An International Journal of Music Technology
Organised Sound
Cambridge University Press

The Analysis of Generative Music Programs

Nick Collins

Organised Sound / Volume 13 / Issue 03 / December 2008, pp 237 - 248
DOI: 10.1017/S1355771808000332, Published online: 03 November 2008

The Analysis of Generative Music Programs

NICK COLLINS

Department of Informatics, University of Sussex, Falmer, Brighton, BN1 9QJ, UK
E-mail: N.Collins@sussex.ac.uk

Composers have spent more than fifty years devising computer programs for the semi-automated production of music. This article shall focus in particular on the case of minimal run-time human intervention, where a program allows the creation of a musical variation, typically unravelling in realtime, on demand. These systems have the capacity to vary their output with each run, often from no more input information than the seeding of a random number generator with the start time. Such artworks are accumulating, released online as downloads, or exhibited through streaming radio sites such as rand()%. Listener/users and composer/designers may wish for deeper insight into these programs' ontological status, mechanisms and creative potential. These works are challenging to dissect; this article makes a tentative start at confronting the unique problems and rich behaviours of computer-program-based generative music, from the social and historical context to the backwards engineering of programs in relation to their sound world. After a discussion of exemplars and definitions of generative art, strategies for analysis are outlined. To provide practical examples, analyses are provided of two small scale works by James McCartney.

1. INTRODUCTION

Algorithmic composition is a familiar territory in computer music, explored since the early mainframe era of *Push Button Bertha* (1956) and the *Illiac Suite* (1955–59) (Ames 1987), where programs to produce a few musically relevant symbols would run overnight. Taking the example of Gottfried Michael Koenig, *Project I* (1964) would output a list of discrete note and instrumentation instructions, derived from probabilistic serialist procedures, which had to be hand-transcribed into a musical score for performance (Koenig 1989; Laske 1989). With the backdrop of such labour-intensive work and the technological benefits of the oft-quoted Moore's Law, it is not surprising that most contemporary explorations favour realtime systems for speed of feedback, a development that also supports complex interactive possibilities (Chadabe 1997; Rowe 1993). Yet systems have also been deliberately devised for the limiting case of minimal intervention in performance. In the asymptote, a human designer makes all decisions pertaining to the potential of the musical device in advance. A single starting trigger might cause the seeding of a random number generator with the current time, allowing the program to progress on a new path with associated musical output revealed in realtime.

It is hoped that this article will promote understanding of contemporary art practice in algorithmic music. There is also some potential that tackling the limiting case of non-interactive algorithmic systems (which shall themselves present many rich problems) can contribute to an understanding of more general interactive music systems. This research may assist composers employing such mechanisms who yearn for greater insight into the relation of process and product. Some care must be taken with any claim for the existence of an independent discipline of 'analysis'; yet analysis, even in the most practical senses, is necessitated so that greater control of musical outcomes can be exhibited in algorithmic systems. Such analysis also points to the assumptions of current systems and potential of future systems, and satisfies innate human curiosity regarding such artefacts.

With these aims in mind, it is fruitful to spend a while unpicking the various strands of debate and terminology, for the existing literature is not always consistent, as detailed below. The reader has probably already picked up on a tension between the terms 'algorithmic' and 'generative', and the mention of process and product may raise questions concerning the compositional responsibility, social context and analogies to, say, improvisation of such work. Ultimately, the issue of where the music itself resides – implicit in an executable program, explicit in a given 'play' – forms an overarching theme of discourse on such musical objects.

After treating terminology, examples of particular generative music systems will be given that help to clarify the objects under discussion. With an awareness of the human prevalence to categorisation (Lakoff 1987) it is helpful to consider central exemplars, and avoid delimiting hard boundaries around categories; overly prescriptive taxonomies deny the continuums of real practice, though they can form helpful tools as long as we bear their claims to full description as approximate. The article then proceeds to introduce some particular analytical methodologies that may prove useful in discussing and learning from algorithmic systems. These are applied in the final stages of the paper in treating two necessarily smallscale examples; there is enough to discuss even in these cases to demonstrate the scope and challenges of such work.

2. DEFINING GENERATIVE AND ALGORITHMIC MUSIC

In a recent seminar, Margaret Boden outlined a taxonomy of electronic and interactive art devised with Ernest Edmonds (Boden 2007). In their terms, *G-art* (generative art) is composed of works that are 'generated, at least in part, by some process that is not under the artist's direct control'. A further sub-category of *CG-art* refers to specifically computer-generated work, 'produced by leaving a computer program to run by itself, with minimal or zero interference from a human being'. Phil Galanter (2006) is also inclusive of non-computer-based work in his discussion of generative art. The rich artistic heritage underwriting this stance in the specific case of music might pay tribute to more than forty years (perhaps counting from Riley's *In C* of 1964) or fifty years upwards (perhaps counting from Earle Brown's *December 1952*) of process, system and indeterminate music. These musical movements themselves parallel conceptual and new media art historically, and homage must be given to Sol LeWitt's famous assertion that 'the idea becomes a machine that makes the art' (LeWitt 1967). Conceptual art's currency of often illogical and unmechanisable concepts, instantiated by human beings, is replaced in much digital generative art by well-defined (computable) concepts enacted via machine; yet the human originator is ever present, even if acting by proxy.

Any definitional complexity in this debate is essentially due to the rise, ubiquity and indeed indispensability of computers in contemporary (digital) art practice. Computers simply require formalisms, and must be programmed with exact instructions, that is, *algorithms* (taking the spirit of the computer science definition of an algorithm rather than admit any less rigorous art critic's notion in using such a term). However, in seeking to differentiate their art (often for publicity) some artists have adopted *generative music* (Eno 1996) as a more rigid description of *algorithmic music that happens to produce output in realtime*. 'Good Old Fashioned Algorithmic Composition' might therefore be distinguished as an offline tool (Ames 1987; Loy 2006; Pearce, Meredith and Wiggins 2002), providing suggestions as a composer's assistant, following rules for the production of musical scores or recordings and otherwise not making so central the algorithmic process itself in the final product.

