AUTOMATING THE DESIGN OF SOUND SYNTHESIS TECHNIQUES USING EVOLUTIONARY METHODS

Ricardo A. Garcia[*]

MIT Media Lab
Machine Listening Group
20 Ames St., E15-491, Cambridge, MA 02139
rago@media.mit.edu

ABSTRACT

Digital sound synthesizers, ubiquitous today in sound cards, software and dedicated hardware, use algorithms (Sound Synthesis Techniques, SSTs) capable of generating sounds similar to those of acoustic instruments and even totally novel sounds. The design of SSTs is a very hard problem. It is usually assumed that it requires human ingenuity to design an algorithm suitable for synthesizing a sound with certain characteristics. Many of the SSTs commonly used are the fruit of experimentation and a long refinement processes. A SST is determined by its *functional form* and *internal parameters*. Design of SSTs is usually done by selecting a fixed functional form from a handful of commonly used SSTs, and performing a parameter estimation technique to find a set of internal parameters that will best emulate the target sound. A new approach for automating the design of SSTs is proposed. It uses a set of examples of the desired behavior of the SST in the form of *inputs + target sound*. The approach is capable of suggesting novel functional forms and their internal parameters, suited to follow closely the given examples. Design of a SST is stated as a search problem in the SST space (the space spanned by all the possible valid functional forms and internal parameters, within certain limits to make it practical). This search is done using evolutionary methods; specifically, Genetic Programming (GP).

1. INTRODUCTION

Sound synthesizers are usually implemented as computer programs and algorithms that run in digital computers and produce digital sound samples (waveforms). These algorithms for sound generation are termed *Sound Synthesis Techniques (SSTs).* An SST can be dissected into a *functional form* and *internal parameters*. The functional form (also known as *topology*) describes the relationship between the functions and elements in the algorithm, while the internal parameters are variables that take a particular value at the moment of implementation of the algorithm (depending on the desired behavior).
Design of a SST is customarily limited to the selection of a functional form from a set of algorithms (i.e. "classic" SSTs) followed by application of a mathematical technique for estimation of the internal parameters to match a target sound. The design of SSTs, more specifically their functional form, is a very hard problem. It is usually assumed that it requires human ingenuity to design an algorithm suitable for synthesizing sound with certain characteristics. Many of the SSTs commonly used are the fruit of experimentation and a long refinement processes.
The efforts for automating the design of SSTs have been mainly focused into automating the parameter estimation stage of the internal parameters for a given functional form.
Horner et al. [1] proposed an approach for automating the internal parameter estimation of FM synthesizers using evolutionary methods, in particular Genetic Algorithms (GA). Johnson [2] proposed an interesting approach to use evolutionary methods and human listeners in an interactive system to explore the parameter space of (*Fonction d'Onde Formantique*) FOF synthesis. Wehn [3] used GA to explore the parameter space of FM-like synthesizers, and allowed some degree of variation in the functional forms.
Our goal is to propose a general approach capable of suggesting valid functional forms and internal parameters for a SST to synthesize a target sound, using a known set of inputs (time varying signals). This problem is related to the system identification, or symbolic regression problem stated in control theory. The inputs and outputs of the system are known, but the system is unknown.

1.1. Approach

The SST space is defined as the space spanned by all the possible combinations of a given set of functional elements and their connections. Every point in the SST space defines completely a functional form and its set of internal parameters. Design of a SST is then regarded as a search in the SST space. The goal is to "find" a point in this space that is capable of producing a sound "close" to the target sound using the given inputs. This measure of "closeness" is done using an "error metric". The search in this space is performed using a class of evolutionary computation method called Genetic Programming (GP). GP has shown outstanding empirical performance in searching complex multidimensional spaces [4-6]. Custom SST representations in the form of *topology graphs* and *expression trees* are used along

[*] Now with Chaoticom. 83 Lafayette Road Hampton Falls, N.H. 03844, USA

with their required mapping rules. Topology graphs are the most widely used representation for SST, but they are difficult to manipulate. Expression trees facilitate the level of manipulation required to use GP for exploring the SST space. A complete discussion of this research can be found in [7].

