger notes. Alsop describes his intent with this system as forcing the user to abandon preconceptions about the causal relationship between input and output, and is part of his interest in **Confounding the gestural domain of a user interface**.

The 'Canon' section of 'Compose/Canon' is an example of **automatic harmonization** and processes subsequent incoming notes mathematically by selecting the pitch interval and difference in inter-note offsets between the two notes, multiplying these values by some floating point number and storing the result. This process is performed on four subsequent note pairs and the four results are combined to generate either a chord or arpeggio on the fifth incoming note. Alsop is the first example in this essay (and the only, given space constraints) of the compositional use of sound engineering tools and methods. To this he applies the kind of processes that can be applied to midi note information. His systems are not learning systems, but there are examples of reflexivity (Selectnotes) and real time improvisation (Guitar21). Many of the systems are designed to create stimulating novelty and surprise for the user. Indeed, Alsop finds the output of the systems challenging the boundaries of his musical taste domain:

> "I try to be very careful not to allow personal taste to overly dictate the workings of a piece or the selection of pieces for public display. While it is admirable to edit so that only the strongest pieces are aired, I feel it is important for a frank and less manufactured representation

> of the composer to also be available, for myself and for the wider audience." (Alsop, 1999:94)

In 'Groovy Neural Networks', Tidemann and Demiris set out to develop a cost effective system of creating drum tracks that model the playing style of human drummers as opposed to the use 'noise' introduced into the timing of the drums, which has no relationship to the way an actual drummer might expressively deviate from metronomic time and rigid velocity settings (Tidemann and Demiris, 2008). They describe the use of an Echo State Network (ESN) to model timing, velocity and large scale tempo and dynamics changes in the approaches of a variety of drummers to playing certain prescribed drum patterns and accompanying a given melody. In the training procedure, only weightings on the outputs are adjusted. The system is implemented on Matlab, which is effective for proving the concept but would need to be translated to a format compatible with extant Digital Audio Workstations to be of general use. Small scale variations (timing and velocity for Kick- and snare drum and Hi Hat) on a given drum pattern were recorded by various drummers and the ANN was able to learn and reproduce the stylistic variations of the various drummers within acceptable limits. Drummers were also sampled accompanying a melody and the timing and dynamic fluctuations over the course of an entire song structure were also modeled with sufficient accuracy. This indicates the use of a neural network to cohesively model long forms.

Genetic algorithms are iterative procedures used to focus the traversal of a large probability space using variations generated on a modeling of the process of genetic evolution, with fixed fitness criteria to evaluate success. Potential solutions are encoded in a data structure resembling a chromosome and relatively successful solutions are given more opportunity to recombine or 'mate' with other successful solutions to further reduce the deviation from the optimal. Furthermore, chromosomes may be 'mutated' to introduce a stochastic element to the search process. Interactive genetic algorithms replace the automated fitness function with a human operator.

GenJam (Biles, 1994) uses this procedure to generate jazz solos. The system is modeled as an inexperienced jazz student with a mentor. A genetic algorithm generates melodic ideas for jazz solos, which are evaluated by a human operator. GenJam generates two types of musical information. Firstly, the algorithm creates one bar phrases and databases the models that survive evaluation by the human mentor. These phrases can be mapped to different types of scales, so the interval structure can vary during implementation. Secondly, the algorithm

strings together groups of four 1 bar models as a musical phrase, which is also evaluated for fitness by the mentor. Furthermore, Biles details a set of 'musically meaningful' mutation operators such as reversing phrases, rotating notes and using random phrase replacement to limit the size of the new information to be evaluated. These musically meaningful genetic operators are designed to limit the genetic mutation process to processes that have a good chance of creating meaningful results. A problem in a user evaluated system like this is the user fatigue resulting from a large volume of evaluations, resulting in inconsistent test results.

This brings a knowledge based approach (database of 'acceptable' musical fragments) to an essentially non knowledge based approach (genetic algorithm). In suggesting further research for this project, Biles mentions the possibility of including a system of capturing mentor input to be used in a neural network based automated fitness evaluation, which makes the future GenJam a good example of a hybrid system.

Pachet, in his Continuator (Pachet, 2004) uses a Markov chain based system to evaluate and mimic user input. The overarching goal of this system is to promote an engaging dynamic interaction between the user and the computer software. This interaction is based on the concept of 'Flow' as described by Mihaly Csikszentmihaly in his book *Flow: The Psychology of Optimal Experience*, and summarized in Pachet (2008). Flow represents an optimal state between anxiety and boredom where "challenges match skills, and in which people experience 'optimal' states , are able to concentrate, to forget time, and create new goals in a totally autonomous way" (Pachet, 2008:10). Pachet describes the Continuator as a class of Interactive Reflexive Musical Systems (Pachet, 2004:2). He further proposes the following list of characteristics which define an interactive systems' reflexivity (Pachet, 2004):

- Similarity or mirroring effect – the output of the system must be recognizable by the user as based on the users' input.
- Agnosticity – the system learns the musical personality of the user by the interaction with the user and nothing else.
- Scaffolding of complexity – the system needs to adapt to the adaptation of the user to the output of the interactive process.
- Seamlessness – output is sonically indistinguishable from input.

Furthermore, the focus of the interaction is more about the process than the output, so strategies were developed to remove the presence of computer screen,

mouse and keyboard from the interactive process. The only interface is the piano keyboard and the audio output. The Continuator takes three types of input which can be one and the same thing depending on the type of interactive role that the system is performing. These are learning input, which is analyzed to build the model of the user, real time input, which triggers output, and contextual input, which provides a preset domain for the focus of the output in certain situations. During operation, the following processes are carried out by the Continuator:

- Segmentation of input
- Incremental analysis
- Consideration of global parameters
- Generation of output

The analysis module takes a segmented musical phrase and builds a series of probable continuations for various parameters such as pitch, duration and velocity. One advantage of this is that once a certain tree has been decided upon, generating the output is a matter of reading the data, which is computationally light and ideal for uninterrupted live playback. Furthermore, strategies for dealing with imprecision play an important part in creating a more versatile system of generation. An example of this would be analyzing input pitches in terms of pitch areas rather than exact notes. Even if a note appears in the output that was not in the input and therefore has no representation in the probability tree, the system can attach this note to a nearby note (in the same pitch area) and find a way forward from this information. Pachet describes a variety of interactions that have been implemented with this basic system (Pachet, 2004:7):

- Turn-taking – The user plays and the computer plays when the user stops, and then stops when the user plays again.
- Turn-taking with delay – the computer does not stop when the user starts playing again.
- Single note accompaniment – the computer harmonizes an inputted note from previously captured (contextual input) chordal material.
- Phrase based accompaniment – in a similar manner to the abovementioned single note accompaniment, the computer harmonizes an entire phrase rather than each note.

- Collaborative – both player and computer play continuously at the same time.

Pachet has conducted numerous experiments with experienced jazz musicians and children, and the effectiveness of the Continuator as a stimulating musical partner has been established without doubt. In a "musical Turing test", the interaction of the Continuator with a jazz pianist (Albert Van Veenendaal) was evaluated in a blind test by two jazz critics during a live broadcast on the Dutch radio station VPRO in June 2004. The critics could not, on hearing alone, distinguish the playing of the human and the computer (Pachet, 2008:8). In spite of this obvious success, Pachet is quick to point out the flaws of the system. The Continuator is able to produce exceptional results on the level of a musical phrase, but performs rather poorly in terms of a broader musical structure. "Stimulating as they can be, sessions with the Continuator do not produce reusable content i.e. fully-fledged pieces of music" (Pachet, 2008). Pachet proposes a future line of research which would include a grammatical analysis process to generate more meaningful long forms in the output of the Continuator.

These few examples can only hope to mark out the territory of algorithmic composition. Fernandez and Vico (2013) does a study that covers more ground but in significantly less depth and runs on to seventy pages, showing many researchers at work on the diverse problems and possibilities of the field.

Working with the examples cited in this section, we see examples of most of the major computational techniques: genetic algorithms (Biles), neural networks (Todd and Tideman and Demiris), formal grammar (Cope and Keller and Morrison), Markov chains (Cope and Pachet), stochastic processes (Hiller and Alsop), statistical models (Hiller, Cope and Collins) and knowledge based systems (Biles). Missing from this list are systems that employ techniques of cellular automata and self similarity, and this is a bias in my research towards methods that can in some way directly model music composition techniques based on existing formalisms and cognitive understanding of music. Of the systems researched, all of them deal with note level information except for Tideman and Demiris's system which is concerned with expressive timing of drumkit performance in popular music, and the systems of Alsop which employ granular synthesis and comb filtering where the algorithm outputs the final audio signal. In terms of compositional goals, the systems of Hiller, Cope and Alsop are designed to deliver complete works, while the others have a more limited experimental scope. Alsop is maybe the exception here in that his algorithms are each designed for the performance of a single work, and so are quite limited in

scope. The expressive timing research of Tideman and Demiris is aiming towards a long form output but the current extent of the project is a proof of concept that a neural network can successfully model the data rather than an actual system that can generate finished works. Of these systems, those of Todd, Collins, Keller and Morrison, Alsop, Tideman and Demiris and Pachet all could be considered to be experiments limited in scope.

Interactivity

Smuts (2009) analyzes five texts which give, in his opinion, poor or incomplete definitions of the nature of interactivity. He takes the following general guideline, in which he acknowledges the presence of "controversial borderline cases" (Smuts, 2009:endnote 3). He states that interactivity 1) "should be in accord with our best intuitions on how the term should be used", and 2) "it should usefully differentiate interactivity from related but in compatible concepts with which it is often confused" (Smuts, 2009:53). To this end he focusses on the terms 'interactivity', 'responsiveness' and 'control' and comes up with various ways in which their meaning should not be confused. Taking the example of human conversation as a starting point, he differentiates an interactive conversation, where subsequent statements and responses have a thematic link, and a responsive conversation where one statement may trigger a response with no meaningful connection to the statement. To elucidate the idea of control, he takes the example of a car, which could be considered interactive but fails due to the fact that the interaction with the car is wholly predictable and therefore does not constitute interaction. The car does not change the way it responds based on the driver. This would be an example of control rather than interactivity, and brings the idea of unpredictability into the definition. (the car behaves predictably therefore it is not interactive. The relation between the car/driver unit and the traffic would be a case for interactivity due to its unpredictable but thematically linked behaviour). His definition follows:

> Something is interactive if and only if it (1) is responsive, (2) does not completely control, (3) is not completely controlled, and (4) does not respond in a completely random fashion. (Smuts, 2009:65)

He then describes how the areas of interactivity and control may be seen on a continuum, taking the example of tennis: evenly matched players would need to constantly interact with each other to play the game, whereas unequally matched play would be dominated, 'controlled' by the skilled player and the game would

lose its interactive nature. He identifies the quality of 'maximally interactive' where control is impossible and 'minimally interactive' where mastery is easy.

For the purposes of this thesis, Smuts makes two references to music which I enjoy even though they are not necessarily central to his argument, and which could have a bearing on the considerations for the research subject: a musical instrument might be considered interactive to an average user due to the fact that they are not entirely in control of it, whereas a virtuoso would be (more) in control. This highlights the relational rather than intrinsic nature of interactivity. Secondly, he mentions the example of jazz, the philosophy of which has underpinned my research into live use of improvised algorithmic music. Jazz, in an ideal case, is interactive amongst the musicians, but not necessarily with the audience.

In his conclusion he finalizes his definition of interactivity in terms of art with the term "concreativity", taken from Collingwood (1938:323). This term describes a situation where the audience can influence the creation of an artwork. Traditionally this would have been restricted to live performances of artworks, but he argues that technology has made it possible that mass produced artwork have the possibility of concreativity.

Sweller (2010) talks about interactivity in the context of cognitive load theory. Cognitive load theory is an approach to understanding how learning takes place in terms of the pressure it places on the human working memory (which can hold about seven things in memory at any one time). There are three types of cognitive load identified:

- Intrinsic cognitive load – this is the effort required to understand the information being presented.
- Extraneous cognitive load – this is the effort involved in processing the way information is presented.
- Germane cognitive load – this is the effort required to process the information into a schema, which is a permanent store of knowledge.

Sweller identifies the extent to which these loads are dependant on the information, the instructional design or the cognitive abilities of the learner. He identifies an element, which is an item of information which can be understood as one thing by a learner. This depends on the learners cognitive ability and prior experience of the subject matter. He identifies the interactivity between elements as an added cognitive load. Intrinsic cognitive load is increased in cases where there is a high interactivity between elements. If the interactivity of the elements acts on the level of presentation of the material (e.g. explaining a concept in a

colloquial jargon), it is part of the extraneous cognitive load and does not interfere with the nature of the concept, and can be changed to optimize the learning experience without compromising the learning of the concept. If this jargon is part of the understanding of the concept, it is an intrinsic cognitive load. Germane cognitive load is entirely dependant on the characteristics of the learner, and the effort spent on this is proportional to that spent in the intrinsic cognitive load. If the working memory is largely involved with extraneous loads associated with the procedure of instruction, less effort can be spent on understanding the material (intrinsic cognitive effort) and consequently there is less effort required to shape new understandings into schemas for future use.

Various effects arise from learning situations but the one that may have particular importance for an interactive software is the Expertise Reversal Effect. This describes an effect where a certain method of instruction which works for unskilled learners becomes less effective as skills increase. This consideration of scaling of expertise could have an effect on the interactivity of the proposed software with continued use and even if there are no explicit plans for implementing this factor, it does present some possible terms to describe observations.

The Continuator (Pachet, 2002), as described in detail in the section on algorithmic composition on page 22, is an interactive software tool that is able to meaningfully adapt to use by highly skilled jazz musicians and five year old children. Pachet (2004), in describing the Continuator as an example of a Reflective Interactive System, talks of the scaffolding of complexity in relation to the user, and the use of the theory of flow where challenges and skills are balanced. The Continuator is a system which is easily learned, but remains engaging to the user due to the reflexive nature of of the interaction, while not sinking to the level of mere repetition of user input. This makes it an excellent example of a interactivity according to Smuts, and circumvents the issue of Swellers Expertise Reversal Effect by directing the learning towards an increased understanding by the user of their own input to the system.

Superhero software development

In this section I will describe the software tools employed for this project, briefly describe some software development methodologies and clarify some jargon and activities engaged in during a software development process.

Java is a free and open source programming language developed and maintained by Sun Microsystems (*Java Website*, n.d.). Java has been applied to

many different platforms from everyday uses such as personal computers, mobile devices to Mars rovers. Java has several advantages as a programming language. Java is easy to learn, and though it is considered a verbose programming language compared to similar languages like Python, it forces the user to develop good programming practices that make programs easier to read and understand by humans, and therefore easier to debug and extend. Java is Object Oriented, which makes it easy to write modular programs with easily reusable code. Java is also platform independent. One piece of Java code can be run on many different platforms. This is achieved by an abstraction layer called the Java Virtual Machine, which handles all the machine specific interpretation of generic Java code.

Ableton Live is a mature and robust audio production and live performance software (*Ableton Website*, n.d.) currently on version 9.7.5, and is this researchers software of choice for live performance and studio work. Many digital audio workstations (DAW) are modeled around the concept of a virtual timeline and mixer. This mirrors the layout of an analogue era recording studio, with multitrack tape recorder, mixing desk and outboard effects units. The digital domain has allowed this concept to be extended to include MIDI and automation, virtual instruments and audio processing plugins. As with most other DAWs, Live is able to record audio, sequence MIDI and automation data to a timeline. Its uniqueness lies in an alternative yet integrated system for playing music using loops (audio and MIDI), without the need to explicitly connect these loops to the aforementioned timeline, allowing music to be arranged and processed on the fly. These features have gained to software a huge user base, who have adapted the software to deejaying, live electronic music performance, audio production, film and theatre sound design and art installation works.

Max/MSP/Jitter is a visual programming environment for the manipulation of MIDI, audio and video. It was originally designed by Miller S. Puckette at IRCAM in the 1980s, and its current development as a commercial product is managed by Cycling '74. Max/MSP/Jitter is also an extensible software environment which allows for the development of new functionalities by developers not associated with Cycling '74. In fact, MSP and Jitter are extension libraries to the original Max program to cater for audio, and video and matrix processing respectively. The visual programming environment allows users to program complex custom processes for MIDI, audio or video, without having to resort to writing computer code. Developers interested in extending Max can do this in a variety of languages, including C, C++, C#, Java, Python, Ruby and

LISP, to name but a few.

Ableton negotiated a collaboration with Cycling '74 to develop a version of Max MSP to run natively within the Live program. This version of Max is called Max For Live. This allows easy integration between the robust live performance functionality of Ableton Live and the potential for customization afforded by Max for Live. Max for Live, aside from all the manipulation possibilities mentioned above also exposes a detailed application programming interface (API) to control the Live software, allowing a developer to build Max patchers[6] to control Ableton Live in novel ways, especially in relation to various control surfaces. The combination of Ableton Live and Max for Live allow for the integration of this robust performance machine with custom Java code. This will be the platform of choice for testing audio playback of the algorithm proposed in this thesis.

Software development methodologies

Two blogs on the subject of software development, *10 Top Programming Methodologies* (n.d.) and *Software Development Methodologies* (n.d.), offer a variety of software development methodologies:

- Waterfall Model
- Prototype Methodology
- Agile Software Development Methodology
- Rapid Application Development
- Dynamic System Development Model Methodology
- Spiral Model
- Extreme Programing Methodology
- Feature Driven Development
- Joint Application Development Methodology
- Lean Development Methodology
- Rational Unified Process Methodology

[6] A 'patcher' is a term specific to Max/MSP and refers to an application designed in the visual coding environment of Max/MSP

- Scrum Development Methodology

A cursory reading of these techniques runs the gamut from dry methodological descriptions to trendy business jargon, and as most of these are not applicable to a solo developer, rather than explain these methods in detail, I will attempt to extract the issues that all methodologies attempt to balance:

- Requirements: this is the definition of the final software. If this is clear, a simple linear (waterfall method) is the simplest approach to take. This is seldom the case. Systems that cater for flexible end requirements run the risk of losing focus if leadership is not clear.

- Client involvement: In the case of unclear requirements, client involvement is essential. Managing this feedback so that there is enough to reach the end goal but not so much as to derail the time and money budgets is a challenge faced by the management team.

- End user involvement: Allowing the design team easy access to the end user for testing is helpful for the quality of the final product but increases costs. Clear overall goals and leadership are essential to keep methods which employ this on track.

- Skill of developers: some models require highly skilled development teams and do not allow any training on the job. This results in the success or failure of a project to be entirely dependant of the staff . This can be problematic should development staff need to be replaced.

- Documentation: documenting a software project can be time consuming especially in iterative processes with many short term deadlines for working models. This can be problematic for later extensibility of the project and for acclimating new staff to the team.

- Management cost: some processes require a large amount of time for planning and scheduling, especially when there is client and user feedback to be elicited and incorporated.

Some programming practices

- Test Driven Development - This is a software development process where a set of automated tests are written to define the functionality of a software. When the software passes the tests, it is complete. This technique assures

that the developers do not waste time on unnecessary features, and complete the code in as simple a way as possible to pass the tests.

- Top down programming - This is a style of code design that defines the large problem and divides it into smaller problems. This is actually implemented in code using stubbs[7] so it is possible to see how the entire architecture interacts without coding all the details. While in reality, any projects end up being a combination of both top down and bottom up coding (where datatypes are designed first and then pieced together later), one of the many blogs I perused on the subject of solo software development had some interesting advice: because many languages have a large body of extra code libraries available for more specific problems than what the basic language might address, use top-down program design and "Do as little low-level work as possible" (*Best Development Methodology for One Person?*, n.d.). Find as much pre-written code as possible and copy and paste. My experience of large solo development projects favours a top down approach because often details of the code may take weeks or months to complete and often one needs a reminder of the overall plan when one returns to the big picture.

- Version Control - This is a process that allows developers to keep track of the development of software and 'roll back' to a previous working version should a development effort result in unusable code.

- Continuous Integration - Rouse (2008) describes continuous integration as a procedure in which new changes to a software in development are integrated into the project, built and tested regularly (daily or more often) so that errors resulting from new code can be detected early on. This system requires some form of centralized integration system and a computer dedicated to continuous integration tasks where a change can be submitted and the building and testing can happen automatically. Two blogs on the subject of continuous integration for solo developers (*Blog on the subject of Continuous Integration for solo developers* (n.d.*a*), *Blog on the subject of Continuous Integration for solo developers* (n.d.*b*)) suggest that continuous integration for a solo developers is less advantageous than for teams. This is due to the time overhead for setting up the system in the first place and keeping it maintained. One situation where it was advantageous was to check for dependencies that may be specific to the

[7] Dummy code

developer's computer, where a new revision of the software might compile and pass tests on that machine but then fail when compiled by another machine (the continuous integration server). Another situation described was one where a single revision by a solo developer needed to be compiled each time for seven different platforms. There is a distinct advantage to be able to automate this and move on with other tasks.