It must be admitted then that generative music as discussed in recent academic literature (Collins 2003; Cox and Warner 2004; Eacott 2000, 2006; Eno 1996), and in online communities such as generative.net, may actually often refer to computer-generated algorithmic music which happens to be realtime in production, a subcase of both generative music and algorithmic music as more widely defined in art theory and computer music. Analytic work described in this article can profitably engage with any algorithmic music generator. In Pearce, Meredith and Wiggins's (2002) terminology of automated composition research, all works would primarily fall under the remit of 'algorithmic composition' for the sake of art.

Thus, although we have seen that definitions of generative art would not restrict themselves to computer programs alone, programs provide a standard exemplar, where the operation of the artwork is free of complicated human intervention during execution. This pure *computer-generated music* (*CG-music*) will be tackled in the remainder of the article, and we shall refer to generative music programs or algorithmic music where the connection to computation is clear. Thus, *human computation* (von Arn 2006) is barred, so that we also exclude instances of algorithmic art such as live coding, interactive genetic algorithms or text music for human interpretation as outside the remit of the current investigation. The hope, however, is that any insight into the relation of algorithm to musical production will be helpful once live human beings are reintroduced.

Human work has not been entirely sidestepped; we must always accept the precursor of human design. In philosophical terms, generative computer programs are examples of the *derivative intentionality* of writing (Searle 2004: 20), where the code is predetermined by a human author who then yields moment to moment autonomy of execution to the machine. Human intervention is thus reduced to initial conditions alone; control is usually choosing when to start the program, providing the seed for the random number generator. Programs are in principle well-defined; it certainly complicates matters to have human beings sitting inside deciding on moves, as within the famous chess-playing Turk, and there will be enough difficulties already even when composers have specified each action in iron-clad code.

The final hurdle before we progress to consider real examples is to mention the process versus product debate. Art itself is replete with examples of the claims of a human being for an object's 'artistic status' leading to the grant of that status and a transformation of the bounds of art. Some might see this cultural negotiation as having gone to breaking point in the 20th century. A cogent and passionate defence of the status of a generative program rather than an individual production as an artwork has already been presented by Ward and Cox (1999) in discussing Adrian Ward's *Autoshop* application.

In a very practical sense, that musicians are choosing to create programs which create music is a noteworthy development that can be profitably engaged with by musicologists and artists alike. The analysis undertaken in this article will attempt to relate the program itself to its possible productions, giving insight into the human effort (designing the process), but always with a view to the outcomes (the products which are each individual run of the program). There is no absolute division; anyone who has created CG-art knows the intimate negotiation between design of a program and feedback from program output. In this sense, during the design cycle CG-music is highly interactive. Yet development of a program must be frozen at some point and the generative object released (and sidestepping additional machine learning carried out during operation). To cite Gregory Chaitin, computer programs are 'frozen thought' (Chaitin 2007); they stand as beautiful (human), artistic, creative, intellectual objects. Algorithmic music is compositional design at a meta-level, human creativity in musical representations, examination of particular rule sets in a space of multiple music theories, with the composer–designer–musician becoming a 'composer-pilot' (Xenakis 1992) through musical modelling space. Composers model composition itself, and such systems give us valuable insight into the relations of music theory, musical design and aural instantiation.

3. EXAMPLES OF REALTIME COMPUTER-GENERATED MUSIC

Brian Eno's 1996 *Generative Music 1* installation, which utilised the proprietary Koan software to provide a continuous ambient installation environment in a church, helped to provoke interest in 'generative music' in the last decade; we have already discussed how the name of his installation stuck as a definition of realtime CG-music.

As always in human endeavour, the currents are much broader than a simple reading might appear. Aside from algorithmic and interactive music (Chadabe 1997), tribute should be made to the demo scene fostered since the 1980s (http://www.scene.org). This movement grew at first around the programmed 'tags' (calling cards) of game hackers, who provided new introduction sequences to the games they hacked, as advertisements and boasts of prowess (pioneer crews include rival Dutch outfits The Judges on the Commodore 64 and The Lords on the ZX Spectrum, though the earliest crack intros were for Apple II computers). Algorithmic techniques were used by necessity in the production of more elaborate animated demos, as 1980s home computers lacked the memory to load large pre-rendered movie sequences. This heritage is evident in current screen savers, which still use such realtime algorithmic generation to great compressive benefit, to procedural audio and animation for games. Yet, in the demo scene, graphics have been the main focus of generative processes. Music for demos is typically fixed sequence playback, though early demos exploited the sound synthesis quirks of 8-bit audio cards, rendering live from hard-coded tracker files (for example, Charles Deenen's circa 1985 *Game Music 1–9* for the Commodore 64, or the work of other 1980s game music composers such as Jeroen Kimmel and Rob Hubbard).

In contemporary computer-based generative art work, whole communities have grown up, such as generative.net and the associated eugene mailing list, with many artists exploring generative visuals in web software, but many also exploring realtime algorithmic composition and rendering of music. To ignore such scenes is to ignore a rich endeavour in current experimental music. It is also important to note that the scene is astylistic; whilst much of Eno's work might be described (in his own words) as ambient, diverse manifestations are exhibited by those in *The Algorithmic Stream* or *rand()%* streaming algorithmic radio stations, Karlheinz Essl's *Lexicon Sonate* (even in its non-interactive version), or my own *Infinite-Length Pieces*. Indeed, algorithmic artworks in the form of programs produced for various computer music systems from Pd to CM to the SuperCollider demo examples are further examples, devised within the realtime feedback of contemporary computer-music programming languages.

In case it is still unclear, any algorithmic method might be applied, and this potentially includes all artificial intelligence techniques. The extent to which such algorithms have yet to be harvested makes this an open research area; there are favourite techniques, controlled probabilistic expert systems being a typical route. Jem Finer's *LongPlayer* can be characterised as utilising deterministic isorhythmic layers, but whilst this may reveal one strategy, algorithmic music systems do not stop there. Some authors have not appreciated this full scope, favouring a particular technique to the exclusion of others. For example, the phenotype/genotype distinction for programs and outputs drawn by McCormack and Dorin (2001) is not pursued further in this article as being too closely locked to evolutionary computation, only one strand of AI techniques employed in computer-generated music.

