Growing Sound Synthesizers using Evolutionary Methods

Ricardo A. Garcia

MIT Media Lab, Machine Listening Group
20 Ames St. E15-401, Cambridge, MA 02139, USA
rago@media.mit.edu

Abstract. An approach to automate the design of sound synthesis algorithms (SSA) is proposed. It uses custom descriptions of the SSA in the form of cyclic topology graphs and acyclic tree graphs with ordered branches. Evolutionary methods are used in two stages: a) for suggesting topological arrangements for the SSA functional elements, and b) for optimizing the internal parameters of the functional elements. An analytical distance metric that uses the complex spectrum of the target and test signals is proposed. An implementation of the approach is described and some initial results are outlined.

1 Introduction

A sound synthesis algorithm (SSA) is an algorithm that can be implemented in a digital computer to produce digital sound samples. A SSA can be designed to imitate some "natural sounds", or to create some "new" unrealistic sounds with certain desired characteristics. Also, it is common to use SSA as models of "real world" systems to explore and infer information about them. The design of SSA is a very hard problem. It usually requires of a careful analysis of the desired characteristics of the final sound, and good insight of the available modeling techniques. Human intuition is the main tool used for these designs. Some of the steps in the design of SSA have been successfully automated. Horner et al. used genetic algorithms to automate the estimation of parameters for FM synthesizers modeling trumpet, tenor voice and guitar tones (Horner, Beauchamp, & Haken 1993). When used GA to explore the parameter space, and to a limited extent, the connection space of FM-like synthesizers (When 1998). These approaches had in common that the main SSA function was given, and that the automation mainly explored the optimization of parameters, but no extreme changes in the arrangement of the functional elements inside the SSA was made. The proposed approach addresses the design of SSA as a search in the SSA space using evolutionary methods. It uses two descriptions of the SSA in the form of a topology graph and an acyclic tree graph with ordered branches. The tree representations are used by the evolutionary methods (i.e. genetic programming) to "grow" topologies capable of rendering sounds. These sounds are evaluated using a fitness function for similitude with a known target sound. The proposed approach is designed to allow (and encourage) novel SSA to be suggested. A discussion of the preparatory steps taken for this project are discussed in (Garcia 2000), and an overall description of the approach can be found in (Garcia 2001).

2 Sound Synthesis Algorithms (SSA)

2.1 SSA as Topologies

It is a common practice to represent (and depict) a SSA as a topology graph with interconnected functional blocks (Boulanger 1999). The term topology will be used to address a particular arrangement of blocks and their connections. A topology and its related SSA are equivalent representations of the same model; therefore, a change in one of them will be reflected in the other.

$$x(t) = e\sin(a(c + \sin(f))) \quad (\mathbf{1})$$

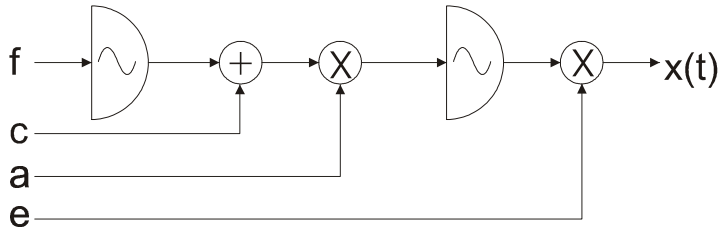

Fig. 1. A simple sound synthesis algorithm (Eq. 1) and its equivalent topology.

2.2 SSA Space

A topology is composed of interconnected functional blocks. Our working definition of a SSA space (topology space) is: *"the space spanned by all the possible combinations of blocks and their connections"*. This space should be restricted to have a countable (and preferable small) number of types of blocks, and also, a restricted number of total blocks per topology. A valid topology is one where all the blocks have all their pins connected (i.e. there are no dangling connections). A not valid topology is not considered as part of our SSA space.

