Return to Index

Creating and Exploring the Huge Space Called Sound: Interactive Evolution as a Composition Tool

Palle Dahlstedt
Innovative Design
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
+46-31-772-1000
palle@design.chalmers.se

Abstract

This paper introduces a program that applies the principles of interactive evolution, i.e., the process of repeatedly selecting preferred individuals from a population of genetically bred sound objects, to different synthesis and pattern generation algorithms. This allows for aural real-time exploration of complex sound spaces, and introduces the task of constructing sound engines and instruments customized for this kind of creation and exploration. Several such sound engines are presented together with sound examples and a discussion of compositional applications.

Keywords: interactive evolution, genetic algorithms, music composition, sound design, sound synthesis

The sound examples mentioned in the paper can be found on the following URL:
http://www.design.chalmers.se/palle/mutasynth

1. Introduction

A number of complex sound synthesis tools are used in the composition of electronic music. These often involve a large number of parameters, which makes them difficult to explore by hand. The algorithms have great potential but are difficult to use. There are different methods for exploring these huge parameter spaces, which may consist of 30-500 parameters or more. In this paper I introduce a program, called MutaSynth, that applies interactive evolution to sound synthesis and pattern generation. An external sound engine is remotely controlled through a genetic representation of its parameter set. Variations on these genomes are generated, and the resulting sound objects are auditioned and selected according to the user's esthetic preferences. This process is repeated until a satisfying result has been reached.

Since the program can be used with any sound engine that can be remote controlled by MIDI messages, many different synthesis techniques can be explored, and widely different sound objects such as rhythmic loops, aperiodic gestures and playable keyboard instruments can be evolved in a simple interactive process. Several examples of sound engines created specifically to be used with this program are presented, and the task of constructing such sound engines is discussed.

There are many previous examples of experiments with genetic algorithms in music generation, most of them dealing with sequence generation. A good overview is given in (Burton and Vladimirova 1999). An example of an application in a musical work is (Jacob 1995). Maybe the most interesting is the work of Johnson (Johnson 1999), where parameter sets for a granular synthesis engine are evolved.

This project is one in a series of experiments on the application of evolutionary algorithms to musical composition, including interactive evolution of score material and coevolution of sonic communication (see, e.g., (Dahlstedt and Nordahl 2001)).

1.1 Genetic Algorithms and Interactive Evolution

During the last few decades, the simulation of biological processes has become a research field of growing importance. The area was given a name, Artificial Life, in the middle of the 1980's by Langton (Langton 1989). One of the main tools in this field is the genetic algorithm (Holland 1975), where artificial objects undergo selection and evolution. For this to work, the properties of objects such as images, sounds, or sorting algorithms, must be mapped onto a genome, which often consists of a string of numbers. Typically, the algorithm works like this:

1. A random population is created.
2. The individuals of the population are evaluated according to some fitness criterion.
3. Those with highest fitness are replicated, often with some random modification (mutations), and/or mated with each other, to create a new generation of individuals. The old population is deleted.
4. The process is repeated from step 2 until a certain fitness threshold has been reached or until the increase in fitness stops.

When evolving esthetically pleasing objects like sounds or images, it is difficult to automate the fitness criteria, since they cannot be explicitly defined. One solution is to let a human evaluate the fitness by some kind of inspection of the whole population before the selection and replication is made. This process is called interactive evolution, and was first considered by Dawkins (Dawkins 1986), who wrote a program generating simple drawings (biomorphs) that could be evolved to look like plants and insects. The purpose was to show how simulated evolutionary processes easily can give rise to complex forms reminiscent of actual biological form. The idea was taken up by Sims (Sims 1991), applying it to the evolution of two-dimensional graphics, where the objects are represented by hierarchical genomes consisting of mathematical formulae. For a number of other applications, see (Bentley 1999).

A multi-dimensional parameter space, such as the parameter set of a synthesis algorithm, is an ideal target for interactive evolution. The individuals can be parameter sets for sound engines or pattern generators, like a synthesizer preset consisting of 100 parameters or more. You can navigate the huge parameter space without any knowledge of its underlying structure, following your esthetical preferences, and still have a very high degree of control.

There are two creative processes involved. First you design a sound engine by hand, which is then explored by way of interactive evolution. Exploring a huge parameter space is essentially an act of creation, when the space is truly unknown. This exploration is carried out in an interactive process. Depending on the sound engine used, the result may be heard in real time. The exploration is not continuous but step-wise. Departing from a number of random sounds, you generate a number of variations on the ones you like. This process is repeated any number of times. In this way, you can search the parameter space for interesting regions in parameter space (i.e. interesting sounds), move between them instantly or in a desired time by interpolation. By combining different sounds, you can find other potentially interesting regions between them.