I shall describe one particular generative artwork in more detail, Thor Magnusson and Runar Magnusson's *SameSameButDifferent v.02* (Magnusson and Magnusson 2007). This system is activated from a simple control panel containing a start/stop button. The source material for the work is a large set (262 MB) of sound recordings from Iceland, which are layered and processed live with each run of the work, though processing is relatively restrained in keeping with soundscape recomposition. Three to seven distinct soundfile layers will appear in a given 5–7 minute production, sometimes leading to impossible environmental pileups:

cavern, bird, boat and bubbling rocks in one observed instance. Sufficiently long soundfiles are prepared to avoid the sensation of looping – for example, some birdsong segments are up to two minutes in length – and are further processed in playback to maintain essential variety. There is a facility to store and recall particular random number seeds, represented as character strings for easy 'composition', and users can record favourite passages as soundfiles for their own linear listening away from the generative system itself. The authors characterise this as playing with the conventions of use of digital media, for their program is not restricted to a single output, but its productions can be judged as particular instances of music when favourites are selected for archiving.

4. VARIATIONS ON ANALYSIS

Rather than storing a fixed recording of finite length, an algorithm can embody the potential exploration of massive parameter spaces, allowing infinite (or effectively infinite, see Collins 2002) play times. Yet this very compression means giving up the idea of a single product and exploring a potential panoply of outcomes. The game in generative music is for the composer to enable a suite of possible products; integrity demands that each production be of sufficient merit to honour the system that produced it. The extent of variation of the system on multiple levels is claimed to be of aesthetic worth by generative artists, and becomes the very factor that challenges analysis.

A key question then is the extent to which any system can truly claim a varied output, and how to measure and analyse the extent and content of productions; in short, how many runs are representative? Music's innate combinatoriality of material lies in wait for us here (Prokofiev 1978: 46–9). As Sergei Sergeyevich noted even in the case of a classical melody, the explosion of the parameter space size with each new choice of note pitch guarantees a world of useful productions. Most parameter spaces for programs are bigger than the number of atoms in the observable universe (Collins 2002). If linked to a single random seed, this scope is reduced to a number of paths (possible productions) controlled by the number of distinct values of the seed. To give an example, in SuperCollider the seed is given by three 32 bit numbers, or of the order of 10^{28} options, so that it would take more than 10^{23} years to listen continuously to all possible three-minute productions; the universe is estimated to be 14 billion, or of the order of 10^{10} years old. In an 'infinite play' situation, just running from one seed, the pseudo-random number generation draws an infinite number of times with a period of 2^{88} and the full combinatorics in interaction with the mathematical parameter space is frighteningly large.

Fortunately, for the purposes of analysis, the psychological space of outputs is typically much more constrained than the mathematical space of a program's runtime avenues. Analysis must confront the relation of the internal space of a program (initial state and algorithms) to the psychological space of potential outputs. Indeed, the essential problem of musical analysis is never sidestepped, in that there are a number of relevant levels and viewpoints from which to consider the cognitive affect of a piece of music; different listeners may more quickly or slowly discount the variability of a given generative artwork depending on cultural exposure to music of that nature and choices of attending.

A fundamental aim in analysing a computer-generated musical work therefore is to explore the relation between source and production. Rather than misleading or incomplete analogies to object-oriented programming and the aforementioned genotype/phenotype distinction, the clearest expression of this task is to explain a given output (a production) in terms of the originating program (a source). Knowledge about such mechanisms allows one to predict the space of future outputs, and thus truly delimit the scope of a given compositional model/music theory/program with respect to both craft (emic) and aesthetic (etic) outcome – that is, poeisis and esthesis. The next sections relate tools to approach this aim. Mathematical parameter space analysis founded in software analysis techniques drawn from software testing can break down the program, and are combined with psychological analysis of associated musical resultants.

4.1. Black box and white box testing

Investigation of the relation of source to productions can be informed by a consideration of the difference between black box and white box testing in software engineering (Lano and Haughton 1993; Myers 2004). In black box testing, the program is left unopened, perhaps because it is only available as a binary executable that cannot easily be cracked and returned to human readable code (many comments and clues would be lost forever even if this could be achieved). In white box testing, access is granted to the original code itself, from which a more detailed analysis of possible productions should be possible. In practice, software testing is a hard pursuit; Myers (2004) claims it to require more ingenuity than devising the actual program.

Testing within a framework of software engineering may rest upon knowledge of the task specification, from which unit tests can be devised; yet in the worst case we may have a sealed executable program with little or no information from the author as to its intended outcomes. Anti-archivists might imagine a future civilisation somehow desiring to reverse engineer the generative musical works of a lost digital media scene. It is always more helpful to have access to the original representational source rather than indirect evidence of that source. In some cases, reasonable assumptions may lead one to reverse engineer the contents of a black box. This process is perhaps already familiar from technical analysis of electronic music works released without their studio sources (for example, the actual sequencer data which gave rise to a piece).

A note of caution: assumptions of reasonable behaviour are essential. Marsden's Maxim might be borne in mind:

> To establish a system for representing any aspect of music is almost a challenge to a composer to invent music which the system is incapable of representing. (Marsden 2000: 168)

Composers can always devise generative music programs that are difficult to analyse. Imagine a program that is set to generate bland clicks on its first hundred runs, but fires into a wondrous world of sound on the hundred and first. Deliberate statistical perversity like this has to be discounted in keeping analysis tractable.

In some cases, the exact details of the generative program may not be recoverable, or necessarily essential. Indeed, dissimilar program code can still achieve a sufficiently similar or identical musical output, as judged from psychoacoustical and cognitive considerations (determining similar categorical boundaries) or from multiple software representations leading to the same 'answer'. Listening tests (Pearce and Wiggins 2001) or further automated analysis on a sample of recordings may be preferred approaches, focused more strongly on the production side. But, wherever possible, white box testing is preferred as giving additional insight.