2.3 Suggested Functional Blocks

After analyzing several "classic" SSA (Boulanger 1999), a set of functional blocks (in Table 1) is suggested. Most of the common SSA use these kind of blocks to achieve their results and some of the more complex functions can be reproduced as a combination of them. In addition, we use a "standard" interface for the connection of the blocks. We define four types of blocks: Source (1 output), Render (1 input), Type_A (2 input, 1 output) and Type_B (1 input, 2 outputs). Every type of block can host different types of functions inside, but the number of inputs and outputs restricts their connections. This standardization gave us a solid framework to develop our approach and at the same time to make it realizable in a digital computer (see section 4)

NAME	TYPE	FUNCTION
Oscillator	Type B	sin, triangular, square
Filter	Type B	(two poles, two zeros
Split	Type B	splits a value in two streams
Constant	Type B	
Add	Type A	
Mult	Type A	
Filter+Oscillator	Type A	oscillator with filtered output
Source	Source	
Render	Render	

Table 1. Basic building blocks, type and function

2.4 Manipulation of Topologies: Alternative Representation

Search in the topology space requires manipulation of the topologies. These manipulations include adding, deleting, and moving blocks and their connections. If a cyclic topology graph representation is used for the manipulation, it is very complicated to keep a topology valid after a series of transformations. In example, if a new block is added, it is possible that the number of inputs/outputs available at the time is not balanced (more inputs or outputs available), then the topology at this step will not be valid, and further changes (additions or deletions) have to occur to make it valid. This becomes intractable when the topology is too big.

Problems similar to this one that involved search in spaces represented by topology graphs were faced by Koza et al. (Koza, Bennett, Andre, Keane & Dunlap 1997; Koza, Bennett, Andre & Keane 1999) and Gruau (Gruau 1992). Koza used evolutionary methods to evolve descriptions of analog circuits. Gruau represented families of neural networks. The main idea is to represent the cyclic topology graph by an acyclic tree graph with ordered branches. The nodes of the tree will have instructions or primitives that affect the development of an embryonic topology. The trees actually encode "how the topology grows". The result after applying the instructions in the tree to the embryo is a fully developed and valid topology. The advantage of this extra representation is the flexibility in manipulation compared with the cyclic topology graph. It is easier to copy, mutate or do crossover on the trees.

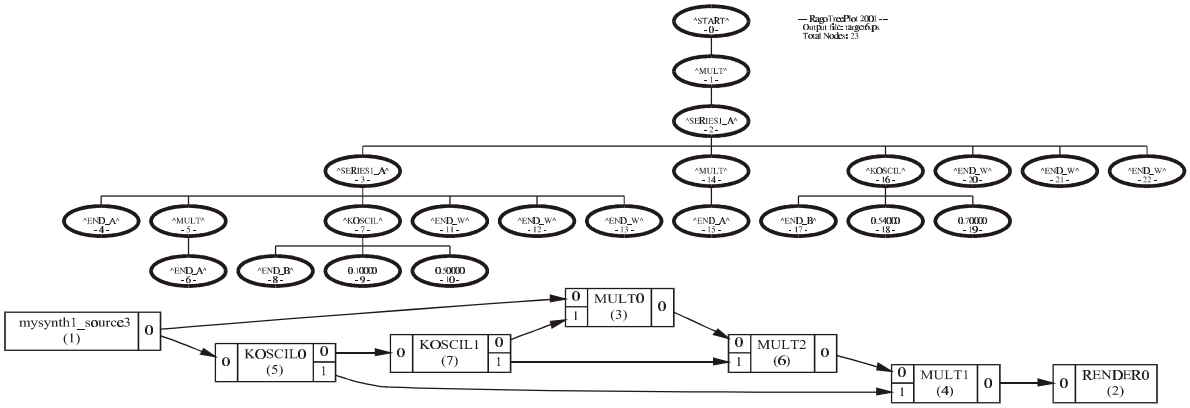

Fig. 2. Tree (top) and evolved topology (bottom) for a particular SSA

The instructions present in the nodes are divided in three main types (Koza, Bennett, Andre, Keane & Dunlap 1997):

Topology Modifying Functions (TMF): These modify the topology by doing a series or parallel operation on the selected block. Any of these operations inserts new blocks and does the necessary connections to keep the topology valid.

