Automatic Parameter Settings For A Commercial Sound Synthesizer

Laurent Droguet

Internship Report – Master's degree

March 26th,2012 – August 26th, 2012
Mama's Lab, Vancouver, under the supervision of Philippe Pasquier

Abstract

How to reproduce a given sound using a synthesizer? This is the aim of this project and internship. Several synthesis techniques such as additive, subtractive of FM synthesis have been successfully used in replicating instrument-like sounds. Nonetheless, these techniques require some parameter setting, task that can be hard and time consuming because manual and not so intuitive. People who worked on that kind of problem have used a number of optimization techniques including genetic algorithm to automize the search of those parameters. Naturally, each synthesis comes from a different theoretical environment, each parameter search space has a different structure and require a specific technique.

On this project, we worked with “Teenage Engineering”, a company from Sweden, which provided us one of their commercial synthesizers. This one possesses among others seven kind of audio engines, several effects and lfo and will be used to automatically generate believable instrument-like sound or any kind of sounds, using a genetic algorithm system.

1 Introduction

A classic synthesizer is an electronic instrument capable of producing a wide range of sounds. Synthesizers may either imitate other instruments or generate new timbres. They can be controlled via a variety of different input devices (including keyboards, Midi instrument controllers). Synthesizers use a number of different technologies or programmed algorithms to generate signal, each with their own personality. We can name subtractive synthesis, additive synthesis, frequency modulation synthesis, physical modeling synthesis and sample-based synthesis.
There are different approaches of using a synthesizer. First the performance aspect: by different ways (keyboard, pad, gestural performance…) the player has the choice of the note (frequency), note length and volume. Then, the perceptual and timbre aspect are controlled by a set of input parameters (audio engine, effects, low-frequency-oscillator...).

It can be hard for the user to find the sound he wants for several reasons: parameters are not usually instinctive for non technical-users, there may be a very large number of them, sometimes a small parameter change can cause a large change in the sound or user doesn't have time.

For those reasons, the synthesizer use can appear unintuitive, especially for the beginners, and even for the experimented ones who prefer having immediate sound feedback rather than engross himself in a complicate analytical parameter-setting mode. That's why we could find the use of Genetic Algorithm more flexible and accessible for the composition than usually because it allows us to automate this parameter-setting step and gain some time and effort.

More specifically, the Genetic Algorithm (GA) will be used to bread parameter settings that minimize a distance function to the target sound: the fitness function. On the one hand we will take a fitness function based on an Euclidian distance between the spectrogram frames of the target and the input sound and then we will try to enhance our model with others sound descriptors like the Mel-Frequency-Cepstra-Coefficients (Mfcc) or Chroma. We will also try others distance than the Euclidian distance.

We first implemented a static fitness function, and then we tried to work with dynamic fitness function coefficients: indeed our main purposes using several distances is to be more accurate with creativity problem [Dimitrios Tzimeas, 2009]. This article suggests that the values of the coefficients can be assigned dynamically in such a way that the system will be able to control the influence of the less-selective properties and their corresponding fitness function components.

We'll see in the following sections how did we decide to use these particular fitness functions, based on previous works, a description of our synthesizer, the way we implemented our system and finally our results and comments.

2 Related Works

Horner et al. in their seminal journal article from 1993 were one of the first to work on the problem of parametric optimization for automatic tone matching sound. They used a genetic algorithm and FM synthesis architecture [HOR93]. They used GA to optimize the modulation indices and carrier and modulator frequencies for various numbers of carriers. The relative spectral error between the original and matched spectra served as fitness function in guiding the GAs search for the best FM parameters. Most matched instruments required three to five carriers for a good match.

In 2001, Garcia [GAR02] used a tree based genetic programming model to grow topologies capable of rendering sounds. He worked with an analytical distance metric that uses the complex spectrum of the target and test signals.

In 2005, Mitchell et al. [MIT05] extended Horner's worked with frequency modulation sound matching with clustering evolutionary strategy. Indeed, the principle of FCES is to partition the search population into sub-population that locally recombine and progress. With an adequate number of clusters and population size, all of the locally optimal peaks can be identified and so, a global optimum is constantly found.

Aucouturier [AUC05] described an effective way to model the textures found in a given music signal and their applications for music similarity. In a first example, he compares the timbre models of different songs to compute their timbral similarity. He used an Expectation-Maximization (EM) algorithm with Gaussian-mixture-model (Gmm) learning model.

In 2006 Lai et al. [LAI06] revisited Horner's FM tone matching once more. The originality of the study is that they combined a timbre feature, the spectral centroid, and the spectral norm as a fitness value in place of the power spectrum.

Chinen and Osaka's Genesynth [Chi07] from 2007 worked with a model flexible enough to represent noisy, inharmonic and harmonic audio. They evolved Noise Band sound model which contains three parameters: amplitude, frequency, and a bandwidth ratio. Their fitness function computes a score by comparing a windowed spectrum estimate of the chromosome with cached and compressed bins from the STFT of the input.

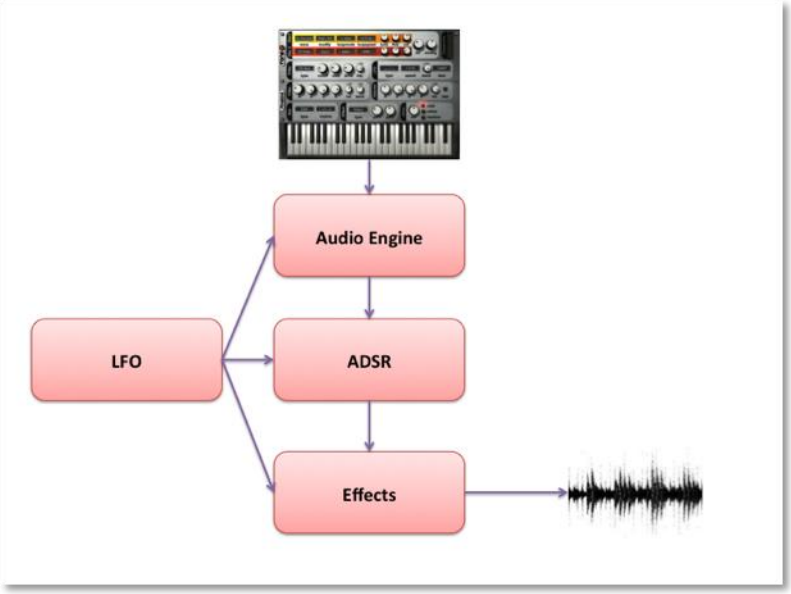
King [YEE11] worked on comparisons of parametric optimization techniques for musical instrument tone matching, using as error metric an Euclidean distance in MFCC feature space. He found the best results with a genetic algorithm but also tested the strong performance of hill climber and data driven approach.

Recently, Macret and Pasquier [MP12] compared two Fm methods to replicate an instrument sound. The first synthesis is a "classic Fm" and the second one, "Modified Fm synthesis" (a recently discovered distortion) is derived from the first one. Our evaluation shows that Mod FM synthesis coupledwith GA seem to overcome the

couple Classic FM synthesis - GA studied in a previous work. This comparative study also shows that GA gives generality and efficiency to parameter matching.

3 Synthesizer Specifications

In this section, we will explain the mechanism of the commercial synthesizer from Teenage Engineering (which we made a non disclosure agreement with), its characteristics, strength and weakness. In order to have a general vision of how the sound is created, let's have a look at the sound path: this is the way the sound moves from the moment the musician hits a key on the musical keyboard or press play tape on tape, until it reaches the speaker or line out.