4.2. Representatives

We could always run a generative music program once only, harvest a single production of five minutes, and claim this to be representative of the work. Any conventional aural and musicological analysis can then be applied to the fixed product so obtained. Unfortunately, this would be a gross abuse of the reality of generative music systems, which are designed to create multiple productions; we would have learnt nothing of the mechanisms by which such programs operate, of the musical model underlying them, and of the scope of future productions from that program. The paradigm shift from composing single works to creating machines that embody a compositional theory creates a more difficult analytical problem. Multiple representative productions must be drawn from the artefact; the question then becomes, how many are actually representative?

This question can be posed as a statistical problem. Unfortunately, there is no guarantee that productions will follow a normal distribution within the parameter space. Random sampling (running the program over and over) must provide good representatives so that tests on the sample are indicative of true population behaviour using standard statistical procedures (Spiegel, Schiller and Srinivasan 2000). Nevertheless, a pragmatic approach is reasonable; most statistical tests are relatively robust, and there is little else that can be done than take a sampling of outputs.

It should be noted, however, that the choice of how many to take may be influenced by the decision on how to analyse and compare the productions themselves; for what may seem a reasonable cross-section of representatives with respect to one musical facet may not be exhaustive of productions for the generative program with respect to another. This may form the key basis for criticism of any individual analysis of a given work, and at the least, if publishing work, one must take care to make assumptions explicit over sampling procedure. For the white box testing case, given the relation of code to outputs, in certain situations it may be possible to make an argument that coverage of program avenues is sufficiently complete to provide an accurate sample.

To sample more productions, it would be convenient if we could run an automated critic/analyser; this is a further computer program that embodies a particular analytical model, just as the generative music program may represent a compositional model. Yet we must collate the database of output versions first. This collation may not be possible at anything less than realtime speed, given the performance character of those generative works we have been considering (and there are algorithmic composition programs that are slower than realtime too). In certain cases, with access to the source, the code may be adaptable to a faster than realtime rendering mode to assist this. Nevertheless, it must still be argued that the database so formed is indicative of productions, which itself may require realtime playback of representatives (an argument from code would be necessary to circumvent this, proving yet again the intimate relation of source and productions in this analytical dialogue).

Once created, a database is amenable to automated analysis methods, essentially music transcription methods from current disciplines such as music information retrieval. The usual issues of computational auditory analysis are inevitably raised, for we are depending on signal processing routines instead of the human ear, and computational methods have not exhibited anything like the acuity of the human auditory system. A caricature of current research might acknowledge certain gross timbral measures and transcriptions with success rates from 40 to 80% or so as currently available (Klapuri 2004, or see the results of recent MIREX algorithm evaluation contests); yet transcription is sometimes ill-defined and most auditory mechanisms remain beyond the state of the art. However, there is some hope that mechanisms may be found that (verified through independent psychological tests) can be employed with some reasonable trustworthiness. Indeed, the domain of computer-generated work is challenging enough that investigation of such technology may be enforced. To place one last proviso, effective auditory models are not necessarily going to calculate at any rate much faster than the human brain itself, and may be orders of magnitude slower. Since the human mind is the arbiter of musical appreciation and social musical discourse, and the automatic transcription technology will always fall somewhat short of being a real human being, it would be unsurprising that the human mind remain the best judge.

The reader may be wondering if reductionism might help us? Where we have access, can we simplify the procedural code (if not the psychological space model)? We surely risk losing increasingly obvious nuances of the composition; in fact, we may irrevocably damage the composition with any such step, particularly for emergent non-modular behaviour. An example will be given below where two characters added to a program substantially alter its audible output. Spotting such situations requires some heavy duty software engineering, and justifying substitutions may require exactly the same analysis that we were seeking to make in the first place.

For those who may have reservations on the programme I have outlined here, I note that I make no rigid stipulation of the exact musicological assumptions that must be applied. I favour myself a full grounding in psychophysics and the psychology of music; others might consider the timbral and spatial concerns of spectromorphology primary, and these are not irreconcilable. But I hope I have provided some justification for treating both program and productions, and indicated a need to bridge between process and product if we are to truly analyse and explore algorithmic music programs.

5. EXAMPLE ANALYSES

Having raised many potential problems within the analysis of computer-generated music, it is still I believe feasible to tackle such analysis, taking care to be explicit about our sampling and musical assumptions. This section will include two analyses of short works by James McCartney, the original author of the SuperCollider audio programming language (McCartney 2002), which illustrate the points made in the preceding sections of this article. The pieces are demo examples available in the (open source) SuperCollider distribution, and were chosen in part for their easy availability and in part because the terse but dynamic work of McCartney rewards study. McCartney is not an official establishment composer (which may be for the better!) and is known far more as a computer-music system designer; these demo examples provide a small-scale test commensurate with the space available. Future articles may confront in depth the cases of particular larger-scale generative music programs.

All code is open source under the GNU GPL licence for SuperCollider and this allows white box testing and easy modification for assessment. I have preserved James McCartney's original comments, added a few additional comments, and tried to make the code generally presentable so that even a reader with little SuperCollider knowledge can at least have a chance of seeing what is going on. To assist understanding, natural language pseudocode of each program is also provided, though the provisos about reductionism must be borne in mind; these reductions should be taken as aides for the reader, and cannot encapsulate every detail of the full programs. Full understanding of the code itself necessarily rests on studying the programming language in question, an essential skill for generative music analysis which employs white box testing. Certain lines are marked in the comments with a /*x*/ number, which allows them to be further discussed in the main body of the text, and the pseudocode also maintains these markings at the critical place as carefully as possible given the reductionism.

5.1. *Synthetic Piano* by James McCartney

To accompany the analysis here, figure 1 gives the original SC code, figure 2 pseudocode, and figure 3 compares spectrograms for two 30-second productions created with the *Synthetic Piano* program. In order to keep the visuals uncluttered, figure 3 was created using a mono version of the program with only three voices rather than six. These spectrograms may be useful for the reader to visualise the exact repetition of each note at a particular phase and period; since the top and bottom spectrograms are from consecutive runs, they provide a limited view into the variation of the program, but we must be very careful not to read too perfect an aural analysis into the spectrograms, nor take two runs as necessarily representative.

