2006－MUS－66 (19)
2006／8／8

Genesynth: 遺伝的アルゴリズムによる音合成

Michael Chinen 小坂 直敏
東京電機大学
〒101-8457 東京都千代田区神田錦町 2-2
mchinen@gmail.com, osaka@im.dendai.ac.jp

ユーザが定義する粒度での音モデルを自己参照する機構により、オーディオ探査空間を探求するために遺伝的アルゴリズムを用いた分析／再合成モデル Genesynth について述べる。

Genesynth: Sound Synthesis via Genetic Algorithm

Michael Chinen, Naotoshi Osaka
Tokyo Denki University

Sound synthesis model, Genesynth, is described which is an analysis/resynthesis framework using a genetic algorithm. The model explores audio search space by embedding self-referential sound models with a user-specified granularity.

1. Introduction

Analysis/synthesis frameworks are important tools, appreciated for the abilities of reproduction and artistic creation of sound, and also for the sound models they form. These frameworks can be categorized into families, such as Serra's SMS[1] and the McAulay-Quatieri MQ[2] algorithm, or as variants or derivatives within these families, such as Pampin's ATS[3] of SMS. However, there is little work done on using genetic algorithms for robust analysis/synthesis that can compare to the aforementioned frameworks. Genetic algorithms, (and search algorithms in general,) have the property of creating a path from one point in the search space to a solution, generating many intermediary solutions along the way. In audio space, a traversal of a solution path can be viewed as sound morphing. Furthermore, the area in the search space surrounding a good solution contains other sounds of artistic interest. Thus, if a search algorithm could be used for analysis/synthesis, it could also be used for sound morphing and variation in a manner more directed and meaningful than manually or randomly adjusting the parameters of non-search based frameworks.

Perhaps because of the large search space of sound, existing genetic algorithm systems often use restrictions, such as assuming the target sound has a harmonic model in wavetable synthesis[4], disregarding the input sound as a goal for the purpose of creating novel sounds[5], or wandering around the search space without a fixed target[6]. These techniques, while certainly valid for creating certain types of sound, limit the possibilities of search to small areas of the search space. Furthermore, the precision of the acceptable area of the search space required by a good analysis/synthesis framework prevents these search algorithms as candidates in general purpose analysis/synthesis.

Genetic algorithms are search algorithms that search the problem space by combining solutions in a way that is analog to the natural passing of genes from parents to offspring. A unique characteristic of genetic algorithms is that its search paths produce a family tree for every solution. In the case of sound, these variations may useful for art. However, a naïve genetic algorithm representing its chromosome with n genes of 16-bit samples, where n is the number of samples in the input sound, will not be able to conduct its search efficiently. Furthermore, if the optimal solution is found, this type of representation will not yield a meaningful model that says anything more than looking at the binary samples of the input data. In order for this problem to use a genetic algorithm as an analysis/synthesis tool, it must use an underlying representation with some kind of model takes advantage of the general structure of sound.

In this paper, we describe an implementation using genetic algorithms in analysis/synthesis frameworks. Common design issues that would make the genetic algorithm impractical in this application are discussed, and their solutions follow. Improvements on the genetic algorithm that

are advantageous for sound are shown and explained. An implementation of a genetic algorithm using a sinusoidal based model is presented. The results of this implementation are given, demonstrating that this method is a viable option for analysis/synthesis and sound morphing.

This paper is organized as follows: First, the basic structure of genetic algorithm is reviewed, along with specific issues that apply to genetic algorithms for analysis/synthesis. We then present our implementation of an analysis/synthesis framework. Finally, our results and conclusions are considered.

2. Review of the Genetic Algorithm

A review of genetic algorithms is useful for understanding how they might be adapted to the problem of analysis/synthesis. The term "classical genetic algorithm," as used in this paper, refers to the simple genetic algorithm that is regarded as the simplest, earliest, and most common structure of genetic algorithms, although it is not strictly defined. All genetic algorithms maintain a certain number of candidate solutions, known as the population of chromosomes. Each chromosome is represented with a sequence of abstract data, which can be turned into a real solution, known as the individual.