The initial sound is made from one of the seven ***audio engines***:

- **Pulse**: square wave engine
- **Fm**: frequency modulation synthesis
- **Phase**: phase distortion type engine
- **Dr Wave**: rax 8-bit style engine
- **Cluster**: up to 6 oscillators chained in a cluster
- **Digital**: pure digital raw engine
- **String**: physical modeling of a string instrument

Then we add an ***envelope***:

The envelope controls the amplification of a sound and is triggered when a note is played. The attack, decay, sustain and release are controlled this way. This is called an ADSR envelope.

We can put some ***effects***:

- **Delay**: solid state delay
- **Grid**: three dimensional feedback plate
- **Phone**: hacked telephone system
- **Punch**: hard hitting low pass filter
- **Spring**: mathematic reverb

and some ***Lfo***:

- **Element**: let the player use external elements like the built in microphone, line-in, G-force sensor of Fm Radio to modulate a sound. Select the element amount, destination (engine, envelope or effect) and the destination parameter.

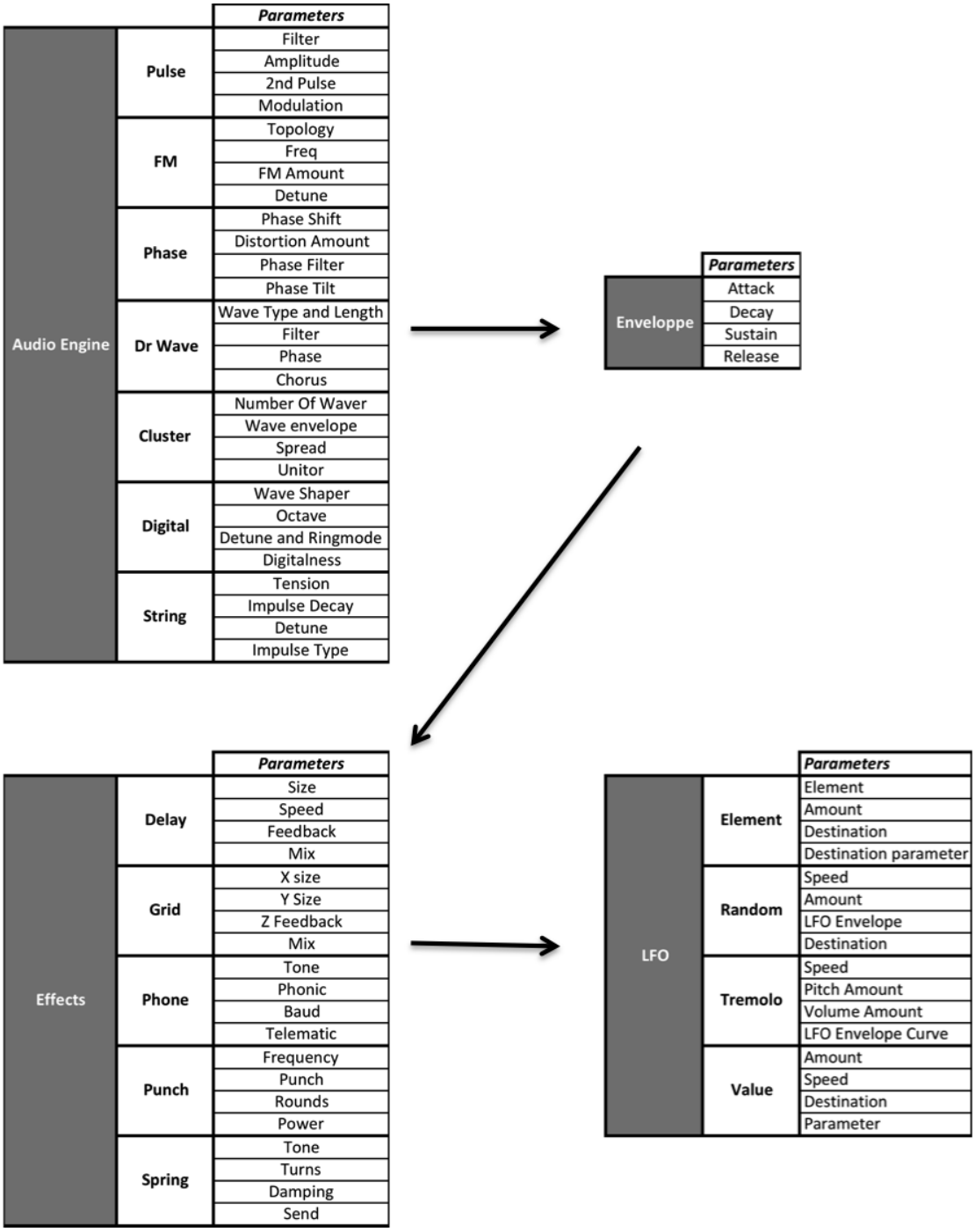
- **Random**: randomize all parameters in a module. Set the speed, amount, Lfo envelope and destination (engine, envelope or effect).

- **Tremolo**: let the user create different types of vibrato effects to your sound by modulating the pitch and volume. Set the speed, pitch amount, volume amount and Lfo envelope curve.

- **Value**: Classic Lfo type which allow the player to change one parameter only. Set amount, speed, destination and parameter.

Each of these modules has 4 parameters the user can tweak. Then we can choose the key/note, the octave and the duration of the key press. Altogether the synthesizer allows us to make a sound with 22 parameters.

The company, which made the instrument, gave us access to a part of the synthesizer, so we could use it as a C++ library to make a sound. We wrapped this library into Matlab which allowed us to built up any sounds from the Teenage Engineering synthesizer.



All these parameters have bounds (appendix at the end of the document), so if a wrong number is entered in the function which is outside the bounds of a parameter, the output sound will be empty. We took it into account and made sure that all numbers are in the right boundaries.

4 Genetic Algorithm

For the parameter optimization of the synthesizer we used a Genetic Algorithm (GA). GA belongs to the evolutionary algorithms which generates solutions to optimization problems using methods inspired by natural evolution such as selection, crossover and mutation.

The evolution starts with a first generation of randomly generated candidate, which will evolve toward better solutions. At each generation, the fitness of every individual in the population is evaluated, knowing that the fitness determines how close a candidate is from a target sound.

Then multiple individuals are stochastically selected based on their fitness from the current population, and modified with genetic operators (cross-over, mutation, reproduction) to form a new population.

The new population is finally used in the next iteration of the algorithm. Usually, the algorithm finish either when a maximum number of iteration has been reached or a satisfactory fitness level has been reached or if the weighted change in the fitness is less than 10^{-10} over 50 generations.

The space of all feasible solutions is called search space. Each feasible solution can be evaluated by its fitness for the problem. The looking for a solution is then equal to a looking for a minimum in the search space.

We explain in the following paragraphs all the mechanisms which happen during the algorithm.

4.1 Candidates

We have 22 parameters to optimize in our system. A parameter is called a "gene" and the 22 parameters together form an "individual" (also called a "chromosome"). All these individuals are in fact set of parameters (which allow us to make a sound), which are candidates to be as close as possible to the target sound. At the beginning of the process, all the candidates have the same length as the target sound and are randomly generated: a uniform distribution is used to calculate each parameter (respecting the boundaries) of an individual.

4.2 Evaluation

Each chromosome is evaluated depending on its distance with the target sound. Remember that the distance is calculated with the fitness function (next section). This evaluation is performed on the whole population of chromosomes.