The example code (originally for SC2, 1998) is devised entirely as a specification for a sound synthesis patch, defining a particular unit generator (UGen) graph to run on the SuperCollider synthesis server. However, each time the patch is run, it generates a graph differing in certain properties sufficient to give varied aural output. The sound synthesis method is a simple but effective source plus filter physical model, using an excitation pulse (a smoothed impulse train) passed through a tuned comb filter, thus approximating a Karplus-Strong model. The patch has n voices, n being the sole input parameter specifiable by a user, defaulting to n=6 'notes' as set initially at /*1*/. There is also a hidden parameter, the seed for the random number

```
(
var n= 6;// number of keys playing /*1*/

play({
Mix.fill(n, {           // mix an array of notes
        var delayTime, pitch, detune, strike, hammerEnv, hammer;

        pitch = (36 + 54.rand); // calculate delay based on a random note /*2*/

        strike = Impulse.ar(0.1+0.4.rand, 2pi.rand, 0.1); // random period for each key /*3*/
        hammerEnv = Decay2.ar(strike, 0.008, 0.04); // excitation envelope

        Pan2.ar(
                Mix.ar(Array.fill(3, { arg i;  // array of 3 strings per note

                        // detune strings, calculate delay time :
                        detune = #[-0.05, 0, 0.04].at(i); //chorusing
                        delayTime = 1 / (pitch + detune).midicps; //convert midi note pitch to frequency

                        // each string gets own exciter :
                        hammer = LFNoise2.ar(3000, hammerEnv); // 3000 Hz was chosen by ear /*4*/
                        CombL.ar(hammer,        // used as a string resonator
                                delayTime,      // max delay time
                                delayTime,      // actual delay time
                                6)              // decay time of string
                })),
                (pitch - 36)/27 - 1 // pan position: lo notes left, hi notes right /*5*/
        )
})
})
)
```

Figure 1. Original SuperCollider code for *Synthetic Piano* with added comments.

```
(
n= number of desired notes (default 6) /*1*/

Mix together n notes, each created using the following synthesis recipe:
        Draw a MIDI note number linearly from 36 to 89 as the note pitch /*2*/
        Create an impulse generator with random period and phase /*3*/
        Mix together three strings per note with chorus detuning:
                Each string is rendered using a comb resonator driven by a
                personalised noisy excitation triggered by the impulse generator for this string /*4*/
        The note is panned in the stereo field based on the pitch /*5*/
)
```

Figure 2. Pseudocode reduction of figure 1.

generator, which is controlling the rolls of the dice for /*2*/ and /*3*/. A separate random number generator is implicated in the LFNoise2 UGen on line /*4*/, but it runs during the live synthesis rather than in the initialisation of the UGen graph.

Listening to the patch a number of times will quickly convince a listener that the process is relatively constrained; indeed, the shortness of the code example here is sufficient to convince oneself of the limits of the psychological parameter space. Since the program effectively determines the base (MIDI note) pitch of the strings at each run, from a range of 54 options (54.rand at /*2*/), there are 54^n possibilities, thus only 24794911296 for n=6. This number might be limited further by considering transpositional equivalence, though there is a further timbral implication; but the mathematical combinatoriality is evident.

/*3*/ introduces a further proliferation of options, by determining the strike rate of each string in terms of period and phase. Whilst floating point ranges are used, psychological considerations of temporal perception might limit the options. The range of possible periods is from 2 to 10 seconds (frequencies of 0.1 to 0.5 Hz), and phase can take any value within the cycle; since human temporal perception is unreliable for periods greater than 2 seconds (London 2004), we might substantially

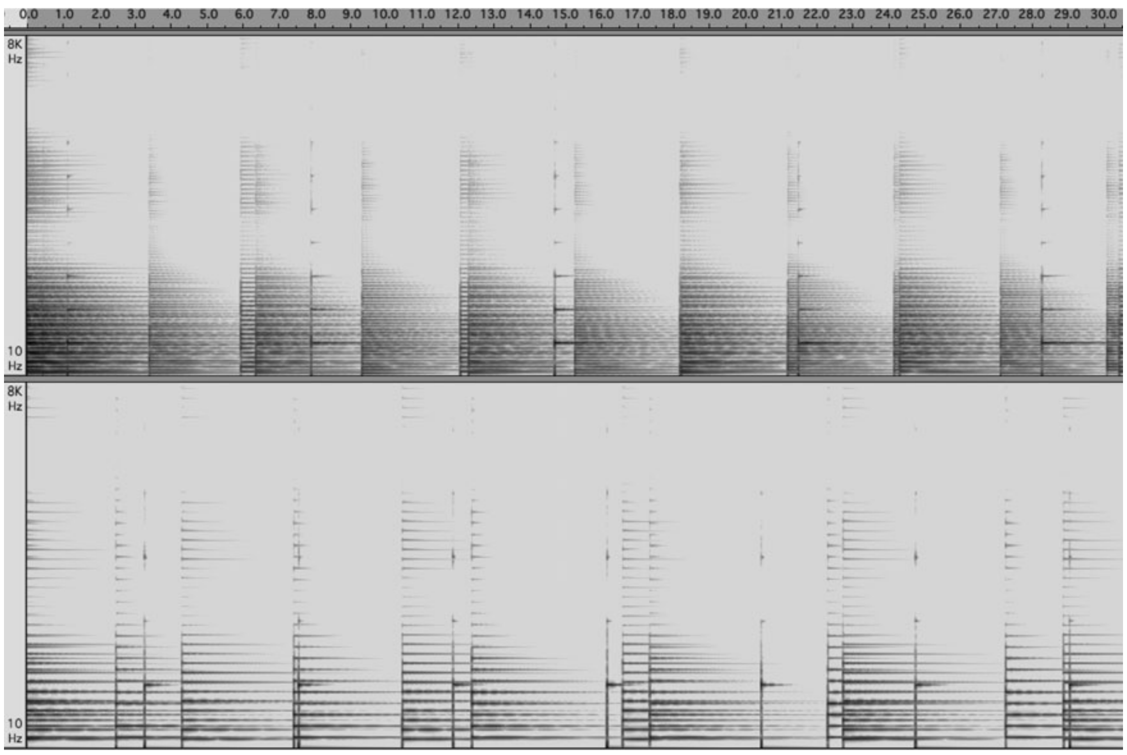

Figure 3. Comparing spectrograms of two (mono) productions from the *Synthetic Piano* code, each 30 seconds long.

