Techniques for Algorithmic Composition of Music

Adam Alpern
Hampshire College
aalpern@hampshire.edu

Fall, 1995

Abstract

This paper is an overview of various techniques used by the author over the last 3 years in the production of computer-generated music. It covers work using semi-random noise, chaotic dynamical systems, contour analysis, and evolutionary programming.

1 Introduction

1.1 A Brief Overview of Automated Composition

The area of automated composition refers to the process of using some formal process to make music with minimal human intervention. This is often done on a computer, with the aid of various *formalisms*, such as random number generation, rule-based systems, and various other *algorithms*.

The nineteenth century mathematician Ada Lovelace, colleague to Charles Babbage, inventor of the "calculating engine", the precursor of computers, had this to say about the possibilities of automated composition:

> "Supposing, for instance, that the fundamental relations of pitched sound in the signs of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent" ([7], p. 4).

1.1.1 Randomness

The simplest form of automated composition involves using random numbers to piece together melodic fragments. One very early example of this is the *Musikalisches Würfelspiel* (Dice Music) attributed to Wolfgang Amadeus Mozart. The Dice Music game involved assembling a number of small musical fragments, and combining them by chance, piecing together a new piece from randomly chosen parts.

Chance has also played a part in the compositions of more modern composers. John Cage has made notable use of chance in his works. His piece *Reunion* is performed by playing chess on a photo-receptor equipped chessboard. The players' moves trigger sounds, and thus the piece is different each time it is performed. Cage has also used natural phenomena to compose music (his *Atlas Eclipticalis*, 1961, composed from astronomical charts), as has Charles Dodge, with his piece entitled *The Earth's Magnetic Field* (1970), for which a computer translated fluctuations in the magnetic field of the Earth into music.

1.1.2 Hiller, Isaacson and Baker

Lejaren Hiller (with the help of Leonard Isaacson and Robert Baker) was one of the first to successfully utilize a computer to compose music. In 1957 he used the Illiac computer to compose the *Illiac Suite*. Unlike the work of Iannis Xenakis (see p. 3), who used computers as an aid to human composition, Hiller and Isaacson attempted to simulate the compositional process itself on the computer. They employed a generator/modifier/selector approach to simulate the composition process. This approach uses some technique to generate "raw materials" as a compositional base, then applies various techniques (such as permutation and geometric transformations) to further manipulate the generated material, and then applies selection rules to choose suitable material to become a composition.

1.1.3 MUSICOMP

MUSICOMP [1] was one of the first systems for automated composition. Written in the late 1950s and early 1960s by Robert Baker and Lejaren Hiller, it consisted of a library of subroutines for the Illiac computer. The routines in MUSICOMP used the generator/modifier/selector paradigm to compose music seen in earlier work which produced the *Illiac Suite*. The author's program *Contour* uses this paradigm to compose music based on $1/f$ noise. (see p. 4).

MUSICOMP was first used by Hiller and Baker to realize the *Computer Cantata*, which they used to demonstrate MUSICOMP's flexibility

[1]MUsic Simulator Interpreter for COMpositional Procedures

and generality. Since it was written as a library of subroutines, it made the process of writing composition programs much easier, as the programmer/composer could use the routines within a larger program that suited his or her own style. This approach has recently been used by the author in organizing various works (see p. 4 and p. 7) into a library of compositional formalisms. This approach, that of building small, well-defined pieces and assembling them together, is an approach well suited to automated composition. Large, monolithic systems tend to suffer from a lack of flexibilty, a lesson taken from work in the field of AI expert systems. By building small parts with well defined behaviour and linking them together, we can great a great variety of methods and compositional output.

1.1.4 Iannis Xenakis

Iannis Xenakis is probably the best known name in the use of computers for music. Xenakis used computers to aid in the composition of scores for live ensembles, using statistical and probabilistic methods. His "stochastic music program" [31] would "deduce" a score from a list of note densities and probabilistic weights supplied by the programmer, leaving specific decisions to a random number generator. *Morsima-Amorsima*, for 4 instruments, was composed in this manner.

