BlueJam: A heuristic-based approach to evolutionary music generation

Ciarán Rowe
Computing Laboratory
University of Kent
Canterbury, UK

E-mail: csr2@kent.ac.uk

Abstract

Visions of a comprehensive algorithmic composer have been pursued over the years. Recent attempts at this have incorporated evolutionary algorithms, a notion whereby populations of candidate solutions compete to make it to the next generation and be used as the seed for further solutions. These algorithms have performed well where the problem is well defined but only an approximate solution is achievable. BlueJam is a program written in Java and Pure Data, that implements algorithms inspired by evolutionary techniques, along with heuristics to guide the evolution in the hope of producing more salient output. We attempt to discern if this heuristic-based approach can improve upon previous efforts to generate music that is not purely deterministic.

1 Introduction

This paper investigates the application of heuristic structures to evolutionary techniques used for generating music. Generally speaking, heuristics can be used to provide meta-information to algorithms, in order to aid computation of a solution.

We describe the implementation of BlueJam, a program that experiments with the combination of these methods to provide a partially interactive environment in which to explore the evolution of note sequences. BlueJam is designed to evolve musical solos on the blues scale, so our heuristics will be attuned to this style.

1.1 Creativity and Aesthetics

Creativity in the domain of computation has been a point of contention among many researchers. Some prominent examples of discussions in this area can be found in literature by authors from a variety of backgrounds - from mathematics and physics (Penrose[27]), cognitive neuroscience (Hofstadter[11]) and computer science (Dreyfus[7]) to name a few. All are attempting to illuminate the grey areas of our grey matter that afford us this mysterious faculty of creativity.

Computational aesthetics have been unable to capture the capabilities of humans in the appreciation of the arts. Neural networks have shown potential in this area, by treating the judgment as a problem of fuzzy classification (Yang et al's Fuzzy k-NN[37]), but approaches like this still lack the essence of reflective judgement, remaining fairly inflexible beyond their training data. Flexibility after training can vary; in research by Spector and Alpern[32] a neural network was trained on samples of jazz musician Charlie Parker, and subsequently responded in a positive way when given musical samples with similar melodic properties (e.g. Hendrix, Coltrane and others), showing some sense of musicality.

Manifestations of creativity in nature tend to be taken for granted. We are often blind to considering creativity as the product of natural processes; for example, could a spider's web or patterns flower petals be considered the byproduct of natural creativity? Broader studies have shown a wealth of connections between nature and mathematics, investigations into constants such as the golden ratio[31] have strengthened these links and enticed further inquiry. Whatever form creativity appears in, "Beauty is in the eye of the beholder." It's fair to say that this old adage can lend some wisdom on the perception of these forms.

Barrow[2] discusses whether it is possible to work within the limits of formal systems and their derivatives (i.e. computation) to access the creative domain. Despite suggestions that this may be frivolous, additional insights from Hofstadter[11] propose that formal systems, mathematics and music may be intrinsically connected, and any formal system (i.e. grammar for music) that captures this will re-

flect those connections.

1.2 Computational Implications

It is common for composers to make observations of their internal creative processes, and follow those through with reflective comprehension and refinement of those processes. By shifting creativity into the computational domain, we must look toward imbuing our algorithms with this ability to become an "artificial art critic"[20]. We remain stuck with a recurring problem; "computation" as a process cannot adequately reflect upon itself. Evolutionary algorithms introduce the notion of the fitness function, allowing the computer to rank solutions that are produced while searching for an optimum result. The fitness function can be considered as an evaluation metric - a measure of how good the system is performing. However, this is still limited to an arbitrary formula, and weaknesses in the fitness ranking will be easily exposed.

Penrose depicts the astonishing faculties of Mozart[27] as a composer and artist, in that he was able to comprehend his creations in their totality from glancing at the score. This ability to read and judge music without hearing it played, throws a mysterious spanner into the algorithmic models we create. Since the conscious perception of this act (i.e. hearing the music) would take far longer than it does for such an individual to conceive, Penrose uses this to suggest that the phenomena of creativity escapes physical constraints. In computation however, constraints on hardware speed, complexity of the algorithm etc, are determinable and fixed. In this sense, complexity theory requires that computers take an order of time to "perceive" their creations, thereby governing performance of the algorithm.

With musicality and creativity being ill-defined in the computational domain, we must ask if their vague definitions present a problem. With the computer trying to achieve a pleasing melody, through convergence or serendipitous discovery, can this process (and the result) be considered a "solution"? Assigning the mantle of creativity or aesthetic value as an attribute of phenomena may be a dire categorical error, in the same way as labelling something characteristically musical a "solution" to the problem of creativity.

2 Overview

BlueJam provides a framework written in Java that implements a variety of evolutionary computation paradigms to generate musical solos over a given scale (by default, the blues scale). It is interfaced with Pure-Data[28] to provide access to MIDI and give real-time output. Inspired by the previous efforts of GenJam, written by John Al Biles[3] in the mid-90s, BlueJam uses special constructs called heuristic trees (see 6.3) to guide the evolutionary processes, along with a set of fitness functions, instead of evaluation by a trained neural network.[1]

Using BlueJam's representation of pitch and rhythm along with heuristic trees, guidance can be prescribed to the program with a preconfigured set of these heuristics, defining the shape of a melody. These heuristics also define how that shape may be transformed by the evolutionary processes (see section 6.3).