Some case studies

Zieliski and Szmuc (2005) discuss the results of a large experiment undertaken to identify code quality differences between solo or pair programming, and classic test-last methodology versus test driven development, after finding that existing literature was often contradictory on the subject. They found that solo and pair programming made no difference to code quality, but that test driven development resulted in a significantly lower code quality according to the assessment metrics being used. For the purposes of this study, pair programming is not an option as i do not foresee having a coding partner for the project. This study does suggest that learning the process of test driven development may be time best spent elsewhere.

Groen et al. (2015) discuss four software development projects that were developed in an academic rather than commercial space. They track the contributions of developers involved, the use of software engineering techniques and the academic publications that resulted from the development over the course of 2006 to 2014. The techniques they observed and analyzed were

- Test Driven Development
- Pair Programming
- Unit Testing
- Functional Testing
- Continuous Integration
- Profiling Testing
- SS-Scrum Sessions
- Code Review

They conclude that the application of some of the above mentioned software engineering techniques and principles improves the quality of the software as well as contributing to a larger publication output.

3 Methodology

As stated before, this study will test the use of new combinations of repetition schemas and musical features to generate new music that may result in material that has a better-than-random chance of containing some musical cohesion.

For this chapter, the issue of software development as it applied to this particular situation will be discussed, after which the workings of the algorithm will be demonstrated in two steps. Firstly there will be a simple explanation highlighting the pattern matching and recombination technique employed in the algorithm, and an explanation of the software architecture. Secondly, a simple four bar melody and accompanying chord progression will be used as input, and the behaviour of the algorithm will be demonstrated with some actual output. This will serve to show that the algorithm is behaving as expected.

Computationally, the algorithm described in this chapter can be described as a knowledge based system, as it contains a particular type of music composing related methodologies within the code. It also contains elements of systems that learn, in that it learns from input material and this has (within limits) an impact on the output. It may also be seen to contain elements of formal grammars, with a set of symbols describing the repetition of a musical feature which can then be replaced according to some production rules to generate new material. Even though it deals with the identification of repeated elements in music, the temptation to call it a system using self similarity as mathematical self similarity deals with objects that are similar on several different scales must be resisted, as this system does not explicitly encode that approach to the information. It does not explicitly use stochastic or statistical processes, artificial neural networks or genetic algorithms. The compositional goal of the algorithm is to create an interactive software to explore the parameter space of the encoded musical information, so it would definitely fall under the computer aided algorithmic composition classification of Fernandez and Vico (2013). It outputs MIDI note information of musical phrases and does not deal with expressive timing, nor does it synthesize the output waveform. The choices made by the user directly influence the output, so there is no complex mapping of some other situation to generate musical parameters, as with procedural audio in a video game.

The development of this software has been constrained by the skills of the developer (myself) and by the size of the development team (again, myself only). I have an aptitude for computer programming but have no formal training in the subject. As this project is still in a proof-of-concept stage, there has been no overwelming reason to expand the size of the development team, thus as a solo

developer, there is little scope for any of the team management concepts uncovered during the reading on the subject. Some of the other concepts have proven to be helpful. A top down approach to the system design is very useful as this simplifies the understanding of the communication between various components. In particular this allowed components such as the chord progression analysis module or the recombination algorithm to be developed and tested separately. This resulted in a more modular software design where these components now exist as reusable code for future projects. Another example would be that all the software components can operate independently of the graphic user interface, which has allowed me to easily envision the practical implementation of future research. Continuous integration was not a technique that in any way changed the approach to handling the work. Once the general software architecture was developed, it was a matter of getting each module to behave as expected, and finally integrating the analysis and recombination algorithm with GUI, playback and score output repository. During the process of development, the Mercurial version control system (*Mercurial*, n.d.) was adopted, linked to the Bitbucket online software repository (*Bitbucket*, n.d.). There has not, as yet, been any need to roll back to previous versions of the software, so this is mostly just a security function in case my laptop crashed or got stolen. During the development of the chord progression analysis module, applying a test driven development methodology was extremely helpful. It became apparent early in the development of this module that it was unlikely to cover every single eventuality that existed, so an incremental approach to developing the logic was taken. A chord progression dealing with a certain harmonic issue was selected and the code was tweaked till the results were as expected, after which the input and output was formalized as a software test. As more challenging harmonic situations where added, it was quite easy to run the test suite of previously implemented chord analyses to see whether the new code had broken something that worked before. Interestingly this was the only component of the project that derived any benefit from this approach. Aside from these software techniques, a general month by month deadline framework for software and thesis chapter completion was laid out. There was no need to change these deadlines significantly, nor was it neccesary to compromise the expectations as to the shape that the final software application took.

A description of the functioning of the algorithm

Firstly the corpus material is prepared and captured into a format that is compatible with the analyzer algorithm. Preparation consists of quantizing notes in position and duration to either the nearest eighth or sixteenth note (depending on the material) prior to ingestion into the algorithm. For the purposes of this research there will be no consideration of expressive timing, although it could be incorporated in future. The melody and its optional accompanying chord progression data is loaded into the algorithm. When refering to the melody the implication is that there are no overlapping notes, which is a constraint that has been decided upon for this research and by no means a limitation of the system as a whole. Each melody comes with time signature information, a start and end point and a list of notes. When refering to each note the implication is that it has a position, pitch, duration and velocity parameter. The notes are ordered according to their position on the timeline. The first pass over the data takes each note and expands the scope of the parameters in the following way: the position parameter, which is a floating point number that indicates the notes placement in time in relation to a timeline where the duration of a quarter note equals 1, is expanded to include positions which are offset values from the previous or subsequent barline. This is for situations where the time signature of the output material is different from that of the melody, so that material can retain its relationship to the barline.

Duration is expressed in a similar way to position. Besides this absolute duration value, which is usable as a musical feature, duration can also be expressed as either staccato, portato or tenuto. While there is considerable debate over the interpretation of these articulations in performance literature, for my purposes staccato indicates a default shortest value, either an eighth or sixteenth note depending on the material. Tenuto indicates a note held until the onset of the next note, and portato indicates a duration which is a percentage of the distance till the next note, and possibly quantized to a default quantization value. A further value is generated from the position parameter and this is the gap between the onset of the note in question and the subsequent note[8].

At this point in the process no further development of the velocity is possible. At this stage of the development of the algorithm, all velocity features are calculated in the next pass over the data once notes have been divided into cells.

[8]For those who have spotted the problem for the last note in a melody (i.e. no subsequent note), this is provided by the first note of the melody, which is positioned as it would appear if the melody were to loop and replay based on its start and end points.

The pitch parameter of a note is an integer from 0 to 127 and is consistent with the note numbering range for notes in the MIDI specification (*Midi 1.0 Specification*, 1996:10). The value 0 indicates a C note and each subsequent number represents the note a semitone up. (1 = C#, 2 = D etc). As a result, just over ten octaves can be represented. In the absence of an accompanying chord progression, the note parameter can be developed into the modulo note, where the octave positioning of the note is removed and a number from 0 and 11 remains. The pitch value can be turned to an interval value by subtracting the pitch value of the previous note[9]. Furthermore there are values derived from the melodic contour. The simple melodic contour, proposed by Friedmann (1985) generates a value of -1, 0 or 1 which is the signum of the aforementioned interval calculation. A further melodic interval is generated in a similar way with a range of -2 to 2, where 0 indicated a repeated pitch, 1 or -1 a relatively small melodic interval and 2 or -2 a relatively large one.

This relatively simple contour parameter comes into its own when there is an attached chord progression for the melody. The chord progression is analyzed to derive the root and quality of the chords, as well as the root and quality of the key that any part of the chord progression may represent. This gives a further variety of parameters that can be derived from the pitch and position of the note, namely the interval from root of prevailing key and the interval from root of prevailing chord. These features can be either chromatic, where the measure is a simple calculation of semitone distance, or diatonic, where the distance includes whether it is diatonic to the prevailing key/chord or not. The implication of this second analysis is that for instance, if the melody had a major 3rd over a major chord, this would be analyzed as a diatonic third. Were the generation algorithm to use this to generate a note over a minor chord, it would generate a minor third, which would be diatonic for that chord. A final pitch analysis derived from Keller and Morrison (2007) involves classifying a note as either a chord tone, a scale tone or a non scale tone. These chord and key related analyses require contour information to give melodic direction.

For the limits of this research, it is assumed that the diatonic analysis of intervals over a prevailing chord would be best suited to generation of a new melody to a new given chord progression, but future research might include a melody derived from the key which would use an as yet undeveloped algorithm to fit a chord progression to this. There is also the possibility that interesting

[9]similar to the last note issue mentioned in Footnote 8, the previous note for the first note of a melody would be the last note positioned as if it were played on a previous repetition of the melody

material might be discovered in using material where a harmonious result is not guaranteed. It is expected that many areas will be discovered where the chosen models fall dramatically short of representing cohesive music.

Repetition and Pattern Matching

The second pass over the data involves grouping the notes into cells of consecutive notes of a certain length, and identifying them according to various musical parameters. For instance, given a series of notes A, B, C, D, and a cell length of 2, three cells can be derived using as a start point each note that will allow a cell of length 2. The following three cells are extracted: A,B, B,C and C,D. It is clear from this simple example that each cell is unique. Given that each note has a position, note value, duration and velocity, the example in Table 2 starts showing a more interesting picture in terms of repetition. Because the position parameter is an absolute position on the timeline there is no possibility of any repeated values as this is a single voice melody with no overlapping notes, as mentioned before. There are no repeated groups in the note parameter of each pattern, but this is for the simple reason that there are no repeated notes in the original dataset. Looking at the duration and velocity, it is plain to see that the durations in cell 2 are repeated in cell 3 and likewise the velocity of cell 1 is the same as cell 3.

Table 2: A simple data pattern

	dataset				**cell 1**		**cell 2**		**cell 3**	
	A	**B**	**C**	**D**	**A**	**B**	**B**	**C**	**C**	**D**
position	0.0	1.0	1.5	3.0	0.0	1.0	1.0	1.5	1.5	3.0
note	48	50	53	55	48	50	50	53	53	55
duration	1.0	0.5	0.5	0.5	1.0	0.5	0.5	0.5	0.5	0.5
velocity	100	50	100	50	100	50	50	100	100	50

Table 3 shows how the position and note parameters can be expanded to demonstrate repetitive behaviour. The inter onset distance is calculated for each note, being the timeline distance between the note and the subsequent note. It can be seen that cell 1 and cell 3 repeat the inter onset distance. The note parameter is expanded to an interval value and a simple melodic contour value. The interval value is the difference between the note value and the note value of the subsequent item. In this case it can be seen that cell 1 and 2 both have a

Table 3: Expanded position and note data

	dataset				**cell 1**		**cell 2**		**cell 3**	
position	0.0	1.0	1.5	2.5	0.0	1.0	1.0	1.5	1.5	2.5
inter onset distance	1.0	0.5	1.0		1.0		0.5		1.0	
note	48	50	52	55	48	50	50	52	52	55
interval	2	2	3		2		2		3	
simple contour	1	1	1		1		1		1	
duration	1.0	0.5	0.5	0.5	1.0	0.5	0.5	0.5	0.5	0.5
velocity	100	50	100	50	100	50	50	100	100	50

melody that rises by 2 semitones. The simple contour is calculated by ascribing a -1, 0 or 1 to the note depending on whether the melody to the next note descends, stays the same or ascends. In this case, each cell has an ascending melody.

From these (and further analyses as mentioned earlier) a list of features is derived, where a feature is defined by its timeline position, the type of parameter (inter onset interval, note, interval etc), the value of that parameter and the attached cell.

Table 4: Feature list example

position	**feature**	**value**	**associated cell**
0.0	inter onset interval	1.0	cell 1
1.0	inter onset interval	0.5	cell 2
1.5	inter onset interval	1.0	cell 3
0.0	duration	1.0, 0.5	cell 1
1.0	duration	0.5, 0.5	cell 2
1.5	duration	0.5, 0.5	cell 3

.etc...

Table 4 shows an incomplete list of the features, to demonstrate the identification of repetition schemas. Using a pattern matching process, a repetition schema groups together items whose feature and values are the same, and from this can be derived a list of positions on the timeline where some musical feature is shown to repeat. Table 5 shows this process for the data in Table 4. It can be seen

that the inter onset value of 1.0 occurred twice, whereas the value of 0.5 appeared only once. If any musical feature were considered for repetition, the information in this list could be used as a template for positioning that repetition on the timeline. Indeed for the purposes of this study, it will be discovered just how freely musical features can be mapped to unrelated repetition schemas, and still reflect the cohesion of the original material.

Table 5: List of repeated features

feature	**value**	**position list**
inter onset interval	1.0	0.0, 1.5
inter onset interval	0.5	0.5
duration	0.5, 0.5	1.0, 1.5
duration	1.0, 0.5	0.0

To generate a simple musical output, a master repetition schema is chosen for its list of positions. Thereafter, an inter onset interval, duration, velocity and pitch feature is chosen. The inter onset interval is used to derive the position of the notes offset from the positions of the master repetition schema. Thereafter duration, velocity and pitch data are added. If the duration value is longer than the inter onset interval it is made equal to this value, as the output is limited to a monophonic melody. Table 6 shows a list of selected parameters from our simple dataset. Once this has been decided upon, the final note list can be generated as Table 7. This process demonstrates that musical features can be identified, their position can be noted, and through pattern matching, the structure of the way they repeat can be extracted. The list of features, combined with a master repetition schema, can be used to generate new musical output which will bear some resemblance to the original in terms of structure and content. The demonstration in the text is perforce very simple (and results in a very simple output) and as the explanation proceeds, various strategies that are used to add complexity to the output will be described.

Table 6: Example generation parameters

master repetition schema		
feature	**value**	**position list**
inter onset interval (but could be any)	N/A	0.0, 1.5
note parameters		
inter onset interval	0.5	
duration	0.5, 0.5	
velocity	100, 50	
pitch	48, 50	

Table 7: Example final note list

new note list				
master schema position	**0.0**		**1.5**	
note position based on note parameter inter onset interval of 0.5	0.0	0.5	1.5	2.0
duration (0.5, 0.5)	0.5	0.5	0.5	0.5
velocity (100, 50)	100	50	100	50
pitch (48, 50)	48	50	48	50

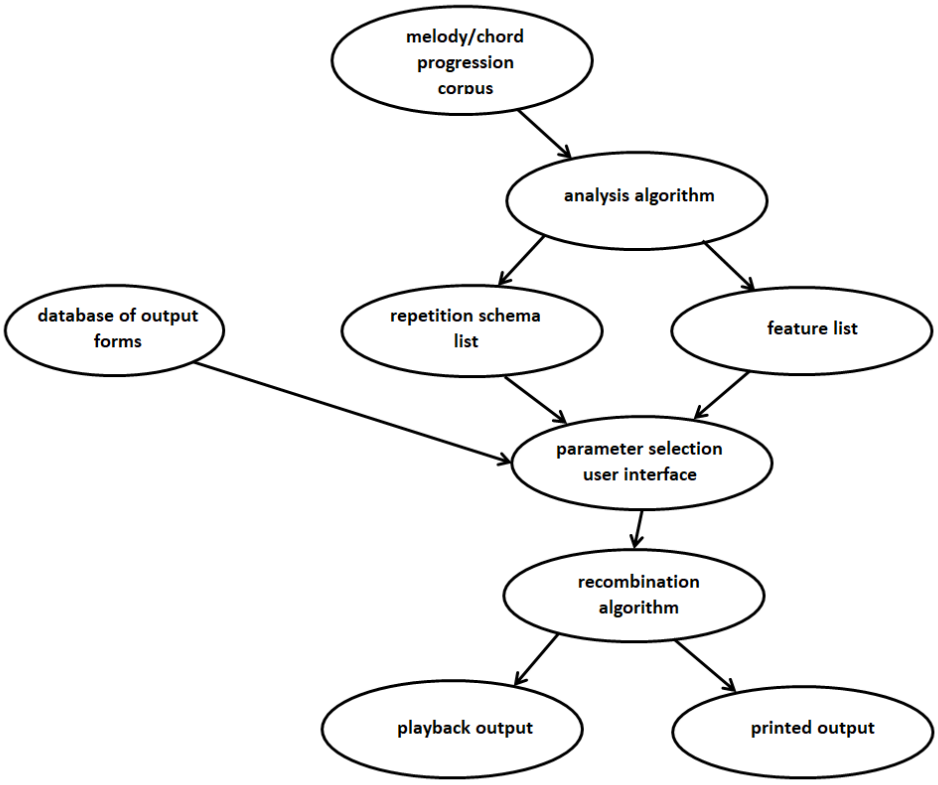


Figure 1: Software architecture

Workflow and Software Architecture

Figure 1 represents a flow diagram of the workings of the software. The melody and chord progression corpus is prepared in a separate process. The melody file represents the notes of the melody, the start and end points of the form and the time signature of the piece. There is a separate but related file which represents the chord progression of the piece represented as the notes of the chords, start and end points and time signature. The analysis algorithm generates the list of features and the list of repetition schemas. The database of output forms is represented the same way as the chord progression files, with notes, start and end points and time signature (these chord progression files can be used as output forms). It is essential for there to be start and end point and time signature information in this file. It is not essential for there to be actual chord information in this file, but that does mean that features that relate the melody to an underlying chord progression cannot be used in the recombination algorithm. The chord progression and output form files are subject to a chord analysis process that identifies the root note, chord quality and extensions of the chord,

and which also makes an assessment of the prevailing key and the chord function within the key. The parameter selection user interface (UI) allows the user to select a combination of features and output form. This UI tests whether the parameters selected represent a complete parameter set that will allow the recombination algorithm to generate output. The output for playback is a custom format which is passed to Ableton Live for immediate playback. The printed score output is in MusicXML format which can be opened in score notation software. Every time a new complete parameter set is selected by the user, the UI automatically passes this to the algorithm and output is generated, meaning that parameter changes can be heard immediately.

Test output demonstrations

A variety of tests have been designed to demonstrate the behaviour of the algorithm. Random choices are not part of the basic workings of the algorithm, so a direct correlation between the input material and parameters and the output should be seen. This is an essential part of the development process to iron out software bugs. For the first test, simple input data with readily identifiable attributes is used, and a simple feature set consisting of the following:

- Inter onset interval
- Absolute duration value
- Loud-or-soft velocity measurement
- Absolute pitch

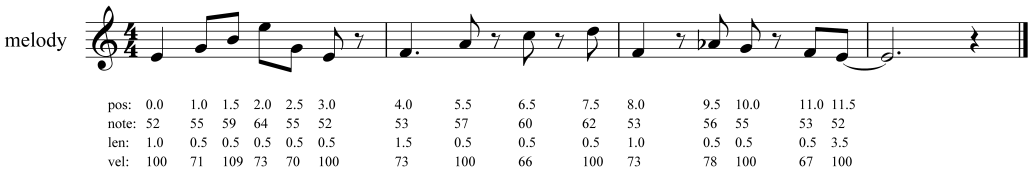
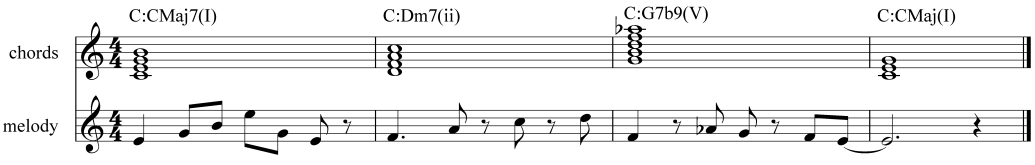


Figure 2: Low level symbolic data

To begin with, the ability of the algorithm to correctly identify low level symbolic data is tested. Figure 2 represents the melodic parameters (position, note pitch, length and velocity). This is the simplest information required for playback of note information and is the extent of information embedded in a Standard MIDI file or Ableton Lives midi clip for the playback of notes. The melodic material of

the analysis corpus, chord progression and output forms are captured in this form. Aside from their relationship to the timeline, the pitch range and the velocity range, these parameters do not represent any awareness of how the notes might function in a musical context.



rhy_gap:	1.0	0.5	0.5	0.5	0.5	1.0	1.5	1.0	1.0	0.5	1.5	0.5	1.0	0.5	4.5
pit_abs:	52	55	59	64	55	52	53	57	60	62	53	56	55	53	52
pit_mod:	4	7	11	4	7	4	5	9	0	2	5	8	7	5	4
pit_int:	0	3	4	5	-9	-3	1	4	3	2	-9	3	-1	-2	-1
pit_csd:	4	7	11	4	7	4	5	9	0	2	5	8	7	5	4
pit_cct:	4	7	11	4	7	4	3	7	10	0	10	1	0	10	4
pit_dsd:	3,0	5,0	7,0	3,0	5,0	3,0	4,0	6,0	1,0	2,0	4,0	5,1	5,0	4,0	3,0
pit_dct:	3,0	5,0	7,0	3,0	5,0	3,0	3,0	5,0	7,0	1,0	7,0	2,0	1,0	7,0	3,0
pit_csn:	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
dur_abs:	1.0	0.5	0.5	0.5	0.5	0.5	1.5	0.5	0.5	0.5	1.0	0.5	0.5	0.5	3.5
dur_spn:	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.67	1.0	0.0	1.0	0.78
dyn_lsd:	96	96	96	96	96	96	96	96	96	96	96	96	96	96	96
dyn_sim:	2	0	2	1	0	2	1	2	0	2	1	1	2	0	2
cnt_sim:	0	1	1	1	-1	-1	1	1	1	1	-1	1	-1	-1	-1
cnt_hsc:	0	1	1	2	-2	-1	1	1	1	1	-2	1	-1	-1	-1

Figure 3: Expanded data

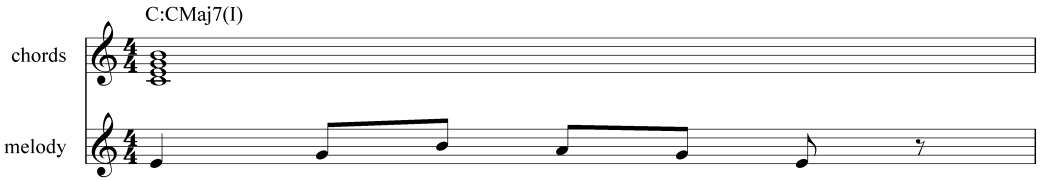
Figure 3 gives a demonstration of how the data has been expanded in the analysis phase of the process. The new information is identified by an abbreviated feature identifier which is described in the key in Table 8 and will be further expanded in the text. Some of the data is a simple duplicate of information in Figure 3, such as the absolute pitch (pit_abs) and absolute duration (dur_abs). Easy further calculations generate the modulo value of each note (pit_mod) which is effectively the name of the note expressed in the range 0 to 11 where a C would be 0 and subsequent numbers would indicate a note a semitone up. Another way of thinking of this would be the note without its octave information. The simple dynamic value (dyn_sim) takes the velocity information of each note and compares it to the mean and deviation of the velocities of every note in the melody and makes an assessment as to whether the note is of soft, medium or loud dynamic. Once these simple calculations are complete, the next step involves linking each note to the next and previous note in the sequence.