Xenakis' work was strongly influenced by the aesthetics of mathematics, and also by earlier works in serial polyphony, which he strongly criticized—"Linear polyphony destroys itself by its very complexity; what one hears is in reality nothing but a mass of notes in various registers The enormous complexity prevents the audience from following the intertwining of the lines and has as its macroscopic effect an irrational and fortuitous dispersion of sounds over the whole extent of the sonic spectrum" [24]. Xenakis wished to have a language for manipulating music derived from mathematics and logic. It is perhaps this concern for higher aesthetics which is why Xenakis' work, while fascinating and thought-provoking, has not been a direct influence on the authors' work, which has been more concerned with exploring the compositional process.

2 Specific Works

2.1 $1/f$ Noise

In 1978, Richard F. Voss and John R. Clarke conducted an experiment in which they analyzed the spectral density of the audio data from many forms of music and speech. Clarke and Voss analyzed spectral densities down to a frequency of $5x10^{-4}Hz$, which allowed them to characterize the fluctuations of a melody over large lengths of time, thus taking into account large scale structure in the samples. They found that many forms of music exhibit a behaviour of fluctuation that varies as $1/f$, where f is the frequency, a characteristic also found in many naturally occurring phenomena [2].

Based on this result, Voss and Clarke played examples of computer-generated melodies based on $1/f$ noise, brownian noise, and random (white) noise to several hundred people, with the general consensus amongst the listeners being that the melodies based on $1/f$ noise were far more interesting than the others. Clarke and Voss then suggest that $1/f$ noise may be a means of adding the correlations and structure found in music over a wide range of times, something which purely random, or white, noise is incapable of doing, since it lacks any long-term correlation over time. In the next section, we will see how $1/f$noise can be used as a raw material for composition.

2.2 Contour

Contour is a program, written in Common Lisp, which generates new melodies by expanding existing melodies based on an analysis of their contour. The significance of pitch-contours in human perception of music has been proven by [16], and , [20], both of which showed that people will often mistake an unknown melody for one they have heard before if the contour of the melody is similar to one they know. The means of analyzing the contour is derived from work by P. N. Johnson-Laird ([19]). Johnson-Laird used a means of encoding a melody by the ups and downs of its notes as part of his hypothesis on how jazz bassists improvise a bass line.

The encoding represents a tune by a series of symbols: S denotes the first note in the tune, U denotes a note which is higher than its predecessor (an up interval), D denotes a downward interval, and R denotes a repeat of the previous note. The actual size of the interval is not noted, merely the direction (see figure 1).

[2]The full list, quoted directly from Voss and Clarke, is: vacuum tubes, carbon resistors, semiconducting devices, continuous or discontinuous metal films, ionic solutions, films at the superconducting transition, Josephson junctions, nerve membranes, sunspot activity, and the flood levels of the river Nile.

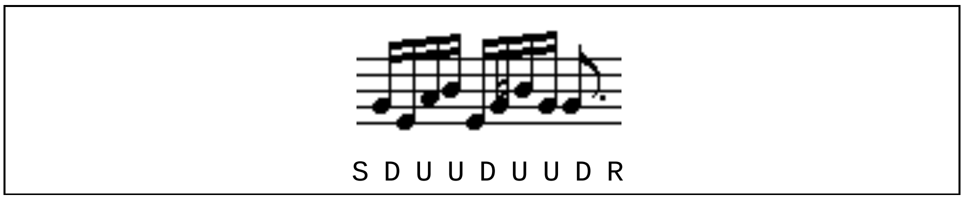

Figure 1: *A melodic fragment from Charlie Parkers Yardbird Suite and its contour encoding.*

2.2.1 Overall and Meta Contours

The Contour program extends the contour concept in two ways. The first is the *overall* contour, the second is the *meta-contour*. The overall contour is a measurement of whether a melody seems to be rising or falling. It is calculated simply by counting the number of ups and downs in the contour. If there are more ups than downs, the overall contour is said to be up, and vice-versa.[3] A meta-contour is represented the same way as a melodys contour, as a series of intervals, but it is constructed from the overall contour of sub-segments of the melody. For instance, one could break up a 12 bar piece into 1 bar segments, and generate the meta-contour from the overall contours of each individual bar (see figure 2).