Development Control Functions (DCF): these don't modify the topology, but insert "nop" or "end" operations that delay or stop the branch of developing the topology.

Construction Continuing Subtrees (CCS): Selects between three main options: assign a type to the actual block, or perform a TMF or DCF to the block.

3 Searching the SSA Space

The original problem of searching for a topology capable of synthesizing a sound close to the target sound is restated as: "search for a tree description that will grow a topology capable of synthesizing a sound close to the target sound." Every tree maps into a single topology (but the converse is not always true, a topology can be represented by many different trees). We will work under the assumption that the tree space maps a big portion of its associated topology space; therefore, a search in the tree space will be good enough to find an acceptable solution in the topology space.

Tree space has many dimensions. To make things more complicated, every tree could have "different" dimensions when related to the other trees. The dimensions are related to the number and type of functional blocks and connections of the topology associated with the tree. Evolutionary methods as genetic programming have shown good results searching in these complicated types of spaces (Koza, Bennett, Andre, Keane & Dunlap 1997; Gruau 1992).

3.1 Genetic Programming

It uses a population of candidate solutions that are evaluated and assigned a fitness value. A new population is created using genetic operations (copy, crossover, mutation) that modify the internal structure of selected individuals. The individuals are selected probabilistically based on their fitness value (the better the fitness, the more probability of been selected). The fitness function measures the individual's performance in some sort of test or comparison with respect to a target. This loop is repeated until an individual with the desired fitness threshold is found, or the maximum number of generations is reached. This approach has shown to converge to acceptable solutions in problems involving exploration of topology spaces (Koza, Bennett, Andre, Keane & Dunlap 1997; Koza, Bennett, Andre & Keane 1999;Gruau 1992).

For our approach, the individuals are tree descriptions of SSA, gropued in populations. The fitness evaluation comprises: mapping of the tree in a working topology and sound being generated from it. A fitness function is applied to compare the generated sound with the target sound to find some kind of distance metric between the two.