BlueJam is not intended to engender creativity, but to explore the possible recombinations of given heuristics, refining a melody in-situ, and perhaps adapting itself to the users after a reasonable number of iterations (see 7.6.2 on heuristic building)

2.1 Definition

It may be temerarious to call an algorithmically produced melody "composition", so for the purposes of this paper, "music generation" is referred to with the suggestion that there may be originality (or at least, randomness) in the algorithm, depending on the views of the reader.

2.2 Algorithms in Music Generation

Algorithms have previously been based around fixed and deterministic models. Key induction and chord detection[30] have been present in modern synthesizers since the '90s. These have been labelled inflexible an unoriginal. By altering these fundamentals to combine with evolutionary methods however, we can look toward a formula for generating music that uses contextual information to parameterize the composition process and be hopeful of generating salient output.

Jacob[12] provides a concise overview of a small system that composes using evolutionary algorithms. Improving upon this, Markov models and complex grammars have been extended by some authors[21] to re-shape the probabilistic element of the compositional process.

Recent research by Khalifa et al[18] produced a program that recombined a library of motifs using a grammar to generate music, with notable success. BlueJam is influenced by the idea of using a motif collection, extending this idea by mapping it onto a tree-based heuristic, which enables limited modification of motif phrases during the evolution.

3 Music Theory

While BlueJam is not deeply rooted in music theory, it takes cues from the author and the opinions given by a sam-

[1] This does not imply that neural networks are inappropriate for the task, nor that they should be excluded (see section 9.2 on Further Work)

ple of other musicians familiar with the art of improvisation.

3.1 Interviews

While developing BlueJam, some interviews were conducted as a precursor to building the program. In addition to the author's experience, the advice and insight of other musicians regarding the improvisational process was very useful in constructing a default set of heuristics, and assisting with the definition of fitness algorithms.

The idea of the heuristic tree is an affirmation that there are particular stylistic rules applicable to certain kinds of music, and that these styles are apprehended by musicians as they learn [24]. Much like a chess player will only see the legal moves given a board configuration, the musician manages to trim away all impractical maneuvers while improvising. This process may be considered as part of a search for the best musical move to make. Copley suggests that the influence in our search processes stems from unconscious habit and behaviour that defines these heuristics[9]. This doesn't preclude us from exploring insightful new direction, but we normally stick to what we know.

3.2 Modelling the Musician's Approach

The target of heuristics in the program is to act as an instructor or set of guidelines for the algorithm. This can be seen as modelling an evolutionary paradigm with "practice makes perfect" trial-and-error style learning, where we apprehend a new pattern by imitation. Nielsen[24] shows this to be an proven way of achieving musical proficiency.

By modelling the approach of being "taught" in this way, we remove the burden on the genetic algorithm (GA) of behaving autodidactically. The fitness function is also a hint to the algorithm, in the sense that it gives the computer some measure of aesthetics to use against the generated music.

4 BlueJam Goals

BlueJam has been developed to the point where it was sufficient for the purposes of the research conducted, although further investigation and extensions of the framework may yield more insight and better results.

4.1 Building on the Genetic Programming Method

The genetic programming paradigm centres around evolutionary strategies and algorithms. Tree-based genetic programming as inspired by Koza[19] is one of the main techniques used by BlueJam, a custom tree structure is implemented. Experimentation with newer structures such as Linear-Tree and Linear-Graph[15] have shown performance increases on certain problems, which evidences the suggestion that genetic programming works best when the underlying structures are customized to suit the problem being solved.

4.1.1 Additional Structure

BlueJam introduces a new structure (the heuristic tree, see 6.3) that can be used to guide music composition. This can be thought of as a heuristic, in the sense that it hints at the shape of the music, defining points where crossover and mutation can occur.

While we would still like to obtain a piece of good sounding music, we would rather not converge on a solution as if we were maximising our fitness criteria; this may lead to undesirable results in contrast with our goals[1]. The set of criteria we provide is vague and perhaps a little untenable - and deliberately so - due to its grounding in aesthetics. Instead of specifying fitness functions to check for discrete characteristics, we can design them to test for the presence of musical qualities.

4.2 Integrating Approaches

BlueJam uses a collection of different techniques to generate and evaluate note sequences. An evolutionary algorithm is employed, populating the fitness landscape with a number of individuals. It is common for interactive evolutionary algorithms to suffer a bottleneck, since it is not possible for a human to evaluate every member of the population. Prior work by Johanson and Poli[14] used automated fitness raters based around neural networks (NNs) to overcome this limitation. BlueJam follows in this approach, replacing a trained NN with the aforementioned heuristics and qualitative fitness functions to encourage positive development and exclude unfit individuals.

Grammar-based approaches not ignored in BlueJam. Research by Brooks et al[5] was an early demonstration of how Markov models could be tuned to a collection of musical pieces, and subsequently used to generate new music. BlueJam integrates arbitrarily tuned models by making selections of pitches from a given scale using a probability matrix (1-order Markov model, see 7.1.2).