To generate new melodies, the Contour program encodes a melody in the manner described above, then performs a number of steps to create a new melody by expanding the original. The steps are:

1. After encoding the original melodys contour, determine if the contour, on the whole, has the characteristic of rising or falling.

2. Use each note of the melody as the starting note for a new melody, which must satisfy the constraint of having the same overall quality of going up or down, as the first note did in the original melody. In other words, if an up note is used as the starting point, the new melody as a whole should go up.

3. Append the new melody fragments together to form a whole melody, which will contain the original melody, interspersed with new material, and whose meta-contour will be the same as the original pieces contour.

[3]This is a very poor heuristic for determining the overall ascending or descending quality of a melody. A first improvement would be to take into account the relative positions of the first and last notes, as well as the number of ups and downs.

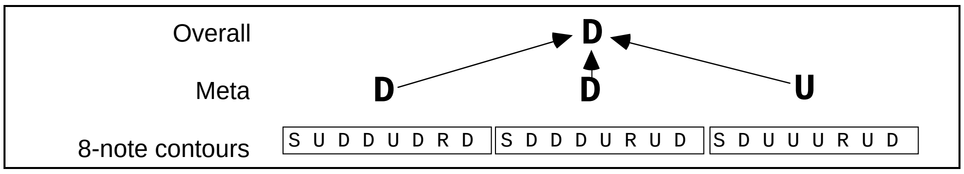

Figure 2: *An illustration of the contours of a 24-note melody, broken into 8-note segments. At the highest level, the overall 'D', this piece has been judged to be descending, since 2 of its' 3 segments are descending*

A few other operations are performed on generated melodies as well. First, they are constrained to a diatonic key, which is either supplied by the user, or guessed at, by assembling all the unique different pitch-classes[4] ([14]) of the original melody, and comparing these to a table of known scales, and choosing the scale judged to be 'most similar' to the collection of pitch-classes making up the melody. Experimental work with the program used melodies generated from $1/f$ noise, as well as purely random noise, as the raw compositional material, using the dice-casting algorithm described in [6]. (for example - in step 2 above, the new melodies being generated are being generated by $1/f$noise, although the original melody may be supplied by a human being.) The greatest success with the program, has been when using $1/f$ melodies as input to the expansion process. As noted earlier, $1/f$ noise may be used to create musical material which is pleasing to the ear, but, since it depends ultimately on random sources, there is no large scale structure, such as repetition of a theme ([23]). However, when put through this process of expansion, initial $1/f$ melodies begin to take on some structure. Since the new melodic fragments are generated and thrown out until they meet the criterion of the new piece having the same meta-contour as the original pieces contour, we begin to see some movement to the piece in its pitch space. While we do not yet have repetitions of a theme, we do have a piece which has some sense of going somewhere, rather than merely wandering aimlessly.

2.2.2 Implementation

Contour is implemented as a library of functions in the Lisp programming language. The current implementation was done in Macintosh Common Lisp, but (with the exception of the user interface) uses pure Common Lisp,

[4]As an example, the note C represents a *pitch class*, which encompasses all C notes, while middle C is a specific *pitch*.

and thus is extremely portable to other systems. Output is in the form of standard Type 1 MIDI Files. There is also an experimental, interactive version of Contour. This version allows a musician to interact with the program via real-time MIDI input. The program listens for melodic fragments of a user-defined length, then uses those melodies to compose a response, based on the original melody augmented with $1/f$ noise-based fragments. In its' current form, the program is too slow to support more than rudimentary interactions, and as such is a tentative first step in the direction of interactive algorithmic composition, but one that is worth investigating further.