2. Implementation

Working with MutaSynth involves two different modules – the sound engine which generates the actual sound, and the program which manages the selection process and calculates the genetical operators. This section will concentrate on the program, how it works and how the sounds are represented genetically.

Any possible sound generated by a certain sound engine can, using an engine-specific mapping table, be represented as a string of real numbers corresponding to the actual parameter values, ordered so that parameters related to each other (e.g. all filter parameters or all parameters controlling a certain oscillator) are located close to each other. We call such a string a *genome,* and every single real number in the string is called a *gene*.

To make the parameters behave in a musical way, i.e. to maximize the percentage of good sounds generated, the genes are mapped to their corresponding synthesis parameters according to different translation curves, to make the most musically useful values more probable, while maintaining the possibility of more extreme values. An example is a vibrato amount, i.e., a low frequency modulation of the pitch of an oscillator. Too much modulation is often undesirable, and only values near zero are relevant for most applications. So, the gene range is mapped to the parameter range in a non-linear way to make very small values more probable. This technique is implemented using interpolated lookup tables. Any curve could be used, but simple exponential, logarithmic and cubic curves have proven most useful. To audition a sound, the corresponding genome is sent to the sound engine according to the mapping table and the translation tables, and the engine produces the sound.

In MutaSynth, the working space is a population of nine sounds or genomes. The number nine is chosen because it corresponds to the number keys on the numerical keyboard, which is a convenient interface for auditioning. Nine is a small number compared to many other applications of interactive evolution, because the auditioning process takes some time compared, e.g., to browsing images. If no suitable offspring is found, one always has the choice of repeating the last genetic operation with the same parents, to obtain more alternatives.

As a starting point for the operations, the user can begin either with a set of randomly generated genomes or any previously stored genome. Sometimes it is also possible to import genomes, if the sound engine is capable of transmitting its parameter values. This may be useful for example when mating factory sounds in a synthesizer.

2.1 The Genetic Operators

Genetic operators take one or two parent genomes and generates variations or combinations of them, resulting in a new population of genomes to be evaluated. In MutaSynth, the genetic operators used are *mutation*, *mating*, *insemination* and *morphing*. These are described individually below. Mutation and mating (also known as *crossover*) are standard genetic operations modelled after the genetic replication processes in nature. Insemination is a variation on crossover where the amount of genes that are inherited from each parent can be controlled. What I call morphing is a linear interpolation on the gene level.

Every operator creates a set of new genomes that can be auditioned and further bred upon in the interactive process. Any sound can be stored at any stage in a gene bank, and the stored genomes can be brought back into the breeding process anytime, or saved to disk for later use. The parents used in an operation can be selected from several sources: a previously stored genome, either of the most recently used parents, any uploadable sound in the current sound engine or an individual from the current population (i.e., the outcome of the last breeding operation).

A genome is really just a constant length string of numbers. Another sound engine would interpret these numbers differently. This means that a genome is meaningless without the sound engine it was created for, and it will not work with any other engine.

It is sometimes useful to be able to prevent a number of genes from being affected by the genetic operations. For instance, when certain parameters of a sound (e.g., the filter settings) is good enough and the user does not want to run the risk of messing them up in further breeding operations, she can disable them, and they will stay as they are. If a gene is disabled, it will be copied straight from the first parent.

Mutation

A new genome is generated from one parent sound's genome by randomly altering some genes. A *mutation probability* setting controls the probability of a gene to be altered and a *mutation range* sets the maximum for the random change of a gene. Together, these two allow control of the degree of change, from small mutations on every parameter to few but big mutations.

Mating (crossover)

Segments of the two parent genomes are combined to form a new genome. The offspring’s genes are copied, one gene at the time, from one of the parent genomes. A *crossover probability* setting controls the probability at each step to switch source parent. The starting parent for the copying process is selected randomly. Each parent will provide half of the offspring’s genes, on average. The genes keep their position within the genome during this copying.

Insemination (asymmetrical crossover)

For a new offspring genome (Q), the following process is applied, based on two parent genomes (P_1 and P_2): P_1 is duplicated to Q, then a number of genes are overwritten with the corresponding genes in P_2. An *insemination amount* controls how much of P_2 should be inseminated in P_1, and the *insemination spread* setting controls how much the genes to be inseminated should be spread in the genome - should they be scattered randomly or appear in one continuous sequence. If the *insemination amount* is small, the resulting sounds will be close in character to the sound of P_1, with some properties inherited from P_2.

Morphing

A linear interpolation is performed on every gene of the two parent genomes, forming a new genome on a random position on the straight line in parameter space between the first parent (P_1) and the second parent (P_2).