Table 8: Key to music feature abbreviations

rhy_gap:	Inter onset distance between this note and the next
pit_abs:	Absolute midi pitch of the top note
pit_mod:	Modulo value of the top note (pitch without octave)
pit_int:	Absolute interval in semitones between this note and the previous note
pit_csd:	Top note expressed as semitone distance (range 0-11) from root of prevailing key centre
pit_cct:	Top note chromatic chord tone expressed as semitone distance (range 0-11) from root of prevailing chord
pit_dsd:	Top note expressed as diatonic interval (range 1-7) from root of prevailing key centre, with a diatonic/chromatic signifier(0/1)
pit_dct:	Top note expressed as diatonic interval (1-7) and a diatonic/chromatic indicator (0/1)
pit_csn:	Top note described as a 0)chord tone, 1)scale tone or 2)non scale tone
dur_abs:	Absolute duration on the timeline where a quarter note = 1.0
dur_spn:	Duration: staccato, portato or tenuto
dyn_lsd:	Loud or soft dynamic. loud = 1, soft = 0
dyn_sim:	Simple 3 value dynamics: loud = 2, medium = 1, soft = 0
cnt_sim:	Simple melody contour: range (-1, 1)
cnt_hsc:	Half simple melody contour. Range (-2, 2)

This allows for easy calculation of such parameters as the inter onset distance between notes (rhy_gap), the absolute semitone interval between subsequent notes (pit_int), the simple melody contour (cnt_sim) where the value is -1 if the note is lower than the previous, 0 if it is the same or 1 if it is higher, and the half

simple melodic contour which has a range of -2 to 2 and divides each melodic movement into large intervals (a fourth or greater up or down, represented by -2 or 2 respectively), small intervals (less than a fourth but greater than a unison (-1 or 1)) and a unison (0). The inter onset interval is expanded to assess whether the note is staccato, portato or tenuto and is expressed in the range 0.0 - 1.0, where 0.0 is staccato, 1.0 if tenuto and anything in between is portato, and represents the percentage of the inter onset interval that will be assigned to the duration of that note. At this point some explanation of the loud soft dynamics feature (dyn_lsd) is required. It will be noticed that there is a single value for each note. This feature only starts to acquire variation once the notes are divided into cells of a length greater than 1. Whereas the simple dynamics (dyn_sim) takes its calculation from the average of all the velocities in the entire clip, the loud soft dynamics calculates its separation into loud or soft from a comparison of the velocity values within the cell. Figure 4 demonstrates data for the first bar of our example where the cell length is 3. Each data item contains 3 values which represent the data for that note and two subsequent notes. Taking the original velocity data in Figure 3, the first three notes have a velocity value of 100, 71 and 109. Taking the dyn_lsd value for the first note in Figure 4 a feature value sequence of 96_64_96 can be seen. This indicates that the sequence is loud-soft loud and this is confirmed from the original data. The average of the three original values is 93.33. The first and the third value are above that, and hence loud, and the second is below and soft. 96 and 64 in Figure 4 are the default velocity for loud and soft respectively for this analysis.



rhy_gap:	1.0_0.5_0.5_	0.5_0.5_0.5_	0.5_0.5_0.5_	0.5_0.5_1.0_	0.5_1.0_1.5_	1.0_1.5_1.0_
pit_abs:	52_55_59_	55_59_57_	59_57_55_	57_55_52_	55_52_53_	52_53_57_
pit_mod:	4_7_11_	7_11_9_	11_9_7_	9_7_4_	7_4_5_	4_5_9_
pit_int:	0_3_4_	3_4_-2_	4_-2_-2_	-2_-2_-3_	-2_-3_1_	-3_1_4_
pit_csd:	4_7_11_	7_11_9_	11_9_7_	9_7_4_	7_4_5_	4_5_9_
pit_cct:	4_7_11_	7_11_9_	11_9_7_	9_7_4_	7_4_3_	4_3_7_
pit_dsd:	3,0_5,0_7,0_	5,0_7,0_6,0_	7,0_6,0_5,0_	6,0_5,0_3,0_	5,0_3,0_4,0_	3,0_4,0_6,0_
pit_dct:	3,0_5,0_7,0_	5,0_7,0_6,0_	7,0_6,0_5,0_	6,0_5,0_3,0_	5,0_3,0_3,0_	3,0_3,0_5,0_
pit_csn:	0_0_0_	0_0_1_	0_1_0_	1_0_0_	0_0_0_	0_0_0_
dur_abs:	1.0_0.5_0.5_	0.5_0.5_0.5_	0.5_0.5_0.5_	0.5_0.5_0.5_	0.5_0.5_1.5_	0.5_1.5_0.5_
dur_spn:	1.0_1.0_1.0_	1.0_1.0_1.0_	1.0_1.0_1.0_	1.0_1.0_0.0_	1.0_0.0_1.0_	0.0_1.0_0.0_
dyn_lsd:	96_64_96_	64_96_64_	96_64_64_	64_64_96_	64_96_64_	96_64_96_
dyn_sim:	2_0_2_	0_2_1_	2_1_0_	1_0_2_	0_2_1_	2_1_2_
cnt_sim:	0_1_1_	1_1_-1_	1_-1_-1_	-1_-1_-1_	-1_-1_1_	-1_1_1_
cnt_hsc:	0_1_1_	1_1_-1_	1_-1_-1_	-1_-1_-1_	-1_-1_1_	-1_1_1_

Figure 4: Expanded data for bar 1 with a cell length of 3

The final analyses involve the consideration of the accompanying chord

progression for this melody. The top staff in Figure 3 represents the notes of the accompanying chord progression and chord and key analysis of this information. Taking the second bar chord symbol as an example (C:Dm7(ii)), the prevailing key (C, capital for major or small for minor), the chord root and quality (Dm7) and the function of the chord in the prevailing key (ii, Roman numeral notation where the number indicates the scale degree as related to the major scale, and where capital letters indicate a major chord and small indicate a minor chord[10]) is displayed. The analysis of each of these chords can identify the following information:

- The overall quality of the chord and the nature of any extended tones.
- The overarching key and its quality (major or minor)
- The function of each chord in the key as indicated by the bracketed Roman numerals.
- The chord tones and scale related to each chord.

By way of example in the accompanying chords of the example melody, a very obvious I - ii - V - I progression in C major can be seen. However the V chord is a 7b9 chord which is diatonic to the key of C minor. Because of the overwhelming presence of chords indicating a major key, this chord is not temporarily identified as being in a minor key. When the Ab melody note in that chord is considered, it is analyzed as being non diatonic when considered against the prevailing key (C major), however it is analyzed as a both diatonic and a chord tone when considered against the prevailing chord[11]. With this information at our disposal, the last analyses can be made, all of which relates to differing perspectives of the pitch of the note. The chromatic scale degree (pit_csd)

[10] As yet, the analysis algorithm does not do fully featured Roman numeral notation (mostly because it was not deemed necessary for the algorithm to function. The contextual awareness of the chord progression analysis does extend to differentiating between chord progressions centred in a single key but using secondary dominants, and chord progression where there are series of ii-V progressions outside the overarching key, which in some cases may be secondary dominants (such as in bebop). In the first case the chords are identified as V7/x in the overarching key and in the second case the chords are identified as a ii-V progression in a temporary key.

[11] This is not necessarily conflicting information as either one or the other might be used to construct a new melody, but it indicates the contextual awareness of the chord analysis algorithm. The chord progression analysis algorithm comes with limitations based entirely on the time that was available for development: it performs well on multi key harmonic settings; it does not, however, identify extensions beyond the ninth on chords, nor does it handle polychords, suspended chords and modal settings. It also does not deal with sloppy playing, notes held over several chords nor is it able to discard a wrong note. It is therefore necessary to prepare the source files with some care.

represents the note as a distance in semitones from the prevailing key centre and similarly the chromatic chord tone (pit_cct) represents the note as distance in semitones from the prevailing chord root note. Both of these values would fall in a range of 0 - 11. The diatonic scale degree (pit_dsd) is represented by two values: the first indicates the degree of the prevailing key in the range of 1 - 7 where 1 represents a unison and 7 a seventh interval. The second value represents any alteration to the note. A zero would indicate no alteration and a 1 or -1 would indicate an sharpened or flattened scale degree respectively. The diatonic chord tone (pit_dct) has similar parameters and does a similar evaluation but using the mode related to the prevailing chord. Footnote 11 describes the issue of the Ab in bar 3 which is an altered 5th according to the prevailing key (C major) but is a diatonic 2nd degree according to the mode of the prevailing chord (harmonic minor 5 mode related to the 7b9 chord)[12]. The last analysis takes the aforementioned mode of the prevailing chord and assesses whether the note is a chord tone (0) a scale tone (1) or neither (2). This feature was an idea taken from the work of Keller and Morrison (2007) which used as similar way of classifying melodic material for use in an algorithm that generated jazz improvisations.

After the various feature values for individual notes are established, these are grouped into cells of sequential notes with each note in the piece as a starting point, and with a length set by the cell length parameter. Figure 4 is an example of this information with a cell length of 3 and the complete information is available in Appendix A. Programmatically, each cell can return a string of characters for the result of each of the features described, for use in a pattern matching algorithm.

Table 9 contains an excerpt from the Feature List showing every instance of the dyn_lsd feature, which analyzes the velocity of the notes in the feature cell (once again the cell length is 3) as being either loud (96) or soft (64). The process of generating the feature repeat list involves identifying where data repeats itself. This is done for each feature. In this case it can be seen that the cells highlighted with an asterisk (*) represent repeated information (96_64_96_). Each item in the feature list is grouped with similar features. Table 10 shows a list of all the different values for this feature as well as the position where they repeat. The list has been ordered from the most to the least number of repeats.

[12]It is arguable that this could or should be a harmonic major 5. Under the circumstances, i) it can be conceded that yes it could ii) there is no certainty if it should and iii) it isnt, entirely due to constraints of development time.

Table 9: Feature list example

dyn_lsd:	position=0.0	96_64_96_*
dyn_lsd:	position=1.0	64_96_64_
dyn_lsd:	position=1.5	96_64_64_
dyn_lsd:	position=2.0	64_64_96_
dyn_lsd:	position=2.5	64_96_64_
dyn_lsd:	position=3.0	96_64_96_*
dyn_lsd:	position=4.0	64_96_64_
dyn_lsd:	position=5.5	96_64_96_*
dyn_lsd:	position=6.5	64_96_64_
dyn_lsd:	position=7.5	96_64_64_
dyn_lsd:	position=8.0	64_64_96_
dyn_lsd:	position=9.5	64_96_64_
dyn_lsd:	position=10.0	96_64_96_*
dyn_lsd:	position=11.0	64_96_96_
dyn_lsd:	position=11.5	96_96_64_

Table 10: Feature repeat list sorted by repeat count

dyn_lsd	64_96_64_	1.0, 2.5, 4.0, 6.5, 9.5,
dyn_lsd	96_64_96_	0.0, 3.0, 5.5, 10.0,
dyn_lsd	64_64_96_	2.0, 8.0,
dyn_lsd	96_64_64_	1.5, 7.5,
dyn_lsd	96_96_64_	11.5,
dyn_lsd	64_96_96_	11.0,

Table 11: Comparative feature repetition

cnt_sim:	position=0.0	0_1_1_	dur_abs:	position=0.0	1.0_0.5_0.5_
cnt_sim:	position=1.0	1_1_-1_	dur_abs:	position=1.0	0.5_0.5_0.5_
cnt_sim:	position=1.5	1_-1_-1_	dur_abs:	position=1.5	0.5_0.5_0.5_
cnt_sim:	**position=2.0**	**-1_-1_-1_***	**dur_abs:**	**position=2.0**	**0.5_0.5_0.5_**
cnt_sim:	position=2.5	-1_-1_1_	dur_abs:	position=2.5	0.5_0.5_1.5_
cnt_sim:	position=3.0	-1_1_1_	dur_abs:	position=3.0	0.5_1.5_0.5_
cnt_sim:	position=4.0	1_1_1_	dur_abs:	position=4.0	1.5_0.5_0.5_
cnt_sim:	position=5.5	1_1_1_	dur_abs:	position=5.5	0.5_0.5_0.5_
cnt_sim:	position=6.5	1_1_-1_	dur_abs:	position=6.5	0.5_0.5_1.0_
cnt_sim:	**position=7.5**	**1_-1_-1_**	**dur_abs:**	**position=7.5**	**0.5_1.0_0.5_**
cnt_sim:	**position=8.0**	**-1_-1_-1_***	**dur_abs:**	**position=8.0**	**1.0_0.5_0.5_**
cnt_sim:	**position=9.5**	**-1_-1_-1_***	**dur_abs:**	**position=9.5**	**0.5_0.5_0.5_**
cnt_sim:	**position=10.0**	**-1_-1_-1_***	**dur_abs:**	**position=10.0**	**0.5_0.5_3.5_**
cnt_sim:	position=11.0	-1_-1_0_	dur_abs:	position=11.0	0.5_3.5_1.0_
cnt_sim:	position=11.5	-1_0_1_	dur_abs:	position=11.5	3.5_1.0_0.5_

Table 11 represents the data for the simple contour (cnt_sim) feature placed next to the data for the same cell for absolute duration. It is clear that an exact repetition of the values for the cnt_sim feature occurs at each of the four positions. When the dur_abs data is considered, the repetition is not quite as complete: the first and the third items are the same but the second and fourth are different. Programmatically, this repeat schema would return a list of indices describing the particular incomplete repeat schema for dur_abs, which would be 0, 1, 0, 2. (a similar index list is available for all other features, and by way of example, the index list for cnt_sim, the feature used to identify the repetition, would be 0, 0, 0, 0) This incomplete repetition information may be used when recombining new features together in the recombination algorithm. For instance, should the repeat schema identified for the cnt_sim feature with an output string of -1_-1_-1_(as detailed in the left columns of Table 11) be used, a list of start positions of 2.0, 8.0, 9.5 and 10.0 would be obtained. Then some other feature would be decided upon to complete the rhythmic aspect of the recombination (call it rhythm_A) and attach it to the cnt_sim index list. This would mean that when the new melody is generated, rhythm_A would be used at each of the four

positions (2.0, 8.0, 9.5 and 10.0) to derive the rhythm of the newly generated cells. Furthermore, should a list of three different pitch features (pitch_A, pitch_B and pitch_C) be chosen and attached to the dur_abs feature above, the dur_abs index list mentioned above (0, 1, 0, 2) would be used to decide which pitch item would be used for each repetition, and get the following combinations:

position=2.0	rhythm_A	pitch_A	(index 0)
position=8.0	rhythm_A	pitch_B	(index 1)
position=9.5	rhythm_A	pitch_A	(index 0)
position=10.0	rhythm_A	pitch_C	(index 2)

Using variations of perfect and imperfect repetition and relating them to new combinations of features, it is expected that new ways of examining source material for new compositions will be uncovered and put a user in the position to explore whether these repetition schemas indeed hold one or several of the keys to making musical sense, and possibly expose universal or style specific repetition schemas that underpin music.

The recombination algorithm requires a list of parameters to be able to successfully create musical output. Firstly it requires the proposed form of the output, which would include the time signature and form length, and may include an underlying chord progression. Obviously if the chord progression were left out, the algorithm would not be able to make use of any of the chord and key specific pitch features described in the working of the analysis algorithm, as there would be no harmonic context, but this would not mean the algorithm would be incapable of working. Next, the overall repeat schema would be selected, which would give the list of start positions[13] and the aforementioned feature repeat index lists which describe imperfectly repeated features.

Lastly, a set of features which would allow the algorithm to correctly calculate a value for position within the cell, pitch, duration and velocity is needed. At this point in the development of the algorithm, position is entirely dealt with be a combination of the position value from the overall repeat schema and the inter onset interval (rhy_gap) feature. Duration is either absolute duration

[13] While positions has been refered to as a single value indicating a position on a timeline , these positions are actually stored by the algorithm as an offset from the nearest bar line. For instance, given the understanding that a value of 1.0 would indicate the duration of a quarter note, in a piece of music in 4/4 time as position of 3.0 would indicate the third beat of the first bar. This would be represented instead as the value pair (2, -1.0) where the first value is a bar number and the second value an offset value in quarter notes. Were the time signature of the ouput changed, a note one quarter note before the beginning of bar 2 would be present.

or proportional duration features (dur_abs and dur_spn respectively). Velocity is delivered by the simple duration or loud/soft duration features (dyn_sim or dyn_lsd respectively). When the issue of pitch is considered, things are a little more complicated. When using the absolute pitch feature (pit_abs) the pitch value is easy to derive but obviously does not take into account any harmonic context that may have been chosen for the output. When using any of the other pitch features, there are several issues that need to be resolved before the algorithm can generate a final pitch value. Take for example the pit_mod feature for the first note in Figure 4 on page 45 with the values 4, 7, 11. This represents the notes E, G and B. The first question to resolve is octave starting point. If the algorithm is unclear on this point there are a variety of options available and these can be set by the user: the algorithm can use the original octave information, the algorithm can select the E closest the the average of all the notes in the source clip, or the user can set any value that takes their fancy. Moving on to the next note, the question is whether the melody will rise or fall: in the absence of further information the algorithm would use the original contour of the source melody, or the user could define a separate contour feature (cnt_sim or cnt_hsc, being the simple and half simple contour feature respectively) to dictate the contour of the melody. Lastly, once the cell pitch data is generated the algorithm would need to consider the tessitura of the generated melody. To this end, register information for the final melody would need to be set. Furthermore the user would be able to set whether notes outside of the register of the instrument would be dealt with piecemeal (so a note that was too low or too high would be adjusted by an octave in the appropriate direction, effectively changing the melodic contour) or whether the entire cell would be adjusted, preserving the contour. Consideration would need to be made for melody which might not fit into the register defined, in which case a piecemeal approach might be chosen or the option which deviated least from the preset instrumental range would be chosen. It remains to be seen whether parameters such as these would make a significant difference to the user control over the final melody.