2.2.3 Future Directions

Contour touches very lightly into the area of automated musical analysis, and the little analysis is does is not very sophisticated. Specifically, the areas of movement in a melody (whether the melody is ascending or descending), and key-determination. [18] is an excellent example of what can be done with automated analysis of music. Contour could also be extended to include more primitives for building structure out of sequences of notes, similar to those found in such music oriented languages as Canon [10], and Common Music [28], both of which are also built on top of the Lisp language. However, it is the authors' opinion that the area of automated musical analysis that holds the most exciting potential, for providing quantified stylistic information which can be used in the generation of new works (for example, as a culture in the system described below, in the section entitled Techniques for Evolving Computer Musicians).

2.3 Nonlinear Dynamical Systems and Chaos

In recent years, the behaviour of systems of nonlinear dynamical equations when iterated has generated interest into their uses as note generation algorithms [25, 13, 17, 5, 23]. The systems are described as systems of mathematical equations, and, as noted by Bidlack and Leach, display behaviours found in a large number of systems in nature, such as the weather, the mixing of fluids, the phenomenon of turbulence, population cycles, the beating of the human heart, and the lengths of time between water droplets dripping from a leaky faucet.

In order to produce data which can be used musically, these systems of equations are put through a process of iteration, whereby the solution is calculated, and then fed back into the equations to be the input values for the next iteration. The solutions represent a point in n-dimensional space (the

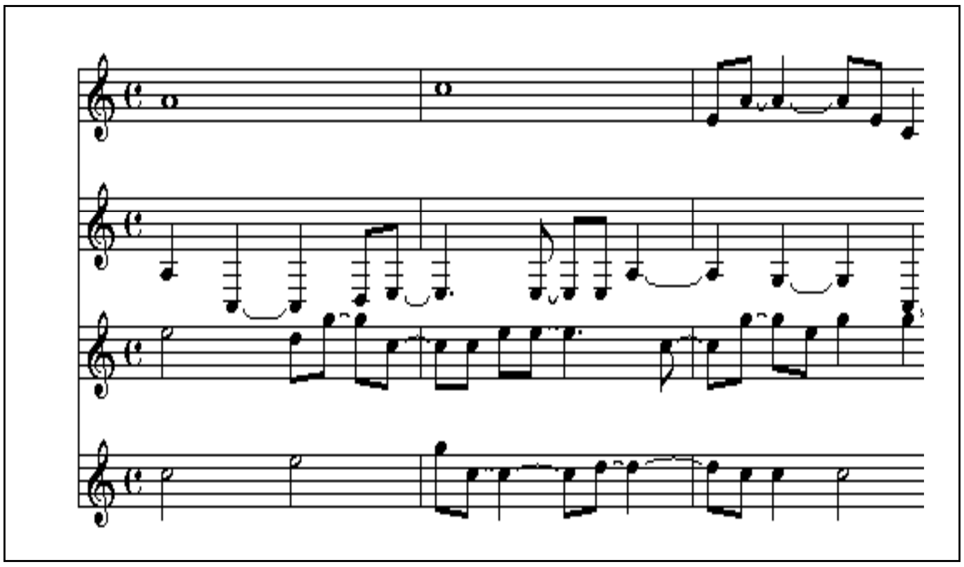

Figure 3: *The beginning of* rwIV*, a piece composed from 2 orbits of the Henon system and 2 orbits of the Standard Map.*

n being determined by the number of variables). The set of these solutions over time is referred to as an orbit of the system. There are three categories of behaviour into which these systems can fall upon iteration, often referred to as attractors, since they represent the set of points to which the values of the system tend. They are: constant, where all points in the orbit are the same; oscillatory, where all points belong to a repeating set of points; and chaotic, or strange attractors, where the orbit behaves seemingly randomly, never visiting exactly the same point twice. The reader is referred to [12] for a more in-depth discussion of nonlinear dynamics.

