MUSICAL METACREATION

The Use of Interactive Genetic Algorithms in Sound Design: A Comparison Study

By Matthew John Yee-King

Matthew John Yee-King
Computing, Goldmiths, London, U.K.

Two sound design methods were compared: modular synthesis and Evosynth, a novel variable architecture synthesizer programming system using an interactive genetic algorithm. They were compared using surveys, classification into established ontologies of creative systems and output analysis. Two surveys examined users' opinions about the two synthesis methods. 430 modular synthesizer users and 14 Evosynth users responded. Both user groups valued unexpected output from the systems and tended to use exploratory approaches to sound design. Placed into ontologies of creative systems, the systems share characteristics such as autonomous signal and pattern generation, interactivity and the ability to generate novel output that was valued by their users. During a month long analysis period where Evosynth was online, 3552 breed events were recorded from 229 unique IP addresses and 90 'fit' sounds were saved to the Evosynth server. The output and other analyses suggested that both systems can generate a wide range of timbres and that they allow a gradual exploration of timbre space.

CCS Concepts: .**Applied computing → Sound and music computing; Computing methodologies** → *Search methodologies;*

Additional Key Words and Phrases: Musical metacreation, sound synthesis, genetic algorithms, audio analysis

1. INTRODUCTION

This paper describes and compares two sound design methods: hardware modular synthesis and a novel virtual modular synthesizer called Evosynth, which is programmed via an interactive genetic algorithm. Each of the two methods are investigated through a survey, classification into ontologies of creative systems and analysis of their timbral outputs. The surveys sample user opinions about modular synthesis and Evosynth, relating to the way they present sonic novelty to the user and other aspects of the programming experience. The classification analyses identify established ontologies of creative systems and place the sound design methods therein. The output analyses consider the timbral range of the two methods and how users explore that range. The complete dataset and code discussed in this paper is available in a source code repository on Github [Yee-King 2015]. The motivation for the work presented in this paper is a desire to explore the boundary between familiar audio tools and novel creative systems and the desire to hear sounds that have never been created before.

1.1. Structure of this paper

The paper begins with a discussion of methods that can be used to examine sound and music creation tools: HCI methods in section 1.2 and classification methods in section

1.3. Previous work using genetic algorithms for sound design is considered in section

1.4. In section 1.5 is a discussion of the first sound design technique, modular sound synthesis. Evosynth is described next, in section 2. The three methods used to analyze the two systems are described in section 3, the results of the analysis are presented in section 4. Finally there is a conclusion in section 5.

1.2. Evaluation methods for creative systems

There is considerable work describing and evaluating novel sound synthesis techniques, user interfaces and generative systems in the research fields of computer music, human computer interaction (HCI) and new interfaces for musical expression (NIME). Stowell et al. provide a review of HCI techniques for evaluating musical systems, concluding with a list of recommended techniques for various types of musical system [Stowell et al. 2009, p13-14]. The recommended techniques include a Turing Test, audience and performer interviews, surveys with Likert scales and a task-based approach. Kiefer et al. present an HCI methodology for evaluation of musical controllers [Kiefer et al. 2008]. They combine a qualitative, grounded theory analysis [Stern 1980] of interviews with users and a quantitative, statistical analysis of interface telemetry data, concluding that the most 'interesting' results came from the coded interview data, indicating the trend in HCI towards *experience* as opposed to *task* focused HCI [Kaye 2007]. Tubb and Dixon evaluated two timbre

exploration interfaces, one providing access to the complete set of 10 synthesis parameters and the other to a parameter space reduced to two dimensions using a space fitting curve [Tubb and Dixon 2014]. They were interested in Guilford's divergent and convergent phases of creativity, which they describe as follows: 'Divergence is the generation of many provisional candidate solutions to a problem, whereas convergence is the narrowing of the options to find the most appropriate solution' [Guilford 1967]. Their evaluation involved a detailed interface telemetry based analysis of the changes users made to the synthesis parameters during an editing session and it suggested various phases of convergence and divergence in the synthesizer programming process. Moving from movement in parameter space to movement in timbre space, Yee-King investigated the timbral effects of a user varying sound synthesis parameters in different synthesis architectures [Yee-King 2011, p56]. The focus here was examining the geography of timbre space and its relation to parameter space and this can be called an output analysis, where the output of the tool is analyzed through feature extraction.

Table I. Evaluation methods for creative systems.

Evaluation	Example method
comparative	system vs. human/ Turing Test
experiential	survey, interview
task based	user interface telemetry
output analysis	creative space analysis

Table II. Classification of creative systems.

Description	Who
Input/Model/Output	[Rowe 1992]
Generative systems: generation and interaction	[Boden and Edmonds 2009]
Systems creating artefacts with quantified novelty and value	[Wiggins 2006]
Musical Metacreation: levels of autonomy	[Eigenfeldt et al. 2013]

Considering the output analysis approach, Collins states that a key question is 'the extent to which any system can truly claim a varied output, and how to measure and analyze the extent and content of productions' [Collins 2008]. Also, he tells us that 'the psychological space of outputs is typically much more constrained than the mathematical space', which emphasizes the requirement for the analysis to consider the perceptual or experiential effect of a system's output. So output analysis can be considered as a quantitative form of experiential analysis, especially if the analysis focuses upon human, perceptual aspects. In summary, we have highlighted several approaches that can be applied to the evaluation of sonic creation systems: comparative (e.g. system vs. human/ Turing Test), experiential (survey, interview), task based (e.g. user interface telemetry) and output analysis (e.g. audio analysis). It should be noted that these approaches are not mutually exclusive, e.g. task analysis might consider the output of the system as well as the process used to achieve it. The evaluation methods described above are summarized in Table I.

1.3. Classification of musical systems

Another approach to the analysis of sonic creation systems is to classify them. Seago et al. define 3 modes of interaction between user and sound: user specified synthesis architecture, fixed synthesis architecture, and direct engagement [Seago et al. 2004]. Dealing with the classification of processes that can be applied to an audio signal, Wilmering et al. present an ontology of audio effects which covers the effect type, effect format, parameters etc., a system that could be adapted to synthesis [Wilmering et al. 2013]. Boden and Edmonds provide a taxonomy of systems that generate artistic works, considering how the art is generated, the control the artist has over the process and so forth, leading to categories such as interactive art, generative art and evolutionary art [Boden and Edmonds 2009]. Wiggins develops a more formal framework for the description and evaluation of creative systems [Wiggins 2006]. It builds from the basis that creative systems exhibit creative behavior that generates artifacts having quantifiable novelty and value. Moving to musical systems, Rowe describes 3 key areas of variation between interactive music systems: how they are driven (input), how and what they generate (output) and where they fall on the continuum from instrument extension to autonomous player [Rowe 1992]. Eigenfeldt et al. describe a taxonomy of musical metacreation systems, mapping from least to most autonomous. The categories are independence, compositionality, generativity, proactivity, adaptability, versatility and finally, volition. These taxonomies will be used when describing the sound design techniques later on. Key classification methods are shown in Table II.