2. REPRESENTATION OF SSTS

SSTs are computer algorithms designed to produce sound samples. Algorithms are usually represented using: mathematical formulas, instruction lists (pseudocode) or topology graphs (flow diagrams). All of them are equivalent representation (posses the same information), but offer different advantages from the viewpoint of a human designer.

a) $y = 2\sin(x) + 0.5$

b)
```
evaluate sin(x)
assign result to P
evaluate 2*P
assign result to Q
evaluate Q+0.5
assign result to Y
output Y
```

c)

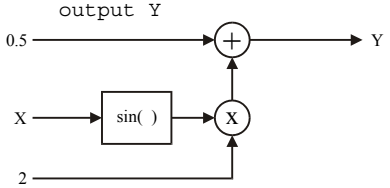

Figure 1. Representations of algorithm: a) formula, b) pseudocode and, c) topology graph

2.1. Representing SSTs as expression trees

Topology graphs of complex SSTs are usually in the form of cyclic graphs (with closed loops). Manipulating this kind of graph is very difficult. Addition, deletion or reconnection of functional elements can render a topology invalid very easy. In addition, if the "designer" in charge of manipulating these topologies is a computer program (as it is the goal), the easier the manipulation, the less probability of creating useless SSTs. Expression trees are graphs that are very easy to manipulate by following a small set of construction rules. It makes sense to try to find a mapping between an "easy-to-manipulate" expression tree graph into a "difficult-to-manipulate" topology graph.

An ingenious idea borrowed from developmental biology suggests a way of doing this. The idea is to encode in the expression tree the instructions for the "development" of an embryonic topology. The process begins with a very simple embryo, and following the instructions it "grows" the fully developed topology. It can even include the development for the internal parameters associated with the functional elements. Problems similar to this one that involved development of topology graphs from expression trees were suggested by Koza et al. [6, 8] that used a mapping into analog circuit topologies; and Gruau [9] that mapped trees into neural network families.

2.2. Suggested mapping: expression tree into topology graph

The nodes in the expression trees are instructions that when executed will result in a fully developed topology graph. Every expression tree will render a unique topology graph, but it is possible to have more than one tree to render the same topology graph. The initial topology graph is called *embryo*, and in our case it is a simple topology with four blocks, as seen in Figure 2 The embryo has a single modifiable object TYPE_A with no functional element assigned yet, and connected to two sources and one renderer. The sources and the renderer will remain the same during the whole development process, but the modifiable object will change and new blocks and connections will be created. This configuration of the embryo could be different (to suit the design specifications, i.e. the number of time varying inputs), but has been chosen for explicatory purposes here. Figure 2 shows a simple expression tree, embryo and first steps of development of a topology. The first node of the expression tree