The abovementioned parameters are selected from a user interface pictured in Figure 5 This graphic user interface was developed for testing with access to detailed parameters as a priority, so aesthetic considerations were not foremost. There are four areas dealing with different aspects of the algorithm. There is a standard menu bar along the top of the screen which allows for the selection of the output form (under the Output tab), the melody to be analyzed and its associated chord progression (RepetitionAnalysis tab). The Live tab deals with initialization

of the playback system and is not directly involved with the functioning of the algorithm. **Area A** allows displays information about the output form, such as the file name, number of times the form is repeated (formCount), the bar length of the form, the time signature and an indication of the chord progression. At the bottom there are buttons that allow for transposing the output form into other keys. All of these values can be adjusted to change the output of the algorithm. **Area B** is concerned with the management of the output of the algorithm, which can be directed to Ableton Live for direct playback, can be saves in a musicXML file for perusal in a scoring program like Sibelius or MuseScore, and saved as a text file in the same format as the output forms and analysis source material. Furthermore it is possible to buffer previous output files so the application can save a musicXML file with multiple outputs in the same file. This was implemented for the purposes of the thesis and for comparative analysis of various settings. **Area C** displays information about the source material for the feature analysis and allows for the setting of the cell length and the selection of the repetition template. Each repetition is displayed on the list as its list of cell start points, and is ordered from the template with the most number of repetitions to the least. **Area D** is where the choice of each feature is made with clearly indicated modules for rhythm, duration, dynamics, pitch and contour. Each module contains three windows. The left window displays all the options available for selection. Double clicking one of these items adds it to the middle window, which contains a list of the features selected. Buttons below this middle window allows the order of items in this list to be changed. The right window contains the possible repetition index lists to which the list of selected features can be attached.

Every time a parameter is changed on the GUI, the application tests whether the algorithm has enough information to generate a new output and if so, outputs this information to the various output destinations as selected in Area B. The background colour of the panel is also designed to turn green if the information for generation is complete and grey if not. This was implemented to aid in debugging the interaction of the user interface and algorithm. Area B changes colour every time new output is rendered.

Example outputs of the algorithm

To demonstrate the algorithm is working correctly the following output examples are offered. For all of these tests the clip in Appendix A is used as input material. Figure 6 demonstrates the annotated scored output of the algorithm. Notes are represented in standard music staff notation. Currently the only articulation to be

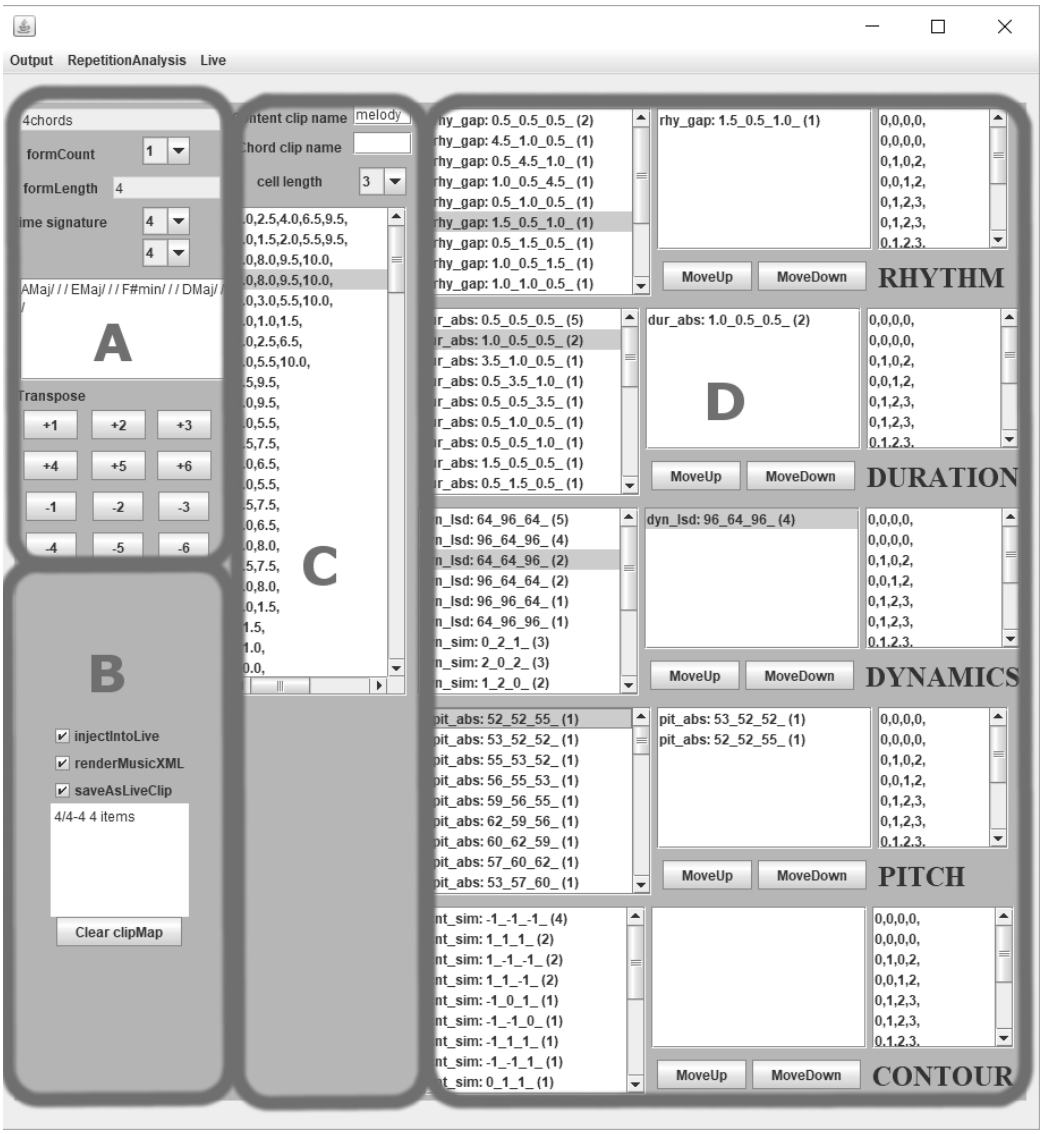


Figure 5: The graphic user interface

found will be the accent, which is used on notes which are designated as loud by the loud soft dynamic feature (dyn_lsd) or the simple dynamic feature (dyn_sim). There is no differentiation between the medium or soft notes for the latter feature in the scored output.

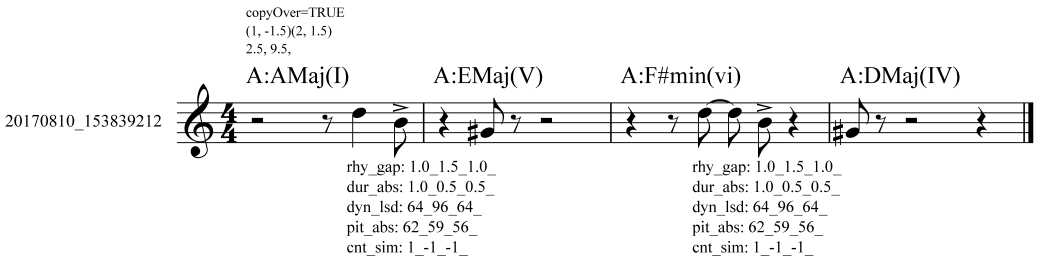


Figure 6: Annotated score output example

The chord symbols above the staff represent the analysis of the prevailing chord of the output form in the format X:Y(Z) where X is the prevailing key (capitalized for major and small for minor), Y is the chord symbol and Z is the analysis of the chord in the prevailing key. The staff name (in this case 20170810_153839212) is a composite of the date and time the clip was rendered and is not directly relevant to the musical output of the clip. Above the staff in the first bar is the indication as to whether the copy over rule is active or not, as well as the start position of each feature from the chosen repetition schema represented in bar relative terms (e.g. (1, -1.5) which represents the position one and a half beats before the beginning of bar 1 (count starts from 0)) and absolute terms (e.g. 2.5 which represents a position 2.5 quarter notes from the beginning of the timeline). Lastly, under the first note of each cell position are the parameters that make up that particular cell, preceded by the abbreviated code identifying which type of feature is being used, but always including rhythm, duration, dynamics, pitch and contour values.

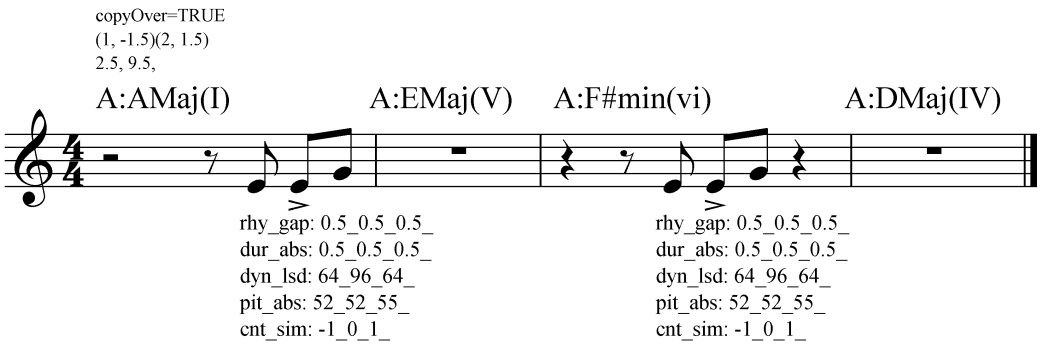


Figure 7: Barline centred positioning example

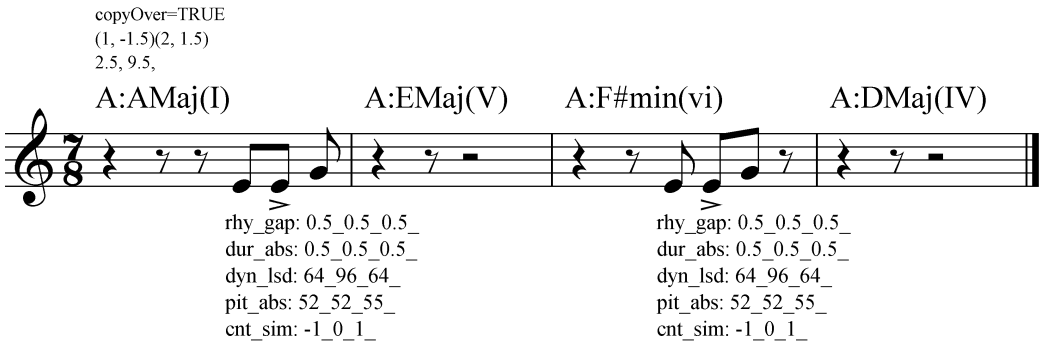


Figure 8: Barline centred positioning with data from Figure 7 in new time signature

Figure 7 shows the output from the algorithm with two cells at position (1, -1.5) and (2, 1.5) (this being an offset of -1.5 quarter notes from barline 1 (count starts at 0) and 1.5 quarter notes from barline 2) Figure **??** shows output with exactly the same criteria except the time signature has been changed to seven eight. The beginning of each cell retains its relationship to the nearest barline even though in absolute timeline position is different.

In Figure 9 two examples of output with two cells that overlap each other can be seen. In the top example the copy over rule has been set to false. The first three notes, D, B and G#, represented by the absolute midi pitch value of 62, 59 and 56, can be clearly seen in the pitch (pit_abs) parameters of the first cell. The second cell begins an eight note later and does not overwrite notes already placed by the first cell. The second cell has its parameters in parentheses to indicate that some of the notes have been omitted due to notes placed by the previous cell in the same positions. In the second example, the copy over rule is set to true: the middle two notes are now drawn from the parameters of the second cell and there are no parentheses.

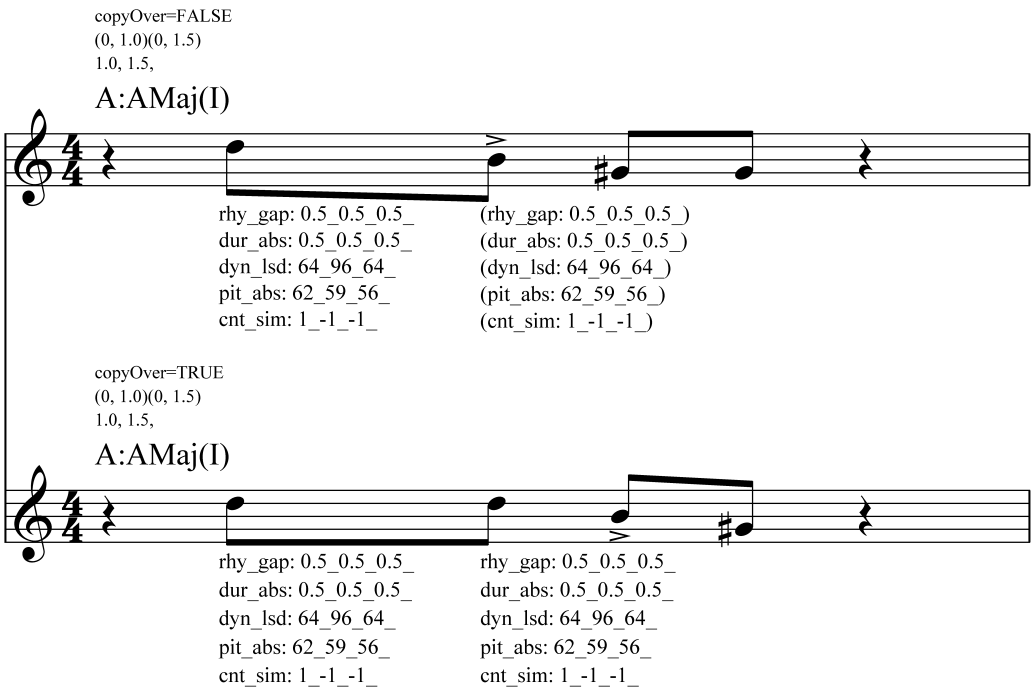


Figure 9: Copy over rule demonstration

Figure 10 shows the output of inexact repetition. The both examples contain the same three cell start positions and the same parameters except for pitch. In the top staff the pitch sequence is the same for each cell, but in the bottom staff three different pitch sequences have been chosen and attached to a parameter that does not repeat in each cell. Figure 11 demonstrates a similar example but where the rhythm has been varied in the lower staff.

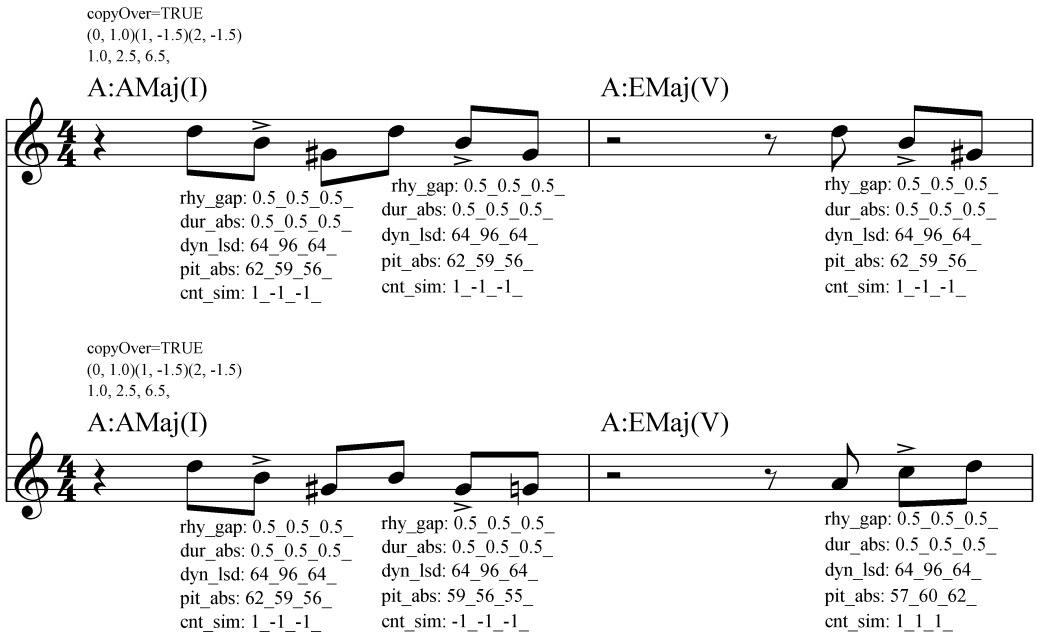


Figure 10: Inexact repetition with three different melodies

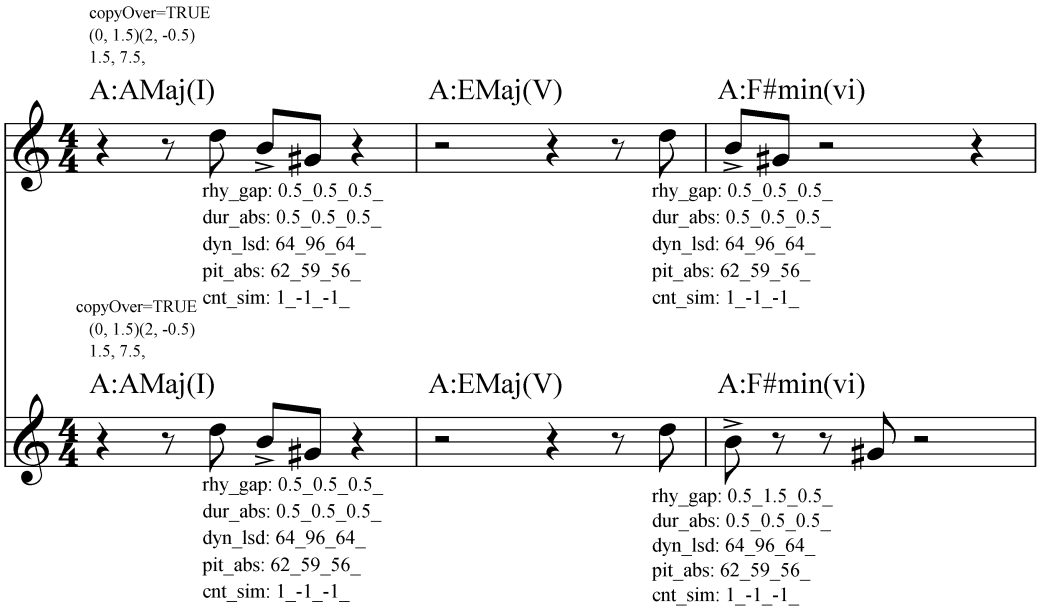


Figure 11: Inexact repetition with two different rhythms

This concludes a description and demonstration of the workings of the analysis and recombination algorithm, showing that it behaves consistently with expectations. In the next chapter, a qualitative assessment of the music qualities and possibilities of the output will be made. What follows is a framework for a series of tests to explore the scope of the output.

Testing the algorithm

The output of the algorithm will be measured qualitatively as to how its output may be appropriate for a selection of musical scenarios.

- Using a given accompaniment track, the algorithm's ability to deliver interesting new melodic material for a live electronic music performance will be assessed. The scope of this output will be limited to music with a generally relaxed, downbeat[14] feel, a tempo below 100 bpm and chord progressions drawn from popular music. This is a matter of personal taste. The requirement here would be that the algorithm produce stylistically appropriate material, and might do so rapidly enough that changing the input parameters might be judged part of a performance.

- A contrafact melody on existing jazz chord progressions and attempt to evaluate how well the new tune would work in my jazz quintet will be generated.

- Using appropriate input material, compositions for another project of mine which plays a fusion of Balkan and middle eastern music will be generated.

Furthermore, for each output scenario the effect that various choices of repetition schemas have on the new composition will be assessed. The tunes selected for this are categorized as follows:

- Riff based tunes with obvious repetitive elements and highlighting rhythmic displacement of repeated motifs. The tunes selected are 'Straight no Chaser'[15] by Thelonius Monk and the instrumental melody from '*Opa Cupa*'[16], a traditional Roma tune.

- Tunes with longer, phrase level repeated sections without an emphasis on rhythmic motifs within a phrase. The chosen material is 'Blue Bossa'[17] by Kenny Dorham and 'Black Orpheus'[18] by Louis Bonfa.

- Through composed tunes with obvious repeats happening at the section - rather than phrase level. The tunes selected are the bebop standards

[14] as in the electronic music subgenre, rather than the first beat of the bar

[15] see Appendix B for transcriptions of the corpus material with chord analyses by the chord progression analyzer

[16] see Footnote 15

[17] see Footnote 15

[18] see Footnote 15

'Anthropology'[19] and 'Confirmation'[20], both written by Charlie Parker.