1.4. Evolutionary computation for sound design

Evosynth, the novel sound design tool presented in this paper, uses an interactive genetic algorithm (IGA) to help a user interactively search the space of possible sounds that it can produce. Its technical implementation is described in detail in section 2 but let us consider previous work using similar types of algorithms. IGAs are a subset of genetic algorithms (GAs) that are heuristic search algorithms inspired by evolution theory [Goldberg and Holland 1988]. GAs are used to iteratively search highly complex spaces for useful items, where in each iteration, candidate items are selected and rated using a fitness function. Highly rated items are mutated and recombined to generate the next set of candidate items, until eventually a highly desirable or fit item is found. In order for these 'genetic operations' to be possible, the item is normally encoded as a manipulable data structure called a genome. In a classic example from robotics, the genome is a list of real values that can be

interpreted into a neural net controller for a robot, fitness is the ability of the controller to move the robot forwards, and fitness is measured by using the candidate controllers to move a robot in simulation [Husbands et al. 1998]. Typically, GAs have an automated fitness function which allows many generations of candidates to be assessed without human intervention. Contrastingly, IGAs have an *interactive* fitness function, where a human user assigns fitnesses to the candidates. In a discussion of IGAs based on Dawkins' seminal IGA, Biomorphs [Ruse 1987, p55-74], Smith states two characteristics of optimization problems to which IGAs can be applied: 1) candidates can be generated and assessed in real time; 2) candidates can be assessed by humans but not by an algorithm [Smith 1991].

GAs with automated fitness functions and IGAs with human fitness functions have both been used to design sounds. The seminal work where parameter settings for fixed architecture FM synthesizers were evolved with an automated fitness function is attributed to Horner et al. [Horner et al. 1993]. Here, the fitness was the distance between the frequency spectra of candidate sounds and a target sound and the GA was able to automatically program the FM synthesizer to make a desired sound. Yee-King and Roth introduced the idea of automatically programming commercially available, fixed architecture synthesizers, and the use of the more perceptually meaningful MFCC feature vectors with their SynthBot system [Yee-King and Roth 2008]. Based on the reported performance of SynthBot, the problem of automatically programming basic, fixed architecture synthesizers to make desired sounds can be considered partially solved as it was able to consistently locate sounds in its sound space, though only for fairly simple synthesizers [Yee-King 2011, p97]. Addressing the problem of variable architecture sound synthesizers as opposed to optimizing parameters for fixed architectures, Macret and Pasquier evolved complete PureData patches to match a target sound [Macret and Pasquier 2014]. As a side note, they state that a single objective GA was not successful in evolving presets for the OP-1 synthesizer, which is more complex than the synthesizers Yee-King and Roth used in Synthbot. The work with variable architectures can be traced back to that of Garcia and perhaps Chinen, both of whom evolved variable synthesis architectures [Garcia 2001], [Chinen and Osaka 2007]. Moving to IGAs, Yee-King's AudioServe system evolved variable architecture modular synthesis patches, working as an online tool and an interactive sound artwork, using a human fitness function [Woolf and Yee-King 2003]. Dahlstedt implemented an IGA that could search the space of synthesis patches for the Nord Modular synthesizer [Dahlstedt and Clavia 2006]. That system was actually built into the commercial Nord editing software.

Figure 1. The Eurorack format modular synthesizer used in the case study, featuring a range of different modules including sequencers, oscillators, filters and envelope generators. Photo credit: Dom Mino.

1.5. Modular synthesis

In this section, background information is provided about modular synthesis, providing context for the comparison between modular synthesis and Evosynth presented later in the paper.

Modular sound synthesizers are synthesizers that are made out of several discrete modules, where a range of different module types provide different functionality. Typical module types are oscillators, filters, envelope generators and step sequencers. New modules can be added to increase the complexity of the synthesizer. They are programmed by connecting the inputs and outputs of the modules together with patch cables. It is the ability to add new modules and the patch programming method that distinguish modular synthesizers from fixed architecture synthesizers. Modules from different manufacturers can be combined in the same synthesizer and the most popular module format is probably Eurorack. A typical modular synthesizer is shown in Figure 1. The modular synthesizer market has grown quite rapidly from 2009 to 2015: James describes the transformation of the Eurorack modular synthesizer market from a handful of manufacturers making a few modules in 2009 to 80 manufacturers making over 700 different modules in 2013 [James 2013]. In 2015, the number of different modules in Eurorackdb, a database of Eurorack modules, stood at 1033 [Noble 2015]. There are a range of factors contributing to the rebirth of the modular synthesizer. Dieter Doepfer, the inventor of the Eurorack format attributes it to the hands-on aspect, the fullness of the sound and the potential for unique sounds [Ableton 2010]. One might also consider other factors such as the rise of small-scale manufacturing, hardware hacking and so forth but this is beyond the scope of this study. Modular synthesizers are normally thought of as real, typically analogue devices but there are many virtual modular synthesizers which aim to recreate the modular synthesizer experience using a two dimensional representation of modules and cables. The Arturia Modular V is an example of a software modular synthesizer. The Evosynth system described in this paper can be thought of as a virtual modular synthesizer in that its synthesis engine uses a modules and wires metaphor.

1.5.1. Description of a modular synthesizer programming session. In order to provide a more direct description of modular synthesis, this section contains a brief reflection upon the author's first experience programming a modular synthesizer. Figure 1 shows the system that was used for this

programming session, which consisted of 4 ‘tracks’ of modules without a keyboard.

Most modules had unfamiliar names meaning I did not know what the function of most of them was. I did not have a manual to hand, so decided to start with the most obviously named modules: oscillator and mixer. I patched the oscillator into the mixer and heard a simple sine waveform. I then began to look for sources of modulation, first locating something that had pulsing lights and patching that to the frequency input, resulting in a slow, pulse wave frequency modulation. I explored various sequencers and other modulation sources in the rig, and eventually ended up with a quite rich, complex timbre, using a combination of triggers, envelopes and oscillators. The process I went through was exploratory, testing out the outputs at various sockets against the frequency input of the familiar oscillator to see what kind of signal they produced. I was searching in a quite haphazard way for increasingly complex and dynamic timbres. I began using layering, where a single modulation is patched to multiple oscillators, some of which respond in non linear ways to the input. This produced a rich, dynamically synchronized movement that I was able to gradually adapt. Listening back to the recordings of the session, I found that the sound moved quite gradually from one timbre to the next, with occasional leaps in timbre space when I discovered a new layer or new modulation.