Manual mutation

Manual mutation is not a genetical operator, but still something that affects the current genome. When the user changes a parameter on the synthesizer, the program is informed about the change and applies the change to the corresponding gene in the currently selected genome. The manual change then lives on through further breeding. This allows for the same level of direct control that the advanced synthesizer programmer is used to, and makes the method useful to both beginners and experienced users. Manual mutation may not be possible with all sound engines, depending on if they transmit parameter changes via MIDI.

2.2 User Interface

MutaSynth is made to be simple. It is also designed to give quick responses to user actions, to minimize all obstacles in the creative process. The current userface looks like this:

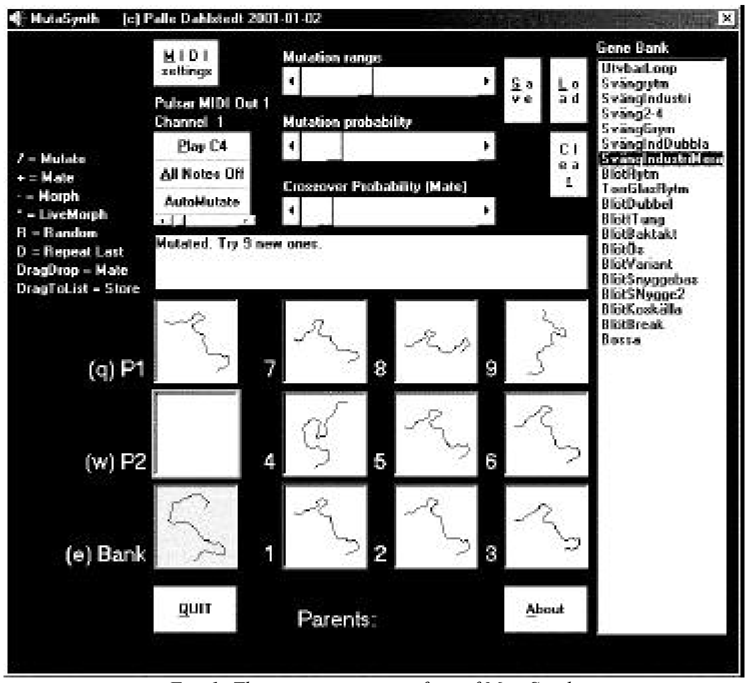

Fig. 1: The current user interface of MutaSynth.

The display shows a number of boxes, representing the population, the last used parents and the currently selected genome in the gene bank. The layout is chosen to correspond to the nine number keys on the computer keyboard. To listen to any individual from the current population, the user presses the corresponding number key, and the parameter interpretation of the genome is sent to the sound engine. The keys +, -, * and / invoke the different breeding operators. With these keyboard shortcuts the composer can keep one hand on her instrument and one on the number keyboard, for quick access to the operators and the individual sounds.

2.2.1 Visual Representation of Genomes

The genomes are represented graphically, to aid the aural memory and to visualize the proximity in parameter space between genomes. For this purpose, I have developed a simple mapping technique. The gene values are interpreted as distances and angles alternatively (scaled to a reasonable range) and drawn as turtle graphics. The resulting worm-like line is scaled to fit in a picture box.

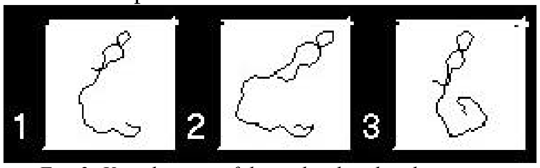

Fig. 2: Visualization of three closely related genomes.

When the different genomes are closely related, this is clearly visible (see fig. 2). Also, very often the graphics may have a resemblance to something figurative, which may aid your memory.

2.2.2 Communication and customization

All communication with MutaSynth and the sound engine, be it a hardware synth or another software, is by way of MIDI, using Midi Continuous Controllers (CCs) or System Exclusive Messages (Sysex). These messages can be customized to control virtually any hardware or software sound engine that supports MIDI remote control. This is done by creating a *profile* for that specific sound engine. A profile contains information on how to communicate with the synth, such as parameter-to-gene mapping, parameter ranges and communication strings.

It is sometimes useful to have several profiles for the same sound engine, biased towards different subspaces of the parameter space or, musically speaking, towards different sound types. This is easily done by enabling different parameters subsets and specifying different parameter ranges, mappings and default values.

3. Examples

In this section I will describe three different sound engines that I have used with MutaSynth, and discuss what kinds of sounds they can generate and how they can be explored. The examples are chosen to show the wide range of possible applications of this program.

When designing sound engines for MutaSynth, there are two ways to go. Either you take an existing sound engine of some sort, such as a stand-alone synthesizer, a softsynth or a granular software engine. Then you try to make a gene-to-parameter mapping that is relevant for the type of sound you want to produce, by carefully selecting parameters to be mapped and reasonable parameter ranges. Or, you build a sound engine from scratch, with the potential to generate the sounds you want, and probably many other sounds. This second approach is more flexible and open-ended, so I will take my examples from that category.