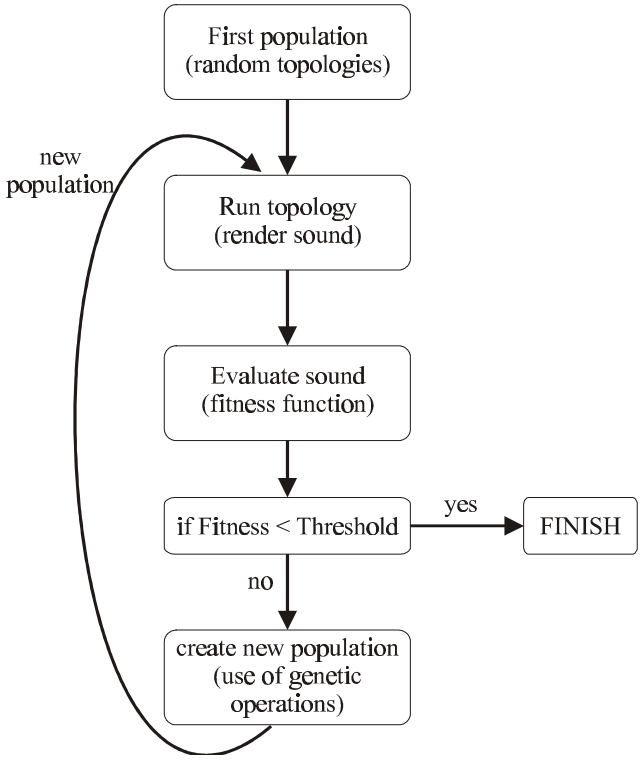

Fig. 3. Genetic Programming loop

3.2 Fitness Function

The fitness function is in charge of evaluating the performance of the individuals compared to a target sound, and guides the search in the SSA space. Different sound distance metrics have been proposed in the literature. They are divided in two main groups: analytical and perceptual. Saint-Arnaud (Saint-Arnaud 1995) proposes a cluster-weighted model of a sound, and uses the analytical distance between respective clusters of the associated models for two sounds. The drawback in this approach is that only works with long sounds, especially sound textures that have repetitive elements over time. Horner et al. (Horner, Beauchamp, & Haken 1993) use an analytical spectral distance metrics that involve the MSE of the magnitude spectrograms of two sounds. Wun et al. (Wun & Horner 2001) propose a distance metric that includes perceptual simultaneous frequency masking. This metric compares the MSE distance of the "perceptually filtered" magnitude spectrums. Our approach requires a distance metric suitable for short duration sounds. Perceptual distance metrics are tempting to use because they could reduce the computation time by orders of magnitude: there are more "perceptually similar" solutions in the space than analytical solutions. Because of the experimental nature of the approach, an analytical distance metrics is proposed and used to develop a better understanding before incorporating perceptual issues. The proposed analytical distance metrics (Garcia 2001) works with the distance between the complex spectrums of the target and testing sounds:

$$MSEmag = \frac{1}{FRAMES} \sum_{j=1}^{FRAMES} \sum_{i=1}^{Freq.Bins} \left[\left(|X_{ij}| - |T_{ij}| \right)^2 Wm_{ij} \right] \quad (2)$$

$$MSEph = \frac{1}{FRAMES} \sum_{j=1}^{FRAMES} \sum_{i=1}^{FreqBins} \left[\left(ang(X_{ij}) - ang(T_{ij}) \right)^2 Wp_{ij} \right] \quad (3)$$

$$MSEtotal = a \times MSEmag + b \times MSEph \quad (4)$$

with:

$$Wm_{ij} = O + (1 - O) \frac{\left(\log|T_{ij}| - \min\left(\log|T_{ij}| \right) \right)}{\left| \min\left(\log|T_{ij}| \right) - \max\left(\log|T_{ij}| \right) \right|}$$

and

$$Wp_{ij} = c \times Wm_{ij}$$

This defines an asymmetric distance metric for the complex spectrum of two sounds: target T_{ij} and rendered X_{ij}. The weight matrix Wm_{ij} has values in the range [0,1]. The offset O should be positive and less or equal to 1. This weight matrix emphasizes the spectral peaks (regions of more energy), and as a consequence, the error in these regions is reduced. The coefficients *a,b* and *c* are used to fine tune the impact of each factor on the overall metric.

3.3 Genetic Operations

Genetic operations (copy, crossover and mutation) are applied to selected individuals when creating new populations. The selection of the individuals is probabilistic, and it is based on their fitness value. The better the fitness value, the higher the probability of being selected to be part of a genetic operation. Because these operations are done in the tree representation of the topologies, they follow the normal approach suggested in (Koza, Bennett, Andre, Keane & Dunlap 1997), but paying attention to the "type" of node that they involucrate.

3.4 Sub optimization

It is convenient to see the search in the SSA space as two separate steps. The first step is to "suggest" a topology (distribution, types and connections of the blocks). The second step is the "fine tuning" of the internal parameters relevant to each type of block. After a topology and its initial parameters are suggested (i.e. an individual is created from genetic operations of individuals drawn from the previous generation), a sub optimization of their internal parameters is performed. This optimization will not change the actual layout of the topology, but just "fine tune" the SSA for the given target sound. In example, an oscillator in the actual topology can be tuned to produce a frequency relevant to the target sound, not a "random" frequency that could be suggested by a genetic operation. In early stages of the development

of the suggested approach, this sub-optimization was not performed, and a tendency of rejecting or penalizing with bad fitness values some otherwise good topologies was noted. The optimization method used is genetic algorithms, but many others can be used (Gershenfeld 1999). Note that at this stage, the number of parameters to optimize is known and fixed, and the fitness function could be the same used for the main GP loop.

4 Implementation

The proposed approach was implemented as a set of programs and scripts in C++ and matlab. The C++ binaries were focused on the manipulation of trees, creation and management of populations, and in the compilation and execution of topologies when rendering sound. The Matlab scripts were in charge of the main Genetic Programming loop, as well as the fitness function. This approach gave flexibility to test different fitness functions in an easy way, but the evaluation speed was greatly hurt. The results of the fitness function testing will be published in the near future.

A custom sound synthesis platform that uses a message driven system to deliver sound samples between the blocks in the topology was used. This approach ensures that any valid topology suggested can be tested.

