Using Ideas from Natural Selection to Evolve Synthesized Sounds

Kaare Wehn
Department of Informatics, University of Oslo
kaarew@ifi.uio.no http://www.ifi.uio.no/~kaarew/

Abstract

This paper describes a system for the automatic creation of digital synthesizer circuits that can generate sounds similar to a sampled (target) sound. The circuits will consist of very basic signal functions and generators that are arbitrarily interconnected. The system uses a "genetic algorithm" (GA) to evolve successively better circuits. First it creates populations of such synthesizers, generates the output and a fitness value of each individual circuit. The ones that are best at imitating the target sound will be kept. They are used for "breeding" to form a new generation where, hopefully, at least some individuals perform better than their parents did. The end result will be a circuit that can create a sound that resembles the target sample. Because it's a synthesizer we can manipulate the different parameters when generating the sound. We can also get a very compact representation of the sound that can be useful when distributing music over a limited bandwidth communications channel (e.g. Internet). As we shall see, it also gives the user a very powerful tool for creating totally new sounds.

1 Introduction

By combining simple signal elements it is possible to create some very interesting sounds. Indeed, this is what analog and most digital synthesizers utilize. The number of ways in which these elements can be hooked together is limitless. Most of these configurations (graphs or circuits) are quite useless, but among all the possible combinations we certainly should expect to find a few interesting ones too. Those utilized by synthesizers today are very basic and relatively easy to analyze. Even the configurations used in physical modeling are a subset of the complex signal graphs that this paper describes. In this context complexity means many connections between elements and many levels of feedback. This makes them hard to analyze and virtually impossible to design. They rather have to be discovered. Therefore we need an algorithm to automatically search this huge space of possible configurations.

2 Genetic Algorithms

In computing there are many tasks that require a vast amount of computing power if we use brute force methods to reach an answer. That is, search through all possible combinations and stop when we arrive at a satisfactory solution.

Many of these tasks are so computationally intensive that we can't hope for an answer in our lifetime. Therefore we must try to find alternative strategies. If we know enough about the structure of the solution to a problem, we can often find search methods that greatly limits the search space. However, when we don't have such clues we have to turn to other strategies. A Genetic Algorithm (GA) may be well suited for this kind of search. GA's are inspired by the theory originally proposed by Darwin and later enhanced by others. One significant discovery was the DNA molecule (the genotype), which is the basis for all life on earth. It's a long chain of amino acids that contains the information on how a body (the phenotype) should grow, although in a very indirect way. In nature all changes in a species are result of random mutations and the combining of DNA's from two different parents. If a new DNA produces a successful body, the changes can be passed on to new generations [3]. These are the ideas that we take advantage of in the Artificial Life System (ALS) of this article.

One difference should be mentioned though. Today this ALS has no genotypes, that is, all the processes of breeding and mutations are done directly to the phenotypes (the graphs) [1, 2]. This design choice where made in an early stage of the development and is likely to be changed soon. The subject will be discussed in "future work".

3 The Synthesizer Signal Elements

The basic elements (nodes) of the synthesizers apply simple functions to their input signals.(See *Figure 1*) All elements have one output and typically two inputs. Table 1 shows the functions that have been implemented in the system today.

Many other, more sophisticated, signal elements could be imagined and shall be explored in future versions of the system.

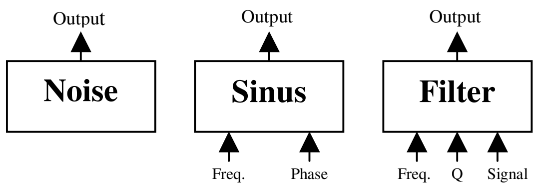

Figure 1. Symbols for signal elements

Function	Inputs1	Input2	Input 3
Noise	-	-	-
Sinus	Frequency	Phase	-
Triangle	Frequency	Phase	-
Square	Frequency	Phase	-
Ramp	Frequency	Phase	-
Adding	Signal 1	Signal 2	-
Multiply	Signal 1	Signal 2	-
Filter, bandpass	Frequency	Q value	Signal

Table 1. The signal elements

4 Connections

Most nodes have input lines. These can each have one connection to the output of any node in the configuration (including it's own output). Every input line is weighted by a value, usually between –1 and 1. If the input line is not connected to an output, the weight itself is used as a constant input signal. A collection of nodes and connections is referred to as a graph.