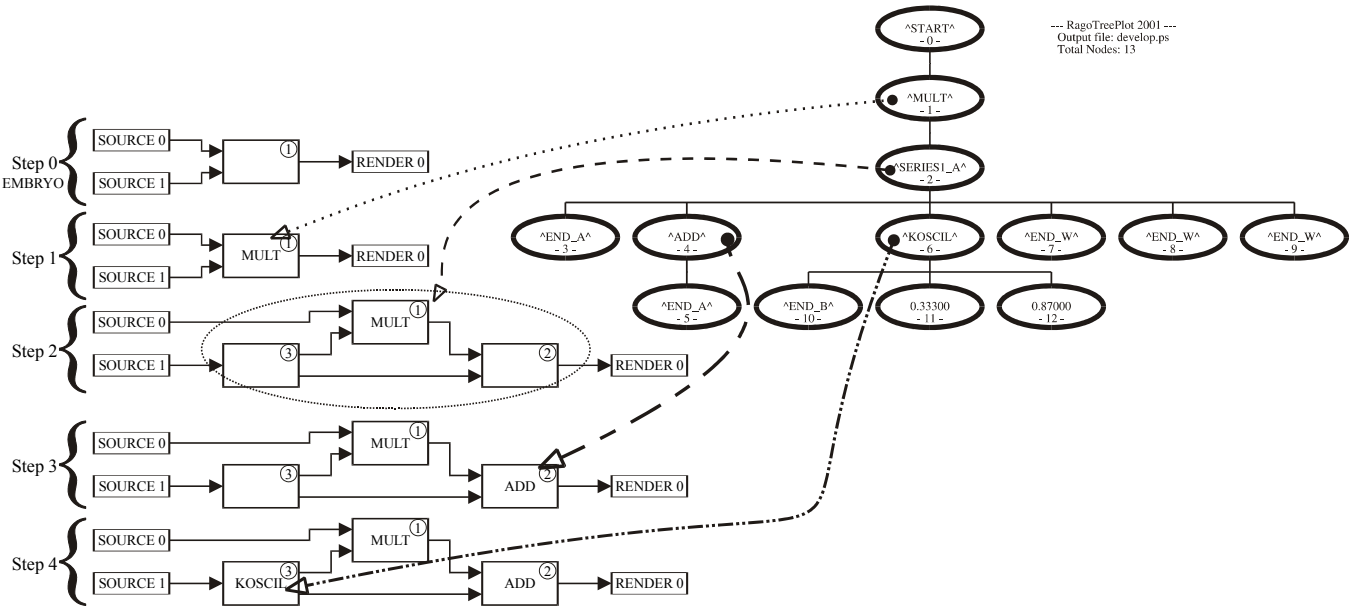

Figure 2. Example of mapping from expression tree into topology graph. Note the development of the embryo into a more complex topology

is the *START* node, and this is ignored during the development process. The second node is "pointing" to the modifiable object number 1. When executed, this node will change the topology graph, more specifically the object that is pointed to in some way. In this case, the node has the instruction *MULT,* so the type *MULT* is assigned to the particular block in the topology graph.
The next node has the instruction *SERIES1*. The effect of this instruction is to add some new blocks and connections to our topology graph, and to create more "pointers" to different nodes in different new branches of the expression tree. After executing this Topology Modifying Function, several new blocks and connections are introduced into the topology graph. Each one of the new objects is modifiable, and has an associated node pointing to it. The rest of the nodes are executed, and this adds, modifies, or changes blocks and their connections into the topology graph. A complete repertoire of topology development functions can be found in [7].

2.3. Functional elements

From an analysis of several "classic" SSTs [10-12], a set of commonly used basic functional elements was extracted. They are called "classic SSTs" because they have been used and studied by researchers and musicians over many years, and offer a good approximation for a set of functional elements. This is a list of the main types of functional elements found in our taxonomy:

- Sinusoidal oscillators (variable amplitude, frequency, phase over time)
- Wavetable oscillator (variable amplitude, read index)
- Delay (memory) for one or more samples
- Controlled gain filter
- Noise generator
- Time varying filters (coefficients can change over time)
- Addition
- Multiplication

3. DESIGN OF SST

Design of an algorithm is defined as "the process that conceives the structural form and internal parameters of an algorithm, capable of producing a desired set of outputs, using a known set of inputs". The specifications for the design are usually given as sets of "examples". Each example comprises a set of inputs and a target output (desired behavior). It is necessary to specify as well an *Error Metric* to measure the performance of a given SST. Design of an SST is usually a two-stage process: first selection of a functional form and, second parameter estimation.

3.1. Classic design

In classic SSTs design a human realizes the first stage of the process. Functional forms are usually never conceived from scratch for a particular target sound. Instead, the designer selects one "template" from a set of known functional forms (i.e. the classic SST) based on the characteristics of the target sound, and the known capabilities of the tentative functional forms. In the second stage, the designer selects an approach for parameter estimation, and uses it to find the internal parameters that better "fit" the selected functional form to produce a sound "close" to the target sound. This part of the process has been automated with high success [1] for FM synthesis parameter estimation. The designer usually tries a handful of functional forms to select the one that results in a better match to the target sound.