Obviously, constant behaviour will produce no interesting musical results. Oscillatory behaviour has the possibility of producing interesting repetitions, if the period (the number of distinct points in the repeating set) is large enough, but in practice the set is usually rather small, resulting in cyclic melodies that circle between only a few pitches. It is this last behaviour that holds the most musical interest.

The attractor of a chaotic orbit defines a certain range of values which the system will wander amongst, often returning to similar, but not quite the same series of points. When interpreted musically, this can often be seen as variations on a theme. This is what makes chaotic orbits attractive as generators of musical material. The material produced has a high degree

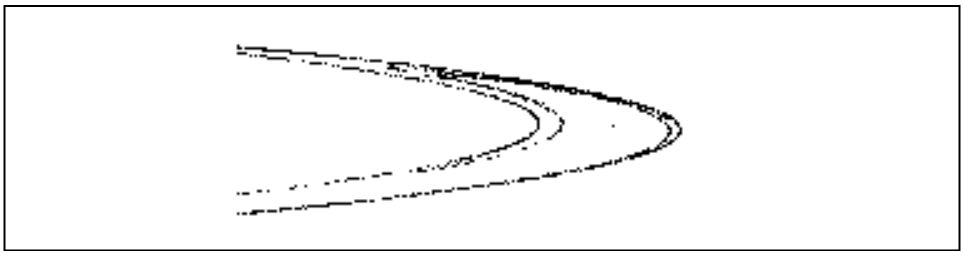

Figure 4: *An orbit of the Hénon system.*

of correlation with its past, but is always producing something new at the same time. Of course, it is important to note, as Rick Bidlack does, that the music produced by such systems is probably best considered as raw material of a certain inherent and potentially useful musicality

2.3.1 The Hénon Map

$$x_{n+1} = y_{n+1} - Ax_n^2$$
$$y_{n+1} = Bxn$$

The Hénon Map is a simple 2 dimensional system, a simplified model of a stars orbit around a gravitational center, discovered by the French astronomer Hénon. While he mad to meticulously calculate and plot thousands of points to arrive at the picture you see above, which depicts the strange attractor which the system is capable of producing.

This system (and others like it), can be implemented very simply in code as an iterative (or recursive) function. Here is an example of the above Hénon system in the language Dylan [5]:

```
define method henon ( x , y , num )
      let a = 1.4, b = 0.3, new-x, new-y;
             for ( i from 0 to num )
                     play-note ( x, y );
                     new-x = y + 1.0 - ( A * ( x * x ) );
                     new-y = B * x;
                     x := new-x;
                     y := new-y
             end for
end method;
```

[5]Dylan is a registered trademark of Apple Computer Inc.

2.3.2 The Logistic (Population Model) Map

$$x_{n+1} = ax_n(1 - x_n), 0 < a \leq 4$$

The Logistic Map equation is a simple, one-dimensional equation that is used to model population growth over time, and has been used extensively as the basis of [25]. This particular equation, though simple, provides a very rich set of behaviours for different values of the parameter a when iterated. For all values of a between 1 and 0 , the system converges on the point x = 0. For all values of a between 1 and 3, the variable x will converge on the fixed point 1 - 1a for all initial values of x. For values of a above 3, there exist cycles where all values of x will cycle between 2, 4, 8, and so on, points. This is called a harmonic cascade, and leads to a point where, for certain values of a, the behaviour of the system is chaotic. These values of a which produce chaotic behaviour are the ripest areas for interpretation as music.

2.3.3 Implementation

Currently, a program in pure ANSI C, labeled Chaosmuse, has been implemented. The user specifies what equation systems, parameters, and musical parameters to use through a text-based, interactive interface. Chaosmuse then generates music and outputs it into several standard file formats, including MIDI Types 1 and 0, Adagio, CSound score file format, and mtr2, a simple format devised by Lee Spector for input into the MAX program. It will run on any machine with an ANSI C compiler. There is currently rudimentary support for reading of parameters from an input file.