5 Fitness value

We now have the building blocks for our synthesizers but before we can search through graph space for new sounds, we need a direction in which to move. So we have to introduce a goal. The target sound will serve this purpose. All new configurations we encounter will have to be measured against this sound, or rather its amplitude spectrum. So, before evolution begins, a spectral analysis is done on the scaled target sound. The amplitude spectrum of this FFT is referred to as *T* and typically has 1024 frequency bands. This normally would constrain the input wave to a length of 2048 samples, but you are able to specify any number of FFT windows to sequentially cover the whole target wave.

When a graph is to be evaluated against the target, the output signal is first tested to check that it actually contains any oscillations. If not, the fitness value is set to a maximum.

If it does contain a valid sound, a frequency analysis is preformed on this scaled signal and the result is denoted *G*. We then use the following formula to calculate the fitness value:

$$e = \sqrt{\sum_{1 \le w < 2} \Big(\sum_{0 \le b < 1024} (T_b^w - G_b^w)^2 \Big)}$$

In the formula w denotes the FFT window number and b is the frequency band. (The calculation of the square root is skipped in the actual system.) The fitness value is the Euclidean distance between the amplitude spectrums of target wave and graph output. Therefore lower fitness values are better and a perfect match gives a zero fitness value.[1]

6 The evolutionary process

Each generation typically consist of 100 graphs. We use a random creation method to generate the first population. Then the evolutionary process can start. The resulting output waveform from each graph is calculated and evaluated against the target sound.

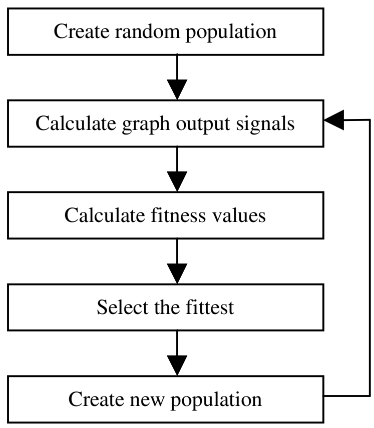

Figure 2. The evolutionary process

The graphs with lowest (best) fitness values are selected for survival and breeding. In this system 20% of the population is singled out in this way and the rest are destroyed. Using different kinds of breeding a new generation is created. The relative success of a graph will determine how many children to be created from it. Now, the process can start over. *(See Figure 2.)*

[1] A perfect match will sound similar, but may have very different phase characteristics.

7 Breeding techniques

Since we have no genotype to work on, breeding is done directly by copying from one or two parent graphs to the new child graph. Three types of breeding techniques are used to form new graphs; cloning, grafting and crossover. After creation graphs are subject to random mutations and a garbage collection that removes inactive nodes and connections.

When cloning the system just makes a copy of one parent. Since copying won't give us any changes from parent to child, mutation rate is set much higher than for the two other techniques.

Grafting takes two parents. They are both copied to the child and a random new connection between the subgraphs is added. Then the graph is subject to mutations and garbage collection. *(See Figure 3.)*

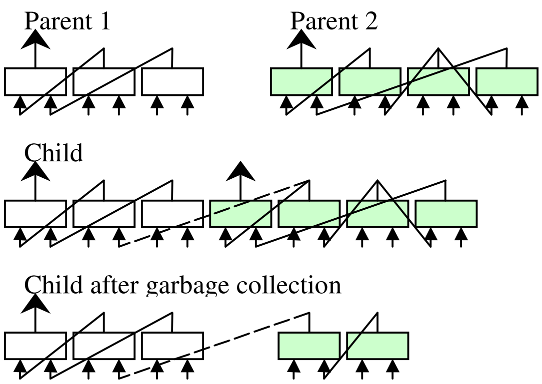

Figure 3. Grafting (without mutation)