5 Concepts and Components

5.1 Note Trees

Note trees are effectively the genotype of the evolutionary algorithm, since they define the structure that stores the solution. By using rhythmic equality (i.e. two quavers are a crotchet, two crotchets are a minim) we can apply a constraint to the number of children dependent on the time sig-

nature (e.g. in $\frac{4}{4}$ time, arity = 2), and this in turn ascribes a rhythmic semantic to the depth of the tree, such that the Rhythm R of a note at given depth d is:

$$R = \frac{1}{2^d} \tag{1}$$

The inverse of R allows us to calculate the depth at which rhythm R occurs in the tree:

$$d = \log_2 \frac{1}{R} \tag{2}$$

The phenotype, i.e. the actual appearance of the sequence, is given by the playing back leaves of the note tree. The *NoteTree* class implements a generic *NoteSequence* interface, so the genome implementation is not fixed to any particular data structure in the BlueJam framework (see 6.5 Modularity). Note trees are also used to implement heuristics (see section 6.3).

5.1.1 Note Trees & Crossover

By enforcing this rhythm semantic, we must also enforce homologous crossover between trees, i.e. nodes in the tree may only be swapped if they are at the same depth. This automatically takes care of balancing the note sequence length in the tree, since a crotchet can only ever be swapped with a crotchet, a minim with a minim (or equivalent), and so on.

5.2 Aesthetic Profiles

The set of models and heuristics that kicks off a particular cycle of evolution can be thought of as an aesthetic profile. Over time, this profile is what evolves into a collection of individual note sequences that will have a candidate for a new heuristic (see 7.6.2).

6 Implementation

6.1 Representation of Notes

BlueJam provides Java interfaces that are isomorphic with some of the basic concepts in GAs. We can illuminate some connection between terms in the genetic algorithm and the structures in BlueJam. The function set is primitive, containing only one abstract function in the current implementation, *play()*. Note trees provide this function by returning their terminal nodes, reading from left-to-right along the base of the tree.

A note is an instance of the *Note* class, which specifies all the basic properties. Further properties are added (such as property locking) by extending classes that represent terminals; the *NoteLeaf* class facilitates this.

6.1.1 Note Pitch

Pitch in BlueJam is represented by an enumeration, containing each of the pitch names (e.g. Pitch.C, Pitch.Cs, Pitch.Db etc), allowing representation of all pitch classes i.e. all white and black notes on a piano, excluding double-flats and double sharps.

Accidentals are treated as attributes and represented by another enumeration. BlueJam can evaluate each note in a given scale (see 6.2) at a given root pitch, and provide the correct accidental for that note (flat, sharp, or natural).

6.1.2 Note Rhythm

Rhythm is also represented in an enumeration supporting up to hemiquaver time divisions ($\frac{1}{64}$). Instances of the enumeration are named after their short rhythm names (e.g. Demiquaver, not Demisemiquaver).

Each note has an additional swing property, which can pair it with an adjacent note. This property can be used to assign a ratio to each note, sharing the rhythmic value of both notes. A percentile is assigned to each note, to make the rhythm syncopated or funky. Setting the share to 100% for a given note creates a note tie. This can also be used to effect dotted notation.

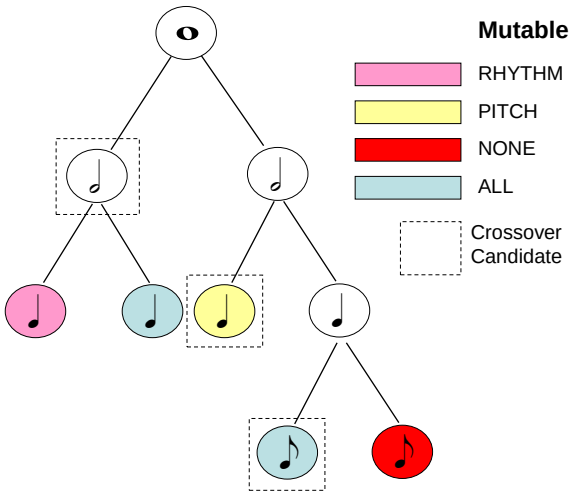
6.2 Representation of Scales

The main scale in BlueJam is the blues, a hexatonic (6-note) scale. Scale notes are labelled with roman numerals, so for the key of C, I = C, II = Eb, V = G etc. All scales (major, minor, modal) are represented as a set of intervals between pitches in the scale, i.e. the number of half-steps starting at the root pitch to progress to the next note, then from that note to the next, and so on. For the blues, that list is 3-2-1-1-3-2.

6.2.1 Chroma

Chroma (or chromatic notes) are the set of notes that do not strictly appear in a scale. Through conducting the interviews, it was clear that the majority opinion of chromatic notes in a solo was that they should be used with due caution. They have a high propensity to decrease the quality of the solo if not used in the right places, and as such are used sparsely. The heuristics in BlueJam use a "clustering" approach to chroma on the blues scale, which imitates a playing technique of the author. Chroma is only programmed to occur in the heuristics between two scale notes at a fast pace, for example, the run of C-Eb-E-F-G shows a chromatic note that is clustered between two other pitches on that scale.

Figure 1. Crossover Points



6.3 Representation of Heuristics

Heuristics are note trees that have interpretable elements. These elements (pitch, accidentals etc) are defined relative to runtime dependent properties, the root pitch or scale for example. "*Runtime*" is analogous to the point in the program where the sequenced music is played to the user.