The session was recorded and is used in the comparative analysis presented later in the paper.

In summary, the key observations were an exploratory programming process with gradual increase of patch and timbral complexity, frequent use of serendipitous timbres and high commitment to a patch, which was not easily reset.

2. EVOSYNTH

In this section, the Evosynth sound design system will be described. Evosynth is a sound design tool that allows its users to interactively evolve sound synthesis patches with variable numbers of modules and connections. It is an example of a computationally creative system, a system which supports a human sound designer by carrying out the synthesizer programming part of the sound design process, leaving the sound designer to simply express their preferences for the sounds that are generated. It will be evaluated using some of the methods listed in Tables I and II: a survey, a classification analysis placing it in various taxonomies of creative systems and an output analysis looking at the range and modes of sound creation it allows.

2.1. Technical implementation

2.1.1. User interface and workflow. Figure 2 shows the Evosynth user interface running on an iPhone and Figure 3 shows an annotated version, illustrating the key features of the interface. A typical workflow consists of the following steps:

(1) Load the page and read the instructions that are displayed when first loading.

(2) Click on the play buttons for some sounds and listen to them.

(3) Click on the breed buttons for sounds that are liked.

(4) Hit the main breed button to carry out the breed operation.

(5) Back to 2.

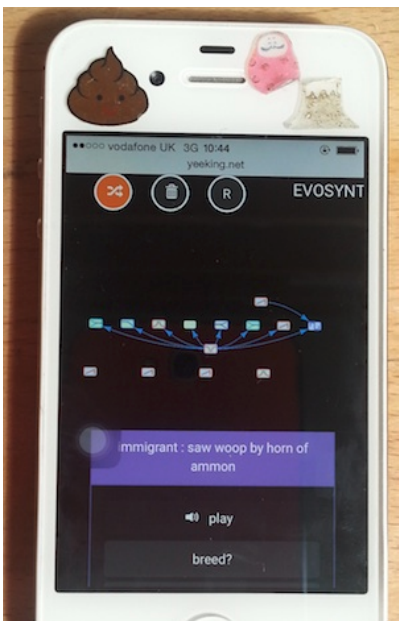


Figure 2. Evosynth running on an iPhone 4S in the Safari browser under iOS 8.

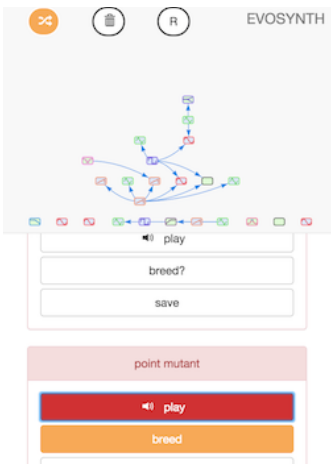


Figure 3. The

Evosynth user interface, with colors switched to white for clarity. 1) breed, 2) reset, 3) record, 4) circuit visualization, 5) mutant type, 6) play, 7) select for breeding, 8) save to server.

Additionally, users can save sounds to the server, at which point they are prompted to name them. A record button allows the user to record the current sound then to download it as a WAV file. Finally, a visualization of the sound synthesis circuit is provided, which allows the user to see which modules the circuit is made from and how they are connected. However, this is automatically generated, and the circuits can be quite complex, so it is not necessarily very comprehensible. An example of the visualization of a circuit is shown in Figure 4.

The system is implemented using the Web Audio API, which enables real-time sound synthesis in compatible web browsers, meaning the audio engine works on desktop operating systems as well as Android and iOS [Rogers 2012]. The user interface is implemented using responsive HTML and CSS, on top of the Twitter Bootstrap API and jQuery [De Volder 2006]. The circuit visualization is generated using the network plotting component of the vis.js library [De Jong and Pazienza 2013].

2.1.2. Sound circuit description and instantiation. The sound synthesis method broadly follows the typical unit generator graph model as first defined by Mathews [Mathews et al. 1969, p126]. It might also be described as a virtual, modular synthesizer. A variable number of different modules are wired together into a circuit, and one of them is wired to the audio output. The sound synthesis circuits are stored as variable length arrays of float values or *genomes*. Each set of 10 floats in a genome (a *gene*) describes a single functional unit in the synthesizer, along with its position in a 2D space and a connection arc within which it will connect to other modules. The 10 parameters contained in a gene and their meaning are shown in Table III.

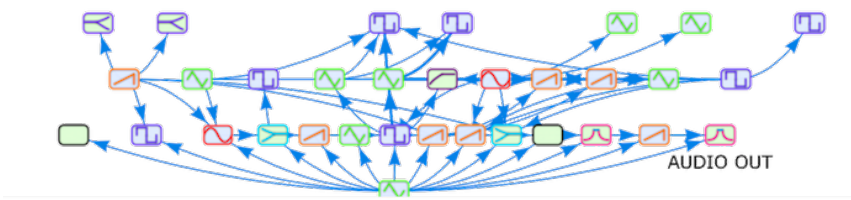


Figure 4. The Evosynth circuit visualizer. The boxes are modules in the circuit and the blue lines are connections from one module to another. One module is labeled with 'audio out' that indicates that this module is the one that is patched to the audio output.

Table III. The 10 synthesizer parameters defined in a single gene.

Name	Description	Normalised range
u	ugen type	oscillator/ filter
s	ugen subtype	sine, square, sawtooth, triangle / lowpass, highpass, bandpass, lowshelf, highshelf, allpass, notch
x	position	0-1
y	position	0-1
θ^1	connection arc angle	0-2π
θ^2	connection arc angle	0-2π
r	arc radius	0-1
p^1	initial value 1	
p^2	initial value 2	
i	connection target	0-no. in ports on target

There are two main module types: oscillator and filter and each has multiple subtypes. Schematics of the inputs and outputs of the oscillator and filter modules are provided in Figures 6 and 7, respectively. The initial value parameters from the gene, p^1 and p^2 are used in various ways to configure the unit. The initial frequency of an oscillator, f_1 which is the frequency at which it oscillates in the absence of incoming modulation, is defined according to equation 1. The frequency gain scalar f_2 for an oscillator, which will scale incoming modulation signals is defined according to equation 2 where p^2 is used as the index in a 10 element array a defined in equation 3. The starting cut off/center frequency of a filter f_3 is defined according to equation 4. The cut off/center frequency gain scalar for a filter is 2000. The

starting Q value for a filter is defined in equation 5 and the modulation gain for Q is 100.