These examples are all implemented on the Nord Modular synthesizer (Clavia 1997), which is a stand-alone virtual modular synthesizer – basically a number of DSPs controlled by a computer editor. One can freely create and connect modules from a library of about 110 different types. This environment was chosen because it is quick and easy to use and does not put any load on the host processor, while allowing for quite complex synthesis and triggering techniques. Any parameter on any module can be assigned a MIDI Continuous Controller, which makes it easy to map up to about 120 genes to relevant parameters. MutaSynth can also be used to control sound engines made with MAX/MSP, jMax, Native Instruments' Reaktor or any other universal sound processing tool, if it supports remote control of its parameters.

For a normal synth programmer these examples may look confusing and unpredictable. This is because they are designed for another working method than the usual manual programming, allowing for all kinds of modulations to happen, maximizing the potential of the engine. Not all modulations will be active or influential all the time, depending on the genome. It would be very difficult to make any sound design by hand with these engines, but with interactive evolution it is quite possible.

3.1 4Sine

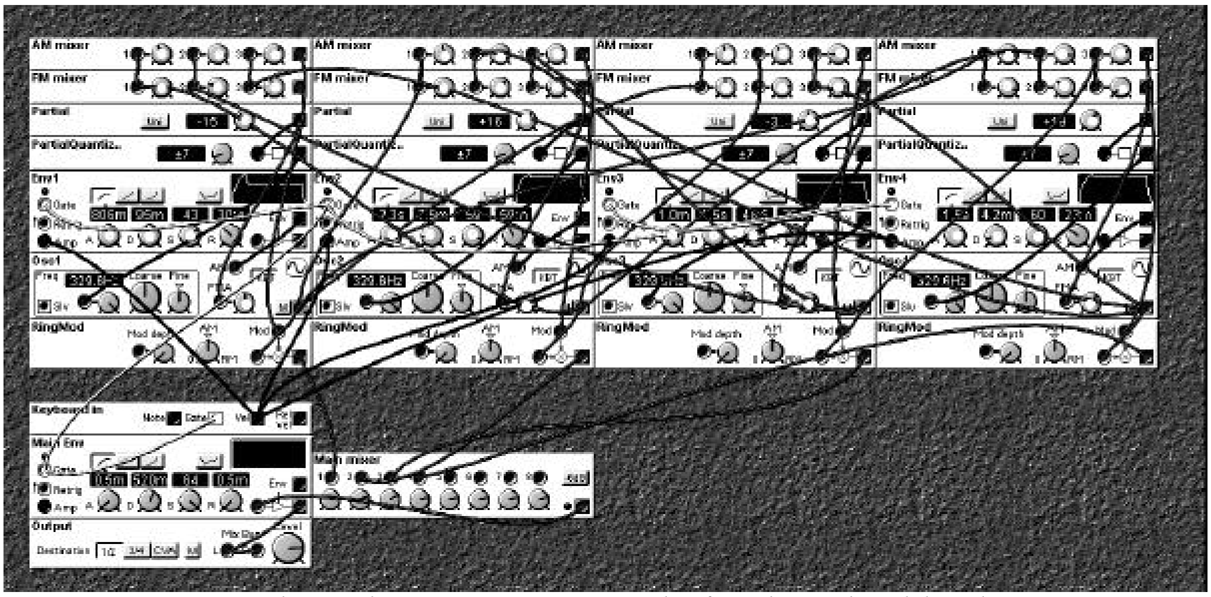

Fig. 3: The sound engine 4Sine, as a screenshot from the Nord Modular editor.
The knobs that are included in the genome are marked with white dots.

This is basically four sine oscillators connected in a network, so that each oscillator is frequency modulated (FM) by a weighted sum of the others' output, and amplitude modulated by the same sum but with different weights (the two upper rows of mixers). Each oscillator has its own amplitude envelope, which is controlled by a MIDI keyboard, as is the fundamental frequency. So this patch generates playable keyboard instruments.

To make the FM part a little bit more controllable, the frequency ratios are not arbitrary, but always integer multiples or fractions of the fundamental frequency. This is controlled by the “Partial Quantizer” (the fourth module from the top). The output is a mix of the four parts.

The levels of the upper mixers, the partial numbers, the FM amount, and the attack, decay and sustain of each envelope are mapped to genes, a total of fourty-four genes, each spanning the parameter’s whole range.

Sound examples 1a-1d show different instruments evolved in a few generations from random starting genomes. Only the final results are played.

Sound examples 2a-2h form a mutation sequence from a strange voice-like sound to a buzzing organ. Note how the attack of the sound is gradually changing from example 2e. The main timbre change happens between 2a and 2b.