4.3 Selection

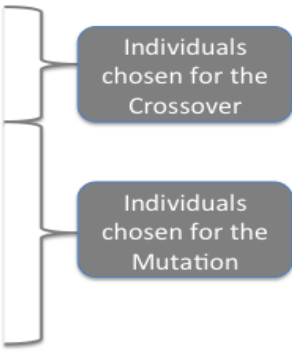
Once every chromosome has been evaluated, the new generation is generated. The N^{elite} best chromosomes are kept and the rest of the population is generated with crossover or mutation. These chromosomes are selected using the binary tournament selection. In this type of selection, two chromosomes are selected randomly in the

population, their fitnesses are compared and the best is selected. The elite individuals are also part of the binary tournament.

Let's make an example of what happens here:
Let's say that we decide to get 8 mutations and 2 crossovers after each generation. We'll need to make 4 tournaments for the crossover and 8 tournaments for the mutation.

Individuals	Fitness
1	0.5
2	3.1
3	2.5
4	1.3
5	1.5
6	0.9
7	0.3
8	3.6
9	0.6
10	1.9

Binary Tournament		
1st ind	2nd ind	Result
6	5	5
4	8	8
3	7	3
2	3	2
6	2	2
7	1	1
2	9	2
1	9	9
1	8	8
8	2	8
5	9	5
3	5	3



We see in this step that the best individuals have more chance to take part to the next generation than the others. We can notice that some chromosome simply disapear because of their bad fitness (4, 6 and 7). Be careful here because the diversity can be questioning: indeed some audio engine can be over-represented at the beginning because they have some good parameters but at the end they will not fit with the target sound. We can avoid it with a big initial population, or a big mutation of audio engine at the beginning, which decrease after.

4.4 Crossover

The crossover step consists in mixing two chromosome parents to make a new chromosome: the child. Our method consists in selecting two random integers m and n between 1 and number of parameters (22). The function selects:

- Vector entries numbered less than or equal to m from the first parent
- Vector entries numbered from m+1 to n, inclusive, from the second parent.
- Vector entries numbered greater than n from the first parent

The algorithm then concatenates these genes to form a single gene. For example, if p1 and p2 are the parents

p1 = [a b c d e f g h]

p2 = [1 2 3 4 5 6 7 8]

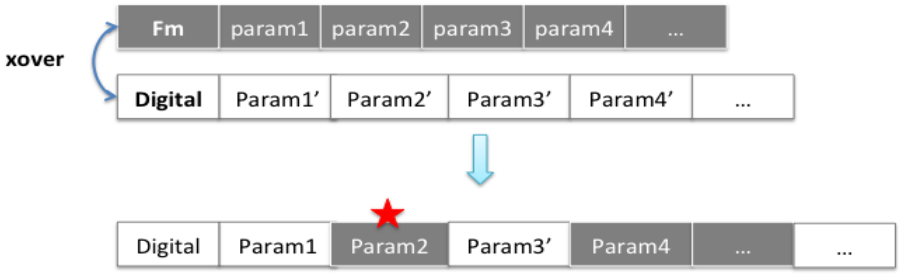
and the crossover points are 3 and 6, the function returns the following child:

child1 = [a b c 4 5 6 g h]

The resulting chromosomes are used in the new generation.

Each parameter range changes depending on the sound engine, or the Fx, or the Lfo chosen. A first difficulty is at this step, indeed we can't use traditional crossovers we want because of the different parameter bounds. If we consider for example a crossover between a FM and a Digital sound engine. Let's have a look at the parameter ranges of the two audio engines:

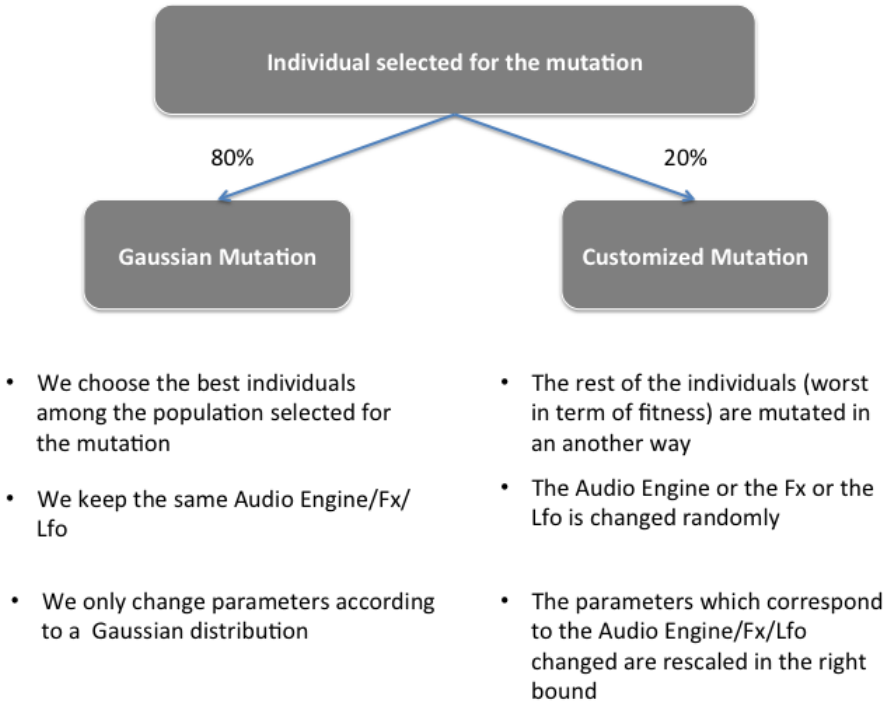
	Parameters	Min	Max
Fm	Param 1	0	32767
Digital	Param 1'	0	32767
Fm	Param 2	0	32767
Digital	Param 2'	2048	26624
Fm	Param 3	1024	17408
Digital	Param 3	-32768	32767
Fm	Param 4'	0	32767
Digital	Param 4	0	32767



We can see that the ranges are not the same and a crossover at the level of the second parameter could cause a sound impossible to synthesize. We solve this problem by picking up a new random parameter between the right ranges. Now if we look back at the example, if the second parameter from the Fm sound is outside the bound of the Digital sound, we generate a new parameter following a uniform distribution between the Digital bound

4.5 Mutation

The mutation is described below:



4.5.1 Gaussian Mutation

A random number from a Gaussian distribution is added to each parameter of the chromosome. The Gaussian has a mean of 0 and the standard deviation is determined by the parameters "Scale" and Shrink" which are set at the beginning of the algorithm.

"Scale" controls what fraction of the gene's range is searched. A value of 0 will result in no change, a "Scale" of 1 will result in a distribution whose standard deviation is equal to the range of this gene. Intermediate values will produce ranges in between these extremes. In other words the "Scale" parameter determines the standard deviation at the first generation.