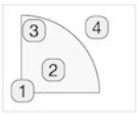


Figure 5. An illustration of how connection arcs work. Only the arc for module 1 is shown. Module 1 will connect to modules 2 and 3 but not 4.

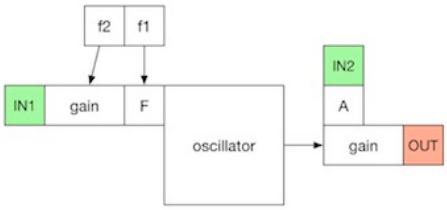


Figure 6. A schematic of an oscillator module. F is the frequency control, set to an initial value f_1 and modulated by input IN1 in the range $0 - f_2$. IN2 allows amplitude modulation. f_1 and f_2 are defined in equations 1 and 2.

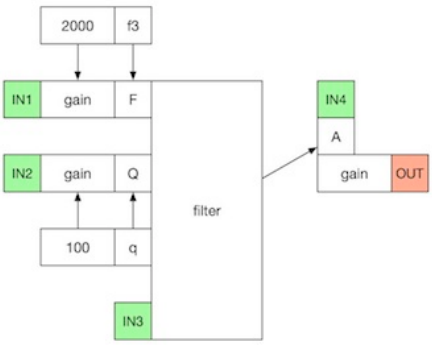


Figure 7. A schematic of a filter module. F is the cut off or center frequency, set to an initial value f_3 and modulatable in the range 0-2000. Q is the filter Q control, set to an initial value q and modulatable in the range 0-100. IN1 is the cut-off/center modulation input, IN2 is the Q modulation input, IN3 is the signal input to the filter and IN4 is the amplitude modulation input. f_3 and q are defined in equations 4 and 5.

$$f_1 = 200p^1 \tag{1}$$

$$f_2 = 200(p^1.a[p^2.10]) \tag{2}$$

$$a = [0.25, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] \tag{3}$$

$$f_3 = 2000p^1 \tag{4}$$

$$q = 100p^2 \tag{5}$$

When a user clicks the play button, the genome for the selected synthesizer is interpreted gene by gene into a set of modules. Per gene, a module is placed into the space, its connection arc is placed over pre-existing modules and connections are made, as shown in Figure 5. Therefore, the first module in the space is likely to be the most highly connected into. The first module is finally wired into a dynamic range limiter to prevent excessive volume and then into the audio output. Considering the circuit shown in Figure 4, the first module is a band pass filter into which are wired a whole network of other modules. The visualization actually uses an automatic hierarchical network display to place the modules in space, as this was found to most clearly show the connections, as opposed to an accurate placement using the x and y parameters. The initial population of circuits with which the user is presented consists of 25 random circuits with 40 modules each. 40 modules was chosen as it tended to provide circuits with a decent range of timbres, without too many silent circuits. Once users had started saving sounds to the server, up to 10 additional sounds from the server would be included in the population each time a breed operation was carried out, to allow the user to cross breed between their population and sounds made by other users.

2.1.3. Genetic operators. Once a user has selected one or more sounds for breeding, the genetic operators are applied, namely point, grow, repeat and shrink mutations and crossover. Point mutation selects several parameters and adds a random, small positive or negative value to them. Grow mutation adds a new, random gene to the genome. Repeat mutation takes a copy of one of the existing genes and adds it to the end of the genome. Shrink mutation deletes a gene. Crossover takes two genomes and knits them together into a new genome. When generating a new population, a variety of types of 'mutant' are generated and placed into the population. The type of mutant is also indicated to the user, as shown in Figure 3.

The Evosynth system has now been described technically. The source code for the complete system and server, including the database of saved sounds is available on Github [Yee-King 2015]. In the following sections, a classification and an output analysis of Evosynth will be presented.

3. METHOD: COMPARING MODULAR SYNTHESIS TO EVOSYNTH

Having now described the two sounds synthesis systems, the methods that will be used to compare them will be introduced.

3.1. Survey

It was judged that a simple survey instrument would be an effective way to sample the 'vox populi' of modular synthesizer users. In order to maximize response numbers, 5 simple but interesting questions were asked with Likert scale response options, with the response options taken from [Brown 2010]. The questions were as follows:

(1) I start with a very simple patch and work from there. (Never, Rarely, Sometimes, Very Often, Always)

(2) I start with a target sound in mind. (Never, Rarely, Sometimes, Very Often, Always)

(3) Unexpected sounds lead me in new directions when patching. (Never, Rarely, Sometimes, Very Often, Always)

(4) The modular synth itself is creative. (Strongly Disagree, Disagree, Undecided, Agree, Strongly Agree)

(5) My modular rig is: (Non-existent, Small, Medium, Large)

A Google Form survey was posted to Twitter, targeted at the user accounts of some well connected modular synth makers and resellers. The survey was made deliberately simple to maximize response numbers. However, it should be noted that not all sections of the modular synthesizer community use Twitter, and this limits the power of the survey. In parallel to this survey, a similar survey was connected to the Evosynth website which aimed to ask related questions to Evosynth users. The questions were as follows:

(1) Evosynth itself is creative. (Strongly Disagree, Disagree, Undecided, Agree, Strongly Agree)

(2) Breeding resulted in similar sounds to the parents. (Never, Rarely, Sometimes, Very Often, Always)

(3) I was able to work towards specific sounds I had in mind. (Never, Rarely, Sometimes, Very Often, Always)

(4) Unexpected sounds lead me in new directions. (Never, Rarely, Sometimes, Very Often, Always)

(5) I enjoyed listening to the sounds. (Strongly Disagree, Disagree, Undecided, Agree, Strongly Agree)

3.2. Ontological analysis

The ontological analysis involved placing the synthesis systems into pre-existing ontologies that have been used in the literature to classify 'creative systems'.

3.3. Output analysis

The output analysis involved working with recordings of the output of the two systems and comparing them in different ways. To create output data for the modular synthesizer, a 15-minute segment from the middle of the author's first modular synthesizer programming session (described in section 1.5.1) was selected. This extract was chosen as it represent a section in the recording where the author felt they were in 'flow', with a steady progression through timbres, akin to the kind of progression that Evosynth was designed to enable.

The 15-minute recording was sliced into 5 second, non overlapping segments which were stored in separate WAV files such that the range of timbres could be further examined. Features were extracted from the segments using Collins' SuperCollider MIR library as it offered a simple means to extract features from multiple files and write them out in CSV format [Collins 2011]. The feature extractors used were MFCC, Loudness, Spectral Centroid, Spectral Percentile, Spectral Flatness, FFT Crest from 0 to 2000Hz, FFTCrest from 2000 to 10000Hz, FFT Spread, FFT Slope, Sensory Dissonance and Onsets. This is the default set used in the SCMIR examples and it offers a sensible range of perceptual features. The resulting feature vectors had 2000 dimensions.