The selection of tunes predominantly from the jazz standard repertoire is entirely due to the authors familiarity with that corpus of works. Musical material from which the output will be reconstructed will be chosen based on various treatments of rhythm, duration, dynamics and pitch and melodic contour. The following criteria are proposed:

- Rhythm: syncopated vs unsyncopated
- Duration: staccato vs legato. The interpretation of staccato needs explaining as it is not the definition from Western classical music, with short detached notes predominating. In this context the playing style of bebop is referenced, which is characterised by legato playing of mostly eighth notes, a strong correlation to an underlying pulse and a focus on accent and syncopation rather than expressive timing.
- Dynamics: this is problematic as the nature of musical phrasing requires some kind of dynamic treatment of melody, so it is difficult to differentiate opposing approaches to dynamics which are not intrinsically tied up with some other aspect of music, for instance phrase accents in jazz are directly connected to syncopation, which for the purposes of this project is a rhythm issue, or the use of accent on off beats in an jazz phrase of eighth notes, which is intrinsic to the time feel of the music, which is beyond the scope of this research. Nevertheless the tests will allow for the behaviour of this musical feature to be assessed.
- Pitch and melody contour: while these can be separated for the purposes of the recombination algorithm, they are linked for the purposes of musical analysis. For this criteria material that is chromatic vs diatonic and predominantly scalar vs predominantly intervallic would be chosen. This last binary poses an interesting problem as melodies such as 'Blue Bossa' or 'Black Orpheus' are statistically predominantly scalar, yet the melodic interval of the first two notes of each of these tunes (an ascending octave in the case of Blue Bossa and an ascending minor sixth in the case of Black Orpheus) contribute significantly to their memorability.)

[19] see Footnote 15
[20] see Footnote 15

The following chord progressions have been chosen as musical forms for the output material. Two chord progressions for each genre have been chosen, with each choice bringing a particular challenge to the test environment.

- Pop/electronica:
  - 'Four chords' chord progression: I V vi IV - the most used chord progression in popular music and also diatonic to a single key.
  - 'While my guitar gently weeps': ii I V6 bVII VI - 10th most popular chord progression in popular music according to Palmer (2011), chosen because it covers a chromatic situation.
- Jazz:
  - Jazz blues: 12 bar chord progression and one of the most used progressions in Jazz.
  - Rhythm Changes: 32 bar chord progression based on 'I got rhythm' by George Gershwin and also one of the most popular vehicles for contrafact composition in jazz.
- Balkan: This style is included to test the possibility and extent that musical style may be to a degree encoded in the software. The choices here do not in any way address the vast complexity of this genre (if indeed it can be boiled down into a single genre), especially in terms of the handling of rhythm and complex time signatures, as well as the diverse approaches to melody and harmony. This authors (limited) experience of the genre has led the criteria for this choice to be, in the first case, a chord progression that plays with parallel major and minor tonalities and in the second case plays with avoiding obvious establishment of a key centre with a V I cadence:
  - I vi V7/V V7 - found in *Forum marked as 'Balkan Music Progressions'* (2007)
  - V bVI V iv

The tunes mentioned as sources for repeat schema material will also be used as material for the recombination algorithm. A broad classification of the material is proposed in the following way.

- Rhythm
  - Syncopated: 'Anthropology', 'Confirmation'

  - Unsyncopated: '*Opa Cupa*', 'Blue Bossa', 'Black Orpheus'

- Duration
  - Staccato: 'Anthropology', 'Confirmation', '*Opa Cupa*'
  - Legato: 'Blue Bossa', 'Black Orpheus'

- Dynamics - chosen because of particular usages of dynamics, potentially intrinsic to other musical features and also dependant on a particular interpretation
  - Motif based accents: 'Straight no Chaser'
  - Dynamics to articulate time feel: 'Anthropology'

- Pitch
  - Diatonic: 'Blue Bossa', 'Black Orpheus'
  - Chromatic: 'Anthropology', 'Confirmation', '*Opa Cupa*'
  - Scalar: 'Blue Bossa','Black Orpheus', 'Straight no Chaser'
  - Intervallic: 'Anthropology', 'Confirmation'

4 Findings and Discussion

The biggest problem when faced with testing this algorithm is the scale of the options. Just taking the three output forms, each with six possible sets of master repeats schemas, each generating on average 350 repeat schemas generates a phenomenal set of possibilities. When the variations of music feature choices for output material are factored in it becomes clear this analysis will not by any stretch of the imagination be exhaustive. Each test involved making adjustments in only one of the three broad parameter areas (choice of output form, master repeat schema material and recombination material). This was done to facilitate a clearer understanding of the effect of particular parameters on the output material. As tests proceeded, further limitations were imposed as interesting parameter spaces were uncovered.

General Finding

As a general finding, the algorithm can produce meaningful musical material. It can also produce a large amount of less than coherent music, which confirms that this is algorithm is not a complete understanding of human musical cognition. As testing proceeded, an understanding the constraints of various choices of musical features emerged quite easily, which suggests that there is some quite clear understanding of musical coherence embodied in the parameters. The following section details observations about parameter choices that stand a better chance of creating meaningful music.

Pitch Feature limitations

The first serious test involved looking at the details of the choice of recombination material, and how the choice of various combinations of rhythm, duration, dynamics, pitch and melodic contour affected the output. The first issue that arose related to the variety of pitch features available, as detailed in the Methodology chapter[21]. Even though all six features related to pitch were useful to identify repeat schemas, only the diatonic chord tone (pit_dct) and the chord/scale/chromatic tone feature (pit_csn) were interesting for continued use. While the diatonic scale degree (pit_dsd) produced some plausible material, it tended to create many repeated series of pitches due to its identification of pitches in terms of the prevailing key, which for many of the output forms does

[21] see Table 8

not change. The chromatic chord tone (pit_cct) and chromatic scale degree (pit_csd) held no awareness of whether the prevailing chord or key respectively was major or minor and so delivered some quite random dissonances. The absolute pitch feature (pit_abs) would produce sometimes hilarious polytonal results due to it identifying notes as absolute pitches and paying no attention whatsoever to the prevailing chord or key in the output form. As a result, these unusable feature where eventually removed from the recombination pitch feature lists. They may return in future work where the harmonic content of the output form is contingent on a generated melody, but for the scope of this project the output forms are perforce preset.

Pitch Endings

When choosing a pitch feature cell for recombination, the most reliable results came from cells which ended on a chord tone. Ending a cell on a note that is not a chord tone often resulted in a sense of incompleteness. If the gap to the next phrase was small enough (tempo dependant, but about 2 - 3 beats) there were instances that the subsequent phrase could be seen as resolving the unresolved note by continuing the phrase. If this gap is too great the final note of the cell was seen as the end note of the phrase and the ending remained incomplete. There were cases when an unresolved phrase end was seen as an early resolution to the subsequent chord, but this was entirely by chance as the algorithm can only identify an eighth note syncopation of the beginning of the bar, and has no way of identifying or planning for an eventuality where this was different.

Handling of chromatic notes

When testing the same recombination parameters on different output forms, diatonic pitch choices were appropriate for the Pop/Electronica styles but not challenging enough for Jazz. Because the algorithm has only a vertical understanding of chromaticism (it will identify that a note falls outside of the mode related to the prevailing chord, but not its relationship with previous or subsequent notes), chromatic notes are not used in a stylistically identifiable way except by chance. Figure 12 gives an example of chromaticism which works by chance. This uses a repeated pit_csn feature of 0_0_1_0_1_2 (chord tone, chord tone, scale tone, chord tone, scale tone, chromatic tone) At point a., the chromatic note is by chance a #11 on a major chord, and quite satisfying to this listener. The #5 at point b is a little more challenging as it is less harmonious with the underlying chord, but the overall melodic line is good so once again this

listener is satisfied. It would be interesting to take this parameter setup and attempt to push it into a less satisfying melodic space in an attempt to discover the boundary of acceptable vs unacceptable. Point b is also a case of one phrase abutting the subsequent one, with the result that the chromatic note is not seen to end the phrase, which would be more unsatisfying. Figure 13 takes the same rhythmic, duration, dynamics and pitch features as Figure 12 and produces further unexpectedly satisfying results. Even though the #5 at point c is out of key, it works melodically. The #2 at point d works as a chromatic lower approach to the subsequent note which is a chord tone and leaves us with no lingering lack of resolution. Notes such as at points b and c (both #5 of a major chord) might be helped were the algorithm able to adjust the voicings of the underlying chord progression. What would be significant even before that would be a way of computationally judging the result to be satisfying even though it flies in the face of 'theoretical' correctness.

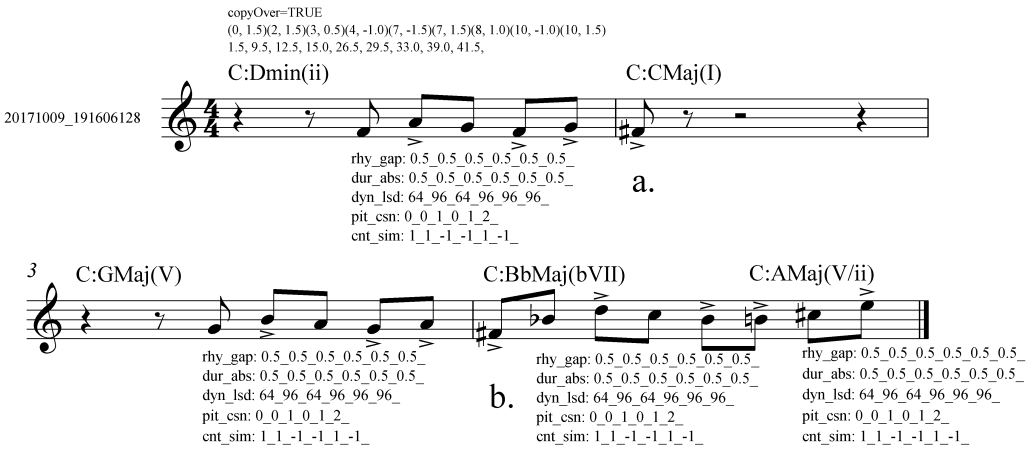


Figure 12: Some interesting chromaticism

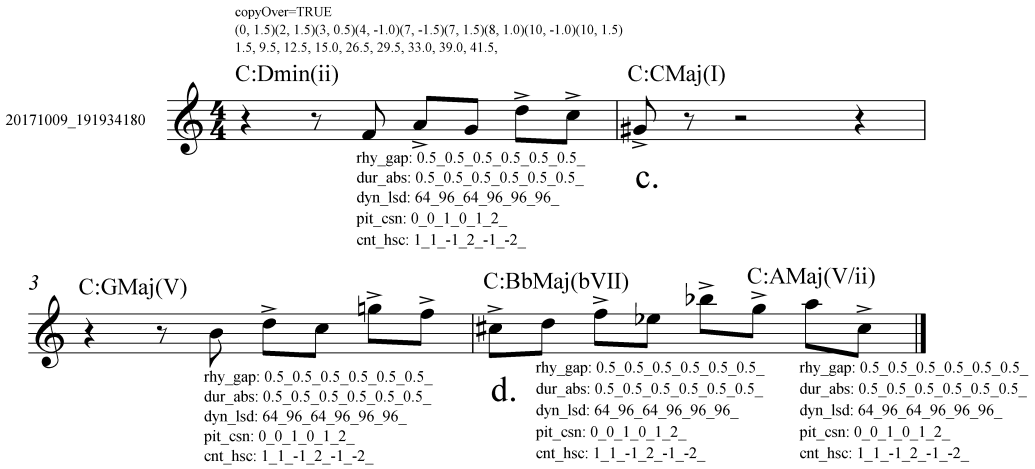


Figure 13: Some more interesting chromaticism

Relationship between tempo and chromaticism

Tempo is not a consideration for the algorithm, and is controlled entirely in the Ableton Live playback engine. It should be noted that at a tempo of 117 bpm Figure 12 was quite attractive. Dropping the tempo to 90 bpm gave a completely different effect. The faster tempo appears to allow the melody to lean more towards the style and energy of bebop (which is the style from which the recombination material is drawn), and hence makes the theoretically dissonant notes more acceptable to the ear. With a more laid back tempo the melody is less effective and the chromatic notes seem out of key.

Contour

The application of contour features in the algorithm can lead to large jumps when combined with a chromatic chord tone feature, which dictates the scale degree of the mode related to the prevailing chord. While this is not a problem per se, often a simple scale passage would be inverted into a series of melodically challenging jumps of a seventh. When no contour feature is explicitly selected, the algorithm uses the contour information of the selected pitch information. This usually yielded a melodically smoother result. Any explicit choice of contour resulted in a melody with large jumps, the overall effect making the melody more angular and the difference between different choices of contour being a mere detail in this general musical effect.

Rhythmic placement

Using 'Confirmation' as a source of recombination material generated a large variety of rhythmic variations, as the tune contains a combination of eighth notes, eighth note triplets and sixteenths. In the tune, eighth note triplets are used as ornaments, with a typical use being demonstrated in Figure 14.

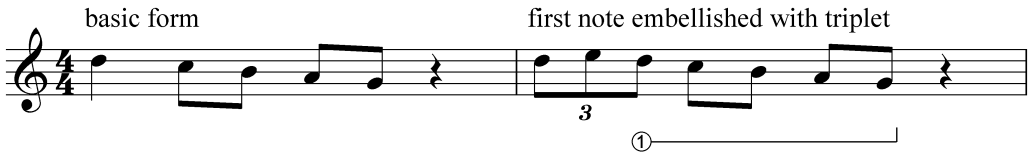


Figure 14: Triplets

However the rhythm feature analysis does not analyze this differently from the notated even eighths, and does not take into account that in a jazz setting with a swing feel even eighth notes would not actually be played evenly. In Figure 14, when the rhythm cell starting on the 3rd note of the triplet as indicated at 1 (and bearing in mind that the algorithm does not analyze time feel such as swing at all) is used as recombination material starting on a beat placement other than the third triplet of a beat, one starts getting unusual placement of subsequent notes. Interestingly enough, when generating material for one of the pop/electronica output forms at 90 bpm these notes do not sound so out of place, referencing styles like Lofi Hip Hop which utilize quirky and atmospheric samples at relaxed tempos with superimposed swing and straight time feels and sometimes a very relaxed approach to beat accuracy. With styles where rhythmic accuracy is important, this is less effective. When generating a Jazz blues form, high number of note spacings with a length of a dotted quarter note and above generally makes the output sound rhythmically obtuse. Generally a combination of eighth and quarter notes were most stylistically appropriate for the Jazz idiom.

With rhythm cells featuring 16th notes and dotted quarter notes, a busy, syncopated line results which is not necessarily supported by the backing track. This suggests a virtuosic jazz fusion style but without a coherent sense of phrasing. At lower tempos (90bpm for this test) quirky timing has more scope to work before sounding weak, as mentioned above. Repeated 16th notes work well. Too many odd multiples of 16ths (like a note of 5/16ths length followed by 7/16ths) make the phrase sound mistimed. An odd 16th multiple followed by even 16th or 8th multiples result in all subsequent notes falling on weak 16th beats which can sound out of time, especially if the backing track does not

support this time feel. An odd 16th multiple can be grounded with a series of notes of 1/16th length before or after it.

Cell length of recombination material

At the low tempo, short cell lengths (in this case 3) were quite effective as it allows space between phrases. In cases like this it is significant to have a harmonious ending note as the end note of the cell is less likely to join into the beginning of the next cell to form a single phrase. With long cell lengths (a cell length of 12 was tested) the choice of end notes was less significant for a reasonably dense repeat schema as the output sounded like a continuous and rambling jazz solo, verging on incoherence, especially when chromatic notes were well represented.

Density of repeat schemas

Overall there seemed to be three broad areas into which the density of repeat schemas fall:

- One or two repetitions in the form of the repeat schema material - these cases do not represent many usable options as there are long gaps between musical material in the output. When the output form is 4 bars long, as with the Pop/Electronica chord progressions, it is possible to generate an appropriate melody with one or two repetitions falling within that space, but with longer forms such as the 12 bar blues and 32 bar 'I got rhythm', such a small number of repetitions navigates the waters between incompleteness and absurdity.

- A repetition point every one to two bars - this represents the most usable material as there is often the case that repeated cells are rhythmically offset, which creates a pleasing effect of irregular repetition. E.g. Figure 15 Shows a repeat schema extracted from 'Straight no Chaser' which delivers a four item repeat schema of (0.0, 3.0, 8.0, 11.0). As can be seen in Figure 15 there is a rhythmic group of four eighth notes starting on the beginning of bar 1 and the repeating on beat 4 of the same bar, creating a pleasing combination of repetition (same rhythmic grouping) and variation (starting on strong and then weak beat) . This pattern is then repeated in bar 3.

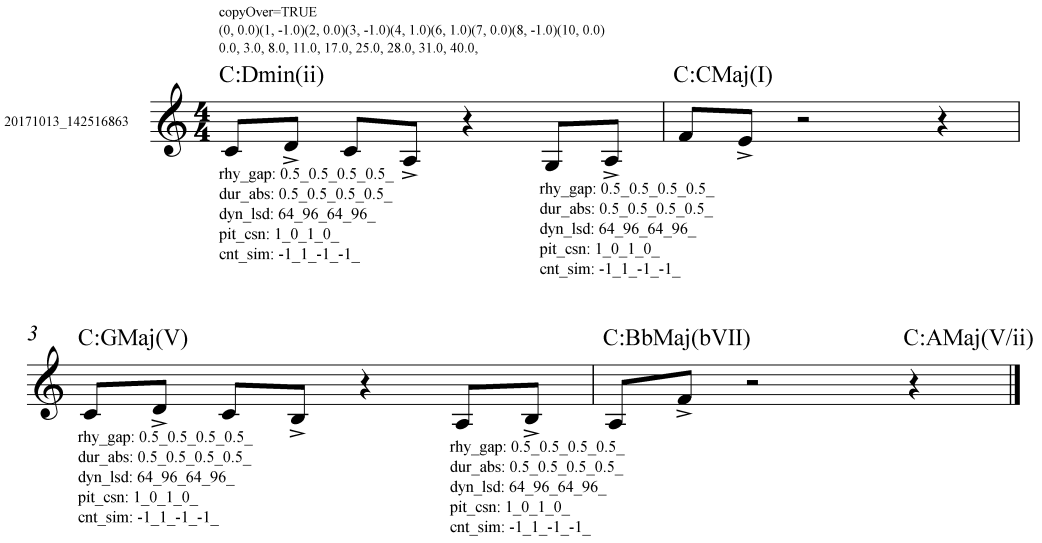


Figure 15: Offset repeats

- A repeat point on every, or almost every note of the original, with a gap between repeat points being less than the length of the cell. This creates a long series of notes as in Figure 16 where there is a new cell starting on each new note, overwriting the notes from the previous cell. What is surprising in this example is that a phrase structure does emerge, with a series of 11 notes starting on the upbeat to bar 1 (because this is a loop, this note is found at the end of bar 4) . The entire rhythmic grouping repeated in bar 3 but starting a beat later, another example of interesting rhythmic displacement of a repeated theme.

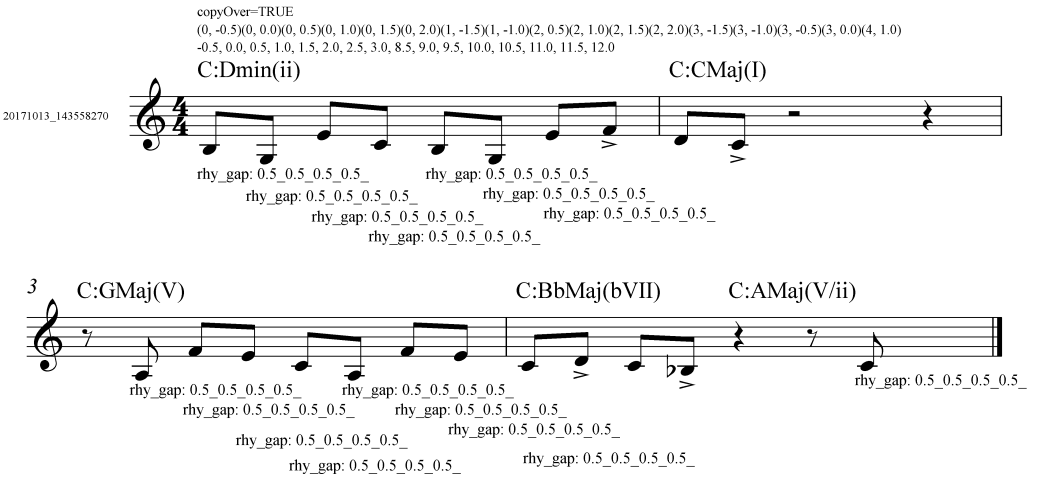


Figure 16: Dense repeat schema example

Rhythmic placement and pitch related to repetition schema

'Straight no Chaser' is a riff based tune and delivers some very useful repetition schemas which fit into item 2 of the previous paragraph. To illustrate the effect on phrasing as affected by choice of rhythmic cells, we take the repetition schema from Figure 15 and demonstrate a variation. In the top staff of Figure 17 we can see this displacement of start point clearly in the first bar and then repeated in the third. When we use a different repeated rhythmic pattern as in the second staff, the repetition is less obvious because the first two and second two repeats join into a single phrase, with no less musical cohesion than the first. This affects the choice of pitch feature as well, as in the top melody it sounds coherent to choose a chord tone as an ending note for each cell. With the choice of rhythm feature in the bottom line, the end note of the first and third cell do not act as phrase end notes, so therefore do not have to be chord tones. The algorithm as yet has no awareness of this situation, but an expert user would be able to deduce this from the information on the GUI.