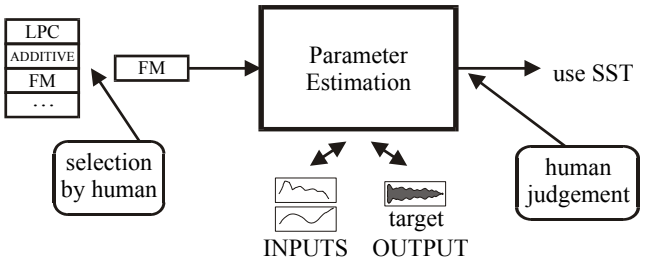

Figure 3 Classic design of SST. Human selects a fixed Functional Form from a pre-defined set (i.e. "classic" SST) and uses a parameter estimation method. Human judges if the results are satisfactory or repeats process with new functional form

3.2. Proposed approach for design

Our proposed approach tries to remove as much human intervention from the design process as possible. The first change (and maybe the most important) is to replace the first stage of selection of a pre-made functional form, with a "functional-form suggesting mechanism". This mechanism will suggest valid functional forms that can be tested to see if they are good or not for the desired goal. The second stage remains the same, and it consists in the parameter estimation for the "selected" functional form to try to match the target sound. Another point where the human intervention can be reduced is in the "error comparison" between the output sound and the target sound. This comparison (error metric) will return a value that will be used for suggesting a new functional form, and it will try to minimize the error. The procedure is repeated until the error falls within acceptable limits.

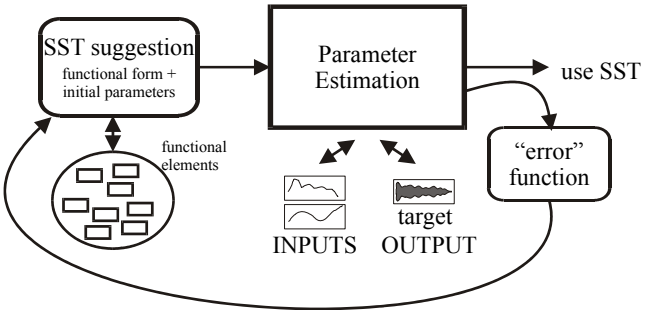

Figure 4 Suggested approach for design. Automated suggestion of Functional Forms, parameter estimation, and automated comparison of target/output sounds fed back in suggestion block.

3.3. Parameter estimation

Once a functional-form has been selected (or suggested), the number and type of internal parameters remains fixed. A technique for parameter estimation can be used to find a set of values for those parameters that will reduce the error between the produced output and the target sound. This is usually done using either *mathematical analysis* or *optimization methods*. The approach selected depends mainly in the type of functional form to be optimized and in the precision needed for its internal parameters. Some of the mathematical analysis tools commonly used include Fourier and Cepstral analysis. When a direct mathematical analysis is not convenient, it is possible to use a numerical method to approximate a "good" set of internal parameters. These methods explore the parameter space in a guided manner, and return usually a locally optimal set of parameters that accomplish the desired behavior for the SST. Note that the error metric plays a fundamental role in the exploration of the parameter space and, ultimately, in the selection of the parameters.

4. DESIGN AS A SEARCH IN SST SPACE

All the possible valid combinations of functional elements, connections and internal parameters compose the SST space.
Hypothesis: *Given a set of inputs, a target sound and an error metric, it is possible to find the functional-form and internal parameters of a SST capable of synthesizing an output sound "close" to the target sound.*
Our original goal of designing a SST (functional form and internal parameters) can then be stated as a search problem in the SST space. The next step is to define a search strategy to efficiently and adaptively explore this space and find an acceptable solution to our problem. The SST space has many dimensions and is highly non linear. Each point in this space represents a different SST (with different functional form and internal parameters). Evolutionary methods have been shown to perform very well in complex search spaces. [4, 13].