2.3.4 Future Directions

The Chaosmuse program will be expanded to allow larger-scale decisions about a composition to be implemented. In it's current state, the music it produces tends to wander aimlessly, with key shifts happening only by accident, if at all. This tends to create a "washing out" effect, with pieces losing the listeners' interest after only a few minutes. Adding larger scale compositional decisions into the process, such as choosing a phrase for later repetition, transposition, and key movement, will greatly increase the interest of the music produces. These higher-level musical decisions will likely be made by a rule-based algorithm. The chaotic routines will also be packaged as a library of subroutines, for easy inclusion in other programs.

2.4 Techniques for Evolving Musicians

2.4.1 Genetic Programming

Genetic programming is a very recent technique for automatic programming of computers. Developed by John Koza ([21]), it is a method which actually uses a process of artificially-created natural selection to evolve simple computer programs. In order to perform this process, one uses a small set of functions and terminals, or constants, to describe the domain one wishes an evolved program to operate in. For example, if the human programmer wishes to evolve a program which can generate or modify music, one would give it functions which manipulate music, doing things such as transposition, note generation, stretching or shrinking of time values, etc . Once the functions have been decided on, the genetic programming system will create a population of programs which have been randomly generated from the provided function set. Then, a fitness measure is determined for each program. This is a number describing how well the program performs in the given problem domain. Since the initial programs are randomly generated, their performance will be very poor - however, a few programs are likely to do slightly better than the rest. These will be selected in pairs, proportionate to their fitness measure, and then a new population of programs will be created from these individuals, and the whole process will be repeated, until a solution is reached (in the form of a program which satisfies the critic), or a set number of iterations has passed. Operations which may be performed in generating this new population include reproduction (passing an individual program on into the next generation unchanged), crossover (swapping pieces of code between two parent programs in order to create two unique children), mutation, permutation, and others.

Genetic programming is a very powerful technique for generating computer programs, as one does not need to know in advance how complex the solution need be, nor even what the best strategy for attacking the problem is. One merely need provide the building blocks and the critic for judging the final output, and let the genetic programming system search the space of possible computer programs in an attempt to maximize fitness.

In [26], we attempted to create a framework for evolving artists on the computer, by providing a cultural context for the evolved artist to utilize, along with the critic. The hypothesis here is that any artwork exists within a rich cultural and historical context, and it is rare for an artist who is ignorant of that context to produce acceptable works. Thus, by providing access to a cultural context, we hoped to produce constructed artists capable of producing works worthy of merit on their own. Harold Cohens Aaron, is perhaps the most well-known example of constructed artist, and one of the few clear successes. Aaron is a computer program which creates original drawings, which have been exhibited in galleries internationally. Aaron was constructed and tutored over more than a decade by Cohen, himself an accomplished painter.

2.4.2 Evolving a Bebop Musician

We illustrate the framework with a system that creates simple programs that produce Bebop jazz melodies. Jazz melody is a good medium for this sort of experimentation for several reasons. First, there are several simple ways to represent melodies in a form that is manipulable by simple programming constructs. Second, the jazz tradition includes several call and response forms, so the idea of creating a new work on the basis of an old work has established precedents within the genre. Third, the jazz literature contains several analytical works that enumerate critical criteria (e.g., [Coker, 1964]), along with many works on technique that provide guidance in creating a function set (e.g, [3]).

Thus, in the framework of genetic programming, what the system produces are simple programs which take a simple jazz melody as their input, and create another simple jazz melody as their output, which is sent to the fitness function to be evaluated.

2.4.3 Architectural Advantages

In many systems for automated generation of artworks (be it music, line drawings, or what-have-you), the cultural context is implicit in the rules programmed in to the computer. For example, this authors work with non-linear dynamical systems as musical generators took raw numerical output from a mathematical system, and forced it into a diatonic scale. This forms part of the programs "culture" available for generating artworks. However, since it is coded into the program, we have no way of making it produce, say, microtonal music, without significantly rewriting the program. While this may seem a trivial point, it illustrates how limitations in what a compositional program can produce can come about—by building in assumptions.