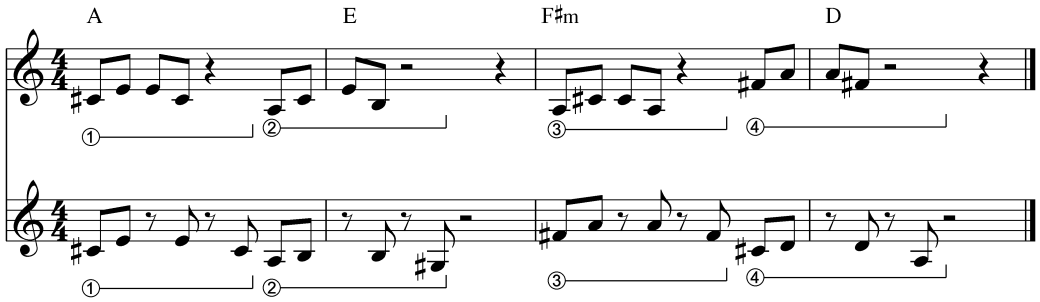


Figure 17: The effect of rhythm feature choice on phrasing

Duration

Duration settings using the staccato/portato/tenuto feature often give rise to problematic sustained notes when the tenuto note falls on a relatively long note, such as the last note in a cell, which is sustained till the beginning of the next cell. This can be especially problematic when the underlying chord changes which makes the sustained note into a stylistically inappropriate dissonance. With repeat schemas that are quite sparse (a repeat every four bars or so), this issue can be especially pronounced. As a result, the best results were achieved using no sustained notes at all. In situations where the duration feature did not produce unmusical results, it seemed that moving between legato and staccato versions of a melody might offer interesting development possibilities, however due to the handling of the duration parameter by the algorithm, it was not as easy

to predict the behaviour of any chosen duration cell (compared to pitch and rhythm), as many different parameters could produce entirely the same musical result. Further testing of duration as a compositional tool suggested some interesting nuances of interpretation affected by this feature. Figure 18 shows two versions of a melody, the difference being the first notes in each bar. This example was the most easily notatable example of the musical effects noted, as generally these nuances would be brought to a performance through the interpretation rather than the notation. The small gap between the first and second note in each bar of the first example serves to draw attention to the second note and give the melody a greater sense of deliberation. This idea of separation of notes by an amount smaller than the prevailing quantization value (in this case 8th notes are the prevalent rhythmic resolution of the fragment) is not explicitly dealt with in the algorithm, but suggests powerful possibilities.

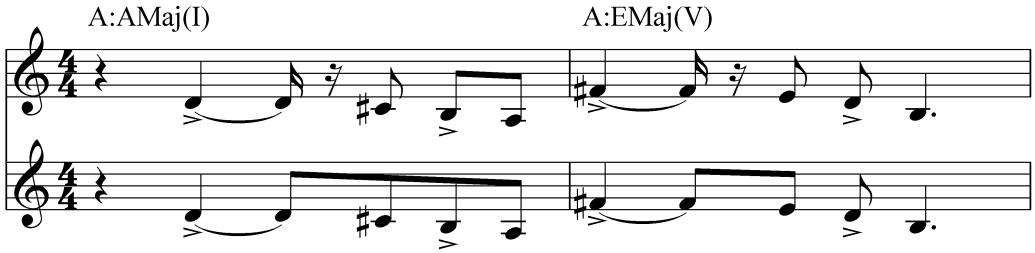


Figure 18: The effect of duration on interpretation of melody

Dynamics

In terms of dynamics, having some dynamic variation was better than having none. When attempting to generate music resembling jazz, any dynamic feature choice that contained an accent on the last note of a cell was better than ending on a weak accent. Given a cell length of 4, the best dynamic choices were strong-weak-weak-strong or weak-strong-weak-strong, which is consistent with this author's experience of use of accents in jazz. In Figure 19 are four examples of dynamics patterns over two 5 note cells. In the first example we have off beat accents articulating the swing time feel, which are effective due to the rhythmic energy of alternating accents on a continuous eighth note melody. (alternating accents on the onbeat are also effective but less idiomatic for jazz). The second example is less effective, due to a poor ending to the phrase with two unaccented notes. The third example is effective as it highlights the last note of each phrase with an accent. In the last example, the rhythm has been changed with the 5th note of each cell moving half a beat later. The accents emphasize the time feel

with alternating accents, as well as the beginning and end of each cell, making this a very strong use of idiomatic dynamics. It must be borne in mind that the above analysis is quite academic: the computer can play back any combination of parameters completely accurately, but the algorithm is not intended for actual live playback of a jazz tune, so the accent schemes are potentially meaningless. Were this music put in front of a band familiar with the jazz performance idiom, it is most likely that no accent markings would be required at all, and any interpretation would far surpass that of the computer for stylistic appropriateness.

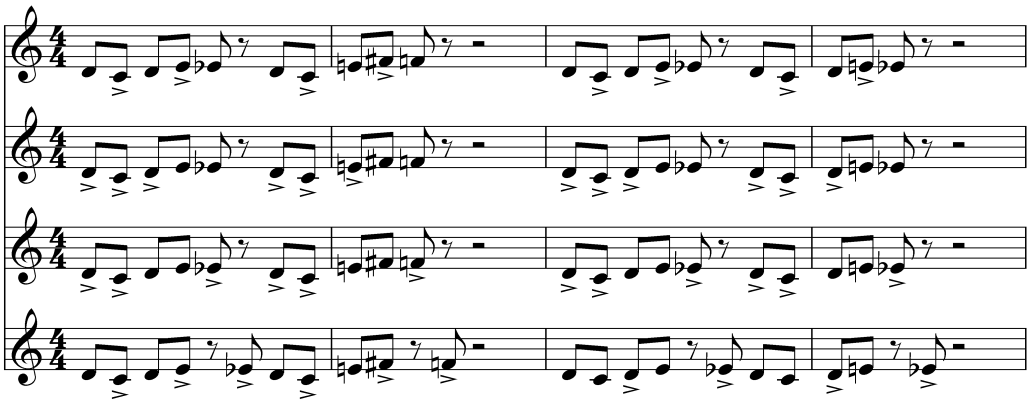


Figure 19: More and less effective dynamic usage

In a test on the pop/electronica chord progressions and given rhythm, pitch and duration parameters appropriate for the style (a combination of short and long notes, a generally legato feel and entirely diatonic melodic choice with good ending notes), the choice of dynamic pattern was less constrained by stylistic considerations than the examples mentioned above. Any dynamic feature was able to significantly affect the perception of the melody, with each variation of accents offering a new interpretation of the tune. In this style, the dynamics feature is definitely usable as a compositional tool and easier to treat on an equal footing with pitch and rhythm. There were seldom instances with this style where a particular choice of dynamic feature was deemed to be 'wrong', a situation which happened frequently with jazz.

Repetition schema coherence related to output form length

The algorithm tended to produce more coherence over short output forms. Unless the output form encompassed only one repetition point, forms of 8 bars tended to have a good success rate in terms of musical coherence. A pleasing 12 bar blues form and some 16 bar Balkan tunes were generated, but they required a lot more experimentation to get a successful result. With the 32 bar 'I Got

Rhythm' form a successful output was not achievable. As the form length increased, there were more and more examples of either rambling improvisation (in the case of a very dense repetition schema) or musically mystifying periods of silence (in the case of one that was less dense). This makes sense in light of the fact that while an analyzed tune may deliver many different repetition schemas, the recombination algorithm uses only one of these to generate a new tune. Were we able to 'fill in the gaps' with another interlocking repetition schema, maybe we would start approaching more cohesive output for longer forms. By selecting musically cohesive sections of several output examples, some quite successful attempts to compile new longer form tunes by hand was made.

Interactivity

According to the guidelines for interactivity laid down by Smuts (2009), the user interface for the algorithm is arguably not interactive at all. It is much more akin to Smuts' example of a car where the behaviour is entirely predictable. However, looking at the interface in terms of the way Flow theory was applied by Pachet to his Continuator, there is arguably a degree of engagement possible with this application, with some limitations. The user interface for this algorithm has proven to be very usable for the purposes of tests. After preparing the corpus material, a working method of identifying the scope of a test and then making a desktop video of the process was established, with the musical output and comments recorded. This video would then be reviewed, generally once only, and emerging themes noted. Because the musical output was directly transferred to Ableton Live for playback, it was possible to immediately hear the effect of a change in algorithm parameters. For the purposes of this thesis, the output was also rendered to a musicXML format file which could be opened in a notation program for display in written form.

Using the analysis of interactivity in terms of cognitive load theory by Sweller (2010), the biggest challenge for this user lies in the area of extraneous cognitive load, which deals with the way that the information is presented. While it was mostly easy for the expert user to understand the presentation of the data in the form of strings of numbers, there were times when this was difficult to decode. This would probably be highly mystifying for a new user. The other two items, intrinsic- and germane cognitive load are not relevant for the experience of a single user who is an expert on the workings of the algorithm. Whether this level of interaction constitutes accepting the GUI as an instrument is less clear. Certainly playback is effectively instantaneous and the parameters are clear to an expert user.

There are two problems however. The first related to the nature of the algorithm: when, for instance, generating a 32 bar output form, once the parameters are set, it is just a matter of sitting back and listening to the output. For testing or composing this could be acceptable, but for live performance there are no obvious assurances of the quality of the output nor any guarantee that further interaction over the course of the form could be helpful to the coherence of the output. Having said that, a live performance use of this tool is far more likely to involve a four bar loop in a pop electronica style than a canned playback of a jazz tune, so in that respect the algorithm and its GUI approaches usability in a live context. The second problem expands on this issue of quality of the output. It would be interesting (and enhance usability for new users) to be able to have some way of classifying parameter choices into groups based on musical mood and style. Furthermore, I personally dislike a mouse in a live performance situation, so mapping the controls to some kind of hardware interface would be part of finalizing a design for stage use.

Conclusion

By way of conclusion, we answer the question of whether the algorithm in its current form can generate appropriate material for the three composing scenarios listed in the 'Testing the algorithm'[22] section. It was found that for the first scenario, being melodic material for a live Pop/Electronica music performance, that the algorithm was, in its current form and (importantly) with an expert user, able to deliver a diverse set of coherent musical material. A new user may struggle with understanding the presentation and interaction of the data and find themselves facing dead ends where the behaviour of the algorithm was inexplicable. Also, there are some non catastrophic software bugs which preclude general use of the application at the moment.

The nature of the algorithm's success with the above material becomes apparent when longer forms are attempted. When approaching the composition of a 12 bar and 32 bar form in the jazz idiom, it became apparent that the form length was critical to the coherence of the musical material. With a combination of some luck, skill and a great deal of experimentation a 12 bar jazz blues[23] was generated which could be termed 'interesting', and would be stylistically identifiable as jazz if performed by my jazz ensemble, who are of an intermediate skill level. The experimentation involved organizing recombination pitch

[22] see page 58
[23] see Figure 21 in Appendix C

material so that the right kind of notes fell in the right place in the form. The temptation to tweak notes by hand had to be resisted and with a bit of work a satisfactory result was achieved. With the 32 bar form it was not possible in the time frame to come up with a single setup that generated a totally coherent tune. A complete tune was developed using 4 and 8 bar sections generated by different parameter setups[24], but all the options of complete tunes from a single setup ended up either being an endlessly rambling and incoherent solo, or with some uncomfortably long gaps between phrases, with no amount of adjusting of the rhythm feature and cell length able to improve the situation. Given that a single analysis tune renders more than one repetition schema, to reverse engineer a coherent tune the software might be adjusted to superimpose several repetition schemas automatically.

Working towards a tune in a Balkan style highlighted an issue of stylistic bias in the automated analysis of the output chord progression. The first 'Balkan' chord progression chosen for these tests (I vi V7/V V7), while arguably a common progression in the style, did not deliver anything remotely 'Balkan' sounding, and the problem lay in the melody. The chord progression was analyzed as being in a major key, with the iv chord borrowed from the parallel minor. Therefore, when modes related to each chord were calculated (to work out diatonic and chromatic notes) the allocation went as follows. (Bear in mind the automated analysis is still a work in progress, with limited contextual understanding of a chord within a progression)

- I - Major (Ionian) mode
- iv - Dorian mode (consistent with borrowing from parallel minor)
- V7/V - mixolydian mode (consistent with the analysis of a V7/V in a major key)
- V7 - mixolydian

This authors limited experience of this melodic style has seen a fairly consistent use of modes of the harmonic minor, and the distinct lack of any modes of this scale emerging from the automated analysis precluded any melodies suggesting this particular sound. The cause of this only became apparent when an attempt was made to generate melodies using the second chord progression (V bVI V iv), which generated much more stylistically appropriate melodic material, due to the

[24] see Figure 22 in Appendix C

analysis of the progression in a minor key, and the related mode for the V chord being the fifth mode of the harmonic minor. This author does not in any way pretend to define melodic construction in this diverse style of music so simplistically. This being a qualitative analysis, the answer to the question 'Would I present this to my Balkan ensemble as a possible new tune?' was sought. With the first chord progression, that answer was consistently 'no', whereas quite pleasing 8 and 16 bar melody on the second chord progression[25] were achieved(bearing in mind that these represent incomplete works). This suggests that, should the first chord progression analyzed with appropriate modes, it could be the basis for stylistically appropriate melodies in this style.

The above issue goes some way to answering the question of whether the algorithm is stylistically biased. There can be no other answer but a conditional yes, especially in the area of this approach to melody construction. The fact that pleasing material was, within limits, generated in three different musical styles suggests that the approach to rhythm, for instance, is dependant on the input material rather than the algorithm itself to generate appropriate content. This would need further testing and experimentation but it is a pleasing result for what it is under the circumstances.

In terms of both duration and dynamics, the conclusion reached is that the algorithm errs on the side of being style-free. Many of the duration and dynamic combinations were a poor or incomplete fit given the rhythmic and melodic content with which these features were combined. This suggests that the treatment of duration and dynamics requires a more sophisticated, or at least different handling to what currently exists in the algorithm, incorporating more of an understanding of the musical context. The most bewildering feature remains the melodic contour. Because contour is an aspect of the pitch feature, its was sometimes difficult to predict the interaction of these two parameters. It may be that there a better ways of presenting the data to the user, possibly using a visual or geometric representation of the interaction, or possibly there are ways of sorting this information better. During experimentation the selection of a contour was mostly to 'shake things up' and the most overt effect was creating challenging melodic jumps. In terms of stylistic bias, this seems to suggest that melodic contour is the parameter that makes music sound like 20th century art music or not, but this is said flippantly and cannot be accepted as a serious conclusion. As with duration and dynamics, the conclusion emerging is that this is not just another musical parameter to be mixed and matched. Contour is a

[25] see Figure 23 and 24 in AppendixC

parameter that needs to be seen (by a future algorithm) in its musical context before the manipulation of this aspect of melody becomes anything more than a blunt instrument, as was the impression created of its behaviour in this algorithm.

Future research

While it has emerged that the analysis of repetition does to a degree embody an model of musical coherence, it is not in any way complete. The testing of this algorithm has proposed many new avenues of research to fine tune that output of the algorithm. There are two areas for future research. The first involves making what we have easier to interact with, and the second points towards new ways of looking at analysis and recombination.

This first area would be quite easy to implement in the existing algorithm and are largely concerned new ways of **sorting the lists of features**. The repeat schema is currently identified as a series of floating point numbers identifying the start point of each repeat. For instance, 0.0,1.0,2.0,3.5 would indicate a repeat schema with a start point on beat 1, 2 and 3 of the first bar (0.0, 1.0 and 2.0 respectively) and lastly on the 'and' of beat 4, or the eighth note upbeat to bar two in a 4/4 time signature. Sometimes, however, these series are quite long and to simplify it would involve shortening the number of start points to include only those that fall within the start and end points of the output form. In situations where the input material was 32 bars long, and the output only 4, it would clean up the interface if only the first four bars of points were shown. There is also significant **repetition of repeat schemas**. In a case where a combined pitch and rhythm were repeated several times in the source material, this would result in the two repeat schemas in the final list, one associated with a pitch feature and one with a rhythm feature. This could be simplified. Furthermore, its was found that there were many cases of feature start points being identified that contained points the same distance apart but with a different start point. This would occur in a situation where repeated material in the input material would be longer than the cell length for analysis. There would be a similar but offset pattern identified for the feature starting on the first and second (and possibly third, fourth, fifth etc) item of the repeated source material. It would simplify things to group these together as often there were interesting differences between these quite similar repetition schemas. This speaks to a larger issue of classifying material, especially for the purposes of recombination. Currently all material is sorted by popularity (i.e. how many times it occurs) but a more flexible approach to classification of material could simplify the navigation of available options. In the case of options

available for recombination, the option to be able to sort lists of pitch items in terms of types of starting or ending notes, and also chromatic content would have been helpful. This would be easy to implement in the current code.

With better classification of the material available, the code could also be extended to be able to include recombination material of **different cell lengths**. For the scope of this research, this did not constitute a debilitating limitation, but given better organization of material this would be interesting and quite possibly useful[26]. A further possibility which would be more time consuming to implement would involve finding more interesting ways of presenting the data. For instance, graphic representations of repeat schemas might be easier to understand that strings of floating point numbers.

The last possible improvement to the application as it stands relates to situations during testing, especially with long output forms, where there are areas of interesting output material with all sorts of musical resonances and development potential, and then other sections of overlong uncomfortable silences. Many repetition schemas were derived from the input material, yet only one was used to generate output. It followed that if the output could be generated from **layering several repetition schemas**, maybe the long form outputs would be able to became more complete.

A fundamental limitation of the algorithm is the discrete parameterization of musical feature material and how musical effect (both effective and ineffective) emerges from these approaches to the material. This second area of improvements point to new ways of seeing the existing musical data, and greater insights into the **interrelatedness of musical features**. It has been noted that arbitrary combinations of pitch and rhythm were generally more successful than a similar approach to duration, dynamics and contour. For example, the treatment of dynamics in jazz is tightly interwoven with phrasing and the articulation of time feel, suggesting that decisions about dynamics would either be contingent on the choices made for pitch and rhythm, or would need to be able to feedback possible changes to these decisions. With pop/electronic musical output, it was found that a more arbitrary choice of dynamics rendered interesting reinterpretations of material rather than being judged to adhere (or not) to a quite strict stylistic usage, suggesting a relationship between features that is dependant on musical style. A future experiment would need to incorporate ways of defining various types of interrelatedness between features. These experiments could take the form of encoding some theoretical approach to melody

[26] Actually it was possible to hack the system to do this but it was not an easy solution, and the possibilities were not actively tested.

construction, or an evaluation process whose results could be stored in a database and data mined for correlations. All of the output of the existing tests has been stored in various digital symbolic forms, so it is already possible to automatically capture this data and start applying statistical analysis to the relationship between features.

A further relationship to look at would be the relationship of pitch material to the underlying chord progression. As mentioned before, the analysis and recombination algorithms do not recognize the **context of chromatic notes** beyond the vertical harmony. This will allow more musically sensible use of chromatic material. In terms of the relationship between rhythm and the underlying form, two issues emerged. Firstly, the analysis of rhythm does not take into account devices such as **ornaments** which need to be analyzed and reproduced as a single item. This would, for instance, deal with the diverse rhythmic material in a tune like 'Confirmation' in a more musically meaningful way. Having said that, there were instances when doing it wrong, like the oddly times notes that result from breaking these ornaments apart, was actually stylistically appropriate, and a way should be found to be able to incorporate both approaches. Similarly, the way that these odd rhythms relate to the construction of repetition schemas could be dealt with in a more contextually aware manner. Secondly, the default handling of **syncopation** involved identifying a note an eighth note before a chord change. If this note were followed by a note on the chord change, the preceding note would be analyzed as being part of the chord prevailing in the bar that it was positioned. However, if there was no note on the commencement of the new chord, it would be assumed that the preceding note was a syncopation of the downbeat of the new chord and would be analyzed as such. This would need to be expanded in future versions of the algorithm. Firstly it might have to cater for melodies where the predominant subdivision was a 16th or a quarter note (or any other beat subdivision for that matter). Furthermore there were interesting melodic phrase endings happening several beats before a chord change which due to the context could be seen as a very early resolution to the forthcoming chord. Discovering the constraints of this musical effect, and making the algorithm aware of this possibility would be interesting. Furthermore, when generating finished works, the treatment of several features in the recombination algorithm would have benefitted from a computational understanding of the **phrase boundary**. The issues around the choice of end notes of cells, legato durations on ending notes to be held over one or more chord changes, and style specific use of dynamic patterns requiring an

accented note at the end of a phrase have been noted in the findings. Given the possibility that closely spaced cells could join into a single phrase (implying that the end note of a cell is not in every case the end note of a phrase), an understanding as to whether a note was actually on a phrase boundary or not would influence the range of choices possible in the handling of this note in terms of pitch, duration and dynamics.