4.1. Genetic Programming (GP)

Genetic Programming (GP) is an optimization/search method that has been gaining popularity in the last decade. It is an extension of Genetic Algorithms, and both belong to the field of Evolutionary Computation. The idea with GP is to have a population of candidate solutions (in our case, suggested SSTs) that will be evaluated and a fitness value assigned to each. The fitness function gives an analytical measure of the performance of the individual and its output. Once all the individuals in the population have computed their fitness value, a new population of candidate solutions is created by probabilistically selecting individuals and performing genetic operations on them. The probability of being selected to be part of a genetic operation is directly related to the fitness of the individual: the better the fitness, the higher the probability. The genetic operations will create new individuals by: *copy* (identical copy of an individual), *mutation* (random alteration of an individual functional form and/or internal parameters), or *crossover* (characteristics of two individuals are fused together to create a new one). The process is repeated until a candidate solution that shows a fitness value that fulfills the specifications is found, or a maximum allowed number of generations have been tested.

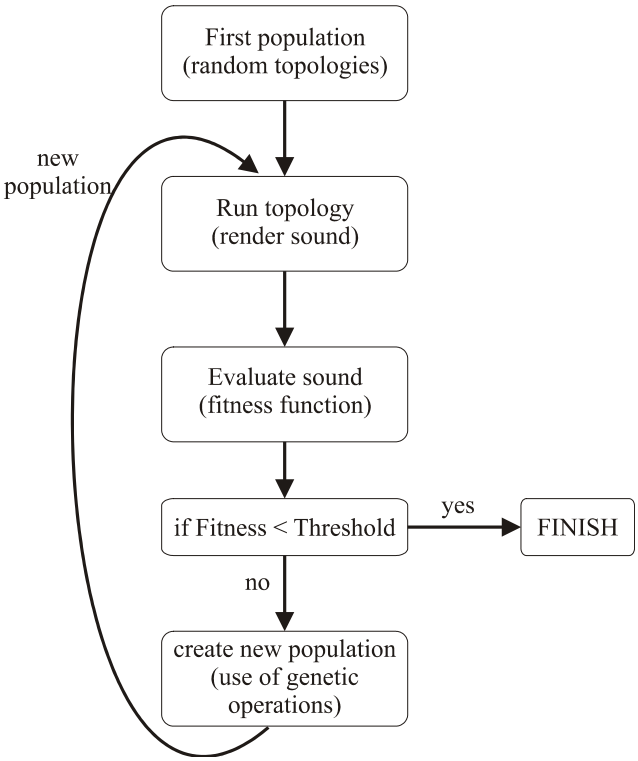

Figure 5 Genetic Programming loop

4.2. Fitness functions

In any kind of optimization or search method, it is fundamental to have a way to measure the performance of the candidate solution. This performance metric is usually called a *fitness function* or *error metric*. Fitness functions (FF) give some numerical grade to the difference between the outputs of the system compared to a desired target. The features that are measured in a fitness function vary from application to application. In our case, for sound synthesis techniques and sound sample sequences (waveforms) as targets, it is usual to define fitness functions that measure the distance between two sounds, or how "similar" they are. An analytical fitness function that uses the Least Squares Error (LSE) of the magnitude spectrograms (of Target and produced sounds) has been successfully applied by Horner et al. [1], and showed good results with this project. An enhancement for this fitness function was to include phase information. The LSE of the phase spectrograms, weighted with the magnitude of the target sound was successfully used in most of the test performed [7, 14]. Perceptual fitness functions are more difficult to compute because of their subjective nature. A fitness function that incorporates a psychoacoustic model of simultaneous frequency masking [15-17] was developed and used successfully in some tests performed [7].

5. AGeSS SYSTEM

A system implementing the proposed approach called AGeSS (Automatic Generation of Sound Synthesizers) has been developed and tested. The system is implemented as a set of binaries (compiled for ANSI C++) and Matlab scripts. The user is required to supply the parameters for the GP run, as well as the "examples" for the system.