3.2 GestureB

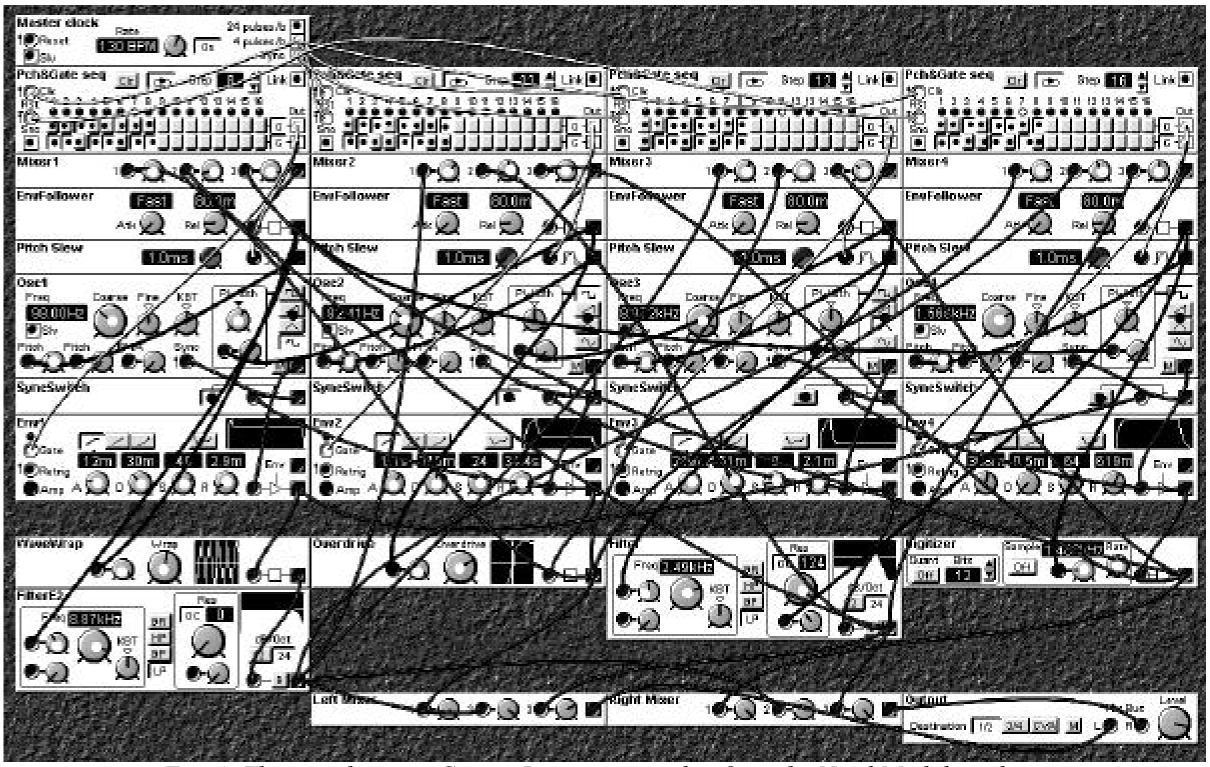

Fig. 4: The sound engine GestureB, as a screenshot from the Nord Modular editor.
The knobs and buttons that are included in the genome are marked with black or white dots.

This is a techno loop generator, basically made of four similar parts, each consisting of : an oscillator, an amplitude envelope, a two-track event sequencer where one track trigs the envelope and the other modulates the pitch of the oscillator, making it glide between two notes with a glide speed controlled by the Pitch Slew module. Each oscillator is optionally synced by the output of the part to the right.

Each part takes a weighted sum (the top mixers) of the the output of the others into an envelope follower, to generate a control signal that is used to modulate some special processing modules in bottom row. These are different for each part, to give them distinct characters: a wave-wrapper into a low-pass filter, an overdrive unit, a low-pass filter and, for the fourth part, a digitizer.

The sequencers have different loop lenghts (8, 12, 16 and 32 steps) to make the pattern more lively, but only the eight first steps are mapped to genes. The others are always zero. The intensity of the patterns can be increased by setting all the loop lengths to eight steps.

Sound examples 3a-f show the wide range of techno patterns possible with this engine.

Sound examples 4a-j shows two different patterns and the results of a mating between them with a crossover probability of 0.1. Note how different elements of the structures make it into the offspring.

Sound examples 5a-j is another example of a mating operation.