Crossover also takes two parents and moves trough the nodes of one parent and copies them one by one. At a random point the system will switch to the other parent and use this as the source. It can also switch back again. Such a switch between the parents will on average happen one time per new graph. As always the child is subject to mutation and garbage collection. *(See Figure 4.)*

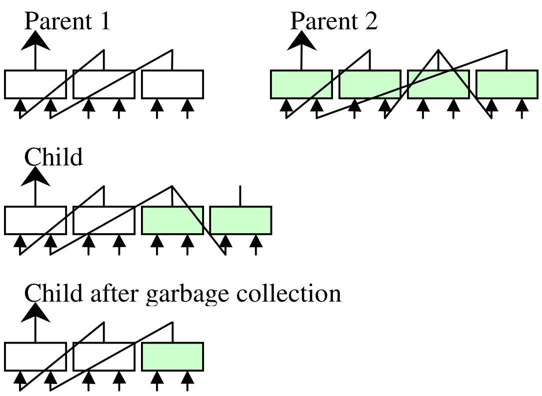

Figure 4. Crossover (without mutation)

8 Mutation

During the mutation stage, one of the nodes in a graph is always replaced. The type of the new element is randomly chosen and it inherits all connections and weights from its predecessor if possible. If not, weights are set to random values and no connections are made. Even without any connections the node can survive the garbage collection if it is "lucky" during the last phase of the mutation stage.

As mentioned earlier the mutation rate may vary. With a rate of 1.0 the following events will usually occur once before the new graph is passed on to garbage collection:

1. A node is added. With random type and weights.

2. A connection is moved or a weight is changed.

It should be pointed out that the mutation strategy has emerged in a very empiric way. By watching the ALS and making observations about where the evolution gets trapped, these steps have been modified several times.

9 What can be expected?

It is important to remember that the graphs that this ALS evolves usually can't give us an exact match of the target sound. There are two reasons for this:

1. The fitness value uses the amplitude spectrum.

2. Real life sounds are too complex.

In theory these graphs could imitate any sound. This is easy to realize if we look to the Fourier transformation. E.g. by combining 256 sinus nodes an exact match for any sound wave with 512 samples could be generated. But this kind of solution is not desirable. The system used to test this ALS has a maximum number of nodes set to 16. This means that the evolution is forced to find more innovative solutions than different flavors of traditional transformations.

10 Results

GA's have been tested extensively during the past decades and have found many applications. Even creationists should be able to recognize the success that these methods have had in the digital world.

Therefore, it should not come as a surprise that genetic algorithm's works in this application too. *Figure 5* shows a typical result.

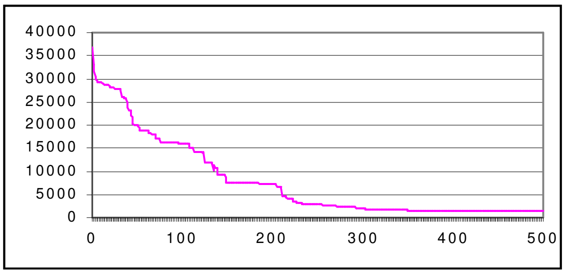

Figure 5. Fitness value vs. generations

As expected the fitness value decreases quite fast in the beginning. Then the ALS has a tendency to get trapped in a local optimal point. To get out of this trap evolution needs a significant change in a graph to escape. This will seldom happen so we can observe long periods with virtually no improvement. During these periods the populations seem to be too genetically homogenous. The result is that evolutionary progress tends to work in steps as seen in the figure. This is a problem that every implementation of GA has to fight.