The initial standard deviation of the parameter i of the individual chosen for the mutation is given by:

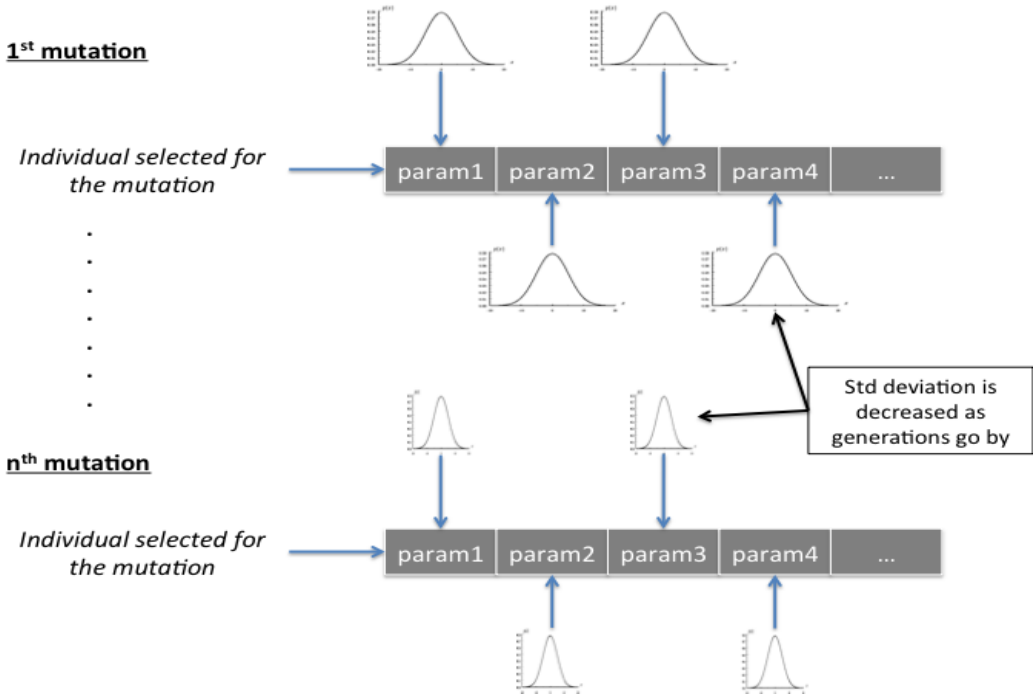
$$\text{Scale} \times (\text{bound\_end} - \text{bound\_start})$$

"Shrink" controls how fast the "Scale" is reduced as generations go by. A Shrink value of 0 will result in no shrinkage, yielding a constant search size. A value of 1 will result in "Scale" shrinking linearly to 0 as GA progresses to the number of generations specified by the options structure. Intermediate values of "Shrink" will produce shrinkage between these extremes.

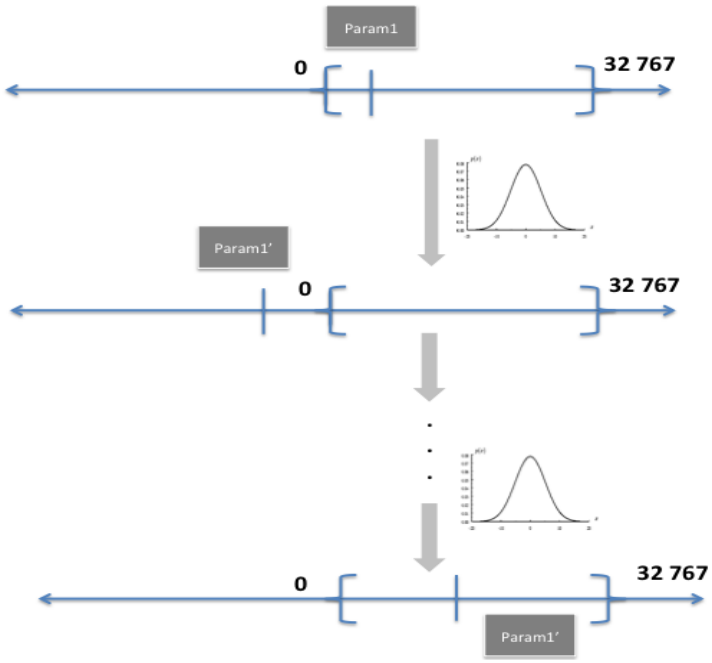
The standard deviation at the k[th] generation, $\sigma_{i,k}$, is given by the recursive formula:

$$\sigma_{i,k} = \sigma_{i,k} - \left(1 - Shrink \frac{Current\ Generation}{Maximum\ number\ of\ Generation}\right)$$

This dynamic standard deviation makes possible a broad exploration of the parameter space at the beginning of the evolution and a fine tune of the parameters at the end of the evolution.

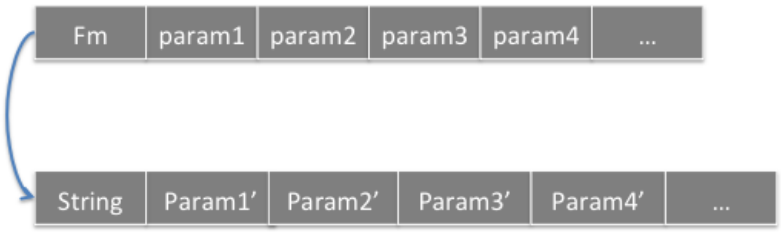


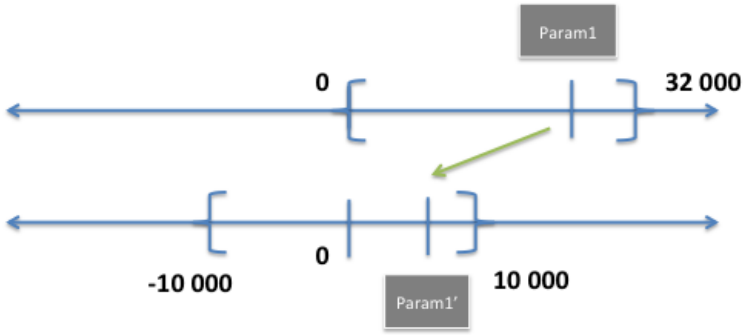
As the crossover step, there is the same difficulty of boundaries. If the new parameter generated by the Gaussian distribution become outside the bounds, we reiterate the process until the new parameter fit the right bounds.



4.5.2 Customized Mutation

The worst individuals in term of fitness are mutated this way because we want to radically change something in their "DNA". We randomly change either the audio engine or the Fx or the Lfo (each have an equal chance to be picked up). We keep the parameter that doesn't have to be changed and we rescale the others. For example, if an individual is mutated according to this rule and the audio engine is chosen to be switched, then its four parameters will be rescaled and the others (Fx, Lfo, key, octave,...) remain unchanged.





4.6 Stopping criteria

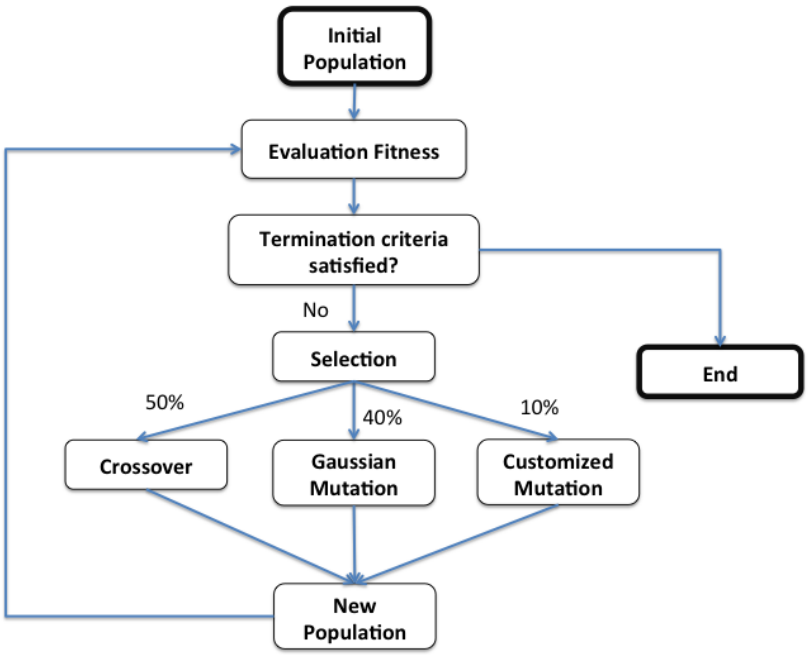
The optimization is stopping if:

- The system reaches the maximum number of generations (a constant chosen to ensure that the system stops)
- The weighted change (equation below) in the fitness is less than 10^{-10} over 50 generations

$$\delta_n = \sum_{i=1}^{n} \left(\frac{1}{2}\right)^{n-i} (f_{n+1-i} - f_{n-k})$$

4.7 Summary

There is a summary of how the genetic algorithm work:



5 Fitness Functions

The fitness function is crucial because it represents the idea of a distance function on the sound space. It got to be as close as possible as the human perception of a sound.
We have made some instrument comparisons using different fitness function, more or less classic. The results are several dissimilarity matrixes, one for each fitness function. For information, there are 12 instruments, some are similar and some not: we compared how much the fitness function matched with our perception of the instrument.
Different techniques are introduced to describe the timbral parameters, in a first time, we decided to choose the Euclidian distance, because it is quite widespread and appears quite intuitive. Secondly, we picked up 2 common ways to characterize a sound: short spectral analysis and Mfcc.

5.1 Short Spectral analysis:

The Short-Term Fourier Transform (STFT) is a common spectral decomposition method for an audio signal, which many spectral analysis techniques are based on. It first breaks up the input signal into small time steps, which is called the frames. Generally, each frame is multiplied by a window function, which is nonzero for only a short period of time, and overlapped one after another. A signal in the each frame is then transformed into the frequency domain by means of the Discrete Fourier Transform.
DFT of the signal is defined as:

$$X(k) = \sum_{n=0}^{N-1} s(n)e^{-iwnk}$$

with $w = 2\pi/N$ and $k = 0, 1, \ldots, N-1$

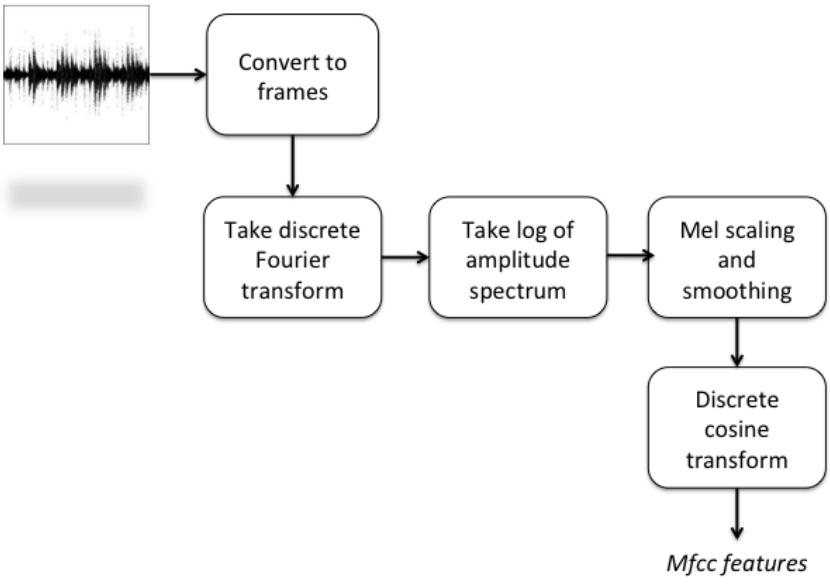
where X (k) is an amplitude of a sinusoidal component at the index number k, and N is the frame size (the number of sample points in the frame). STFT can thus provide a time-varying spectrum for the input signal. Yet, the downfall of STFT is that there is a mutual trade off between frequency resolution and time resolution. DFT with long framesize have better resolution on the frequency axis but poor time resolution. Shorter frame size gives better time resolution but poor frequency resolution.

Characteristics of the short spectral analysis:

Analysis window size	**10ms**
Overlapping	5 ms
Zero padding factor	4
Spectral resolution	20 Hz
Signal portion	2s

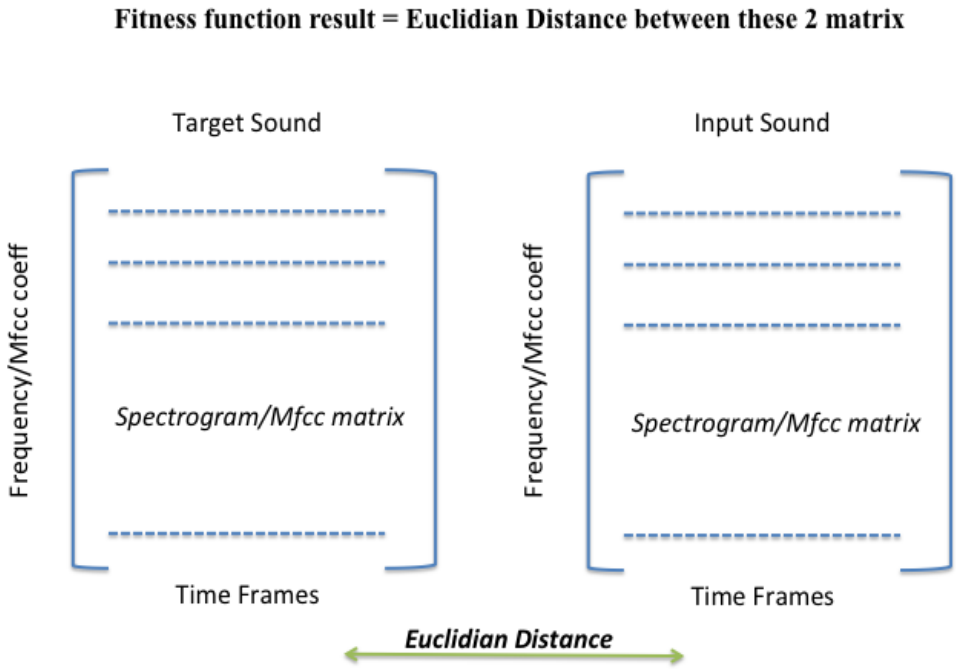
5.2 Mel Frequency Cepstrum:

MFCCs are a short-time spectral decomposition of an audio signal that conveys the general frequency characteristics important to human hearing. The low order MFCCs account for the slowly changing spectral envelope, while the higher order ones describe the fast variations of the spectrum. Therefore, to obtain a timbre measure that is independent of pitch, we only use the first few coefficients.



Most of the Mfcc applications have been for the speech recognition but it is also a good choice for the purpose of timbral description [CAS08]. Moreover, Logan [LOG00] provides a well-cited and convincing justification for the use of the MFCC in musical modeling, supporting the use of the warped frequency scale and the Discrete Cosine Transform. In summary, the MFCC has been widely used and validated as a numerical measure of musical instrument timbre.

