# GENESYNTH: NOISE BAND-BASED GENETIC ALGORITHM ANALYSIS/SYNTHESIS FRAMEWORK

*Michael Chinen* *Naotoshi Osaka*

Tokyo Denki University
School of Science and Technology for Future Life
2-2 Kanda-nishiki-cho, Chiyodaku, Tokyo, 101-8457, Japan
mchinen@gmail.com, osaka@im.dendai.ac.jp

**ABSTRACT**

Genesynth is an analysis/synthesis framework that uses a genetic algorithm to search for a noise band sound model. The framework is written as an open source C++ tool, which allows for both the modification and resynthesis of found sound models and also genealogical exploration of the generations of sound variations created as a side effect of the genetic algorithm.

The genetic algorithm used is specialized for the problem of audio analysis by using variable chromosome length as well as hierarchical chromosome structure. The algorithm's fitness function compares cached and compressed FFT data against estimated noise band models represented in the chromosomes.

The noise band model is synthesized using sinusoids that are stochastically modulated in frequency to achieve a flexible bandwidth.

## 1. INTRODUCTION

Analysis/synthesis frameworks have been proven useful as a method of sound manipulation, offering the benefits of higher-level models to control sound. Frameworks are often classified by their definition of sound models and units of synthesis. Established examples include sinusoidal models such as PARSHL[1] and the MQ algorithm[2], and Hybrid examples such as SMS[3] and ATS[4], and noise-band models such as Loris[5]. These frameworks are often judged on their abilities to reproduce a class of sounds accurately. While accuracy is clearly an important feature, the authors also feel that for those using the frameworks as a means of sound manipulation, it would be valuable if the framework also included a high-level means for said sound manipulation. With this in mind, we have developed an analysis/synthesis framework, using a genetic algorithm (GA) as the means for manipulation and variation of sound, and a noise-band method of synthesis.

A genetic algorithm is a type of search algorithm that searches the problem space by using the concepts in evolution theory such as fitness and selection. Search algorithms usually have the property of creating a smooth path from the starting point to the solution. In most applications, the midpoints along the path are simply discarded. However, when interested in the variations for artistic purposes, these intermediary solutions are interesting. In the case of analysis/synthesis, the traversal of this path means a series of sound variation, starting from noise, gradually getting closer and closer to a sound model that fits the input.

It should be noted that genetic algorithms have been used in many musical and sound based applications[6]. Horner in particular has done related work on using GAs in parameter estimation for wavetable synthesis[7]. However, perhaps because of the large search space of digital audio, the current applications have either restricted search spaces, or require human interaction to aid the fitness function. Because none of these current genetic algorithms fit the needs of a general-purpose analysis engine, we have designed our own.

As this framework does not compute the model directly from the FFT, but instead searches for a solution starting from random noise, analysis is a time intensive operation, during which incrementally better results are found. Generally, the algorithm finds some recognizable features of both noisy and pitched input within a short period of time (seconds,) although an identifiable sound can take much longer, and in many cases a good but far from optimal solution may be the end result. Because the purpose of the system is for music and art, the user will be interested in the sound variations created as a side effect by the GA, an thus an imperfect solution may be more acceptable.

In this paper, we describe a hierarchical chromosome structure as a sound model. We then provide a GA specification that outlines the search component. Next, the equations for synthesizing a chromosome are specified. Automatic side-effect and manual variation and sound manipulation for the purpose of music and art are then discussed. Finally, we conclude with future directions

## 2. GA ANALYSIS SPECIFICATION

### 2.1. Sound Model

The sound model of Genesynth was designed with two values in mind. As a GA, the model must be efficient with space and modular. It should also be flexible enough to represent noisy, inharmonic, and harmonic audio. In Genesynth, sound is described in a data structure called a *chromosome*. The chromosome is defined as a collection of smaller data components described in table 1.

**Table 1.** Data members of Chromosome

| Component | Contents |
|---|---|
| *Chromosome* | List of *SoundCells* |
| *SoundCell* | *NoiseBand*, list of child *SoundCells*, list of *PlaceGenes* |
| *NoiseBand* | frequency, amplitude, bandwidth |
| *PlaceGene* | Sample number, interpolation coefficients for frequency, amplitude, and bandwidth. |

The chromosome structure takes several deviations from that of a standard GA. Firstly, to save space and speed up the GA, the chromosome is of no fixed length – mutations can cause growth or pruning. Secondly, Genesynth chromosomes have a hierarchical structure with allele markings, for the purpose of modeling hierarchical sound phenomenon such as harmonics. As a result, this requires a non-standard crossover algorithm.