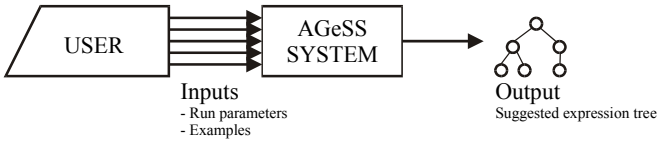

Figure 6 AGeSS system: user input (parameters, examples of control signals and Target). Output (suggested expression tree)

6. EXAMPLE

The developed AGeSS system was used to perform a series of experiments to explore the potential of the suggested approach. One of them is outlined here.

A simple FM synthesis formula was chosen for this experiment [10, 12], as shown in equation 1. This SST has been explored in depth by many researchers and musicians. The value of the internal parameters was taken from the original values suggested by Chowning for simulating a woodwind sound [18].

$$s(t) = A(t)\sin\left(2\pi\frac{C_f}{FS} + 2\pi I M_f \sin\left(2\pi\frac{M_f}{FS}\right)\right) \quad (1)$$

C_f = Carrier frequency = 880 , 988 = *f(t)*

M_f = Modulator frequency = 880/3, 988/3 = *f(t)/3*

I = index of modulation = 2

FS = sampling frequency = 8000

A(t) = time varying envelope

This SST uses two time varying inputs: *A(t)* and *f(t)* and 3 internal parameters I, D, M. For the generation of the Target sound, two time varying signals were generated (using Matlab) to simulate the brass sound of two distinct notes (A880, B988) of 0.3 seconds each. These can be seen in Figure 7.

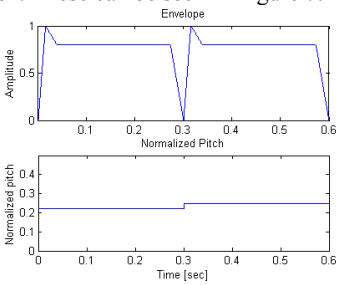

Figure 7 Input signals (top) Envelope for two notes. (bottom) Normalized pitch for two notes (A880, B988).

The selected fitness function uses the simultaneous frequency masking fitness function. It calculates the spectrogram of the target sound and uses this to calculate the threshold of masking of the target. This information, along with the spectrogram of the output sound is used to calculate a distance metric.

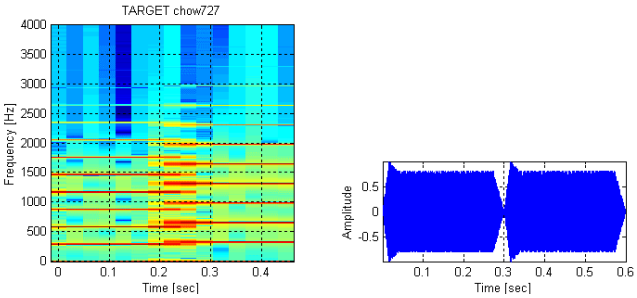

Figure 8 Spectrogram and waveform of TARGET signal formed by two notes (A880, B988).

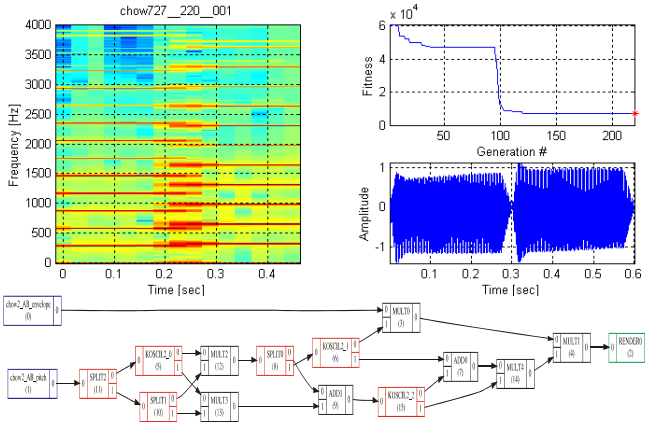

Figure 9 Spectrogram, waveform and topology for best individual of Generation 220

Note that the final spectrum agrees with the target in all the frequencies of the harmonics. But the final spectrum has higher energy at the high end of the spectrum. The topology evolved in generation 220 is shown in Figure 9. It is possible to analyze the functional elements and their connections to find the close form formula representation of the topology. In this case, it is represented in equation 2.

$$s(t) = k_1 A(t) oscil_2\left(k_0 f(t) + f(t) oscil_0\left(f(t)\right)\right) \times \left(oscil_1\left(k_0 f(t)\right) + k_2\right) \quad (2)$$

Comparison between equations1 and 2 shows a close similarity in their functional form.