drop the number of options; the temporal result is somewhat like a slow mobile set in motion, where the recurrence of certain voices and the tracking of phase relationships is inherently difficult. A coarse analysis might claim that the temporal information is of no significance in accounting for inter-run variation at this point. There is one quirk however; an Impulse UGen should usually have a starting phase from 0 to 1.0, as a proportion of a cycle. Those in the synthetic piano patch are initialised from 0 to 2pi; due to an interaction with the low-level specification of the Impulse itself (obtainable only because SC is open source), any phase value greater than 1 will cause an initial strike on starting the patch, such that a near aggregate chord across the keys of the piano is heard commencing each run. This behaviour could be corrected by 1.0.rand instead of 2pi.rand in line /*3*/. It is hard to say whether this is deliberate or a mistake in converting the code from SC2 to SC3 in 2002; one is reminded of the 'errors' in Xenakis's GENDYN code (Xenakis 1992), which are certainly part of its character.

Some elements of the patch are deterministic; in other words, the pan positions depend on the pitch of each key (/*5*/). Various constants scattered through the patch can be modified, perhaps departing from the author's original intent, but providing clues to the empirical construction of the patch; for example, McCartney's original comment admits choosing 3000 Hz by ear at /*4*/. Summarising the generative system, it may be viewed as a soundscape generator; lacking longer term directives of structure and exhibiting a flat hierarchy, form is coincidental, not planned, a consequence of linear probability distributions over initialisation parameters primarily at /*2*/ and /*3*/.

Given the amount that might be said concerning this patch in relation to its size (and whose perceptual effect might be the subject of further listening tests with multiple subjects) the compressive benefit of generative systems is evident. As one final observation, consider the addition of two characters to /*2*/, namely 54.rand becoming 54.0.rand. The switch to a floating point range now brings in microtones; at an estimate of microtonal sensitivity at 24th tones, the parameter space grows by a further 12^n and twelve tone equal temperament is immediately lost! This is pointed out as evidence of the heady consequences of even small changes to source code.

5.2. From a workshop given by James McCartney in Santa Cruz on 6 June 2003

The second realtime algorithmic music program was originally conceived as an example for a workshop (McCartney 2003), and is in two parts, the first comprising a generation of particular recipes for the sound synthesis of metallic percussion sounds, and the second specifying the realtime scheduling of those sounds. In this manner, since either part may be renewed when desired, the work has a certain generative ambiguity; there are issues here of the reusability of code and the potential of different code substituting either part. In the original (http://lists.create.ucsb.edu/pipermail/sc-users/2003-June/004129.html), there is also a further separate effects unit with mouse control, which is added once the process is underway, but this optional extra is kept from this example for reasons of space, and to sidestep further issues of interaction.

Again, the analysis is accompanied by original source code (figure 4), a pseudocode reduction (figure 5) and a visual aide. The spectrogram in figure 6 was created from 20 seconds' worth of one mono production, for a limited demonstration of density and timbral properties only, as discussed below.

Treating the definitions of the metallic sounds first (the first block of code in figure 4 in outer parentheses, with the SynthDef construct), aural comparison confirms the use of metallic plate sounds with varying resonances. The sound synthesis method revealed by the source code is that of a white noise excitation shaped by a decaying impulse linearly drawn from 0.2 to 1.0 seconds (/*2*/) passed through a filterbank represented by the Klank UGen (/*3*/). Parameters of this filterbank are all drawn from appropriate ranges, exponentially for filter centre frequencies and ring times, and linearly for amplitude gains (though this might also have been made exponential through the use of decibels, for instance). The number of filters in the filterbank is defined by n=12 at /*1*/, though again this constant might be modified by the user; tests show that for low n, the spectral recipe is too simple, producing singular ring components that fail to simulate the body responses of real world objects, but instead sound like 1950s studio experiments. Higher n (i.e. n=30) are not particularly different to n=12, but increase the sense of a blacksmith's forge (more violently struck metals). All sounds are panned randomly within a flat plane square four-speaker set-up by the PanAz UGen (/*4*/); McCartney's entire workshop was carried out with such a spatialisation.

```
// mass production of synths..
(
40.do({ arg i;
SynthDef("rperc" ++ i.asString, { arg i_bus = 0, amp = 0.1, rate = 1;
        var n = 12;                    /*1*/
        var exc, out;
        //excitation signal:
        exc = WhiteNoise.ar * Decay.kr(Impulse.kr(0,0,amp*0.1), rrand(0.2,1.0)); /*2*/
        //through a filterbank:
        out = Klank.ar(`[ //holding parameters of the Klank filterbank: /*3*/
                        {exprand(100.0, 10000.0)}.dup(n),   //filter centre frequencies
                        { rrand(0.1,1.0) }.dup(n),          //filter gains
                        {exprand(0.05,1.0)}.dup(n)          //filter ring times
                ], exc, rate);
        DetectSilence.ar(out, 0.0001, 0.1, 2);
        Out.ar(i_bus, PanAz.ar(4, out, rrand(-1.0,1.0))); //4 speaker azimuth pan /*4*/
}).load(s);
});
)

// a process to use them.
(
var s = Server.local;
Task({
var dur=0.2, inst = \rperc0, amp = 0.05; //initialisation /*5*/
inf.do({
        if (0.3.coin, {        // chance of changing instrument /*6*/
                inst = "rperc" ++ 40.rand.asString;
                amp = exprand(0.02,0.2) * 0.5;
        });
        //play current sound:
        s.sendBundle(0.2, [\s_new, inst, -1, 0, 0, \amp, amp, \rate,  exprand(0.5,2)]); /*7*/
        //change current duration?:
        if (dur.coin, { dur = [0.05, 0.075, 0.1, 0.15, 0.2, 0.3, 0.4, 0.6,  0.8, 1.6].choose }); /*8*/
        dur.wait;
});
}).play(TempoClock.default);
)
```

Figure 4. Original SuperCollider code for Santa Cruz workshop piece with added comments.

```
There are two stages:

Stage 1 (prepare sounds)
Create 40 different synthesis recipes based on variants of a source-filter model:
        The source is a decaying impulse amplitude modulated by white noise /*2*/
        The filter is a bank of 12 (/*1*/) resonators with centre frequencies, filtergains and ring times generated from
        exponential (100-10000Hz), linear (0.1-1.0) and exponential (0.05-1.0 seconds) distributions respectively
        /*3*/
        Apply quadrophonic panning with random azimuth location /*4*/

Stage 2 (schedule sounds over time)
Initialise variables for duration (dur), current sound (inst) and playback amplitude (amp) /*5*/
Loop indefinitely:
        There is a 30% chance of re-assigning inst and amp /*6*/
        Play the current sound with the given amplitude, and with a frequency scaling of the filter frequency bands
        from an octave down to an octave above /*7*/
        dur itself is the probability of updating dur to a new value selected from an array of options /*8*/
        wait for dur
```

Figure 5. Pseudocode reduction of Figure 4.