In the following subsections, the components of a chromosome relevant to the sound model are described, starting with the most contained (deepest) in the hierarchy.

#### *2.1.1. NoiseBand*

*NoiseBands* are the form the basis of the sound model in Genesynth, which can be likened to a unit generator or synth. As a data object, a NoiseBand contains three parameters: amplitude $a$, frequency $f$, and a bandwidth ratio $bw$, (the ratio of bandwidth to frequency.) A NoiseBand represents a band of spectral energy centered around its frequency parameter, with power uniformly distributed over it such that its integral will be equal to the amplitude. A bandwidth of zero then represents a pure sinusoid, and a bandwidth of 1.0 represents noise energy from 0hz to $2f$.

#### *2.1.2. PlaceGene*

The NoiseBand has no time information in any of its data members. Thus, on its own, it represents a static, non changing sound. Real sound is always changing; Genesynth uses *PlaceGenes* to enable a NoiseBand to change its power spectrum through time by acting as nodes from which to hang coefficients for amplitude, frequency and bandwidth at a certain sample.

#### *2.1.3. SoundCell*

A *SoundCell* contains a NoiseBand, a list of PlaceGenes, and children SoundCells. Child SoundCells "inherit" the PlaceGenes of its parent, if the parent's PlaceGenes have a flag set. This is intended to model relationships between partials and noise energy bands such as the way partials and noise in an instrument sound tend to increase and decrease in amplitude together. Having this kind of relationship defined also gives a higher level of control for modifying the sound model later.

### 2.2. Search Method

The Chromosome structure determines how flexible the sound model will be. The rest of the GA is concerned with how to manipulate chromosomes to explore the search space (mutation and crossover,) and how to determine if our searching is progressing (fitness function.)

The overall flow of Genesynth's search method is no different than any other GA:

```
Initialize a population of size p
Score the population.
Repeat until we have found an acceptable score:
  Repeat until the next population is of size p
    Randomly select two high-scoring chromosomes
    Mutate the chromosomes with probability m
    Crossover the chromosomes with probability c
    Add the chromosomes to the next population
  Score the next population
  Use the next population as current
```

The implementation of the above pseudocode is broken up into five smaller methods that are described below.

#### *2.2.1. Initialization*

Chromosomes are initialized as having one SoundCell with a NoiseBand and a random number of PlaceGenes with random parameters within sensible predefined parameter limits.

#### *2.2.2. Mutation*

Genesynth chromosomes are defined as hierarchical data objects, instead of the bit strings commonly found in GAs. This requires a custom mutation (and crossover) method. From the top of the hierarchy, (at the Chromosome level,) each of its contained data objects are mutated with a probability $m$. If a complex data structure (SoundCell, PlaceGene, or NoiseBand) is mutated, all of its contained objects are mutated, resulting in a depth first traversal of the chromosome's components. When a number object is mutated, the parameter value is raised or lowered by a random percentage selected by an exponential distribution, and clamped to keep it within the parameter's predefined limits.

The components are then mutated with the same probability $m$ to have its subcomponents deleted, or moved up or down a level in the hierarchy in the case of SoundCells.

The final step is the addition (with probability $m$,) of newly initialized subcomponents, with a few exceptions. Depending on a coinflip, a SoundCell will split its NoiseBand up into smaller NoiseBands that occupy a subset of the original NoiseBand spectrum by adding children and modifying its own NoiseBand. This feature helps speed the search along in a divide-and-conquer methodology.

### 2.2.3. *Crossover*

The crossover method in Genesynth is perhaps the most complicated. It uses a hierarchical crossover similar to one described by Bently[5], which differs in that it uses allele markers to indicate sections in the chromosome that have a certain history and as such SoundCells with identical markers probably have a similar spectral representation. Crossover is done uniformly, so a digital coin toss is performed to see if a given allele should crossover. Two chromosomes that both contain a SoundCell with the same allele marker will always have its subcomponents crossover with each other. Therefore, the SoundCells' PlaceGenes and NoiseBand will crossover with each other, but as the child SoundCells, having alleles, they will try to find their corresponding matches. In the case of a SoundCell that does not have a matching allele, it is simply copied to the chromosome that wins the coin toss.