An initial population of random trees was generated. Every tree was mapped into a working topology, and then it was sub-optimized to “fine tune” the internal parameters of each block. The sound produced with the optimized version of the individual was then passed to the fitness function and a fitness value was assigned to this individual. The GP loop continued performing genetic operations on the fittest individuals and creating new populations until a suitable solution was found.

For the actual test run, a set of target sounds comprising organ, clarinet, piano, and two synthetic tones were used. Their duration varied from 1 to 2 seconds, and they had one to three notes in different pitches. Their sampling frequency was 8000 Hz.

A typical population ranged from 40 to 70 individuals. The total number of generations varied from 200 to 500. The first population was always randomly generated (no knowledge about the target sound was used when generating the first population). The sub-optimization was done using a genetic algorithm, which used a variable number of individuals randomly selected from 1 to 15, and the number of generations varied from 1 to 7.

These test were run in a Pentium III, dual processor of 733 MHz, with 256 MB of ram, running Windows NT 4.0 and Matlab 6.0. Each run took on the range of 10 hours to complete.

5 Results

All the relevant data from every run was stored for further analysis. The information about the “best of generation” individual was paid special attention. The sound produced by the best of generation on every generation was compared to the target sound in an informal listening test. The fitness value for the best of generation was used as a measure for convergence of the approach. An example of the results for a piano sound (note C3) is plotted in Figure 4.