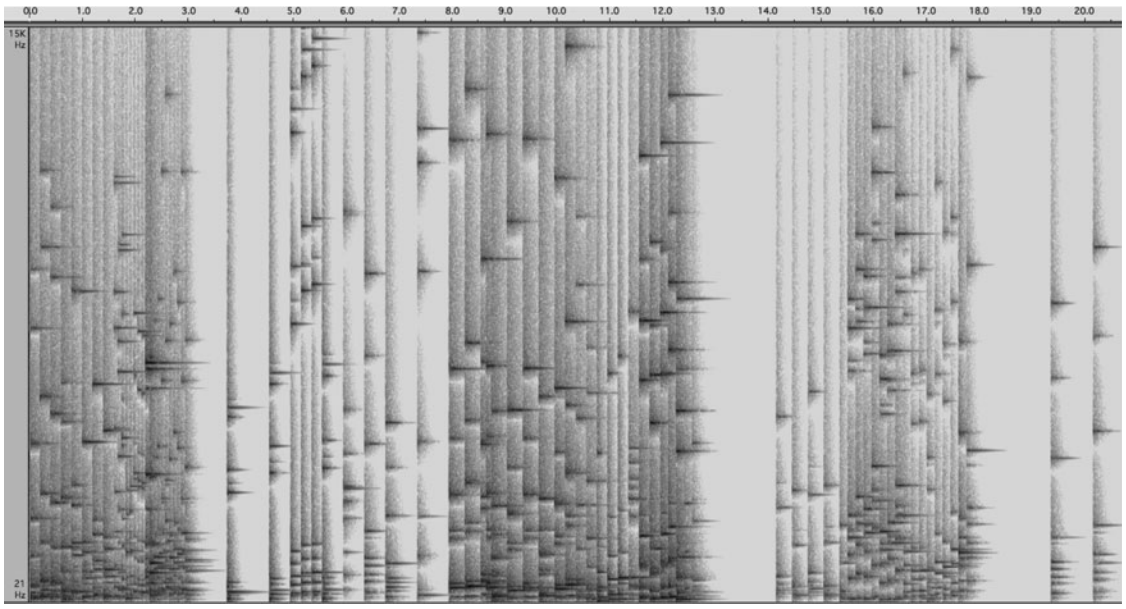

Figure 6. Spectrogram of 20 seconds of sample (mono) output from the Santa Cruz program, demonstrating the rhythmic densities and spectral recipes in play.

It should be noted with respect to earlier comments on automated analysis that we could analyse the spectral variety of sounds in this instance by, say, generating 400 and comparing their spectral recipes. Yet the SynthDef itself already gives us the spectral properties of the filter, and this is a case where the human ear is trusted, in alliance with an understanding of the sound synthesis method employed. There is no doubt that representatives of the sounds in this case are well covered by reasonable sampling of this nature.

Moving onto the scheduling portion of the work, one of the chief motivations in analysing this particular example piece is to point to the beauty of the code at /*8*/. The infinite scheduling loop (within the Task) is designed to wait for a certain duration, the current value of the dur variable initialised at /*5*/, between triggering sounds. The duration is expressed (by default) in seconds, but in /*8*/ is itself used to form the probability of taking on a new value; in effect, long durations give a high chance of change, for isolated slow pauses, and low values give a low chance of change that cause flurries of fast events. Durations over 1 (1.6 being the only option) are guaranteed to only happen once before a new choice of duration. However, the durations are chosen from an array with a linear distribution, so there is a one in ten chance of two 1.6-second gaps in a row, a one in a hundred chance of three, and so on. At the other end of the scale, the 0.05 gap might on average correspond to twenty repetitions before a change of the duration value. In rhythmic terms, ten events per second is the human motor limit for repetitive actions without chunking (London 2004), so 20Hz action is inhuman; there is no jitter or other expressive timing built in, so that the machine quality is evident. This all adds up to an arresting rhythmic sequence that is definitely inhuman; our categorical perception of rhythmic values in relation to metrical structure tends to make it difficult for us to deal with ten distinct time values as accurately as the computer.

Timbral variation is achieved by exploiting the forty stored sounds and their control parameters through /*7*/, which scales the spectra of filter frequencies. A new instrument and amplitude is selected on average every 3.33 strikes due to /*6*/. Yet the overall effect is one of subtle timbral variation, due to the uniform sound synthesis method underlying the instruments; the listener is within a manic metallic machine that has the capacity to pause in dramatic fashion. The patch has the character of a demonstration, but could be a stepping-off point for something much richer; yet, it still provides plenty of food for thought for the algorithmic music analyst, belying its size as a morsel of code.

Neither of the two example programs tackled above shows longer-term form or significant variation of output in the way being designed into some current algorithmic music systems. They are simply entry-level works to enable the grasping of the problems of analysis in this domain, and were picked as much as anything for their terseness, which allows the whole program to be examined here. Future analyses may take an entire article to tackle one single work, and only be able to quote salient extracts of code. It remains an open question to consider how the ploys discussed in this article might scale up to creative systems built to engage in much more challenging and varied behaviour.

6. CONCLUSIONS

Artists are drawn to the medium of generative music for many reasons, from compressive benefits through love of system to love of the infinite. An analytical engagement with these works is productive and healthy to much contemporary artistic endeavour, for the artists as much as the musicologists. This article has sought to clear up some of the history and terminology surrounding generative music, focusing in on generative music computer programs in particular.