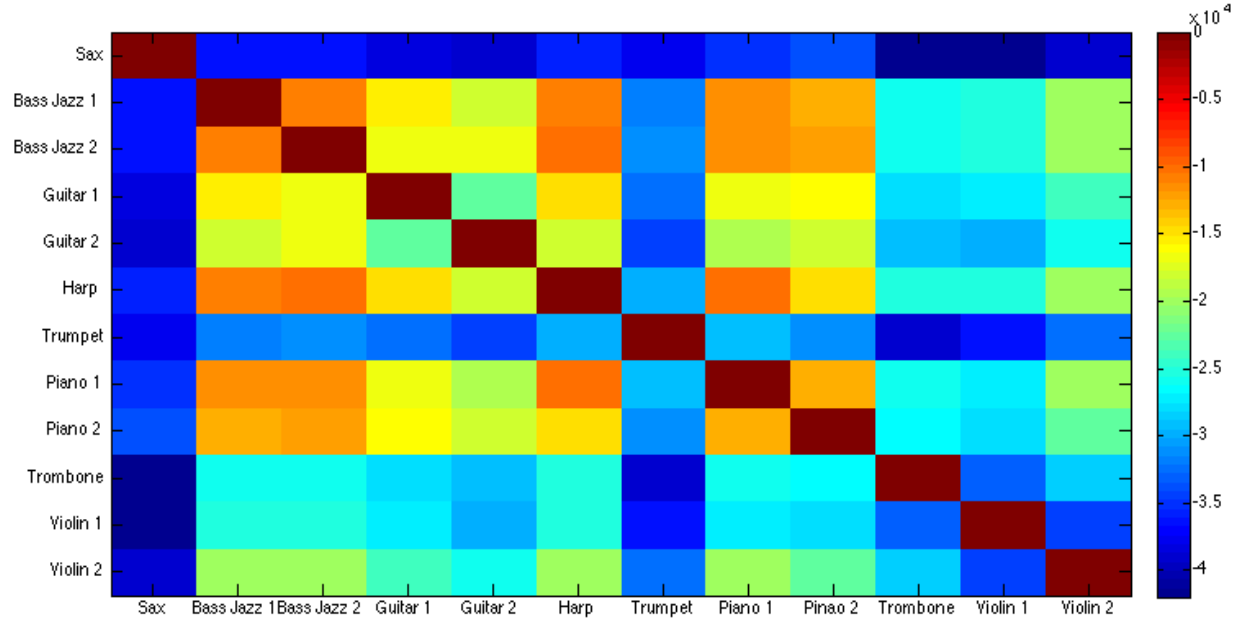
Now we introduced our 2 ways to characterize a sound, our fitness function result is the Euclidian distance between either the spectrogram matrix or the Mfcc coefficient matrix.



We conducted 3 experiments, one for each fitness function and there are the results, which can be also consulted at this URL:

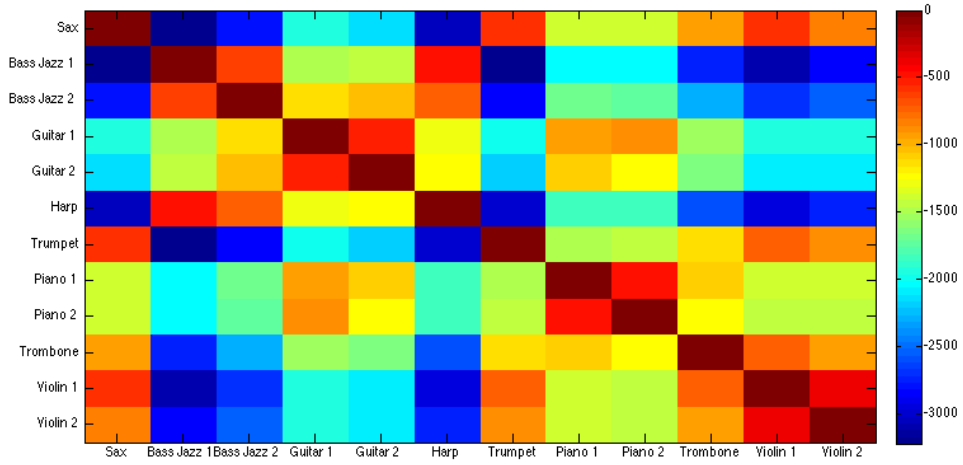
http://metacreation.net/ldroguet/index sound.html

First dissimilarity matrix:



This image shows the distance in term of spectrogram frame between the first second of the 12 instruments samples. The smaller the value, the greater the similarity.

Same thing with the Mfcc method:



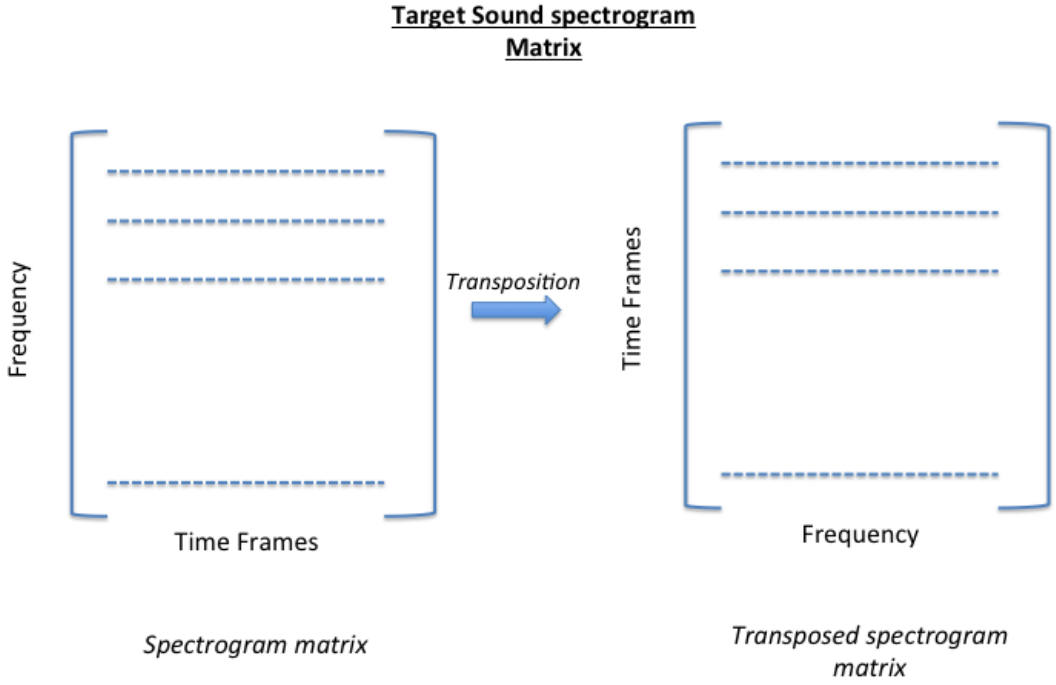
The two different methods show different results but seems making sense with our perception of instrument timbres. The closeness of certain perceptual similar sounds seems to be more pregnant with the Mfcc. Indeed, take a look at the two guitar songs: they are clearly similar with the Mfcc (nearly red) but not that much with the spectrogram (yellow). Generally, plucked and striked string instruments seem to stand out: guitar, bass, harp and piano are rather close, in a way more or less strong but this is a generalized trend.

Now, instead of the Euclidian distance, we are going to introduce another way to compare feature matrix (spectrogram/mfcc) of two sounds: “Clustering”. Very well known in Data Mining and Marketing, this method is often used to classify people

according to their purchase behavior and their characteristics (sex, age,...). We know that sounds possess also a lot of characteristics so we'll do the same. Basically the principle is to set up a k-means model on the target sound in order to obtain a certain number of barycenters. After that, we'll be up to apply this model to the input sound and compare difference of distribution in each group.