A simple instance of a heuristic can be a *NoteTree* with *NoteLeaf* nodes of relative pitch (*Pitch.R*, see 6.1.1). This relativity allows the heuristics to outline the shape of a melody line by defining the pitch relative to the root note at runtime. At runtime, a validation function is used to make properties of heuristic instances absolute, replacing the relative pitches and octave numbers with MIDI note values (see 5.1 for NoteTree implementation details).

6.3.1 Locking the Heuristic

We can execute the customary operations of crossover and mutation over the heuristic, while retaining musically meaningful operations (transpose, increase rhythm etc). In order to maintain melodic shape, the heuristic limits these operations by specifying which properties of a given terminal are mutable. BlueJam then calculates which nodes are eligible for crossover (see figure 1). This is a non-greedy search, so if rhythmic properties are mutable, then the parent of that node will be returned as a candidate for crossover - in contrast, a greedy search would return all nodes under a candidate down to the terminals.

This operation allows us to lock certain qualities into a heuristic, for example, musical resolution, i.e. finishing back at the root note. Research by Katz[16] demonstrates how qualities like this may be intrinsic to the pleasure we experience from music.

Figure 2. Note (.heuristic) Descriptor

```
i(0)r(QUAVER)s(0)p(R,0)m(0)re(0)o(4);

# i  - A unique numerical identifier
# r  - Rhythm of the note
# s  - Swing (swingRatio [, partner i])
# p  - Pitch (pitch [,relativeSteps] )
# m  - Mutablity (enumerated value)
# re - Rest (boolean)
# o  - Octave (int)
```

Figure 3. Example Model (.m) Definition

```
#    C     Eb    F     Fs    G     Bb
C  0.10  0.25 0.175 0.05 0.175 0.25
Eb 0.25  0.10 0.25  0.05 0.175 0.175
F  0.175 0.20 0.10  0.20 0.15  0.175
Fs 0.05  0.05 0.425 0    0.425 0.05
G  0.175 0.175 0.15 0.20 0.10  0.20
Bb 0.25 0.175 0.175 0.05 0.25  0.10
```

6.3.2 .heuristic format

The format of this file describes the properties of terminals in the tree. A parenthesised parameter (fig. 2) is available for every element describing a note, including it's pitch type, swing value and swing partner (the note it is paired with), the rhythm and mutable properties of the note. The example in figure 2 shows an immutable root note of quaver length, in the 4th octave.

6.4 .m format

BlueJam uses a 1-order Markov model to achieve a better-than-random initialization strategy.

In figure 3, we can see the layout of the model is fairly simple. The .m file expresses a matrix of probabilities, effectively a Bayesian $P(A|B)$, where B is the note already in the sequence and A is the subsequent note. Each row in the table should add up to 1.0.

6.5 Modularity

BlueJam's architecture is designed to be modular and pluggable. The root interfaces and abstract classes (*NoteSequence*, *Note*, *Function*, *Terminal*), do not have to be implemented by the tree genome (as per the default). Other systems may choose to use string representation. It is even feasible to speculate the use of a MIDI representation that implements these; at some point the nodes could be output to a MIDI file. This could enable interoperability with other

algorithmic music generators that produce output or accept input as MIDI files.

Key algorithms used for selection, initialization, and fitness are implemented as singleton classes, and populations can choose to change algorithms used on a per-individual basis. Populations and individuals can use any of these at runtime. To implement a new algorithm for selection, fitness or initialization, the user would only have to write the class and update the relevant enumeration for that type to make the algorithm available to BlueJam.

7 Algorithm Design

7.1 Initialization Functions

Comparisons drawn by Brown[6] in his juxtaposition of rule-based and genetic algorithms suggest that using musically meaningful mutation is a better way of encouraging the emergence of musical patterns, as opposed to the application of conventional evolutionary techniques (for example, in Manzolli's Vox Populi[23]). Using heuristics by design, BlueJam already has a musical "clue" to begin with, and our initialization stems from there. We imitate tree growing, as specified by Koza[19], adding a few variations and some initial mutation.

7.1.1 Growing Trees

Tree growth is described as the addition of functions and terminals to a tree. When initializing the population, most individuals are assigned a heuristic. Those without a heuristic are grown from scratch. The functions in our tree are defined implicitly, so BlueJam checks which part of the tree is empty and executes the grow method on these regions, randomly selecting terminals at different rhythmic values until the tree is full; some of these may be rests.

After growing, we also choose a few nodes to mutate. Unlike the two-tiered implementation of Jacob[12] (choosing and layering motifs to make phrases), we already have motifs, so we just flesh them out and shuffle some properties.

7.1.2 Markov Models

Markov Models are used in BlueJam to guide the initialization process. There is disagreement among researchers as to the compositional merit of the output of such models. Järveläinen[13] states that they have little value, due to their tendency of generating high correlations with the training data. However, they are robust and easy to compute, and appear in the earliest attempts at computational composition[36].

These models have been examined thoroughly by prior research. McCormack [21] explored the representation of Markov models as non-deterministic grammars. Conversion into a grammar condenses the model representation and allows for chance selection of transformation rules. This technique has been explored by subsequent research; Papadopoulos[26] mentions several further developments in the application of grammar-based approaches to algorithmic music.