Preparation of the corpus material is also an area into which some efforts could be placed. The existing system requires a distinct preparation process to make the data meaningful. It would be interesting (and challenging) to automate this process. With midi files, this would involve developing a dynamic quantization process but opens up the possibility of extracting information related to expressive timing for onsets and durations. This could possibly then be extended to allowing for live input of corpus material, a compelling live performance possibility. Were an analysis process such as this extended to act upon an audio signal as opposed to midi data, it could open up new possibilities for interaction with other musicians or the capturing of symbolic data from any available audio source.

It is pleasing to note that these results fall into some very broad new areas (interrelatedness of musical features, preparation of the corpus) to the very specific items mentioned above. A last avenue involves adding modules to generate the entire accompaniment track, possibly including variations to the chord progression, a completely new reharmonization, and the possibility of alternate time signatures[27]. Furthermore, and especially for the live electronica performance, it would be compelling to attempt to expand the musical parameters to include controller information such as pitch bend and timbre control information, or mix the use of the algorithm with some of the other live electronic performance possibilities offered by Ableton Live.

[27]The current algorithm can generate material in a variety of time signatures, but the testing procedure involved fixed backing tracks so this aspect was never explored.

References

10 Top Programming Methodologies (n.d.). Accessed 4 November 2017.
URL: *https://www.developer.com/mgmt/slideshows/10-top-programming-methodologies.html*

Ableton Website (n.d.).
URL: *https://www.ableton.com/en/live/ accessed 20 April 2017*

Ahmed, M. (2009), 'Emily howell, the virtual composer making waves in the computer world', *http://technology.timesonline.co.uk/tol/news/tech_and_web/article6884631.ece (accessed 2 September 2, 2010)* .

Alsop, R. (1999), 'Exploring the self through algorithmic composition', *In Leonardo Music Journal Vol 9. Cambridge, Massachusetts: MIT Press. pp 89-94* .

Best Development Methodology for One Person? (n.d.).
URL: *https://softwareengineering.stackexchange.com/questions/59713/best-development-methodology-for-one-person accessed 21 April 2017*

Biles, J. (1994), Genjam: A genetic algorithm for generating jazz solos, *in* 'Proceedings of the International Computer Music Conference 1994, IGMA, SanFransisco'.

Bitbucket (n.d.). Accessed 3 November 2017.
URL: *https://community.atlassian.com/t5/Bitbucket/ct-p/bitbucket*

Blitstein, R. . . (2010), 'Triumph of the cyborg composer (david copes computer)', *http://musicandculture.blogspot.com/2010/02/triumph-of-cyborg-composer-david-copes.html (accessed 2 September 2010)* .

Blog on the subject of Continuous Integration for solo developers (n.d.*a*). Accessed 23 April 2017.
URL: *http://stackoverflow.com/questions/130592/is-continuous-integration-important-for-a-solo-developer*

Blog on the subject of Continuous Integration for solo developers (n.d.*b*). Accessed 23 April 2017.
URL: *https://softwareengineering.stackexchange.com/questions/277325/how-would-a-one-man-team-benefit-from-a-continous-integration-setup*

Christensen, B. (2009), 'Emily howell the computer is the composer', *http://www.techovelgy.com/ct/Science-Fiction-News.asp (accessed 2 September)* .

Collingwood, R. G. (1938), *The Principles of Art*, New York:Oxford University Press.

Collins, N. (2001), Algorithmic composition methods for breakbeat science, *in* 'Proceedings of Music Without Walls. Leicester 2001: pp 21-23'.

Cope, D. (1987), 'An expert system for computer-assisted composition', *Computer Music Journal, Vol. 11, No. 4* pp. 30–46.

Cope, D. (1992), 'Computer modelling of musical intelligence in emi', *Computer Music Journal, Vol. 16, No. 2, pp. 69-83. The MIT Press.* .

Eurola, T. (2003), The dynamics of music expectancy, Master's thesis, Department of Music, University of Jyvskyl.

Fernandez, J. and Vico, F. (2013), 'Ai methods in algorithmic composition: A comprehensive survey', *Journal of Artificial Intelligence Research 48 (2013) 513-582* .

Forum marked as 'Balkan Music Progressions' (2007). Accessed 8 October 2017.
URL: *https://www.guitarmasterclass.net/guitar_forum/index.php?showtopic=41218*

Friedmann, M. L. (1985), 'A methodology for the discussion of contour: Its application to Schoenberg's music', *Journal of Music Theory* **29(2)**, 223–248.

Groen, D., Guo, X., Grogan, J. A., Schiller, U. D. and Osbornee, J. M. (2015), 'Software development practices in academia: a case study comparison', *https://arxiv.org/pdf/1506.05272.pdf accessed 19 April 2017* .

Grout, D. J. and Palisca, C. V. (1996), *A History of Western Music. 5th ed*, W. W. Norton & Company: New York.

Hiller, L. and Baker, R. (1964), 'Computer cantata: A study in compositional method', *In Perspectives Of New Music, Vol. 3, No. 1 (Autumn-Winter 1964). pp. 62-90* .

Huron, D. (2001), 'What is a musical feature? Forte's analysis of Brahms's opus 51, no. 1, revisited', *Music Theory Online* **7**(4).
URL: *http://www.mtosmt.org/issues/mto.01.7.4/mto.01.7.4.huron.html*

Java Website (n.d.).
URL: *https://www.java.com/en/download/faq/whatis_java.xml accessed 20 April 2017*

Keller, M. and Morrison, R. (2007), A grammatical approach to automatic improvisation, *in* 'In the proceedings SMC'07, 4th Sound and Music Computing Conference, 11-13 July 2007, Lefkada, Greece. pp. 330-337'.

Lehrdahl, F. and Jakendoff, R. (1983), *A Generative Theory of Tonal Music*, Cambridge, Massachusetts: MIT Press.

Maurer, J. A. (1999), 'A brief history of algorithmic composition'.
URL: *https://ccrma.stanford.edu/ blackrse/algorithm.html*

Mercurial (n.d.). Accessed 3 November 2017.
URL: *https://www.mercurial-scm.org/*

Midi 1.0 Specification (1996), MIDI association. Accessed 31 August 2017.
URL: *https://www.midi.org/specifications/item/the-midi-1-0-specification*

Narmour, E. (2000), 'Music expectation by cognitive rule-mapping', *Music Perception: An Interdisciplinary Journal* **17**(3), 329–398.
URL: *http://www.jstor.org/stable/40285821*

Nierhaus, G. (2009), *Algorithmic Composition Paradigms of Automated Music Generation*, Germany: Springer-Verlag/Wien.

Pachet, F. (2002), The continuator: Musical interaction with style, *in* 'International Computer Music Conference, Gotheburg, Sweden, ICMA'.

Pachet, F. (2004), *Musical Creativity Current Research in Theory and Practise. Edited by Deliege,I and Wiggins, G*, Psychology Press., chapter Enhancing Individual Creativity with Interactive Musical Reflective Systems, p. 1 15.

Pachet, F. (2008), 'The future of content is in ourselves', *ACM Journal of Computers in Entertainment, 6(3)* .

Palmer, T. (2011), 'The 10 most used chord progressions in pop and rock and roll', https://thornepalmer.wordpress.com/2011/12/29/the-10-most-used-chord-progressions-in-pop-and-rock-and-roll/. accessed 8 October 2017.

Papadopoulus, G. and Wiggins, G. (1999), Ai methods for algorithmic composition: A survey, a critical view and future prospects, *in* 'Proceedings of the AISB 99 Symposium on Musical Creativity'.

Rouse, M. (2008), 'continuous integration (ci)'.
URL: *http://searchsoftwarequality.techtarget.com/definition/continuous-integration accessed 23 April 2017*

Sandred, O., Laursen, M. and Kuuskankare, M. (2009), 'Revisiting the illiac suite a rule based approach to stochastic processes', *In Sonic Ideas/Ideas Sonicus. Morellia: Mexican Centre for Music and Sonic Arts. pp 42-46* .

Schenker, H. (1954), *Harmony*, University of Chicago Press.

Schmidt-Jones, C. (2008), *The Basic Elements of Music*, C O N N E X I O N S Rice University, Houston, Texas.
URL: *http://ufdcimages.uflib.ufl.edu/AA/00/01/16/43/00001/Music.pdf accessed 6 April 2017*

Smuts, A. (2009), 'What is interactivity?', *The Journal of Aesthetic Education, Vol. 43, No. 4 (WINTER 2009), pp. 53-73* .

Software Development Methodologies (n.d.). Accessed 4 November 2017.
URL: *http://www.itinfo.am/eng/software-development-methodologies/*

Sweller, J. (2010), 'Element interactivity and intrinsic, extraneous, and germane cognitive load', *Conceptualizations, Specifications, and Integrated Research Perspectives (2010), pp. 123-138* .

Tidemann, A. and Demiris, Y. (2008), Groovy neural networks, *in* '18th European Conference on Artificial Intelligence, vol. 178, pp. 271275'.

Todd, M. (1989), 'A connectionist approach to algorithmic composition', *In Computer Music Journal Vol 13 No 4. Cambridge, Massachusetts: MIT Press* .

Urbano, J., Llorns, J., Morato, J. and Snchezcuadrado, S. (2011), Melodic similarity through shape similarity, *in* 'Proceedings of the International Symposium on Computer Music Modeling and Retrieval, pp. 338355'.

Westergaard, P. (1959), 'Review of Hiller L. & Isaacson, L. (1959). Experimental Music: Composition with an electronic computer', *In Journal of Music Theory, Vol. 3, No. 2 pp. 302-306. New York: McGraw-Hill* .

Weston, S. (2002), *An introduction to the mathematics and construction of splines*, Addix Software Consultancy Limited.
URL: *http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.8012&rep=rep1&type=pdf accessed 4 April 2017*

Zieliski, K. and Szmuc, T. (2005), *Software Engineering: Evolution and Emerging Technologies pp 113-123*, IOS Press.

A Music analysis data

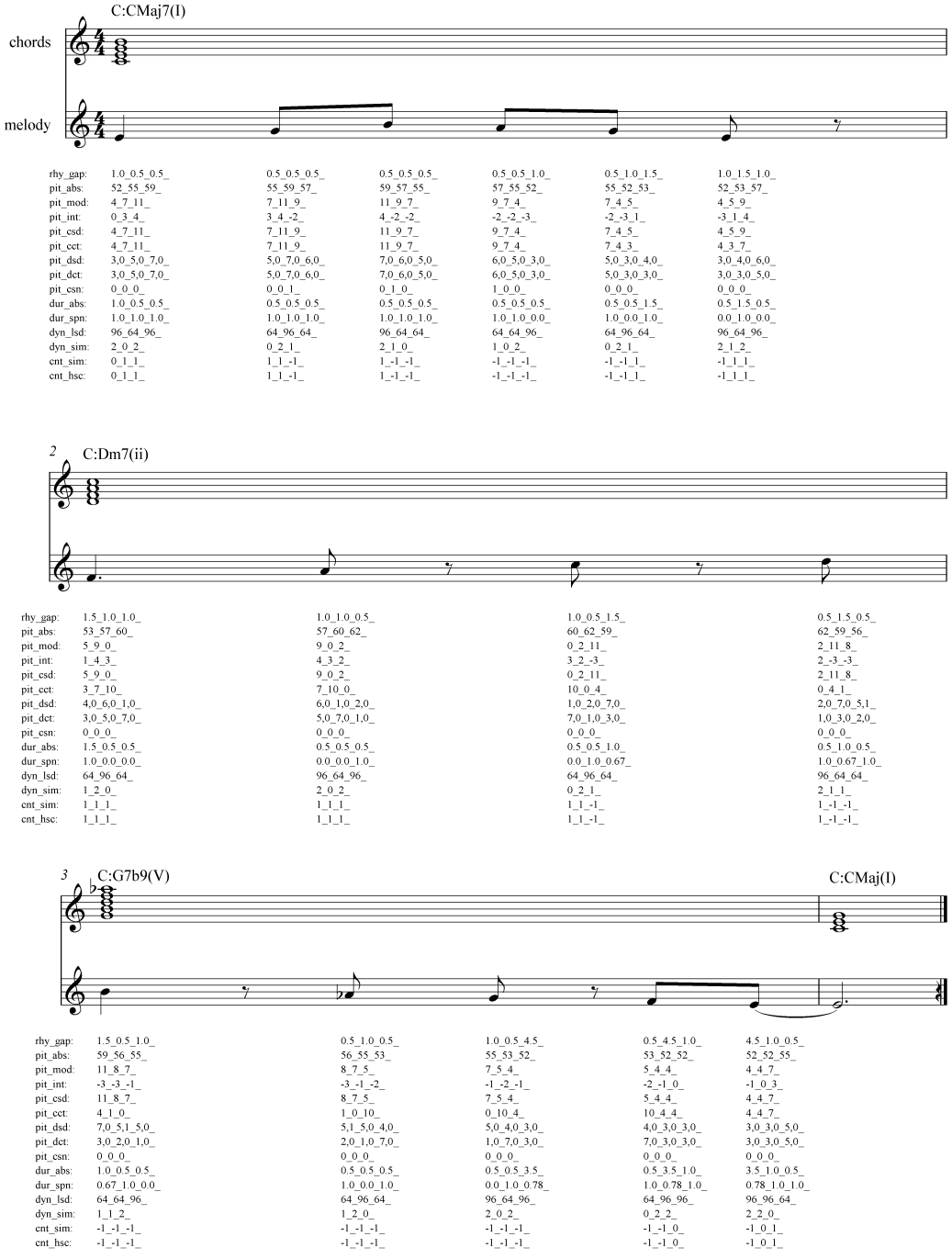
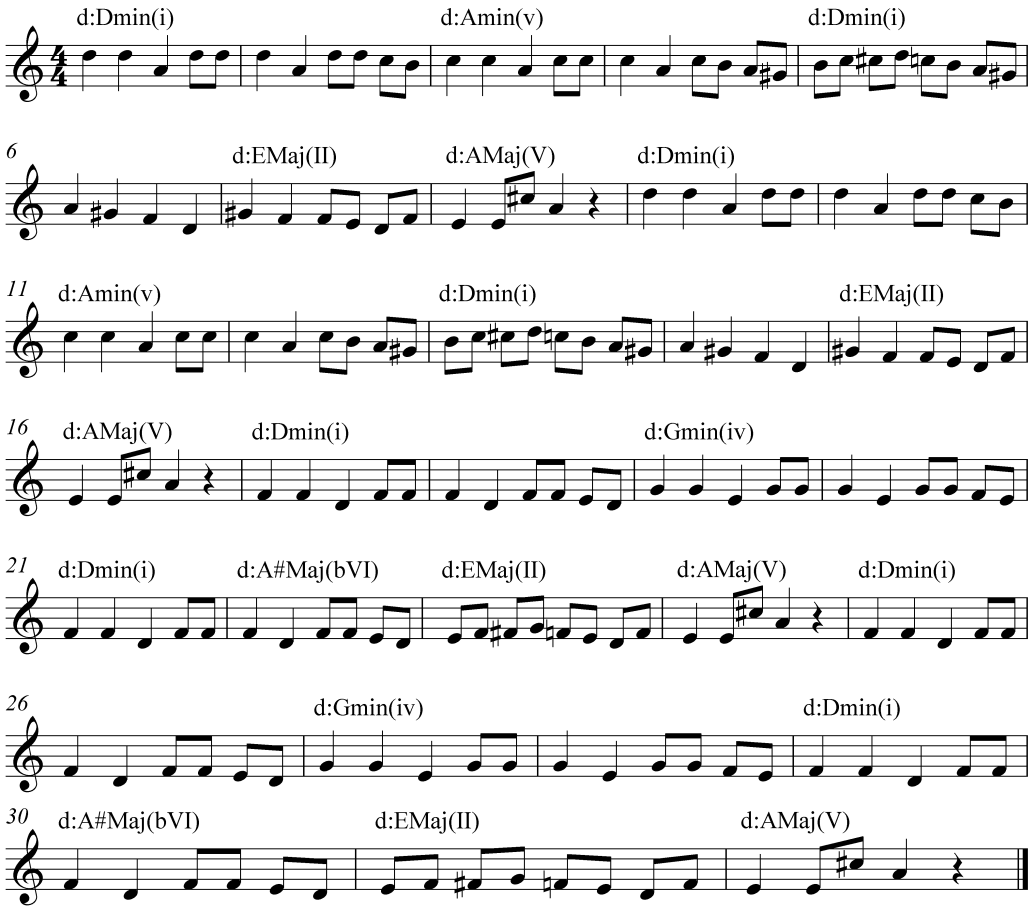