A better approach is "factor out" the context in the same way that the critic is factored out by the genetic programming framework. If we provide such cultural information, we should be able to produce, and judge the success of, programs evolved to create works in different styles, merely by varying the basic tools available, and the body of works for output to be judged against. For example, the system should be able to produce different results when given a library of Mozart as context, than when given a library of Jazz.

In current work with this framework, the provided culture has been a case-base of past, highly-valued works (in this case, melodies from tunes by Charlie Parker). While there is much more to cultural context than a mere library of past works, this provides a good, and necessary, starting point.

2.4.4 The Computational Critic

For a simple problem with a well-defined solution, such as performing symbolic regression (the process of deriving a function to produce a set of data, given that set of data), creating the critic is easy - it merely has to calculate how close each data point generated by a program is to the target set. For a complex domain such as music, where judgments must be made about what is good music, something which is very hard to quantify, creating this critic can be a very difficult process.

Our simple critic assesses the fitness of an individual program by running it with each melody from a case-base of Charlie Parker melodies as input. Each of these runs produces a new melody as a response to the input, and the sum of the assessment of each of these melodies is returned as the fitness of the program. The critic tests melodies on the basis of five critical functions drawn from criteria presented in ([3]). TONAL-NOVELTY-BALANCE returns a perfect score if there is a perfect balance between novel tonal material, and tonal material which can be found in the case-base. RHYTHMIC-NOVELTY-BALANCE works in the same way, but it compares rhythmic structure of the melodies against melodies in the case base, rather than tonal structure. TONAL-RESPONSE-BALANCE compares the notes of the program's output with those of its' input, note by note, and returns a perfect score if there is a balance between equal and inequal notes. SKIP-BALANCE returns a perfect score if there is a balance between diatonic movement and intervallic leaps. RHYTHMIC-COHERENCE reutnrs a perfect score if there is no "stuttering" in the melody, that is, if there are no sixteenth or shorter notes occurring between longer notes.

Another element to take into account when constructing a critic is the ability of the genetic programming system to exploit weaknesses in the fitness function (the critic). For example, one of the criterion our simple critic looks for is a balance of recognizable material and novel material. Thus, a simple program which does nothing but quote the first half of its input, then for the remainder of the new melody, intersperses notes from the input (or from the case-base) with random notes (or even with silence) has a good chance of satisfying the critic. Clearly, this is a weakness in the critic.

2.4.5 Future Directions

Further work is being done to enhance the music critic component, since this is the weakest link of the system. The reader is referred to [27] for the first attempt at building a connectionist jazz critic. The worked described their is being further extended.

3 Conclusion

> "Since I have always preferred making plans to executing them, I have gravitated towards situations and systems that, once set into operation, could create music with little or no intervention on my part. That is to say, I tend towards the roles of planner and programmer, and them become an audience to the results" - Brian Eno ([9], p. 5).

This quote represents part of the authors' attitude towards automated composition. The works presented here have tended to be the sort of system in which the programmer sets up the parameters and algorithms and then says "Go!", letting the system produce a composition (or a musician) with little to no intervention or modification of the final result by programmer/composer. These systems often provide excellent means of testing theories about the compositional process on a computer. For example, the genetic programming experiments tested the theory that an artist exists within a context, and meaningful judgements about the quality of genreated artworks cannot be made without access to that context (see p. 11).

The synthesizer and the computer have both played a large role in freeing composers from traditional constraints. Modern synthesizers allow pieces written for full orchestra to be realized and experienced without the hassle of finding and commisioning an orchestra to play them. Computers can free the composer from note-by-note composition, allowing him to explore theories of music and composition as a process, without worry for the actual notes.

References

[1] Charles Ames. Automated composition in retrospect. *Leonardo*, 20(2):169–185, 1987.

[2] D. Baggi. *Readings in Computer Generated Music.* IEEE Computer Society Press, 1992.

[3] David. Baker. *David Baker's Jazz Improvisation.* Alfred Publishing Co., Inc., 1988. Revised Edition.