The next step was to create equivalent data to the above for evosynth. Noting that the segment of the modular synthesizer session was chosen by looking for a section where the programmer felt they were in flow, smoothly working through a series of programming steps, an Evosynth session was selected where there was a sequence of regular 'select for breeding' events spanning a similar length of time (15 minutes) to the modular synth session. This was possible as every time a user

clicked on the 'select for breeding' button, the genome of the selected synthesizer was stored to the database along with a timestamp. The chosen session belonged to an anonymous user of the online system. The genomes that were selected for breeding were interpreted into running synths and rendered into a set of 5 second long WAV files, from which the set of features described above were extracted. They were also concatenated into a single file for comparison to the modular recording.

4. RESULTS

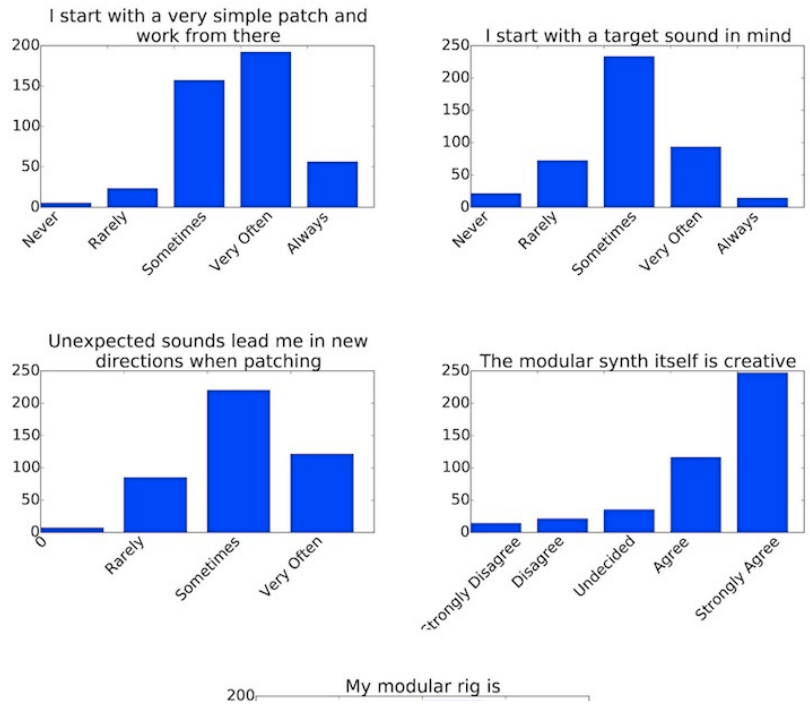
4.1. Survey of modular synthesizer users

432 people responded to the modular synthesizer survey and the questions and response statistics are shown in Figure 4.1. 84% of respondents reported that the modular synth itself is creative. The intention of this question was to find out if users viewed their synth as a creative system in the Boden or Wiggins sense, but it is very likely the respondents did not grasp this intention, not being experts in this field so we cannot conclude much here. However, considering the question 'Unexpected sounds lead me in new directions when patching', 79% responded with 'sometimes' or 'very often'. The intention of this question was to probe if users were assigning value to sounds which they did not intentionally create, by being lead in new directions by these sounds. According to Wiggins, creative systems should generate artifacts having quantifiable novelty and value. Two of the other questions investigate the programming process: working from a simple patch as a starting point is reported as very common, starting with a target sound in mind is reported mostly as 'sometimes', implying that exploratory programming is common.

4.2. Survey of Evosynth users

During the two weeks following the announcement of Evosynth on Twitter, 90 sounds were saved to the server by users and 3552 breed events were recorded from 229 unique IP addresses. Most of the sessions involved less than 10 breed operations, but 60 of the 229 involved 10 or more breed operations, probably signifying engaged use of the system. Six users logged over 100 breed operations in their sessions, implying prolonged use of the system. Unfortunately, only 14 of the 200+ users responded to the survey; the results are shown in Figure 9.

The number of respondents to this survey is much lower than for the modular survey, likely due to the added commitment of having to first learn and use the Evosynth system. To some extent this survey aimed to test how much control users felt they had over the sound design process--a majority felt they were able to work toward sounds they had in mind, despite having no direct control over the circuits. The breeding process seemed to be reasonably effective at producing iterative movement through timbre space as people felt that offspring tended to sound like the parents. People seemed to consider Evosynth to be creative in itself, but again it is not clear that they understood the intent of the question. Finally, users enjoyed the sounds. This is a positive response to the system, but a more elaborate survey would be required to probe more details. It should also be noted that the surveys contain different questions and they were answered by different groups of people. The responding groups are not necessarily representative of any particular group, e.g. modular synth users or Evosynth users.



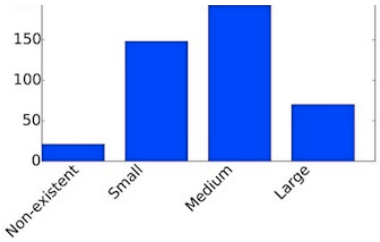


Figure 8. Results of a survey of 432 modular synthesizer users.