The methodology outlined in this article is grounded in software testing techniques and in literature on sound synthesis and music cognition. Two examples of this approach have been given for short generative works by James McCartney. But no claim is made that this article somehow embodies a complete set of tools and approaches for analysis, even if there could ever be a final solution to the problems of analysis. Inevitably, there is a great deal more to learn as we tackle this music. There are no doubt other profitable avenues that might be transplanted to the domain of generative music program analysis, from musicological insights on corpuses and style, to generative linguistics and creativity in musical improvisation. It is also hoped that computer-generated-music analysis may be related to and also give some entry to issues in the analysis of interactive music systems, where responses are founded on generative techniques.

Indeed, the difficulty of treating complex algorithmic music systems has only been touched on; how could a meta-system be approached, created to itself create generative music programs? How can investigation be scaled up to deliberately massive algorithmic works? These are questions that could keep analysis busy for an infinite length of time; it is tempting to build a generative program for the analysis of computer-generated music…

ACKNOWLEDGEMENTS

The author wishes to thank Andrew Brown and four anonymous reviewers, who did much to improve the article's accessibility and remit.

REFERENCES

Ames, C. 1987. Automated Composition in Retrospect 1956–1986. *Leonardo* **20**(2): 169–86.

Boden, M. 2007. What is Generative Art? COGS seminar, University of Sussex, 2 October 2007.

Chadabe, J. 1997. *Electric Sound: The Past and Promise of Electronic Music*. New Jersey: Prentice Hall.

Chaitin, G. 2007. *Meta Maths: The Quest for Omega*. London: Atlantic Books.

Collins, N. 2002. Infinite Length Pieces: A User's Guide. Proceedings of MAXIS, Sheffield, April.

Collins, N. 2003. Generative Music and Laptop Performance. *Contemporary Music Review* **22**(4): 67–79.

Cox, C. and Warner, D. (eds.) 2004. *Audio Culture: Readings in Modern Music*. New York: Continuum.

Eacott, J. 2000. Form and Transience – Generative Music Composition in Practice. In *Proceedings of Generative Art*, Milan.

Eacott, J. 2006. *Contents May Vary: the Play and Behaviour of Generative Music Artefacts*. PhD thesis, University of Westminster, submitted June 2006.

Eno, B. 1996. Generative Music. *In Motion Magazine*. Available online at http://www.inmotionmagazine.com/eno1.html.

Galanter, P. 2006. Generative Art and Rules-Based Art. *vagueterrain03*. http://vagueterrain.net June 2006.

Klapuri, A. P. 2004. Automatic Music Transcription as we Know it Today. *Journal of New Music Research* **33**(3): 269–82.

Koenig, G. M. 1989. Aesthetic Integration of Computer-Composed Scores. In C. Roads (ed.) *The Music Machine*. Cambridge, MA: MIT Press.

Lakoff, G. 1987. *Women, Fire, and Dangerous Things*. Chicago: University of Chicago Press.

Lano, K., and Haughton, H. 1993. *Reverse Engineering and Software Maintenance: A Practical Approach*. Maidenhead: McGraw-Hill Book Company.

Laske, O. 1989. Compositional Theory in Koenig's Project One and Project Two. In C. Roads (ed.) *The Music Machine*. Cambridge, MA: MIT Press.

LeWitt, S. 1967. Paragraphs on Conceptual Art. *Artforum* **5**(10) (June): 79–83.

London, J. 2004. *Hearing in Time: Psychological Aspects of Musical Meter*. New York: Oxford University Press.

Loy, G. 2006. *Musimathics (Vol. 1)*. Cambridge, MA: MIT Press.

Magnusson, T., and Magnusson, R. 2007. SameSame-ButDifferent v.02 – Iceland. *YLEM Journal: Artists Using Science and Technology* **27** (Summer).

Marsden, A. 2000. *Representing Musical Time: A Temporal-Logic Approach*. Lisse: Swets and Zeitlinger.

McCartney, J. 2002. Rethinking the Computer Music Language: SuperCollider. *Computer Music Journal* **26**(4): 61–8.

McCartney, J. 2003. SuperCollider workshop at the Felix Kulpa Gallery, Santa Cruz, California, USA, 6 June 2003. http://lists.create.ucsb.edu/pipermail/sc-users/2003-June/004114.html.

McCormack, J., and Dorin, A. 2001. Art, Emergence and the Computational Sublime. In A. Dorin (ed.) *Proceedings of Second Iteration: A Conference on Generative Systems in the Electronic Arts*. Melbourne: CEMA, pp. 67–81.

Myers, G. J. 2004. *The Art of Software Testing*. Revised and updated by T. Badgett, T. M. Thomas and C. Sandler, Hoboken, NJ: John Wiley and Sons.

Pearce, M. T., and Wiggins, G. A. 2001. Towards a Framework for the Evaluation of Machine Compositions. In *Proceedings of the AISB'01 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*. Brighton: SSAISB, pp. 22–32.

Pearce, M., Meredith, D., and Wiggins, G. A. 2002. Motivations and Methodologies for Automation of the Compositional Process. *Musicae Scientiae* **6**(2): 119–47.

Prokofiev, S. 1978. *Sergei Prokofiev: Material, Articles and Interviews*. Compiled by Blok, USSR: V. Progress Publishers.

Rowe, R. 1993. *Interactive Music Systems*. Cambridge, MA: MIT Press.

Searle, J. R. 2004. *Mind: A Brief Introduction*. New York: Oxford University Press.

Spiegel, M. R., Schiller, J., and Srinivasan, R. A. 2000. *Probability and Statistics*. New York: McGraw-Hill.

von Arn, L. Human Computation. Google TechTalks, 26 July 2006. http://video.google.com/videoplay?docid=–8246463980976635143.

Ward, A., and Cox, G. 1999. *How I Drew One of My Pictures: or, The Authorship of Generative Art*. Available at http://www.generative.net/papers/authorship/index.html.

Xenakis, I. 1992. *Formalized Music*. Stuyvesant, NY: Pendragon Press.