3.3 Struct

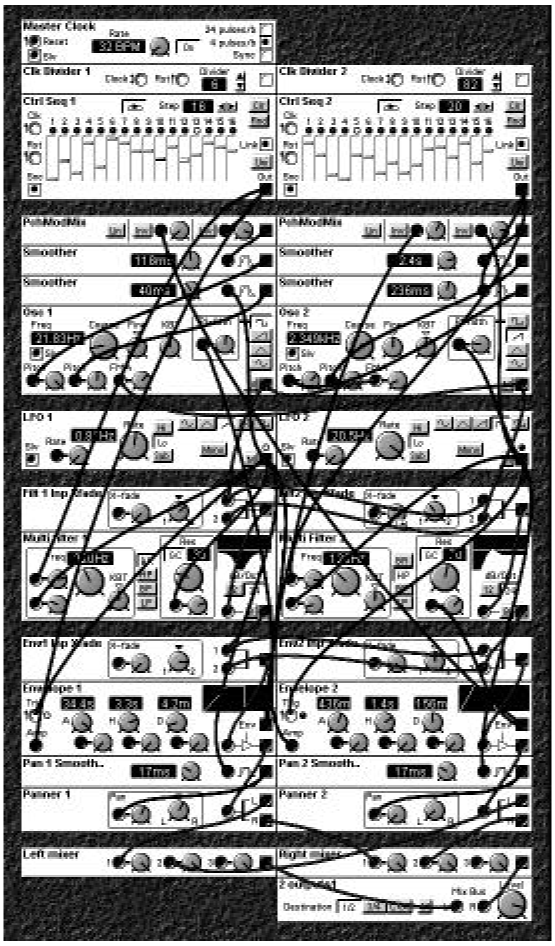

Fig. 5: The Struct engine with all cables.

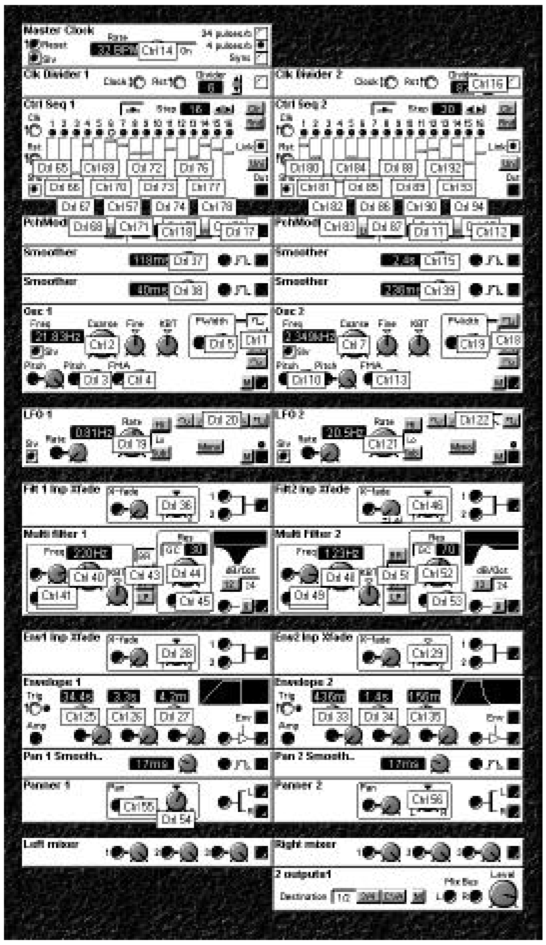

Fig. 6: The same engine without cables, with gene mapping shown.

The idea behind the design of Struct was to allow for widely different sonic gestures to be generated. Two low frequency oscillators (LFOs) and two control sequencers are used to modulate two oscillators and two filters, with rather intricate relationships. The pitch of each oscillator is modulated by one LFO, one envelope and one sequencer, in variable amounts. To make the timbres more interesting, the two oscillators have the possibility of modulating each other's frequency.

The cut-off frequencies of the two filters are similarly modulated by one sequencer and one LFO, in variable amounts. The filters take as input weighted sums of the two oscillators (controlled by the "Filt Inp Xfade" module just above the filter), and the envelopes/amplifiers take weighted sums of the two filters' outputs as inputs.

The oscillators can generate sine, triangle, up- and down-saw and square waveforms. In case it is a pulse wave, the pulse width is modulated by the envelope in the first oscillator and the LFO in the second. The filters can be of any of the basic types (low-pass, band-pass, hi-pass and band-reject). The output is a mix of the two modulated panners located at the bottom of the patch.

Many of the sounds generated from this engine are of the banal, very synthetic "computer game effects" type, but sometimes one stumbles across surprisingly organic and complex gestures and textures, and when these interesting spots in parameter space are found, they often show the way to many good, related sounds, sometimes enough for a whole piece.

Struct was actually one of the first sound engines I made for MutaSynth, and most of the sounds from the electro-acoustic etude *One Minute Man* (sound example 9) are made with it.

Sound examples 7a-e shows some of the potential of the sound engine. These are sounds that have been evolved by mutations from random starting genomes, in 2-12 generations.