2.1 Chromosome Structure

The structure of the classical genetic algorithm's chromosome is simple: a string or array of bits is used to encode a solution. As an example, in figure 1, three chromosomes represent numbers with its binary format.

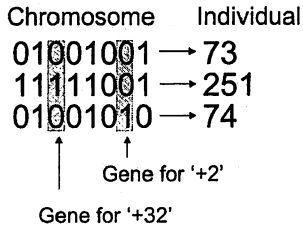

Fig. 1: Binary Chromosome Structure

This representation has the advantage of being general, as it can be used to represent any computational problem using the solution's binary format.

2.2 Chromosomal Operations

Each iteration of the genetic algorithm creates a new population by a crossover process that combines two chromosomes to make two new chromosomes, and also by a mutation process, which probabilistically alters a part, or gene of the chromosome. The mutation and crossover functions are easy to implement if a bit string is used, as can be seen in figure 2.

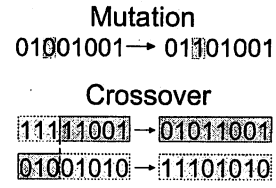

Fig. 2: Mutation and Crossover

A heuristic, called the fitness function, is used to score a chromosome by measuring the distance a chromosome is from a good solution. Chromosomes with higher fitness scores are chosen for crossover and mutation. Fitness scores are computed by measuring the distance away from an optimal solution as in Figure 3.

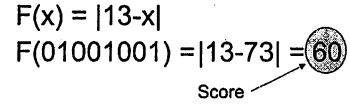

Fig 3: Scoring of the chromosome '01001001' in a search for the number '13'. In this example, lower scores are better.

2.3 Problems Using the Classical Genetic Algorithm for Sound

However, this kind of linear binary representation poses problems for sound. If chromosomes represent audio this way, (that is, each gene being one of n 16-bit samples,) then the chromosome is represented as a sequence of time. Consequently, the operations of crossover and mutation will cause non-linear changes in the time domain, producing sharp clicks and pops in the resultant sound that are unlikely to be desirable.

2.3.1 Size Solution

A representation that is smaller and more meaningful is needed, because many chromosomes will need to be generated and analyzed in

the lifetime of the genetic algorithm. The search space in an audio buffer with n 16-bit samples is huge – on the order of $O(2^{16^n})$. In theory, a smaller representation does not necessarily exclude sections of the search space if stochastic or variable length representations are used. To show an example in the sound domain, the part of search space that white noise occupies is large, but might be represented with just one stochastic gene. This is an example of one chromosome solution that can apply to many points in the search space.

In practice, the search space will be limited to a certain extent by the model that is chosen. If a model takes advantage of the fact that sounds have hierarchical components, such as harmonics and fundamental frequencies using a tree-like structure, the data required can be lessened without limiting the search space.

2.3.2 Continuity Solution

The problem of surviving the continuity of sound through mutation and crossover can be solved using a gene that implies interpolation. Interpolation will also supply the advantage of modeling the way sound components in real sound, like sinusoids, tend to bend and stretch over time. Looking at continuity from a Helmholtzian perspective, sound can also be exploited for data reduction in that there are certain relatively steady-state sections of sound that can be represented by its start and end times, and the information about what stays constant in this section, which could be frequency, amplitude, or phase acceleration. As we cannot know a priori how many "steady" sections a sound of a certain length will contain, this model needs to be implemented as a variable length chromosome, whose size can be arbitrarily restricted. The size restriction can impose a granularity upon the search space if the smallest optimal solution is of a larger size. If this granularity exists, it can be reduced to the knapsack fitting problem, which genetic algorithms have been proven useful for.

2.3.3 Crossover Solution

To implement the variable length and hierarchical structure suggested in the previous two sections, the chromosome can use a tree structure. This will make crossover more complex than in the classical genetic algorithm. This general problem of hierarchical crossover has been worked on since long ago. Bentley[7] describes a useful method of hierarchical crossover that can be applied for this problem.