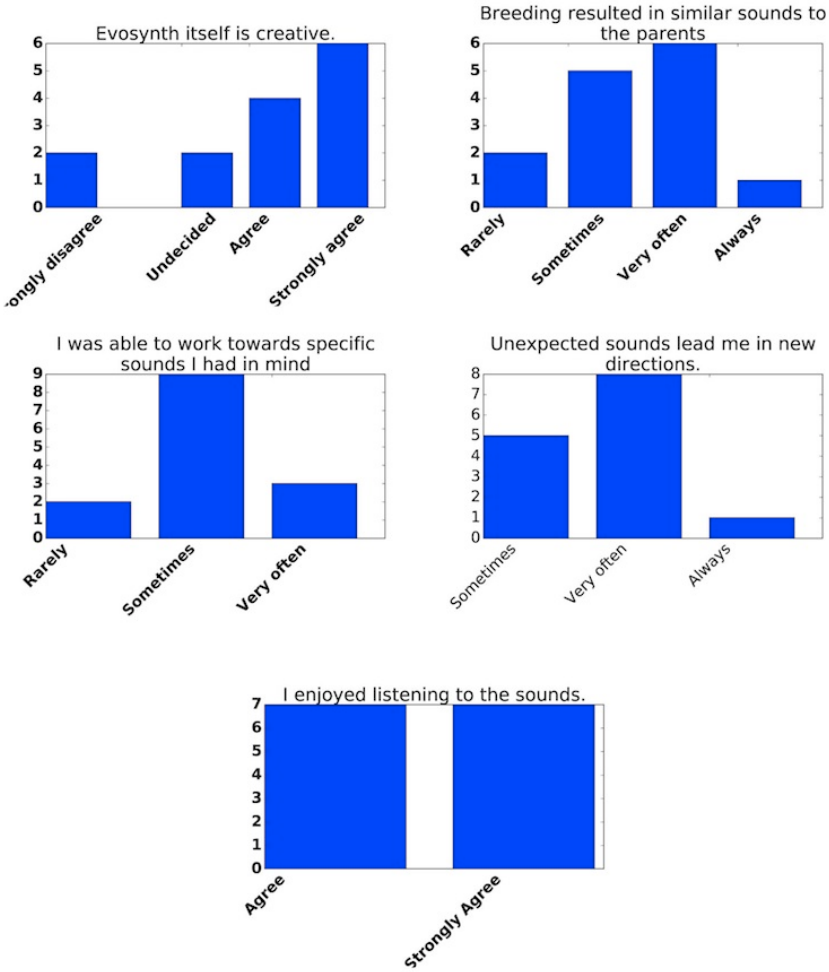


Figure 9. Results of a survey of 14 Evosynth synthesizer users.

4.3. Ontological analysis of modular synthesizers

Where does the modular synthesizer sit in the ontologies of creative systems mentioned at the beginning of the paper? Considering Boden and Edmonds classification of generative art systems, the modular synthesizer is capable of generative art or G-artifacts are made using a process not directly under the control of the artist. It is an interactive (I-art) system, since the sound designer interactively creates the artifact. Considering Eigenfeldt et al.'s taxonomy of musical metacreation, there is a close match to the lowest level of autonomy, i.e. independence, with their level 1 example

of ‘an LFO based upon a complex mathematical function whose output may not be entirely predictable’. The modular synthesizer can possibly stretch to level 2, compositional autonomy: consider a 2D matrix step sequencer controlled by two complex LFOs, as commonly found in modular set ups for drum sound triggering. Finally, regarding Wiggins’ formal descriptions of creative systems, further work is needed to establish what precisely is the value of the ‘unexpected sounds’ encountered whilst programming modular synthesizers, but clearly there is some sort of interplay between sounds proposed by the programming process and sounds explicitly designed by the human programmer.

4.4. Ontological analysis of Evosynth

In the introductory discussion of taxonomies for classifying ‘creative’, computational systems, Boden and Edmonds’ taxonomy of systems that generate art [Boden and Edmonds 2009] provided several terms that can be applied to Evosynth. It makes generative-art (G-art), where artifacts are made using a process not directly under the control of the artist. It makes evolutionary art (Evo-art), where artifacts are ‘evolved by processes of random variation and selective reproduction that affect the art-generating program itself.’ Finally, as a system, it is interactive (I-art), since the selection of individuals for reproduction is carried out interactively by the human sound designer, not autonomously. Next one can consider Eigenfeldt et al.’s taxonomy of musical metacreation systems or G-art systems that work in the musical domain, structured from least to most autonomous [Eigenfeldt et al. 2013]. Evosynth deals with sound, not symbolic musical material but it is certainly *generative*, since it ‘generates and/or substantially varies pre-generated sequences’ and it is somewhat *proactive*, since it is ‘able to initiate [its] own musical gestures’--it proposes new synthesis patches to the user. Therefore, it achieves a moderate level of autonomy on their scale.

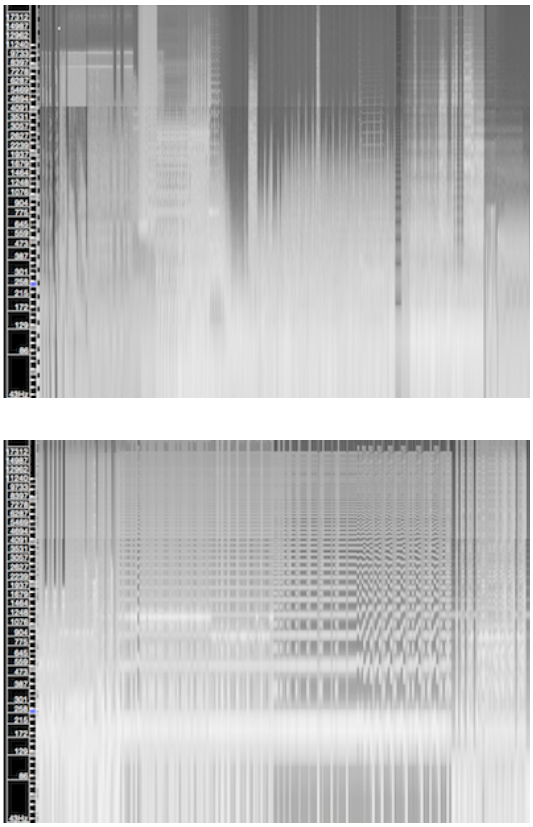
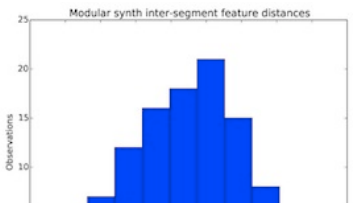
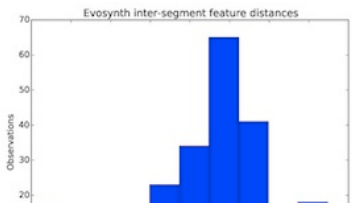


Figure 10. Spectrograms of a 15-minute modular synth session (above) and a 15-minute Evosynth session (bottom).

4.5. Output analysis of modular synthesis and Evosynth

The output analysis was carried out on a single modular synth session and a single Evosynth session, which were conducted by different people. It has not been established if these were typical examples of either, but they do provide some grounds for discussion. The spectrograms of the two sessions are shown in figure 10, with the modular on the left and Evosynth on the right. Considering the modular, It is clear from this spectrogram that a wide range of timbres was explored in this extract. Another feature is the distinct changing in timbre that was associated with adding and removing patch cables. The Evosynth spectrum *looks* less varied, with many correlated regions suggesting exploration of a more limited timbre space.





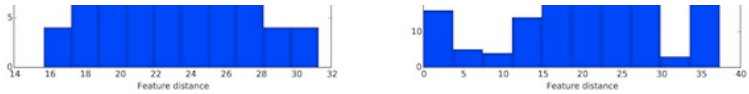


Figure 11. Inter-segment feature distance distribution from the synthesizer programming sessions shown in Figure 10. The modular synth is on the left, Evosynth on the right.