[4] K. E. Balaban and O. Laske, editors. *Understanding Music with AI: Perspectives on Music Cognition.* MIT Press, Cambridge, 1994.

[5] Rick Bidlack. Chaotic systems as simple (but complex) compositional algorithms. *Computer Music Journal*, 16(1):33–47, 1992.

[6] Tommaso Bolognesi. Automatic composition: Experiments with self-similar music. *Computer Music Journal*, 7(1):25–36, 1987.

[7] Edmund Bowles. *Musicke's Handmaiden: Or Technology in the Service of the Arts.* Cornell University Press, 1970.

[8] J. Coker. *Improvising Jazz.* Simon and Schuster, Inc, New York., 1964.

[9] David Cope. *Computers and Musical Style.* A-R Editions, Madison WI, 1991.

[10] R. B. Dannenberg. The canon score language. *Computer Music Journal*, 13(1):47–65, 1989.

[11] Peter Desain. Object oriented style, part i: Lisp as a second language. 1995.

[12] Robert L. Devaney. *Fractal Patterns Arising in Chaotic Dynamical Systems.* Springer-Verlag, New York, 1988.

[13] A. DiScipio. Composition by exploration of nonlinear dynamical systems. In *Proceedings of the 1990 International Computer Music Conference.* International Computer Music Association, 1990.

[14] Charles Dodge. Profile: A musical fractal. *Computer Music Journal*, 12(3):10–14, 1988.

[15] Charles Dodge and Curtis R. Bahn. Musical fractals. *Byte*, 666:185–196, 1986.

[16] W. J. Dowling. Scale and contour: Two components of a theory of memory for melodies. *Psychological Review*, 85:341–354, 1978.

[17] Michael Gogins. Iterated function systems music. *Computer Music Journal*, 15(1):40–48, 1991.

[18] Henkjan Honing. Poco: an environment for analysing, modifying, and generating expression in music. In *Proceedings of the 1990 International Computer Music Conference*. International Computer Music Association, 1990.

[19] P. N. Johnson-Laird. *Jazz Improvisation: A Theory at the Computational Level*, pages 291–323. Academic Press Limited, 1991.

[20] M. R. Jones. *Dynamics of Musical Patterns: How do Melody and Rhythm Fit Together?*, pages 67–92. Academic Press, London, 1993.

[21] John R. Koza. *Genetic Programming*. The MIT Press, Cambridge MA, 1992.

[22] Otto Laske. A conversation with marvin minsky. *AI Magazine*, 14(1):31–45, 1992.

[23] Jeremy Leach and John Fitch. Nature, music, and algorithmic composition. *Computer Music Journal*, 19(2):23–33, 1995.

[24] Gareth Loy. *Current DIrection in Computer Music Research*, chapter Composing with Computers—a Survey of Some Compositional Formalisms and Music Programming Languages. The MIT Press, Cambridge, 1989.

[25] Jeff Pressing. Nonlinear maps as generators of musical design. *Computer Music Journal*, 12(2):35–45, 1988.

[26] Lee Spector and Adam Alpern. Criticism, culture, and the automatic generation of artworks. In *Proceedings of the Twelth National Conference on Artificial Intelligence, AAAI-94*, 1994.

[27] Lee Spector and Adam Alpern. Induction and recapitulation of deep musical structure. In *Proceedings of the IJCAI-95 Workshop On Artificial Intelligence and Music*, 1995.

[28] Heinrich Taube. Common music: A music composition language in common lisp and clos. *Computer Music Journal*, 15(2):21–32, 1991.

[29] Richard F. Voss and J. R. Clarke. $1/f$ noise in music: Music from $1/f$ noise. *Journal of the Acustical Society of America*, 63(1):258–263, 1978.

[30] Iannis Xenakis. *Arts-sciences, alloys.* Aesthetics in Music. Pendragon Press, New York, 1985. thesis defense.

[31] Iannis Xenakis. *Formalized Music. Thought and Mathematics in Composition.* Indiana University Press, 1991.