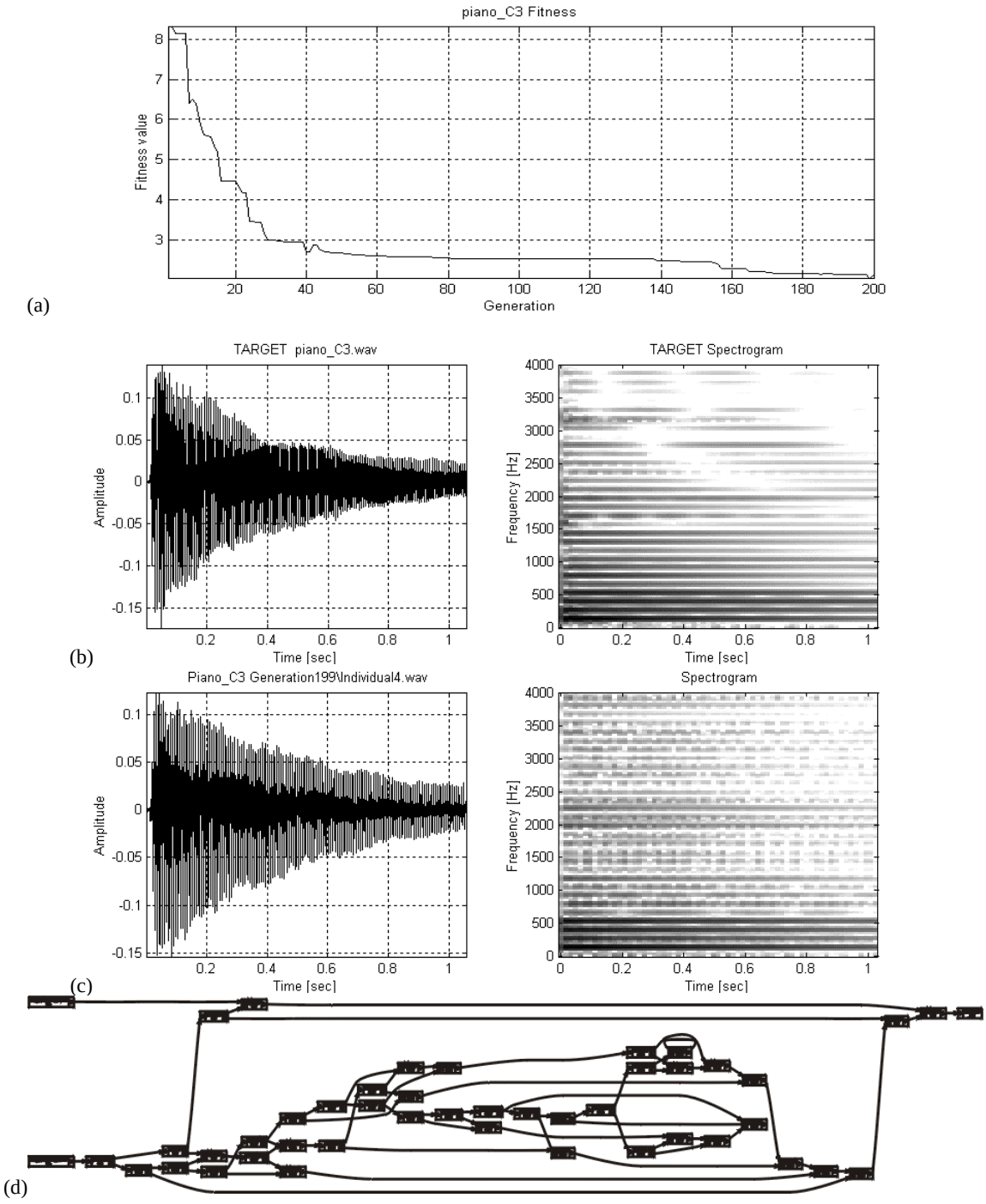

Fig. 4. Results for piano note C3. (a) fitness value over 200 generations, (b) waveform and magnitude spectrogram for the target sound, (c) waveform and magnitude spectrogram for sound produced by the best individual of generation 199, (d) bird-eye view of evolved topology for same individual

6 Conclusions

An approach to automate the design process of the sound synthesis algorithms using evolutionary methods was proposed. The approach was shown capable of suggesting plausible topologies for rendering sounds similar (based on the fitness function criteria) to the target sound. Genetic programming was the evolutionary method used, and a custom description of the SSA and topologies in the form of acyclic tree graphs with ordered branches was used. The proposed fitness function used the analytical distance between the weighted magnitude and phase spectrogram of a test and target sounds. A small number of experiments were run, but their results are significant and show the potential of using evolutionary methods as a credible approach to automate the design of SSA.

7 Acknowledgments

The author wishes to express gratitude to his colleagues at the Machine Listening Group at MIT Media Lab, in particular to Paris Smaragdis for his ideas and support during the development of this research. Special thanks to Una-May O'Reilly for her guidance in the field of evolutionary computation.

8 References

Boulanger, R. (1999). *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming*. Cambridge: MIT Press.

Garcia, R. A. (2000). Towards the Automatic Generation of Sound Synthesis Techniques: Preparatory Steps. *109th Convention Audio Eng. Soc. Preprint 5186.*

Garcia, R. A. (2001). Evolving Sound Synthesis Algorithms. *submitted to IEEE Waspaa'01*

Gershenfeld, N. (1999). *The Nature of Mathematical Modeling*, New York: Cambridge University Press.

Gruau, F. (1992). *Cellular Encoding of Genetic Neural Networks (Tecnical Report 92.21).* Laboratoire de L'Informatique pour le parallélisme, Ecole Normale Supériure de Lyon.

Horner, A., Beauchamp, J., Haken, L. (1993). Machine Tongues XVI: Genetic Algorithms and Their Application to FM Matching Synthesis. *Computer Music Journal*, 17:4, pp. 17-29.

Koza, J. R., Bennett III, F. H., Andre, D., Keane, M. A., Dunlap, F. (1997). Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming. *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 2.

Koza, J. R., Bennett III, F. H., Andre, D., Keane, M. A. (1999). *Genetic Programming III, Darwiniang Invention and Problem Solving*, San Francisco: Morgan Kaufmann Publishers.

Saint-Arnaud, N. (1995). *Classification of Sound Textures*, MS thesis MIT Media Lab.

When, K. (1998). Using Ideas from Natural Selection to Evolve Synthesized Sounds. *Proc. Digital Audio Effects (DAFX), Barcelona,*

Wun, C.-W., Horner, A. (2001). Perceptual Wavetable Matching for Synthesis of Musical Instrument Tones. *J. Audio Eng. Soc.* 49: 250 – 262.