During initialization, BlueJam attempts to fill out any missing notes with either a rest or a new note. When making a new note, the pitch of that new note can be chosen based on the content of our model. BlueJam examines the note sequence prior to the new note, and selects a Pitch from the model based on this information.

7.1.3 Rhythm Initialization

Rhythms are initialized using uniform probability weights to discourage the selection of very long or very short notes. The current implementation is biased toward selecting quavers most of the time. With each new rhythm, there is also a probability of adding "swing" to that note.

7.2 Fitness Functions

BlueJam has a number of fitness functions that are designed to reward or penalize particular qualities in evolved note sequences. Some of the justifications for these functions are related to musical theory or the experience of the author, and some are based around prior research into fitness measurements for evolutionary music.

Fitness measurements in BlueJam are normalized between 0-1. In extreme and borderline cases, candidates that have negative fitness or fitness > 1 are rounded.

Evolutionary algorithms are commonly applied as optimization algorithms. To re-iterate the problem faced here, we cannot easily define what "optimization" is in music, therefore optimization is not necessarily the global goal of these functions; it is to guide the evolution toward good qualities.

7.2.1 Interval Fitness

This fitness measurement is a simple method of rating the coherence of a solo. Concurring with the author, it was established in the interviews (see 3.1) that good soloing technique rarely employs large jumps over scales. Wiggins et al[35] and others have imposed this limitation in their fitness criteria, favouring stepwise progressions that move smoothly over single notes.

BlueJam measures this by examining the note sequence to discover if there are any intervals of width > 3 half-steps. These are classed as large intervals, and are summed to a

variable l. We also wish to penalize repetition in the melody, so we sum successive occurrences of the same interval to another variable r if we repeat more than a certain threshold (by default, 3 repetitions).

Functions 3 and 4 produce two similar curves that normalize our fitness to a value between 0-1 where r or $l > 0$. Function 4 reduces exponentially, providing a harsher judgement of fitness. The ideal candidate in 3 (with no penalties), normalizes to $\frac{1}{\ln 2}$ that later gets rounded to 1. In 4 the ideal candidate normalizes to 1.5, which also gets rounded to 1.

$$f(x) = \frac{1}{\ln\left(2^r + 2^{\sqrt{l}}\right)} \tag{3}$$

$$f(x) = \frac{3}{2^r + 2^{\sqrt{l}}} \tag{4}$$

7.2.2 Contour Fitness

Miranda[22] explores the musicological approach of Todd[33], evolving entities with musical behaviour. A characteristic described in this approach is the melodic contour. A melody has contour if any sequence of notes consistently go in a single direction over a period of time, for example, the sequence C-D-E-F has an upward contour.

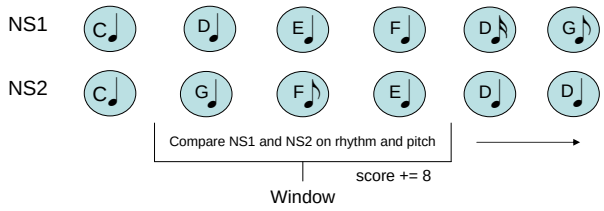
BlueJam implements a simplified measurement of fitness based on melodic contour. In our variation, we look for melodic contour by sampling two points in the note sequence passed to the fitness function. These samples are used to ascertain whether the sequence has a contour at both points. The function returns an arbitrary fitness value depending on the combination found. The most diverse combination receives the highest fitness value.

- Up + Up OR Down + Down = 0.5
- Up + Down OR Down + Up = 1.0
- Up + None OR Down + None = 0.7
- Unknown + None OR None + None = 0.0
- Unknown + Up OR Down = 0.3

7.2.3 Δ Heuristic Distance

This fitness function is somewhat contentious since it relies on the duration of the evolved phrases and heuristic to be around 4 bars in length to accurately classify the fitness. When evaluating shorter phrases, the candidate can score a moderate fitness value by being identical to the heuristic, which we are attempting to avoid. This could be greatly improved in future versions of the program by parameterizing the fitness function with the lengths of the heuristic and generated note sequence.

Figure 4. Distance Comparison



The difference between the individual, and the original heuristic that generated it, provides a good measure for separating candidates that have changed a reasonable amount from those that have undergone too many or too few changes. Initially this is measured by correlating each note in a candidate note sequence with the notes in the original heuristic. We search using a window of comparison along these sequences, every similarity found in the window adds 1 to the score (see figure 4).

The process in figure 4 gives us a linear measure of how similar two note sequences are, in this case how similar an individual is to the heuristic it is based on. The highest similarity scores fall above 150, where the compared sequences are virtually identical.

To level this out, in function 5 we take the natural logarithm of the differential score and apply it to the fitness function to obtain a normalized fitness value in the range -1 – 1 (subzero values are rounded to 0).

Function 5 describes a curve (see figure 5) with a rising front followed a steep downward turn that should cut off all individuals too similar to their parents, returning 0 accordingly.Through trial and error, the best candidates were shown to have a score between 20–60; these are afforded a higher fitness value by function. Individuals that deviate too much (< 10) will end up with a negative or negligible fitness score.