While crossover is complicated at the complex-object layer, it becomes very simple once it reaches the parameter level, because it can then be treated as a regular linear crossover. When crossing over a parameter $a$, we generate the crossover from the current partner values $a_1$ and $a_2$ by linearly interpolating a random amount toward the partner's value. This technique can be looked at as the analog to bit swapping uniform crossover for our floating-point parameters, since randomly bit swapping two integer bits will generate two numbers somewhere between the originals.

### 2.2.4. *Selection*

To form the next population of chromosomes, we rank the chromosomes in the current population by score and then assign to each chromosome a probability of selection proportional to its ranking.

### 2.2.5. *Fitness Function*

The fitness function essentially determines which paths the search should continue to explore. The goal is to compare the input sound to the sound model stored in the chromosome. Ideally, the fitness function should be consistent and efficient. These ideals cannot be easily achieved with the naïve method of scoring by comparing the samples of the original to the synthesized – it is not efficient, as synthesizing and comparing each sample takes too long, and it is not consistent, as stochastically synthesized noise bands would yield many different waveforms, and thus scores for the same model.

With these considerations in mind, Genesynth's fitness function computes a score by comparing a windowed spectrum estimate of the chromosome with a cached and compressed bins from the STFT of the input.

Runs of STFT bins are converted into a bands if they are long enough and are greater than a threshold amplitude A. Within these bands, STFT bins are quantized and grouped into wider bins. The result is that we are able to represent wide bands of noise energy with very few bins. This is a form of compression that helps the GA speed up. This can be seen in fig. 1.

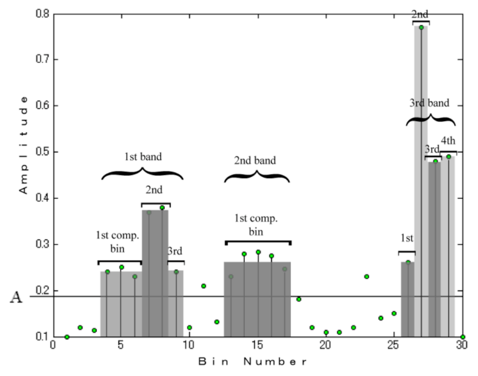

**Figure 1.** Compression of energy bands from FFT bins

There is special case of not compressing bins immediately after a big decibel leap for the purpose of catching sinusoidal peaks, where we need the neighboring bins to "triangulate" where the main lobe's center is.

To compare the compressed spectrum to the chromosome, we need a method of estimating the *windowed* spectrum (including lobes) for a NoiseBand. Then we can compute the score for a frame in a way that resembles SNR:

$$\text{Score} = s\,/(1+w) \tag{1}$$

with

$$s = \frac{\text{energy of input satisfied}}{\text{total energy of input}}$$

$$w = \frac{\text{energy of chromosome not used}}{\text{total energy of input}}$$

The scoring function rewards Chromosome NoiseBands that overlap input bands. So long as the NoiseBand lobe estimator is defined to be monotonically decreasing as outside of the bandwidth area, the genetic algorithm will use the lobe slope as feelers to slide and fit into spectral hills. This effect can be seen in fig.2.

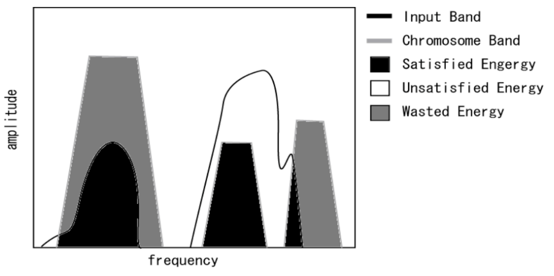

**Figure 2.** Comparison of Input and Chromosome

The spectral score is then just the average of all frames for the chromosome in question. To encourage efficiency, the chromosome score is finally penalized by a small amount proportional to the number of SoundCells it contains.

## 3. SYNTHESIS

The NoiseBand represents a band of energy specified by amplitude, bandwidth, and frequency. To synthesize this, we use a stochastically frequency modulated sinusoid. For the *k*th NoiseBand we can compute the *n*th sample is as:

$$s_k(n) = \mathrm{A}\cdot\cos(\,\theta\,(n)\,) \qquad (2)$$

with

$$\theta\,(0) = \mathrm{c}; \quad \theta\,(n) = \theta\,(n\text{-}1) + 2\pi f(n)/\mathrm{SR}$$

$$f(0) = \mathrm{F}; \quad f(n) = f(n\text{-}1) + [f_{\mathrm{target}}(n)\text{-}f(n\text{-}1)]/[r(n)+1]$$