5.3 Clustering:

There is the explicit methodology (the example uses the spectrogram matrix but it is the same with the Mfcc): the first step consists in translating our spectrogram matrix. That way, we'll make each sound clustered according to its frequency at each time frame.



The second step begins with the "k-means clustering" which aims to partition the time frames observations into k clusters in which each observation belong to the cluster with the nearest mean/barycenter. Generally, we tend to look for population very different between each other.

More precision on the "k-means clustering":

This is one of the most famous unsupervised learning algorithms that solve the well-known clustering problem. The procedure tends to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster.

The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early groupage is done.

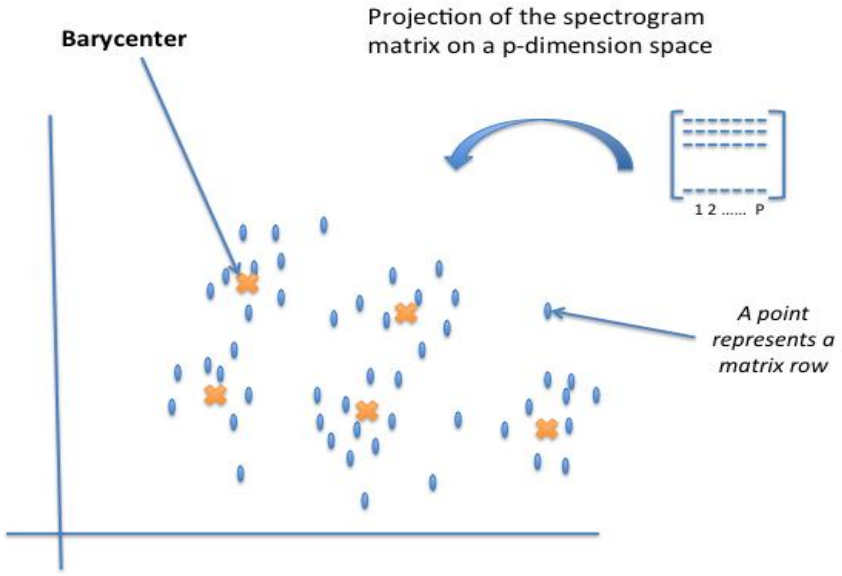
At this point we need to re-calculate k new centroids as barycenters of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop we may notice that the k centroids

change their location step by step until no more changes are done. In other words centroids do not move any more.

Finally, this algorithm aims at minimizing an objective function, in this case a squared error function. The objective function:

$$F = \sum_{i=1}^{k} \sum_{i=1}^{n} \left\| X_i^j - c_j \right\|^2$$

where $\left\| X_i^j - c_j \right\|^2$ is the distance of the n data points from their respective cluster centres.

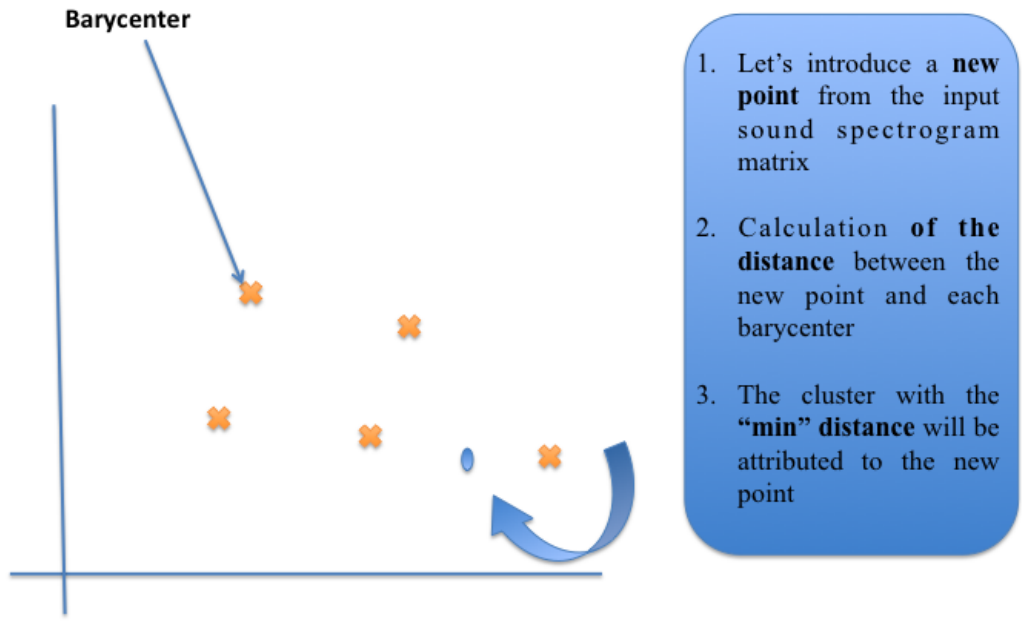


Target Sound model

We then obtain a cluster distribution, that is to say the percentage of time frames in each cluster.

Target Sound model distribution

The third step consists now in attributing a cluster to each time frame of the **input sound** (= sound which is compared to the target sound) according to the target sound model.



Target sound model applied to the input sound

When we obtain the final distribution of the input sound, we are up to compare the input and target distribution like in the graphic below:

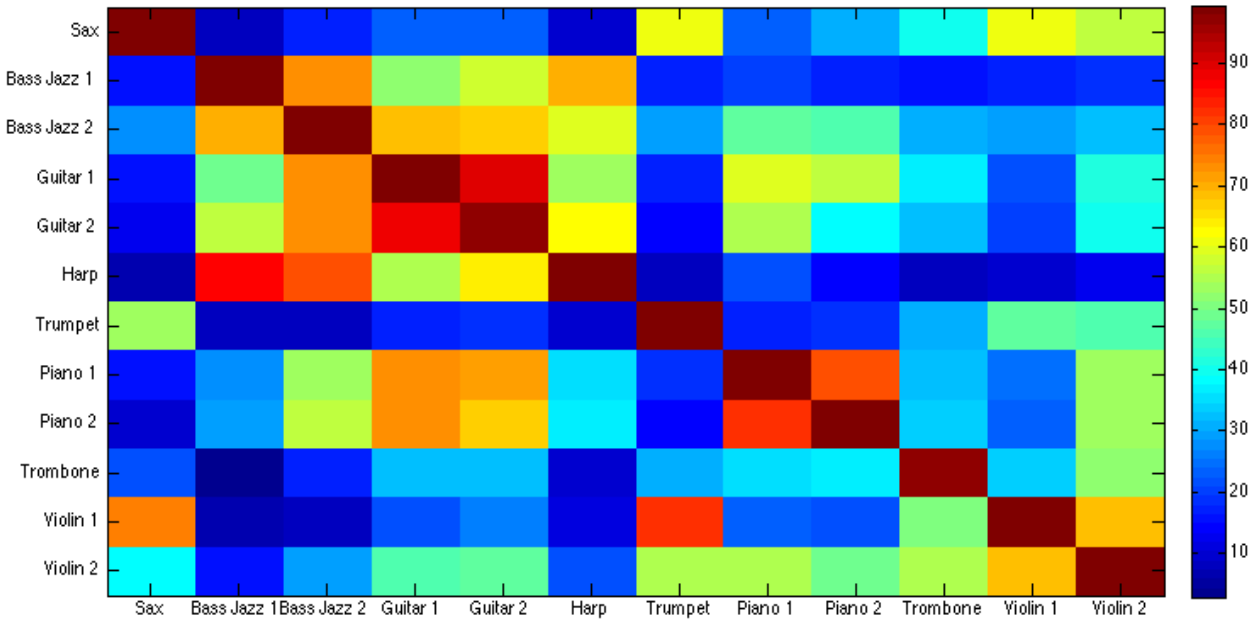
Input and target sound distribution

***The fitness distance** will be the difference of the target and input distribution in each Cluster*

In this example, the target and the input sound seem visibly quite close in view of the distributions. The fitness function will have a value ranging between 80 and 90 percent of similarity.