7. CONCLUSIONS

Design is stated as a search in the multidimensional SST space. Each point in this space will represent a different functional form and set of internal parameters. The goal is then to find a point in the SST space that will fulfill the specifications of design. It is not clear how neighboring points are related in this representation. In addition, the number of possible points in this space is huge, making it impossible to do a thorough search of the space. These characteristics make the search of the SST space a

very complex problem. Evolutionary methods, such as Genetic Programming, have proved satisfactory when dealing with these types of problems. The experiments show that the selected set of functional elements and the representation scheme are effective for the automated design of some common synthesis algorithms, especially the frequency modulation techniques.
For more information about this research project, please visit http://www.ragomusic.com/research/

8. REFERENCES

[1] A. Horner, J. Beauchamp, and L. Haken, "Machine Tongues .16. Genetic Algorithms and Their Application to Fm Matching Synthesis," *Computer Music Journal*, vol. 17, pp. 17-29, 1993.

[2] C. G. Johnson, "Exploring the sound-space of synthesis algorithms using interactive genetic algorithms," presented at Proceedings of the AISB Workshop on Articial Intelligence and Musical Creativity, Edinburgh, 1999.

[3] K. Wehn, "Using ideas from natural selection to evolve synthesized sounds," presented at Proceedings of the Digital Audio Effects DAFX98 workshop, Barcelona, 1998.

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

[5] J. R. Koza, *Genetic programming II : automatic discovery of reusable programs*. Cambridge, Mass.: MIT Press, 1994.

[6] J. R. Koza, *Genetic programming III : darwinian invention and problem solving*. San Francisco: Morgan Kaufmann, 1999.

[7] R. A. Garcia, "Automatic Generation of Sound Synthesis Techniques," in *Program in Media Arts & Sciences*: Massachusetts Institute of Technology, 2001, pp. 98 p.

[8] J. R. Koza, F. H. Bennett, III, D. Andre, M. A. Keane, and F. Dunlap, "Automated synthesis of analog electrical circuits by means of genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 109 - 128, 1997.

[9] F. Gruau, "Cellular encodign of genetic neural networks," Ecole Normale Superiéure de Lyon, Lyon 92-21, May 1992 1992.

[10] C. Roads, *The computer music tutorial*. Cambridge, Mass.: MIT Press, 1994.

[11] G. Depoli, "A Tutorial on Digital Sound Synthesis Techniques," *Computer Music Journal*, vol. 7, pp. 8-26, 1983.

[12] R. C. Boulanger, "The Csound book : perspectives in software synthesis, sound design, signal processing, and programming,". Cambridge, Mass.: MIT Press, 2000.

[13] N. A. Gershenfeld, *The nature of mathematical modeling*. New York: Cambridge University Press, 1999.

[14] R. A. Garcia, "Growing Sound Synthesizers using Evolutionary Methods," presented at Sixth European Conference on Artificial Life. Workshop on Artificial Life Models for Musical Applications, Prague, Czech Republic, 2001.

[15] J. G. Roederer, *The physics and psychophysics of music : an introduction*, 3rd ed. New York: Springer-Verlag, 1995.

[16] K. C. Pohlmann, *Principles of digital audio*, 3rd ed. New York: McGraw-Hill, 1995.

[17] P. R. Cook, "Music, cognition, and computerized sound : an introduction to psychoacoustics,". Cambridge, Mass.: MIT Press, 1999.

[18] J. Chowning, "The synthesis of complex audio spectra by means of frequency modulation," *Journal of the Audio Engineering Society*, vol. 21, pp. 526-534, 1973.