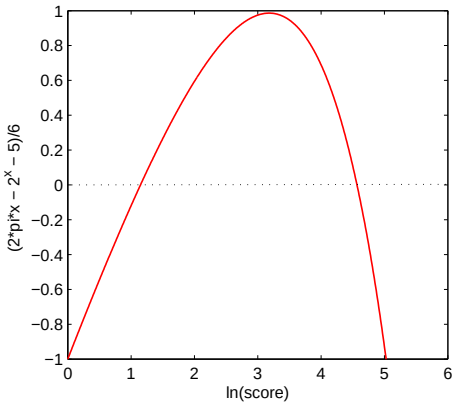
$$f(\ln x) = \frac{2\pi x - 2^x - 5}{6} \tag{5}$$

7.3 Stacked Fitness

This special function incorporates all previously outlined methods to evaluate individuals in a multi-objective manner. The combined fitness value assigned to the individual is a weighted sum of the fitness values returned from each function:

- Intervals (50%)
- Contour (25%)
- Heuristic Distance (25%)

Figure 5. Fitness curve (distance)



7.4 Testing the Fitness Functions

The default set of heuristics were pitted against the defined fitness functions, to affirm that they scored highly. In all cases the heuristics scored over 0.6. This assurance informs us that a heuristic-based sequence will dominate in most generations, so the selected candidate should almost always reflect the melody of one of the heuristics.

7.5 Selection Functions

We implement a variant of tournament selection to choose winning candidates that pass into the next generation.

7.5.1 Tournament Variation

The tournament is divided into two rounds. It's likely that if a candidate does not rate well on the intervals, it will also not rate well on the other criteria. We use negative selection in the first round to eliminate a portion of the individuals; those with the lowest fitness have the highest possibility of losing. This has a complementary effect on the selection pressure, which is set by the tournament size[34].

In the second round, we commence regular tournament selection, picking the fittest individual from the remaining bunch and adding it to our collection of winners. The tournament is carried out n times, where n is the number of individuals in the mating pool for the next generation.

7.5.2 Parameter Tuning

In some cases, this selection scheme produced good results for the first few iterations, but the heuristics would quickly die out of the population, and the program would converge to almost identical patterns that bore little resemblance to the original heuristics. The following changes were made to fix this:

Selection pressure was decreased by increasing the tournament size in order to increase the likelihood of exploring potentially interesting although less fit candidates that would otherwise not be selected. Wieczorek[34] suggests this approach while discussing selection strategies.

Population coherence was maintained by re-inserting the heuristics every generation as if they were elites, along with the tournament winners from the previous generation.

7.6 Adaptation

7.6.1 Elitism and User Feedback

Pure Data provides an interface with available MIDI devices. BlueJam can use a MIDI controller to receive feedback on the quality of it's output. Each time positive feedback is received on a note sequence it is marked as elite, and re-inserted into subsequent generations. This interactive element bridges the fitness gap, knowing that aesthetic judgement has not been adequately represented in our computational model.

7.6.2 Heuristic Building

It is feasible that at any point in the evolution, BlueJam can select an evolving candidate and transform it into a heuristic. Much like the way musicians discover new patterns or phrases that are good to use (see 3.1), BlueJam can output a given note sequence into the .heuristic format (see 6.3.2) and create new individuals based upon it in later generations.

7.7 Pure Data Interface

BlueJam provides an interface to the evolution using the graphical programming language, Pure Data (PD). BlueJam interfaces with PD using the Java plug in for Pure Data, PDJ[10]. The evolution is executed in a separate thread, and the results of this are sent to PD. PD's front end allows the user to kick off the evolution and hear the output in real-time over MIDI. The user can then give feedback if desired, by nudging the pitch wheel on their MIDI controller.

8 Results and Discussion

Samples of BlueJam's output were played to a few critical participants for evaluative feedback. While the output

of the program showed the influence of the heuristics, the results were not completely satisfactory. Most runs were unmusical in their contingency, a likely outcome of BlueJam's lack of global awareness, but despite this there were snippets of usable material.

8.1 Phrase Evolution

Some samples (numbered 1-10) were extracted from the runs of BlueJam to illustrate some of the best output and musical qualities. Most of these samples were evolved in the key of C, at a tempo of around 120bpm (beats per minute). Some were evolved in the key of F at a slightly faster tempo. Each run used a different combination of the default heuristics. A brief overview of these is given.

Samples 1, 2 and 3 The feel of the heuristic is clearly present in the output, most samples managed to show this. After a while however, subsequent runs did not always reflect their original heuristic. Sample 2 shows musical resolution, which was present in most runs.

Sample 4 Following the same heuristic to sample 2, this shows an improvement over the rhythmic feel and and pitch progression in the solo.

Sample 5 This sample shows the use of chroma. Depending on the context, this could sound good or bad, but the program is not sufficiently equipped to know this.

Sample 6 An interesting run started producing "call and return" style motifs, which were not programmed into the heuristics. This may have been a little fortuitous, since it was not reproduced on a subsequent run with the same set of heuristics.

Samples 7 and 8 Showed good examples of two heuristics combined.

Sample 9 One of the best musical runs overall, showing progression, staccato phrasing, and resolution.

Sample 10 Musical and progressive, but does not resemble any particular heuristic, showing how the evolution surpasses the original heuristic specification.