$$f_{\mathrm{target}}(\mathrm{n}) = \begin{cases} f_{\mathrm{target}}(\mathrm{n} - 1) & \text{if } r(n) > 0 \\ \mathrm{random}((1.0 - \mathrm{BW})\cdot\mathrm{F}, (1.0 + \mathrm{BW})\cdot\mathrm{F}) & \text{otherwise} \end{cases}$$

$$r(0) = 0;$$

$$r(\mathrm{n}) = \begin{cases} \mathrm{r}(\mathrm{n} - 1) - 1 & \text{if } r(n-1) > 0 \\ \mathrm{random}(0, \mathrm{SR/F}) & \text{otherwise} \end{cases}$$

where SR is the sample rate, A, F, and BW are the NoiseBand's amplitude, frequency, and bandwidth parameters, and random(*x*,*y*) is a uniformly random value between x and y. This will yield a pure sinusoid if BW is zero and toneless noise if BW equals one.

The entire chromosome waveform can then be synthesized as

$$s(n) = \Sigma\, s_i(n) \qquad (3)$$

## 4. MUSICAL RESULTS

The main goal of this work is to provide a artistic and musical approach to the analysis/synthesis problem. The following sections describe typical usage of the algorithm to generate creative content.

### 4.1. Search Variation Trend

Instead of using the much faster traditional method of computing the solution directly, we use a GA to search for it. Depending on the input, it can take minutes or hours to find a good solution. The musical benefit of using a search method lies in its automatically generated intermediate solutions. Knowledge in the way these sound models tend to form may provide the user some insight. In the results described below, a generation size of 20 chromosomes was used.

The search for any input tends to start with a whitish noise band that quickly shrinks down to a band the width of the spectral range and amplitude envelope of the input within the first 10 to 50 generations. If the input contains narrow bands or sinusoids, this main chunk of noise is quickly broken up into narrow bands that slowly tighten around the peaks. Finally, the frame by frame amplitude differences are sought out.

For pitched sounds the algorithm creates variations that bring in partials out of fuzz-like noise that allows the partials to be out of tune without sounding so harsh. For sounds composed of mostly fast changing noise, such as thunder, dynamic wind-like sounds appear.

The convergence rates for different sounds vary largely based on how dynamic the sounds are, as well as their length. A 3 second bass example was synthesized on Apple PowerBook G4 under 2dB of spectral distortion in 30 minutes, yielding around 20,000 variations. A 7 second thunder sample may take four times longer and still contain artefacts. However, it takes just a few seconds to produce potentially useful sound variations. The running time of the algorithm is improving as development continues.

### 4.2. Family Tree

Each *generation* (population) created by Genesynth is saved to a text file. After the analysis is over, the user can create a family tree of a specified chromosome, tracing either its best parent or worst parent back to the first generation. The user can then use this list as an instrument, by synthesizing different generations one at a time, or several all together at once.

## 5. CONCLUSION

Genesynth is still under development. An OS X GUI is in development and should help make Genesynth more accessible. The project can be found under SourceForge under the project name Genesynth at www.sourceforge.net.

## 6. REFERENCES

[1] Smith JO, and Serra, X. 'PARSHL: A Program for the Analysis/Synthesis of Inharmonic Sounds Based on a Sinusoidal Representation", *Proceedings of the International Computer Music Conference*, 1987.

[2] McAulay, R. J., and T. F. Quatieri, "Speech Analysis/Synthesis Based on a Sinusoidal Representation," *IEEE Trans. Acoustics, Speech, and Signal Processing*, Vol. 34, No. 4, pp. 744–754, 1986.

[3] Serra, X. "Musical Sound Modeling with Sinusoids plus Noise". G. D. Poli and others (eds.), *Musical Signal Processing*, Swets & Zeitlinger Publishers, 1997.

[4] Pampin, J. "ATS: a Lisp Environment for Spectral Modeling" *Proceedings of the International Computer Music Conference,* Beijing, 1999

[5] Fitz, K., Haken, L., Lefvert, S., and O'Donnel, M. "Sound Morphing using Loris and the Reassigned Bandwdith-Enhanced Additive Sound Model: Practice and Applications", *Proceedings of the International Computer Music Conference*, Gotenborg, Sweden, 2002

[6] Dahlstedt, P. "Sounds Unheard Of" *Ph. D. Dissertation,* Chalmers University of Technology. 2004

[7] Horner, A., "Wavetable Matching Synthesis of Dynamic Instruments with Genetic Algorithms," *Journal of the Audio Engineering Society*, 43(11), 916-931, 1995