The system is able to find graphs with output that is similar to the target sound, but there is certainly room for improvement. During testing the output from a known graph has been used. *(See Figure 6.)* The system is able to arrive at this configuration, but only after a large number of generations.

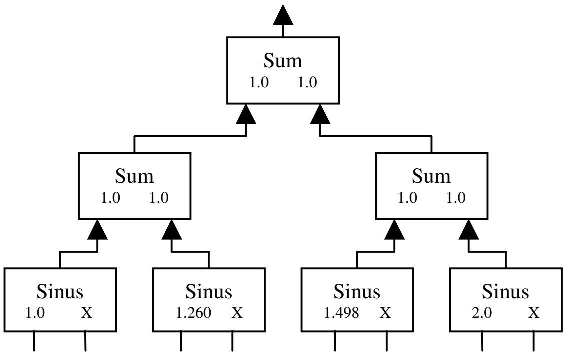

Figure 6. Test target graph. Sinus major chord

On the other hand, to observe the system while it is running can be quite entertaining. New sounds are playing all the time in a tremendous tempo. Most of them are similar to classical synthesizer sounds, but once in a while you may encounter very rich and strange sounds. These and their corresponding graphs can be saved for later use. The operator can also bypass the fitness value selection and choose which graphs that should survive. But this is impractical. It takes to long and it is very difficult to know which graphs that have most potential.

11 Future work

To make the system less prone to be trapped in local maximums many techniques can be implemented:

1. A genotype [4]. Every graph will then be the result of the decoding of a string of bits. This is more in line with traditional GA and will greatly simply and cleanup the process of breeding and mutation. But it requires a thorough consideration when selecting the coding scheme. One solution is to use a Lindenmayer system (L-system) of production rules [5, 6]. This can generate graphs with a fractal structure similar to the one shown in *Figure 6*. Another method is to have a fixed structure of elements and connections and use the genotype string of bits to encode only the input weights.

2. Geographical isolation between several populations with occasional exchange of genetic material. This allows different characteristics to evolve in each population. When evolution converges, fresh gene material from another population could trigger new progress.

3. A geographical location for each member in a population. Mating will only occur between neighbors. This will hopefully preserve genetic diversity.

12 Conclusion

The results presented in this paper are promising. The implementation of this ALS shows that the concept works. It is capable of finding graphs with an output similar to the target sound. These graphs would usually be impossible to design by hand. The system comes up with very complex and innovative solutions. On its way towards a solution it also creates many graphs that fail miserably in mimicking the target sound. But these sounds can have a value of their own. Musicians should be able to utilize the system to discover totally new sounds not necessarily similar to the target. For them the target sound will function as a loose hint given to the system about what kind of sound they are looking for. Then they can sit back and listen while the system gives them different alternatives.

Given the preliminary state of this work there should be room for significant improvements. If this proves to be true other application emerges.

It could be used as a very efficient compression technique to save bandwidth when transmitting sound waves.

13 Acknowledgements

This work is a part of my master thesis. Thanks to my supervisor Øyvind Hammer for his valuable ideas, knowledge and encouragement.

14 References

1. Sims K., "Evolving Virtual Creatures", *Computer Graphics*, Annual Conference Series, July 1994, pp.15-22

2. Sims K., "Evolving 3D Morphology and Behavior by Competition", *Artificial Life IV Proceedings*, ed. by R. Brooks & P. Maes, MIT Press, 1994, pp.28-39

3. Dawkins R., *The Blind Watchmaker*, Harlow Longman, 1986.

4. Whitley D., *A Genetic Algorithm Tutorial*, Computer Science Department, Colorado State University [whitley@cs.colostate.edu]

5. Channon A., *The Evolutionary Emergence route to Artificial Intelligence*, MSc in Knowledge-Based System 1995/96, School of Cognitive and Computing Sciences, University of Sussex.

6. Channon A., *The Artificial Evolution of Real Intelligence by Natural Selection*, Image, Speech & Intelligent Systems Research Group, University of Southampton. http://www.soton.ac.uk/~adc96r