8.2 Interesting Behaviour

None of the heuristics included any chroma, but BlueJam was able to sparsely incorporate some interesting combinations in the test runs (as in sample 5). One run[29] of the program saw the evolution of a modal cadence that was not present in any of the heuristics. It is interesting to consider how the program identified and preserved the emergence of other musical characteristics, whether by sheer luck or coincidence, this kind of output is interesting.

A few surprising behaviours were also noticed in unrecorded runs. In one instance, the program was observed evolving an ornamental style during a demonstration; the program started playing trills in its solo, but no such styles were coded into any of the heuristics.

8.3 Limitations

8.3.1 Representation

There are many limitations in the current representation of notes in BlueJam[2]. Most articulation and dynamics (ornaments, slurs, grace notes) are not supported at all. Neither are triplets or other complex note relationships (e.g. portamento). This could be vastly extended within the current framework to provide comprehensive coverage.

8.3.2 Rhythm and Sequences

BlueJam has no indication of global context in the current implementation. This implies that although it is aware of local changes in pitch and rhythm, it cannot keep track of accentuations, the passing of time or other metadata about the piece. Lacking this is somewhat debilitating, but stripping most of these extraneous dimensions has helped this research focus on the raw effect of the heuristics.

Building a layer of metadata about the note sequences (for example style, rate of swing change, note relationships etc) may prove very useful in developing the system further as has been achieved in works like Özcan and Erçal's AMUSE[25].

8.3.3 Formalistic Limitations

Gödel's incompleteness theorem[8] has far reaching implications for formal systems, including any that might choose to represent a musical format say, in the form of a grammar. It is intriguing to speculate what we can say about the capability of grammars as creative tools, given Gödelian insight. What we might suggest is that there will always be something musical, that cannot be captured by any given formal system of music.

9 Conclusions and Further Work

We have explored the evolution of note sequences using heuristics combined with a collection of methods inspired by genetic algorithms and the work of previous research. Ample extensions could be made to the current version of the program, to improve it's performance and bring more domain-specific knowledge to the driving algorithms.

[2]Further documentation regarding note representation is available in the project corpus

In terms of performance, most of the runs produced output that was not globally coherent. It was however, pleasing to observe the algorithms re-arrange chunks of music into sometimes interesting and novel combinations. Also, the program does not inherently support polyphony, so it can currently only evolve individual musical lines. This would be a worthwhile extension to the program, given more time.

The interactive element of BlueJam (see 7.6.1 suggests it is possible to embed human subjectivity in searching algorithms that do not exist in a purely computational domain, i.e. those that involve some element of aesthetic judgement. This is encouraging, since making artistic decisions on a purely analytical basis leaves something to be desired.

Strong AI would expose appreciation of art or computational "thinking" as just another type of classification algorithm. Dreyfus[7] outright refutes that we might be able to engender insight through the rigidity of algorithms. Having created a small number of musically interesting phrases using a hybrid of techniques, this hopefully contributes to showing that we may be able to outdo these limitations, employing computation to help reflect upon and understand music - and perhaps other phenomena that escape analytical boundaries.

9.1 Practical Applications

Synthesizer manufacturers are constantly looking for new ways to wow their customer base. Embedding a successful generative evolutionary program onto a chip could facilitate a "jamming" functionality. The user could be playing accompaniment to a musical piece while the keyboard itself joins in to fit that particular style of play, allowing original improvisations to emerge.

9.2 Program Extensions

The potential for tweaking the fitness and selection functions in BlueJam is extensive, for example implementing fitness sharing, niching, co-evolution. Additionally, the the default set of heuristics can be modified to any configuration that suits the user of the program.

Additional extensions to BlueJam could include:

- A separate sequencing component to keep constant tempo and time signatures.
- Control via other MIDI input - for example, a footswitch, to turn the jamming on and off.
- Graphical visualization of tree-building in the evolutionary composition process.

It's feasible that with a bit of polishing and some further development it could function as a tool for computer-assisted composition outside of it's intended purposes in research

Particle Swarm Optimization[17] is another method that could be explored in the application of music generation. Research by Blackwell[4] has shown promise in this area, and it's possible that some adaptaion of heuristics could also be employed in conjunction with this approach.

9.3 Final Remarks

If computers ever did have a good representation of aesthetics, we would expect to find that not all of them agreed about the merit of particular works. When we speak about an artwork, we use a language that can describe all the nuances of that piece, and we are constrained to our own immanent, internal representation of that work, which is irrevocably linked to our opinions about it. We must investigate if, by transferring to the computational domain, we can preserve those endemic qualities that afford subjective judgement.

Given the aims of some research in building an algorithmic composer, it's worth considering that evolution has already carved out better and better "solutions" for music composers, and has provided the mold for the model solution in us - an organic one. While we attempt to imitate using these building blocks, we must consider what we are building with, since our imitations will be inherently limited by the components we use.

Acknowledgments

The author would like to thank Dr. Colin Johnson for his guidance and support throughout this project.

References

[1] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 14–21, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

[2] J. D. Barrow. *Impossibility: the limits of science and the science of limits*. 1998.

[3] J. Biles. Genjam: A genetic algorithm for generating jazz solos, 1994.

[4] T. Blackwell. Swarming and music. *Evolutionary Computer Music*, pages 194–217, 2007.