Sound examples 8a-j forms a transition sequence from one sound to another. The different sounds are located on a straight line in parameter space, and the population is generated with the morphing genetic operator.

3.4 Observations

There are different ways of thinking when constructing sound engines to use with interactive evolution, depending on its purpose. Firstly, I need to know what kind of sounds I want to produce. Not an exact description, but I need to know how they should be generated, what techniques to use, what kind of parameters should be mapped to genes and such things. In general, I have to think about the potential of the engine, considering the consequences of the whole range of each parameter and the probabilities of related parameters to have meaningful combinations of values. These are crucial choices – there is nothing like a universal synthesis engine, except maybe to interpret a huge genome as a wave file, but that would be quite useless because of the infinitely small portion of the possible genomes that will actually make any meaningful sound. Most will be just noise. The more universal I make the sound engine, the smaller the portion of useful sounds. If I try to cover many different types of sounds, I will also allow the generation of strange sounds in between the useful areas in parameter space. These may be interesting, but probably only a very small minority.

The genetical representation of the sound, i.e., the mapping of the genes to the sound generation parameters, is crucial and careful attention has to be paid to the choices of parameters and ranges. Sometimes parameters are dependent on each other, like in FM synthesis. Then the modulator frequency should be an integer multiple or fraction of the carrier frequency for a harmonic result. This is of course a simplification, but it illustrates the problem. There are several ways to avoid this. One is to design the sound engine so that one gene maps to the carrier frequency and another to the integer ratio between them (like in the example Struct in the preceding section). Then the probability for useful sounds increase, but the results are also more limited.

Another way is to start the evolution from genomes that sounds good, then limit the use of big mutations, concentrating on mating with other good genomes, and possibly use some insemination. Then the ratio between the dependent genes' values will be inherited from one of the parents, except if a crossover occurs between them, which is unlikely if they are located close to each other in the genome. The disadvantage of this solution is that it puts restrictions on the working method and requires the user to be aware of what is happening at the genetic level, which may not always be the case.

Yet another way of dealing with very large sound engines is to divide the parameters into relevant groups, applying interactive evolution to one group at a time. If the sound engine is an FM synth, one could first evolve the frequency ratios, then the envelopes and modulation parameters, for example. The grouping technique is also effective in limiting the possible damages of bad mutations, by protecting the finished parts – according to your fitness evaluation – of the genome. Grouping may not always be possible, though. For example, if the sound engine is a homomorphic network of intermodulating oscillators, it may be very difficult or impossible to evolve e few parameters at a time, since they all depend so much on each other.

Many synthesis tools keep the event level and the timbral level separate. Personally, I like to use sound engines that combine sound synthesis and pattern generation in one way or another, instead of just evolving keyboard playable sounds and then playing them. It is possible to parametrize both structural patterns on the macro level and the timbre at the micro level, and evolve them simultaneously. In this way, you can explore a continuum in both these dimensions at the same time, which allows for more complex audible relationships between sound objects.

4. Applications

Considering the wide range of possible sound engines and synthesis techniques, there are a number of applications for this program in the field of sound design and electronic composition and performance.

4.1 Material Generation

MutaSynth can be used on the sound level, for developing sound objects, loops, timbres and structures to be used in manual compositional work. In this case, it offers a way of investigating the possibilities of a certain sound engine. A musical advantage is that the material created during a breeding session often is audibly interrelated to a quite high degree, which opens up for compositions based on new kinds of structural relationships.

4.2 Interactive evolution of synthesizer sounds

MutaSynth can be used for programming almost any MIDI synth, without any knowledge about the underlying synthesis techniques. This depends on the existence of a suitable profile for the synth model in question, mapping the genome to a relevant parameter set in the synth. These profiles are quite easy to create, given the parameter set and communications protocol of the synthesizer in question.

The genetic operators provide a simple way of trying out combinations of existing sounds, finding variations or just exploring the parameter space of the synth. Doing this on normal off-the-shelf synths I have found sounds that I never thought were possible. Many computer based synth editors offer a randomization feature, sometimes with a random step size, corresponding to genetic mutations. They do not, however, provide anything like crossover or morphing, to explore the parameter space between existing sounds.

4.3 Live evolution

With a suitable set of sound engines synchronized to a common clock, several layers of musical material can be evolved simultaneously, live or in the studio. Either by evolving one layer while the others play, possibly with headphone monitoring, and sending it out to the speakers when a nice pattern has been found, or by evolving over many sound engines at the same time, like a giant genome consisting of all available knobs/parameters. Also, since the gene banks of MutaSynth are controllable by MIDI Program Change messages, the playback of previously evolved stored sounds is quite easy to integrate in a live set-up. I think this would be a very powerful tool for DJ:s and composers of music for the dance floor.