To investigate this further, the sessions were sliced into 5 second segments which were arranged chronologically. Features were extracted (as described in section 3.3) and the Euclidean inter segment feature distances were measured. The distributions of these distances are shown in Figure 11. Lots of high distances imply lots of large movement in feature space. Both sessions exhibited broadly similar movement in timbre space between segments, though there were more very small and very large jumps in Evosynth. There are several reasons this might occur, for example some mutations would produce tiny jumps and completely different genomes can be selected from the database in in the middle of a session, producing large jumps.

For the final analysis, the 5-second segments from both sessions were combined to a single dataset and a principle component analysis (PCA) was applied to the features. 41% of the variance in the dataset was explained by the first two coefficients of the PCA, so the 2D plot shown in Figure 12 is a reasonable representation of the relative distances between the vectors. (Such plots are typical of the literature, e.g., Grey [1977] and Halpern et al [2004].) There is limited overlap between the two sounds sets, implying the timbres within each where quite different. The spread of timbres for the Evosynth session actually looks slightly greater, implying a wider range of timbres, with the modular timbres bunched together. Again, this is an interesting area of discussion but it should be stated that neither the modular session nor the Evosynth session were necessarily representative of typical programming sessions.

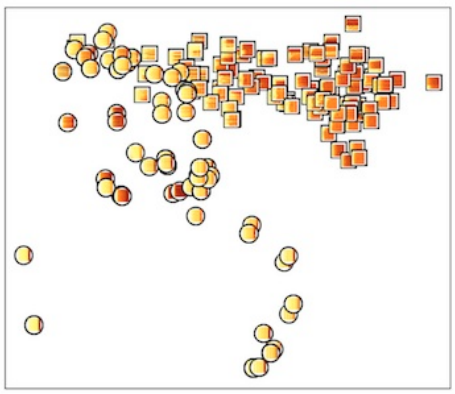


Figure 12. Results of running PCA on both sets of files. The *x* and *y* positions are the first two PCA coefficients, circles are Evosynth sounds and squares are modular synth sounds.

5. CONCLUSION

This paper has described and compared two sound design methods: hardware modular synthesis and a novel virtual modular synthesizer called Evosynth, which is programmed via an interactive genetic algorithm. Each of the two methods were investigated through a survey, through classification into ontologies of creative systems, and through analysis of their timbral outputs. The surveys sampled user opinions about modular synthesis and Evosynth, relating to the way they present sonic novelty to the user and other aspects of the programming experience. The classification analyses identified established ontologies of creative systems and placed the two sound design methods therein. The output analyses considered the timbral range of the two methods and how users explore that range.

The survey analysis demonstrated that in both cases, users were willing to make use of unexpected sounds and that they commonly used an exploratory approach to sound design. The classification analysis indicated that the modular synthesizer encroaches into the creative systems area, with its generation of valued, novel output and its fine and coarse level algorithmic pattern generation capabilities. Similarly, Evosynth generates valued novelty and has medium level autonomous capabilities. The output analysis shed some light on the range of timbres that were explored during modular and Evosynth programming sessions. The modular session contained a fairly tight cluster of timbres compared to the wider range explored in the Evosynth session. The movement in timbre space during the sessions was fairly similar, with the Evosynth session showing more very small and very large movements. It is not clear if this observation is typical of modular and Evosynth programming sessions however.

In future work, Evosynth could be developed to better support the aims of the user, perhaps using the growing dataset of selected and saved sounds to map out the timbral space. It would be interesting to gather a set of different recordings of modular programming sessions, and carry out an overarching output analysis which extends upon the work described here, which might also feed into the design of the underlying Evosynth exploration engine. It would also be interesting to investigate further the culture and techniques that are developing in the modular synthesis scene.

ACKNOWLEDGMENTS

The author wishes to thank all the folk who took the time to use Evosynth and to take the surveys, various people who gave excellent feedback about Evosynth during its development and Dom Mino for lending me the modular synthesizer.

REFERENCES

Ableton. 2010. Dieter Doepfer: Completing the Circuit. (2010). **https://www.ableton.com/en/blog/ (http://www.ableton.com/en/blog/)** dieter-doepfer-completing-the-circuit/

M.A. Boden and E.A. Edmonds. 2009. What is generative art? *Digital Creativity* 20, 1/2 (2009),21–46. DOI:**http://dx.doi.org/10.1080/14626260902867915 (http://dx.doi.org/10.1080/14626260902867915)**

S. Brown. 2010. *Likert Scale Examples for Surveys Dichotomous Scales : Three-Point Scales :*. TechnicalReport.

M. Chinen and N. Osaka. 2007. Genesynth: Noise Band-based Genetic Algorithm Analysis/Synthesis Framework. In *Proceedings of the ICMC 2007 International Computer Music Conference*. ICMA.

N. Collins. 2008. The Analysis of Generative Music Programs. *Organised Sound* 13, 03 (2008), 237–248. DOI:**http://dx.doi.org/10.1017/S1355771808000332 (http://dx.doi.org/10.1017/S1355771808000332)**

N. Collins. 2011. SCMIR: A SuperCollider music information retrieval library. In *Proceedings of the International Computer Music Conference*. Ann Arbor, MI: MPublishing, University of Michigan Library.

P. Dahlstedt and Clavia. 2006. Nord Modular G2 Patch Mutator. (2006). **http://www.clavia.se/products/ (http://www.clavia.se/products/)** nordmodular/MutatorManual.pdf

J. De Jong and G.E. Pazienza. 2013. Browser-Based Graph Visualization of Dynamic Data withVisGraph. In *Graph Drawing*. Springer, 518.

K. De Volder. 2006. JQuery: A generic code browser with a declarative configuration language. In *Practical Aspects of Declarative Languages*. Springer, 88–102.

A. Eigenfeldt, O. Bown, P. Pasquier, and A. Martin. 2013. Towards a Taxonomy of Musical Metacreation : Reflections on the First Musical Metacreation Weekend. In *Workshop on Musical Metacreation*. 40–47.

R. Garcia. 2001. Growing sound synthesizers using evolutionary methods. In *Proceedings of ALMMA 2002. Workshop on Artificial Models for Musical Applications*. Citeseer, 99–107.