2.3.4 Fitness Function Solution

Recalling that heuristics in effective search algorithms such as A*[8] provide an easy-to-compute estimate of distance from the solution, a more efficient fitness function can be implemented by caching a simple model, of the input sound that can be compared to the chromosome without ever synthesizing it.

This is much more efficient than taking the chromosome's sound model, synthesizing it, and comparing the distance to the original audio file by direct sample-by-sample subtraction. It has already been mentioned that the chromosomal representation data needs to be much smaller than the audio data because many chromosomes will need to be created in the course of the genetic algorithm. If the chromosomes are scored by the above sample-by-sample subtraction, much of this advantage is lost, because we must turn every chromosome into an audio buffer.

It is important to realize that the type of model used in the fitness function determines what type of sound the genetic algorithm can synthesize. Other parts of the genetic algorithm, such as the structure of the chromosome affect the type of sound that is synthesized, but mainly exist to make the genetic algorithm efficient.

3. Genetic Algorithm Implementation

Taking the above general considerations, we have implemented a framework for genetic algorithm analysis/synthesis using a sinusoidal model in C++. This framework is flexible in that adding another model on top of it is straightforward, using C++ classes to minimize the amount of code needed.

3.1 Chromosome Structure

The chromosome structure is a tree. The hierarchical unit of the chromosome is called a SoundCell. SoundCells contains two other data structures, the SinRoot and PlaceGene, which determine how sinusoids are synthesized. A table describing the attributes of these objects along with an illustration of a SoundCell and its components can be seen in figures 4 and 5.

Object Name	Attributes
SinRoot	-Frequency -Initial Phase -Amplitude

PlaceGene	-Interpolation Sample -Frequency Coefficient -Amplitude Coefficient
SoundCell	-SinRoot -List of PlaceGenes -List of SoundCells

Figure 4: Table of chromosome data structures

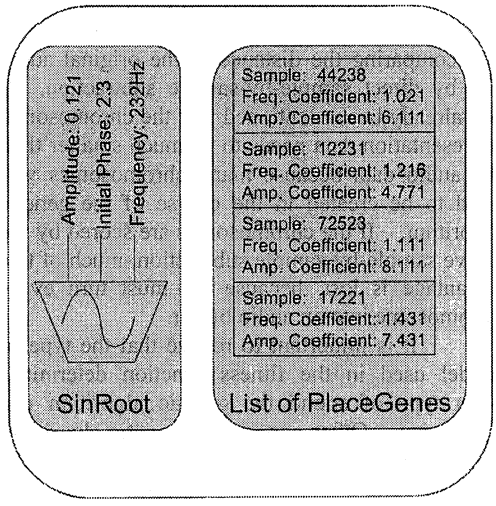

Figure 5: An example of a SoundCell

At the top level of the chromosome, there is a string of SoundCells, each which can have a number of its own SoundCells and PlaceGenes. An example of a what the entire chromosome may look like is illustrated in figure 6.

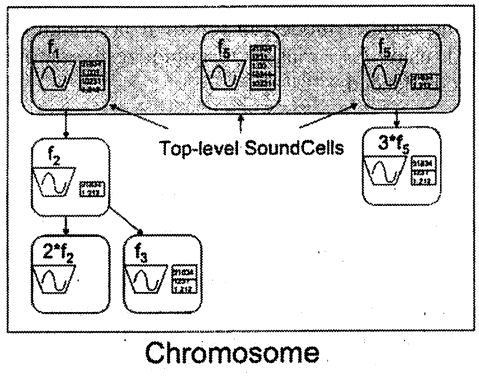

Fig. 6: Example of a chromosome's structure

3.2 Synthesis

Upon synthesis, each SoundCell synthesizes the tree at its root in a preorder fashion by concatenating its PlaceGenes to a list, creating the sinusoid in a buffer from the SinRoot and this list, then passing the list down to its child for the child to synthesize. Next, the SoundCell removes its PlaceGenes from the list, and the traversal continues. Because the PlaceGenes represent the frequency deltas as coefficients, child SoundCells will inherit the coefficients, (instead of a literal delta frequency,) and are thus able to generate sinusoids that bend proportionally to their frequency, as is the case with natural harmonics.