The music for the video *Time and Space* (sound example 10) was composed in this way, in a quasi-live situation in the studio.

5. Discussion

Interactive evolution as a compositional tool makes it possible to create surprisingly complex sounds and structures in a very quick and simple way, while keeping the feeling of control. I am still the composer, rather than a slave of the application. The process is interactive and gives immediate aural feedback, which is crucial for creative work, while it allows using very complex sound engines that would be difficult to use productively in any other way. Interactive evolution does not limit the creativity by imposing templates or modes on the result, like many other creativity aids such as document templates, groove templates, chord sequences or phrase libraries. The sound engine defines a huge space, which is searched without restrictions in all dimensions with the help of chance and esthetic judgement.

MutaSynth can be combined with manual control, through manual mutation and freezing of groups of genes. An expert can easily make a parameter change in the middle of the breeding process, and a manually designed sound can be imported for further breeding.

I like to be surprised by chance and emergence, while other composers may prefer total control. They may feel uneasy working with a technique like this, since it is not a tool for achieving a predefined goal, but rather searching the unknown, with your ears wide open. So, when you know exactly what you want, this may not be the tool of choice. In my view, though, creativity involves a certain amount of unpredictability and surprise.

There seems to be an upper limit of the duration of the sonic fragments that are created with MutaSynth. My experience is that it is possible to evolve either continous sounds that form one small part of a dense structure, or short self-contained structures that can sustain interest by themselves for a short time, but not longer sections or whole pieces. A reasonable duration to expect from these evolved structures could be up to maybe fifteen seconds.

To search an unknown space differs from a typical optimization process guided by an explicit fitness criterion. It is more about finding local maxima in an implicit fitness function, i.e. to find pleasing sounds, than to locate a global maximum. Ending up with a good sound one did not know one needed may be a plausible result. If the goal is to design a good oboe sound and a good trumpet sound is found on the way, it is just a bonus. I often save sounds during the whole breeding session, the last not being any better than the first, but different. Together they form an interesting progression, often appearing in that order in the finished composition. The way is the goal.

In my own work, the impact of using MutaSynth has been significant. Since I often use the bulk-creation-then-filter method because of my personal belief and interest in the unexpected, this is a very suitable tool. Because of the interactivity, it is blurring the border between improvisation and composition – it is more a controlled way of improvisation, with inherent selection of the best takes. Also, the continous audition of new material is changing the focus from concept to result, relying heavily on the judgement of the composer. Composition is turned into a selection process, at least in the material-creation phase. Still, of course, the sound engines have to be created by hand.

6. Future work

It would be interesting to evolve not only the parameter sets of the the sound engines, but the engines themselves, which would require a complex genetic representation of the sound generation and processing modules and their connections. One way of doing it could be to make a genetic representation of a mathematical formula that generates the actual waveform. If this is done with a hierarchically structured genome, it would allow the evolution of arbitrarily complex sound objects.

Automation of fitness criteria would be an interesting extension, allowing for batch evolution of sound objects. This could be based either on some kind of description of the desired sound, or by example, making the program evolve sound objects as close as possible to a given sound file.

References

Bentley, P. J. (ed.). 1999. *Evolutionary design by computers*. Morgan Kauffman.

Burton, A.R. and Vladimirova, T. 1999. Generation of Musical Sequences with Genetic Techniques. *Computer Music Journal* 23:4 (1999), 59-73.

Clavia DMI AB 1997. *Nord Modular* (a virtual modular synthesizer). Stockholm, Sweden. http://www.clavia.se.

Dahlstedt, P. and Nordahl, M. 2001. Living Melodies: Coevolution of sonic communication. *Leonardo Journal* 34:3 243-248, 2001, MIT Press Journals.

Dawkins, R. 1986. *The Blind Watchmaker*, Essex: Longman Scientific and Technical.

Holland, J. H. 1975. *Adaptation in natural and artificial systems*, Ann Arbour, MI: The University of Michigan Press.

Jacob, B.L. 1995. Composing with Genetic Algorithms in *Proceedings of ICMC 1995*. San Francisco, CA: International Computer Music Association.

Johnson, C.B. 1999. Exploring the sound-space of synthesis algorithms using interactive genetic algorithms, in A. Patrizio, G.A.Wiggins and H.Pain (eds.) *Proceedings of the AISB'99 Symposium on Artificial Intelligence and Musical Creativity*. Brighton: Society for the Study of Artificial Intelligence and Simulation of Behaviour.

Langton, C.G. (ed.). 1989. *ALIFE I, Proceedings of the first international workshop of the synthesis and simulation of living systems*. Addison Wesley.

Sims, K. 1991. Artificial Evolution for Computer Graphics. *Computer Graphics* 25, 319-328.