Figure 20: Expanded data for test material with a cell length of 3

B Transcriptions of corpus material

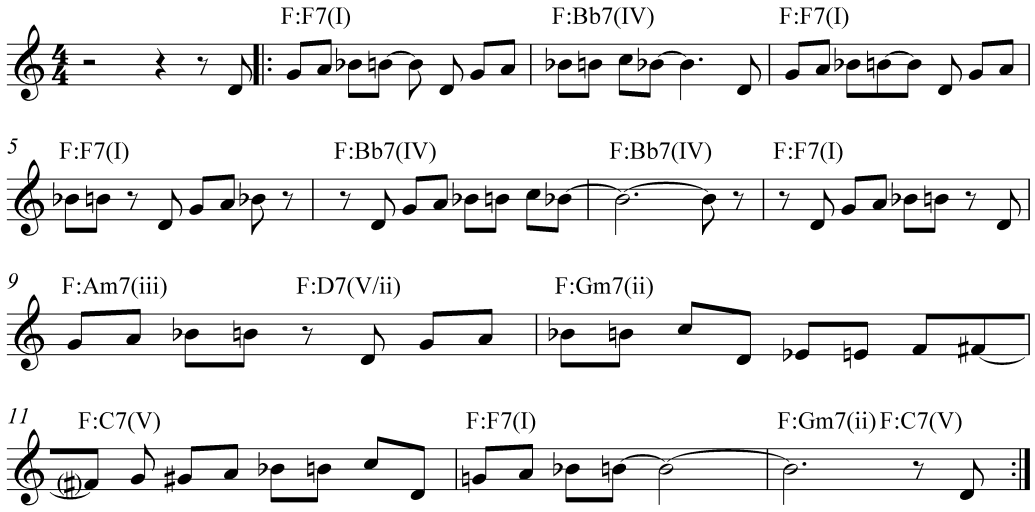
Opa Supa

Traditional



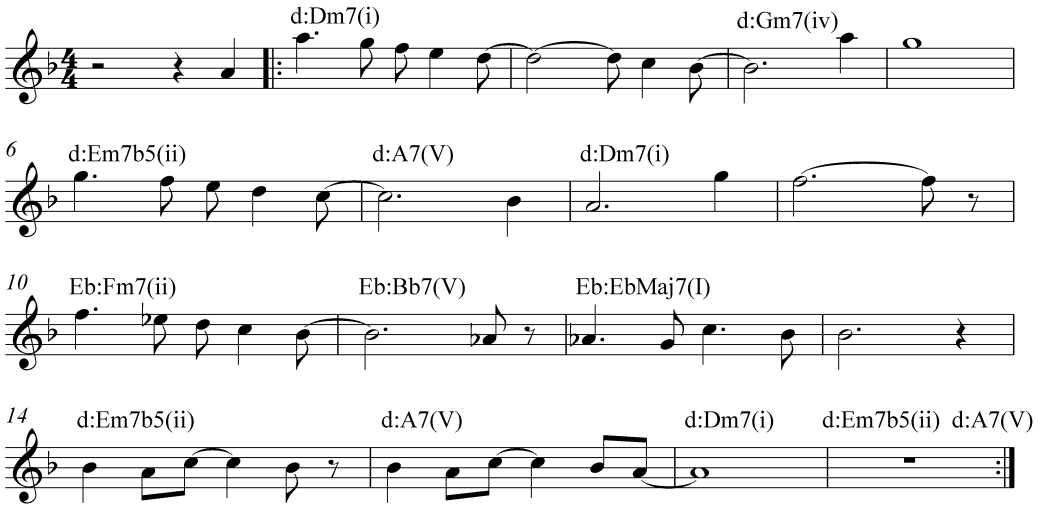
Straight no Chaser

Thelonius Monk



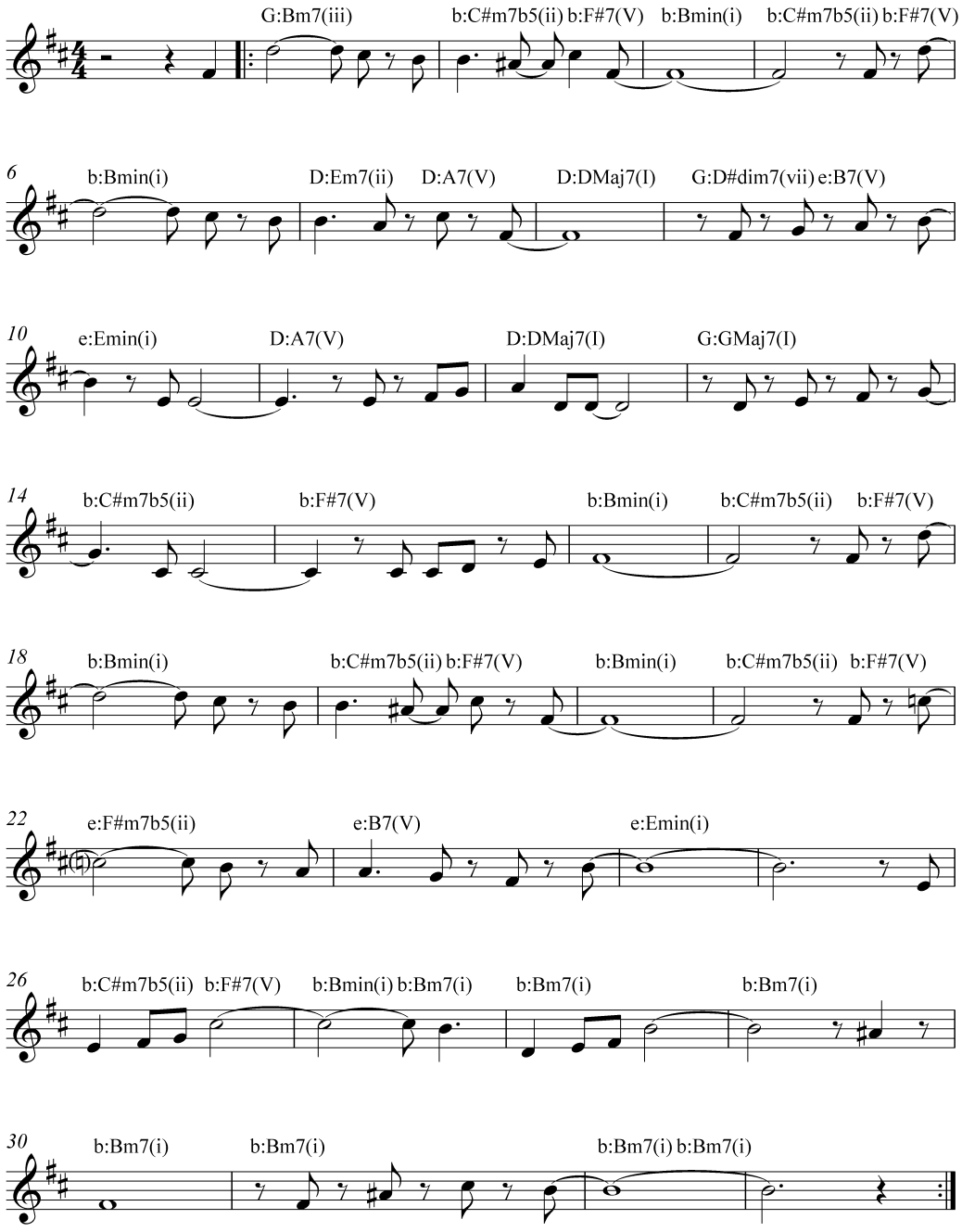
Blue Bossa

Kenny Dorham



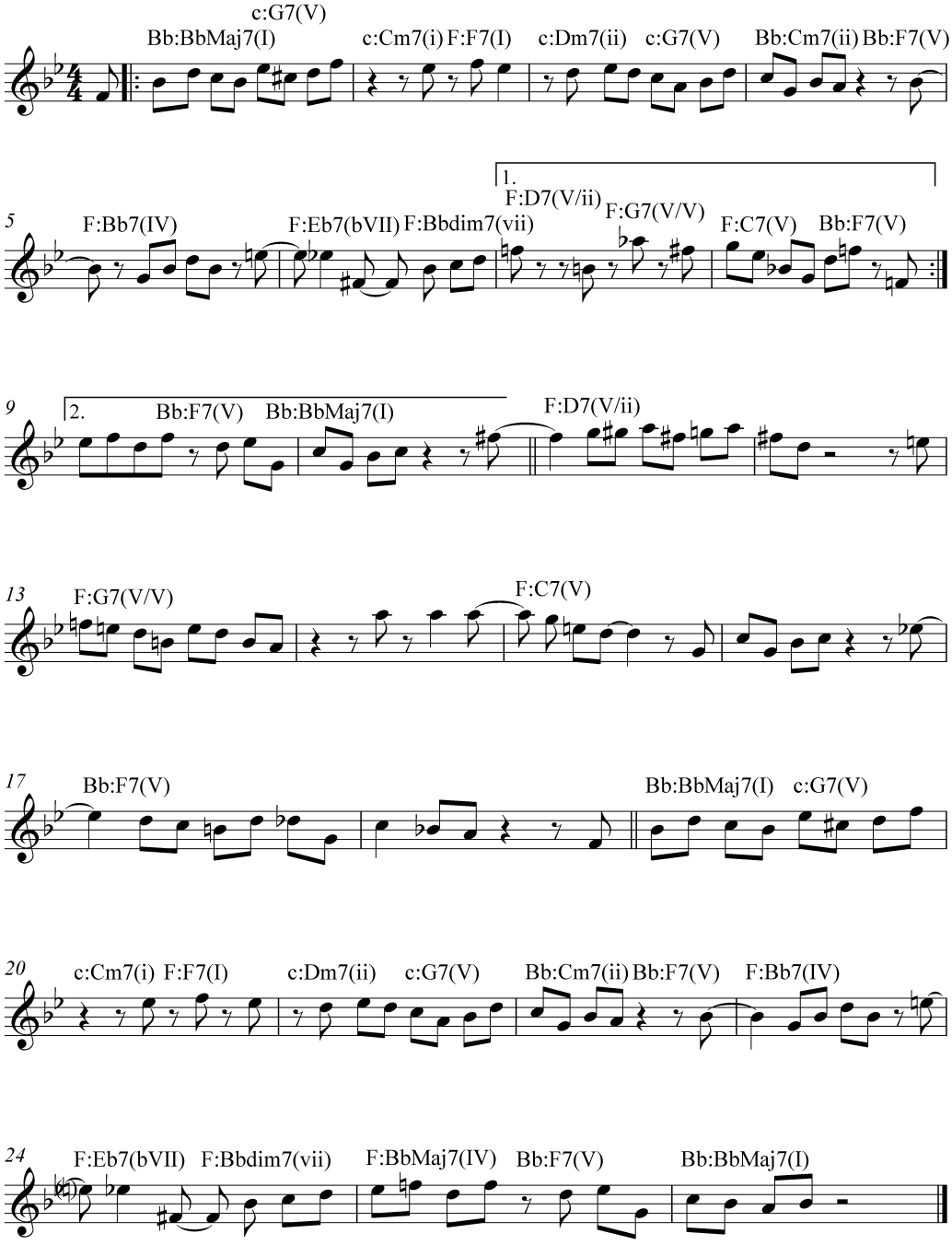
Black Orpheus

Louis Bonfa



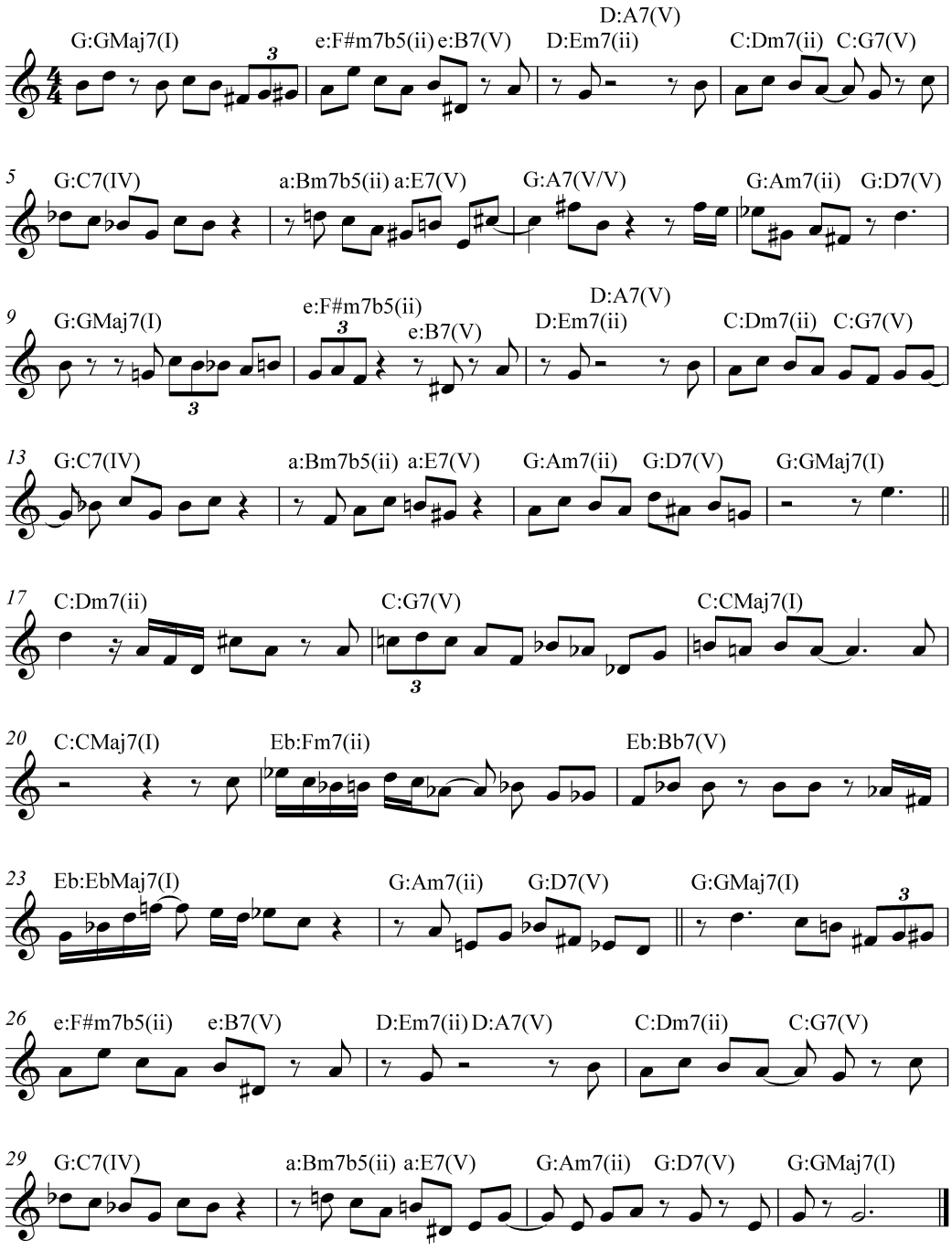
Anthropology

Charlie Parker



Confirmation

Charlie Parker*s*



C Final works

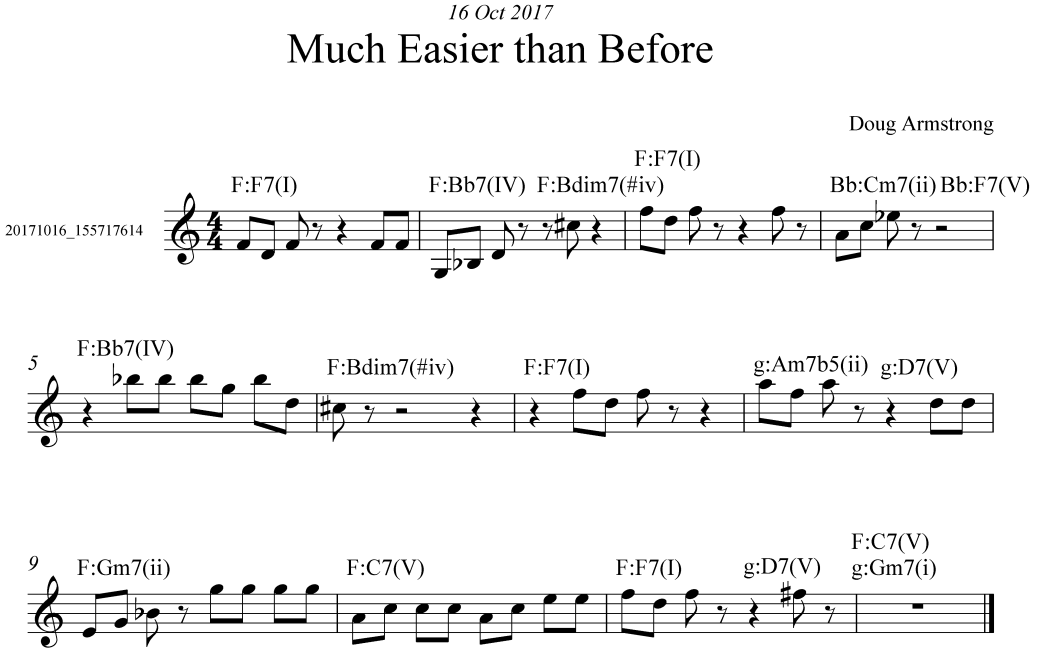


Figure 21: An ‘interesting’ jazz blues

16 October 2017

RhythmMashup1

Doug Armstrong

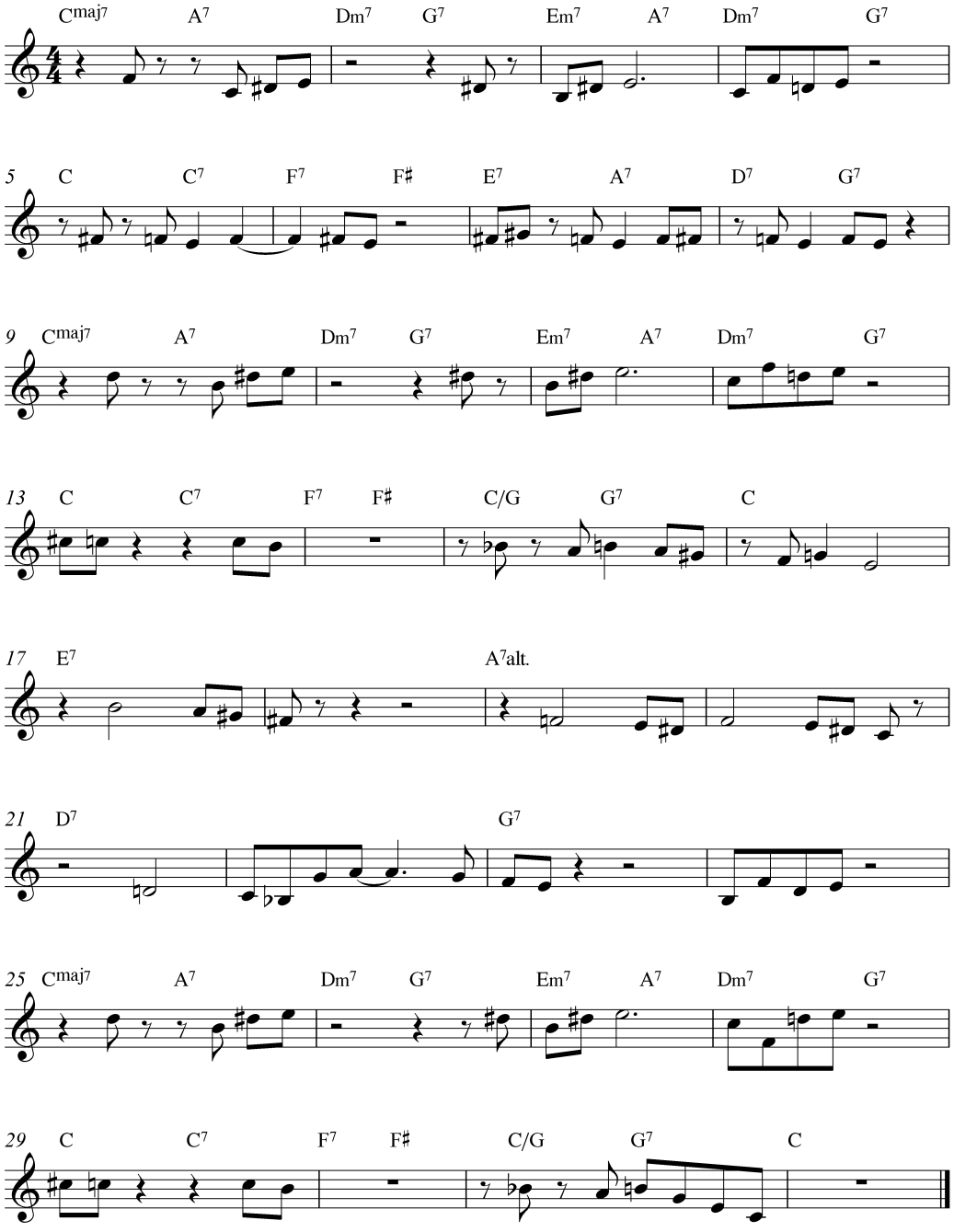


Figure 22: A ‘Rhythm Changes’ contrafact melody constructed from various outputs of the algorithm.

18 Oct 2017

Gypsy Swing Thing

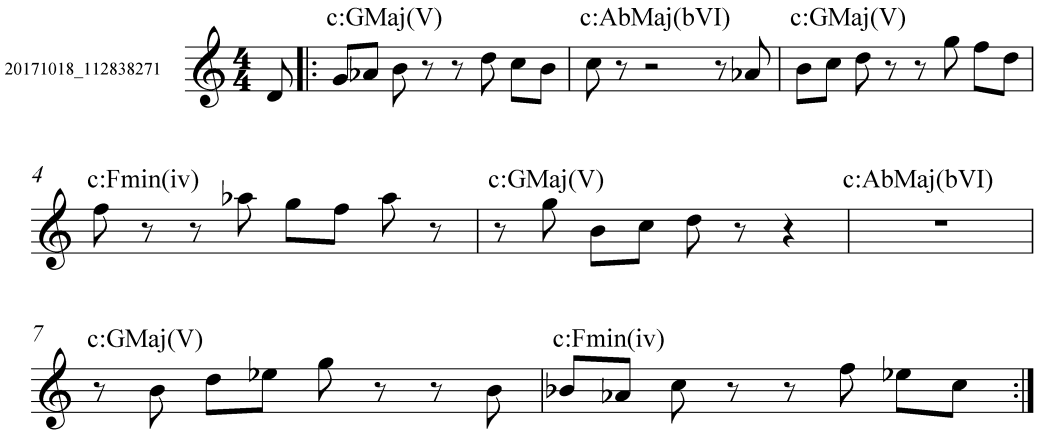


Figure 23: A melody I may propose as the beginnings of a new piece for my Balkan band.

18 Oct 2017

Play it fast

Doug Armstrong

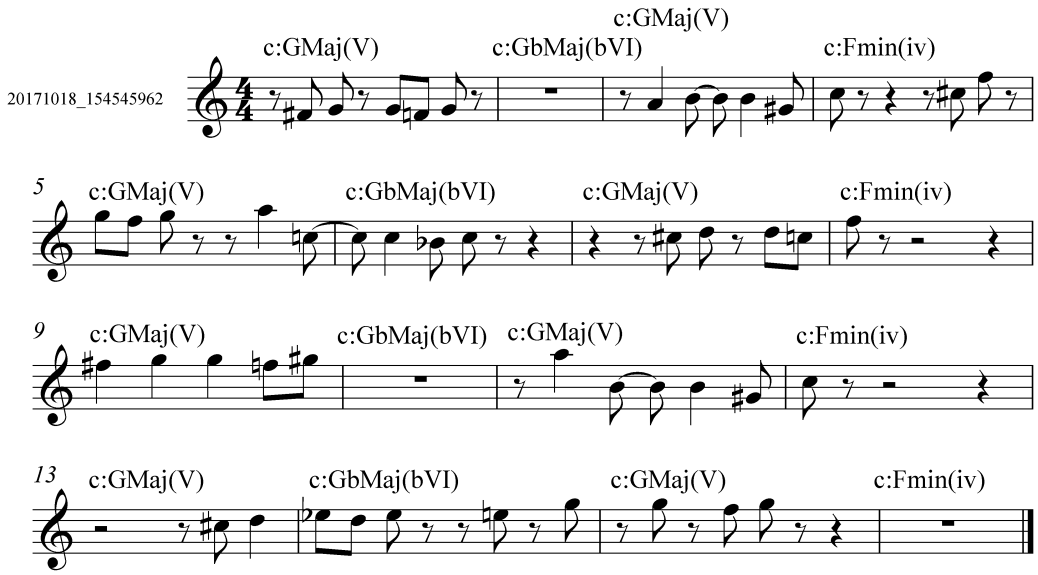


Figure 24: Another melody I may propose as the beginnings of a new piece for my Balkan band.