3.3 Initialization of Chromosomes

The first generation is created by initializing empty chromosomes, adding a random number of components, initialized as specified in figure 7.

Object Name	Initialaztion
SoundCell	A SinRoot and a random number of PlaceGenes are added and initialized
PlaceGene	Interpolation Sample and coefficients are randomized within a predefined range
SinRoot	Frequency, Amplitude, Initial Phase are assigned random values in a predefined range

Fig.7: Initialization Rules

3.4 Mutation of Chromosomes

Mutation is more complicated. Upon mutation, the SoundCell tree of each SoundCell is traversed, and every component is mutated with probability *m*, by changing its value randomly within a certain range. These mutations occur as listed in Figure 8.

Object Name	Possible Mutations
SoundCell	-SoundCells are added with a harmonic SinRoot -PlaceGenes are added by random initialization -PlaceGenes are added by cloning -PlaceGenes are copied to a child SoundCell -PlaceGenes and SoundCell children are deleted -SoundCell children are removed from the tree and added to the root level
PlaceGene	-Interpolation sample -Frequency coefficient

Object Name	Possible Mutations
(PlaceGene)	-Amplitude coefficient
SinRoot	-Frequency -Amplitude -Initial phase

Fig. 8 Mutation Rules

Creating SoundCells and PlaceGenes is complex for specific reasons. For example, copying a PlaceGene to one of the child SoundCells, gives the child a chance to 'override' the parent's inheritance through subsequent mutations. The cloning technique is useful because existing coefficients that are likely to be useful at some other time due to the highly continuous nature of sound. It should be noted that the structure of a SoundCell is initialized as absolutely harmonic, but can easily contain inharmonic parts through mutation. Furthermore, once the child SoundRoot's have their own PlaceGenes that interpolate frequency, the resultant sound will not have a absolutely harmonic structure, but a realistic near-harmonic structure, with the instantaneous frequency of the harmonics being slightly more or less than an integer multiple of the fundamental.

3.5 Crossover of Chromosomes

Crossover is implemented using the hierarchical method described by Bentley[7]. Two chromosomes are randomly selected with a probability proportional to their fitness score. Crossover happens recursively, starting at the top level of the chromosome. A random top level SoundCell is first chosen as the splitting point. As each SoundCell in one chromosome is traversed, an equivalent SoundCell in the other chromosome is searched for. If it is found, they trade PlaceGenes by splicing the PlaceGenes at a random sample. The children SoundCells then do crossover in the same fashion. If there is not equivalent SoundCell, it is simply copied over to the other one if it is past the splitting point.

3.6 Fitness Function

The scoring of a chromosome is done by comparison of its peaks against the input sound's peaks. The peaks in the input file are computed only once and cached, by taking FFTs at a regular interval. The peaks from the chromosome are computed for every chromosome, simply by reading off them off the SinRoots and their interpolated points. The fitness function that compares these peaks is very important, because it determines the topology of the search space. Each peak from the input sound is matched against the chromosome's peaks, and the closest add to the chromosome's score a value based on how close in frequency, and how long the peaks overlap. Subtracted from this value are the amplitudes of all peaks that do not match. This is a rough description of the implementation, but the important part can be grasped from it: adding the present peaks and subtracting the missing peaks effectively maps the search space to a thick vector with the middle area being silence, and one end being the target sound. This topology allows the genetic algorithm to 'retreat' back to silence and towards the goal at the same time, if it generates sound that are dissimilar to the target.

4. Synthesis Results

The genetic algorithm was used to attempt to resynthesize a bowed bass note lasting about three seconds. A population size of 25 chromosomes was used, and 17,500 generations were computed. The running time for this trial was slightly less than five hours.

At the end of the trial, the resultant waveform had established the fundamental frequency, its harmonics, and points of interpolation. The synthesized sound is close, but not the same as the original. The following figure compares the two waveforms at one point in time during the attack, where some of these differences can be seen.