[5] F. Brooks, A. Hopkins, P. Newmann, and W. Wright. An experiment in musical composition. *IRE on Trans. Electronic Computers EC*, 6(1):175–182, 1957.

[6] A. R. Brown. An aesthetic comparison of rule-based and genetic algorithms for generating melodies. *Org. Sound*, 9(2):191–197, 2004.

[7] H. L. Dreyfus. *What Computers Still Can't Do: A Critique of Artificial Reason*. MIT Press, Cambridge, MA, USA, 1992.

[8] J. R. N. E Nagel. *Gödel's Proof, revised edition.*, pages 109–113. New York University Press, 2001.

[9] A. Gartland-Jones and P. Copley. The suitability of genetic algorithms for musical composition. *Contemporary Music Review*, 22:43–55(13), Number 3/September 2003.

[10] P. Gauthier. Java plug-in for pure data. *Available http://www.le-son666.com/software/pdj/ (Last Accessed: 16 Mar 08)*, 2008.

[11] D. R. Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books, 1979.

[12] B. Jacob. Composing with genetic algorithms, 1995.

[13] H. Järveläinen. Algorithmic musical composition. April 2000. Available: http://www.tml.tkk.fi/Studies/Tik-111.080/2000/papers/hanna/alco.pdf, (Last Accessed: 16 March 2008).

[14] B. Johanson and R. Poli. GP-music: An interactive genetic programming system for music generation with automated fitness raters. Technical Report CSRP-98-13, University of Birmingham, School of Computer Science, May 1998.

[15] W. Kantschik and W. Banzhaf. Linear-tree GP and its comparison with other GP structures. In *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 302–312, Lake Como, Italy, 18-20 Apr. 2001. Springer-Verlag.

[16] B. F. Katz. Musical resolution and musical pleasure. In *Proceedings of the AISB-93, Nineth Biennial Conference*, pages 209–18, Amsterdam, 1993. IOS Press.

[17] J. Kennedy and R. C. Eberhart. *Swarm intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[18] Y. M. A. Khalifa, B. K. Khan, J. Begovic, A. Wisdom, and A. M. Wheeler. Evolutionary music composer integrating formal grammar. In *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, pages 2519–2526, New York, NY, USA, 2007. ACM.

[19] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA, 1992.

[20] B. Z. Manaris, P. Roos, P. Machado, D. Krehbiel, L. Pellicoro, and J. Romero. A corpus-based hybrid approach to music analysis and composition. In *AAAI*, pages 839–845. AAAI Press, 2007.

[21] J. McCormack. Grammar based music composition, 1996. Available: http://www.csse.monash.edu.au/ jonmc/research/ Papers/L-systemsMusic.pdf (Last Accessed: 16 March 2008).

[22] E. R. Miranda. At the crossroads of evolutionary computation and music: Self-programming synthesizers, swarm orchestras and the origins of melody. *Evol. Comput.*, 12(2):137–158, 2004.

[23] A. Moroni, J. Manzolli, F. V. Zuben, and R. Gudwin. Vox populi: evolutionary for music evolution. *Creative evolutionary systems*, pages 205–221, 2002.

[24] S. G. Nielsen. Learning strategies in instrumental music practice. *British Journal of Music Education*, 16(3):275–291, 1999.

[25] E. Özcan and T. Erçal. A genetic algorithm for generating improvised music. In *Proceedings of the Evolution Artificielle (EA'07) Conference*, 2007.

[26] G. Papadopoulos and G. Wiggins. AI methods for algorithmic composition: A survey, a critical view and future prospects. In *Proc. AISB'99 Symp. Musical Creativity*, pages 110–117, 1999.

[27] R. Penrose. *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics (Popular Science)*. Oxford University Press, March 1999.

[28] M. S. Puckette et al. Pure data. *Available http://puredata.info (Last Accessed: 16 Mar 08)*, 2008.

[29] C. Rowe. run8.mp3 (00:02:00). *BlueJam Corpus*, 2008.

[30] R. Rowe. *Interactive music systems: machine listening and composing*. MIT Press, Cambridge, MA, USA, 1992.

[31] S. K. Sen and R. Agarwal. Golden ratio in science, as random sequence source, its computation, and beyond. *Computers and Mathematics with Applications*, 2007.

[32] L. Spector and A. Alpern. Induction and recapitulation of deep musical structure. In *Proceedings of International Joint Conference on Artificial Intelligence, IJCAI'95 Workshop on Music and AI*, Montreal, Quebec, Canada, 20-25 1995.

[33] P. M. Todd. Simulating the evolution of musical behavior. *The origins of music*, pages 361–388, 2000.

[34] W. Wieczorek and Z. J. Czech. Selection schemes in evolutionary algorithms. In *Proceedings of the IIS'2002 Symposium on Intelligent Information Systems*, pages 185–194. Physica-Verlag, 2002.

[35] G. Wiggins, G. Papadopoulos, S. Phon-Amnuaisuk, and A. Tuson. Evolutionary methods for musical composition. *International Journal of Computing Anticipatory Systems*, 1998.

[36] I. Xenakis. *Formalized Music: Thought and Mathematics in Composition*. 1971.

[37] Y.-H. Yang, C.-C. Liu, and H. H. Chen. Music emotion classification: a fuzzy approach. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*, pages 81–84, New York, NY, USA, 2006. ACM.