D. Goldberg and J. Holland. 1988. Genetic Algorithms and Machine Learning. *Machine Learning* 3 (1988), 95–99. DOI:**http://dx.doi.org/10.1023/A:1022602019183 (http://dx.doi.org/10.1023/A%3A1022602019183)**

J.M. Grey. 1977. Multidimensional perceptual scaling of musical timbres. *The Journal of the Acoustical Society of America* 61, 5 (1977), 1270–1277. DOI:**http://dx.doi.org/10.1121/1.381428 (http://dx.doi.org/10.1121/1.381428)**

J.P. Guilford. 1967. *The nature of human intelligence*. Vol. 5. McGraw-Hill. 249–256 pages.

A.R. Halpern, R.J. Zatorre, M. Bouffard, and J.A. Johnson. 2004. Behavioral and neural correlates of perceived and imagined musical timbre. *Neuropsychologia* 42, 9 (2004), 1281–1292. DOI:**http://dx.doi.org/10.1016/j.neuropsychologia.2003.12.017 (http://dx.doi.org/10.1016/j.neuropsychologia.2003.12.017)**

A.Horner, J. Beauchamp, and L, Haken. 1993. Genetic Algorithms and Their Application to FM, Matching Synthesis. *Computer Music Journal* 17, 4 (1993), 17–29.

P. Husbands, N. Jakobi, T. Smith, and M. O'Shea. 1998. Better Living Through Chemistry: Evolving GasNets for Robot Control. *Connection Science* 10, 3/4 (1998), 185–210. DOI:**http://dx.doi.org/10.1080/095400998116404 (http://dx.doi.org/10.1080/095400998116404)**

A. James. 2013. The Secret World Of Modular Synthesizers. *Sound on Sound Magazine* (Apr 2013).

J. Kaye. 2007. Evaluating experience-focused HCI. In *CHI '07 extended abstracts on Human factors in computing systems. CHI '07*. 1661. DOI:**http://dx.doi.org/10.1145/1240866.1240877 (http://dx.doi.org/10.1145/1240866.1240877)**

C. Kiefer, N. Collins, and G. Fitzpatrick. 2008. HCI Methodology For Evaluating Musical Controllers: A Case Study. *Proceedings of the 2008 International Conference on New Interfaces for Musical Expression (NIME-08)* (2008), 87–90. **http://sro.sussex.ac.uk/37012/http://www.nime.org/ (http://sro.sussex.ac.uk/37012/http%3A//www.nime.org/)** proceedings/2008/nime2008

M. Macret and P. Pasquier. 2014. Automatic Design of Sound Synthesizers as Pure Data Patches using Coevolutionary Mixed-typed Cartesian Genetic Programming. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*. 309—-316.

M.V. Mathews, J.E. Miller, F.R. Moore, J.R. Pierce, and J.C. Risset. 1969. *The Technology of Computer Music*. The MIT Press.

J. Noble. 2015. The Eurorack Database--Modules. (2015). **http://www.eurorackdb.com/node/h (http://www.eurorackdb.com/node/h)**

C. Rogers. 2012. Web Audio API. W3C Working Draft. (2012).

R. Rowe.1992. *Interactive Music Systems: Machine Listening and Composing*. MIT Press Cambridge, MA, USA.

M. Ruse.1987. The blind watchmaker. *Trends in Ecology & Evolution* 2 (1987), 81–82; DOI:**http://dx.doi.org/10.1016/0169-5347(87)90158-3 (http://dx.doi.org/10.1016/0169-5347(87)90158-3)**

A. Seago, W. Hall, S. Holland, and P. Mulholland. 2004. *Synthesizer user interface design--lessons learned from a heuristic review*. Technical Report.

J.R. Smith. 1991. Designing Biomorphs with an Interactive Genetic Algorithm. In *ICGA*. 535–538.

P.N. Stern. 1980. Grounded theory methodology: its uses and processes. *Image* 12, 1 (1980), 20–23; DOI:**http://dx.doi.org/10.1111/j.1547-5069.1980.tb01455.x (http://dx.doi.org/10.1111/j.1547-5069.1980.tb01455.x)**

D. Stowell, A. Robertson, N. Bryan-Kinns, and M.D. Plumbley. 2009. Evaluation of live human-computer music-making: Quantitative and qualitative approaches. *International Journal of Human Computer Studies* 67, 11 (2009), 960–975. DOI:**http://dx.doi.org/10.1016/j.ijhcs.2009.05.007 (http://dx.doi.org/10.1016/j.ijhcs.2009.05.007)**

R. Tubb and S. Dixon. 2014. The Divergent Interface: Supporting Creative Exploration of Parameter Spaces. In *Proceedings of the International Conference on New Interfaces for Musical Expression.* 227–232. **http://www.nime.org/proceedings/2014/nime2014 (http://www.nime.org/proceedings/2014/nime2014)**

G.A. Wiggins. 2006. A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems* 19, 7 (2006), 449–458. DOI:**http://dx.doi.org/10.1016/j.knosys.2006.04.009 (http://dx.doi.org/10.1016/j.knosys.2006.04.009)**

T. Wilmering, G. Fazekas, and M.B. Sandler. 2013. The Audio Effects Ontology. In *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR-2013)*. 215–220.

S. Woolf and M. Yee-King. 2003. Virtual and Physical Interfaces for Collaborative Evolution of Sound. (2003); DOI:**http://dx.doi.org/10.1080/0749446032000150861 (http://dx.doi.org/10.1080/0749446032000150861)**

M.J. Yee-King. 2015. Evosynth: ACM-CIE DOI with data. (Oct 2015); DOI:**http://dx.doi.org/10.5281/zenodo.32553 (http://dx.doi.org/10.5281/zenodo.32553)**

M.J. Yee-king and M. Roth. 2008. Synthbot: An unsupervised software synthesizer programmer. *Proc. International Computer Music Conference* Oct. (2008), 184–187. **http://citeseerx.ist.psu.edu/ (http://citeseerx.ist.psu.edu/)** viewdoc/download;jsessionid=BCEF14F32EB95858E07402CDCA6D249D?doi=10.1.1.329.2079

M.J. Yee-King. 2011. *Automatic sound synthesizer programming: techniques and applications*. Ph.D. Dissertation. University of Sussex.

You must have an ACM Web Account to comment on this article, Sign In or Sign Up.

Sign In

Sign in using your ACM Web Account username and password to access premium content if you are an ACM member, CIE subscriber or Digital Library subscriber.
If you are a member of the general public, you may set up a Web Account to comment on free articles. ACM membership is not required.

Username

Password

» Forgot Password?
» Create a free Web Account

SIGN IN

Get Access

Join the ACM. Become a member to access premium content and site features, and take full advantage of ACM's outstanding computing information resources, networking opportunities, and other benefits.

JOIN

ACM

(HTTP://CAMPUS.ACM.ORG

/PUBLIC

/QJ/QUICKJOIN

/INTERIM.CFM)