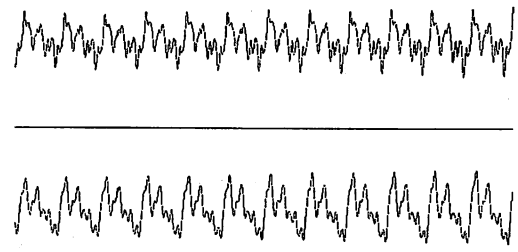

Fig. 9: Original waveform (above,) and Synthesized waveform, (below.)

The spectrum of the synthesized sound matches up fairly well, as can be seen in figure 10. The slight inharmonic deviations from harmonic structure in the real bass sound were also captured. However, the amplitudes of the partials are not precise.

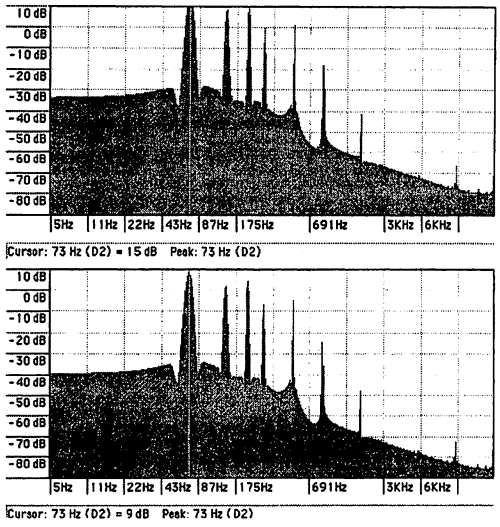

Fig. 10: Original spcectrum (above,) and synthesized spectrum, (below,) from a 8192 sample frame FFT during the attack of the sounds.

The synthesis flavor sounds similar in timbre to other sinusoid models, because the genetic algorithm uses one to conduct its search. The sounds created along the search path show off the model's interpolation and provide an interestingly 'clean' decomposition of the synthesis towards a single sinusoid with the amplitude envelope of the input bass sound. Experiments using an already computed chromosome solutions for one sound as its starting point to morph towards a different sound using the same algorithm were also conducted. The results show that the algorithm does get from one sound to the other, but wanders around the search space instead of taking a perceptually homogeneous and direct path to get to the goal.

5. Conclusions

We have shown the genetic algorithm can be used for analysis/synthesis, and provided a framework and some guidelines for creating a different genetic algorithm analysis synthesis implementation. The model implemented indirectly uses a sinusoidal model. This type of model is very popular amongst analysis/synthesis tools for its reliability, which is why it was chosen as a safe choice to demonstrate the genetic algorithm.

The major achievement of this experiment is that its meaningful model allows good variant sounds to be constructed as a side effect of the genetic algorithm's search. The model created by the genetic algorithm finds relationships behind peaks, and is capable of reaching an area around the target sound.

Future work needs to explore deviations from the sinusoidal model that might better take advantage of the genetic algorithm. Hopefully, this work will open up new paths for future implementations.

6. References

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

[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] Pampin, J. "ATS: a Lisp environment for Spectral Modeling," in Proc. of the Int. Computer Music Conference, Beijing, 1999.

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

[5] Magnus, C., "Evolving Waveforms with Genetic Algorithms"<http://cmagnus.com/cmagnus/ga_overview.shtml>, 2003.

[6] Johnson, C. G., "Exploring the sound-space of synthesis algorithms using interactive genetic algorithms." Wiggins, editor, Proceedings of the AISB Workshop on Artificial Intelligence and Musical Creativity, Edinburgh, 1999.

[7] Bentley, P. J. & Wakefield, J. P., "Hierarchical Crossover in Genetic Algorithms." In Proceedings of the 1st On-line Workshop on Soft Computing (WSC1) ,1996.

[8] Hart, P. E., Nilsson, N. J.; Raphael, B., "Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths"". SIGART Newsletter 37: pp. 28–29, 1972.

[9] Smith, JO, and X. Serra, "PARSHL: A Program for the Analysis/Synthesis of Inharmonic Sounds Based on a Sinusoidal Representation" (ICMC-87).