Now back to our instrument comparisons using this method. We implemented a K-means algorithm with 5 clusters. Five is a good number of cluster: enough information about our population and we avoid an over fitting. The resulting of fitness function is the mean of a hundred k-means based models (it means that we have repeated the experiment 100 times then we made the mean).

There are some results:



We can observe the same trend than the others: plucked and striked string instruments stand out. But generally, similarities between instruments are subtler and closer to our musical perception.

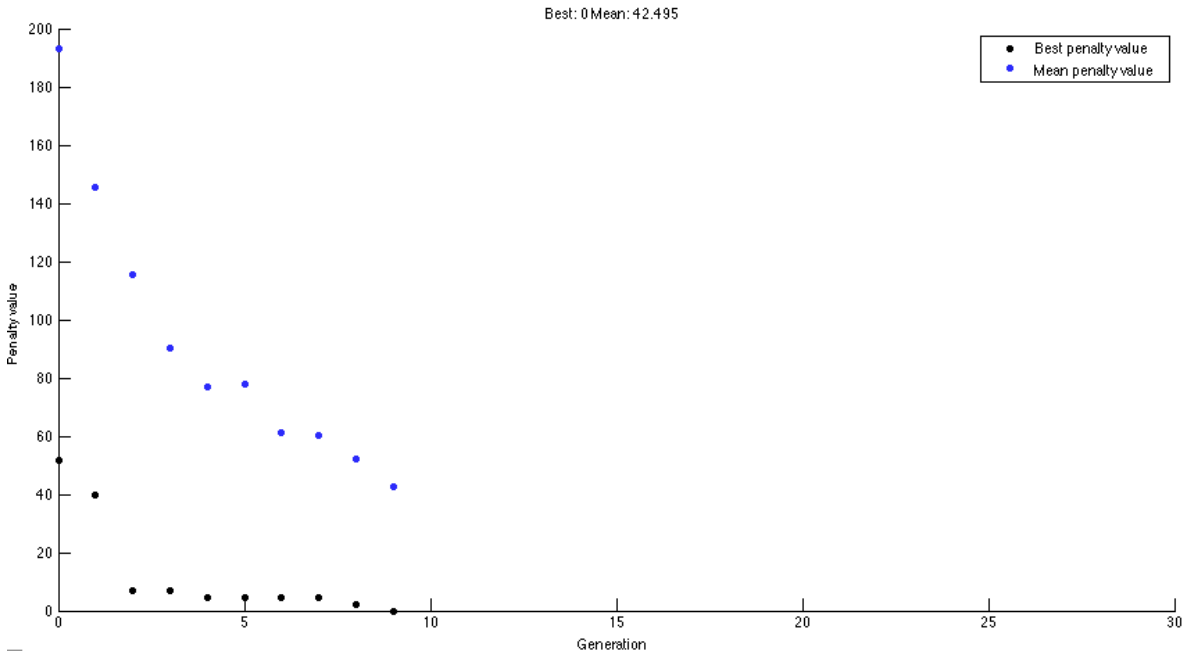
6 Evaluation and Results

6.1 Simulated Sounds

I tried a first implementation with a simple physic plucked synthesizer. First idea is to set up a target sound made by the plucked synthesizer to prove the convergence of our system: this is called "reverse engineering". If it doesn't work, it means that the fitness function doesn't work or then we have to change the GA set up (too much mutation, bad crossover, mistakes in the initial population…).

We can see the results here with a plot of the best fitness value (in black) versus the mean fitness (in black), per generation. Our goal is to reach the lowest

fitness value we can. According to our precedents experiments with the Mfcc matrix of similarity, a sound start to be close to an another sound beginning from a fitness value of 300.



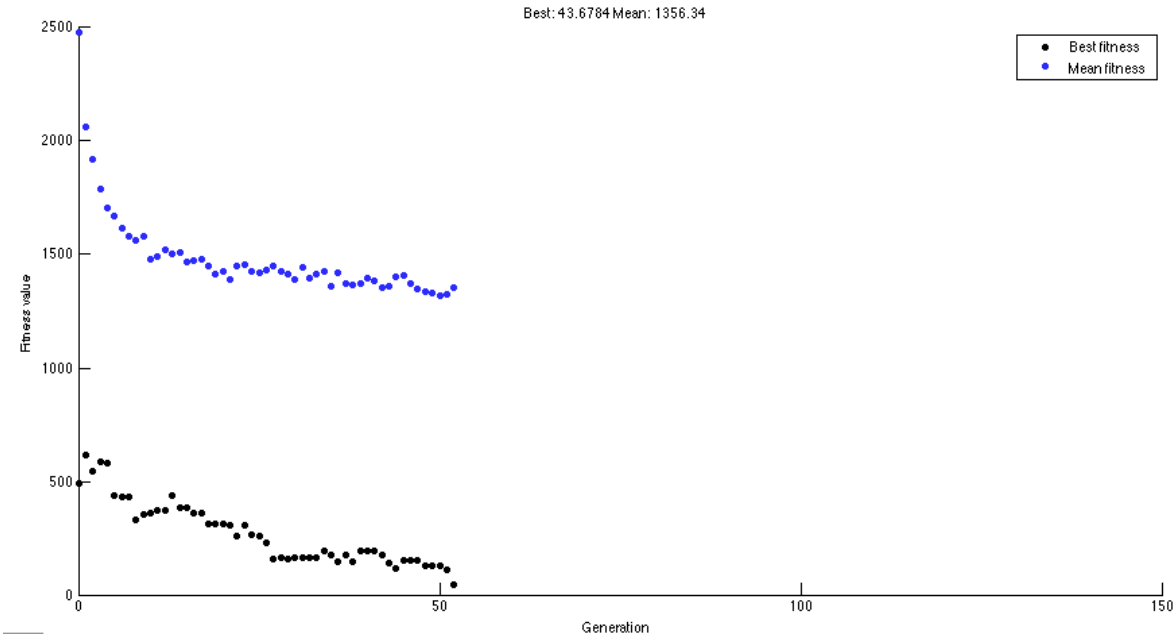
The GA converges at the end of a few iterations very quickly toward the right parameters we used to create the sound. This synthesizer had "only" three parameters to optimize, we expect that it will be much longer with our twenty two parameters.

Now we made some tests on simple synthesizers, the real first experiment on the commercial synthesizer was intended to prove the convergence of our system. As target sound, we used two sounds synthesized with the "FM" and the "Digital" audio engines. We were expecting our system to find the target set of parameters by reverse engineering. We first wanted to conduct 3 experiments per sound, each of these experiments using a different fitness function: "Spectrogram" - "Mfcc" - "Clustering with Mfcc".

We finally used only the “Mfcc” fitness function, why? Those experiments should first have been done on a super-computer, which work way faster than a simple computer, plus the genetic algorithm can be parallelized. It didn’t work and we lost a lot of time trying to make it work. To give an idea, an experiment takes half an hour and it takes a day on a classic computer.

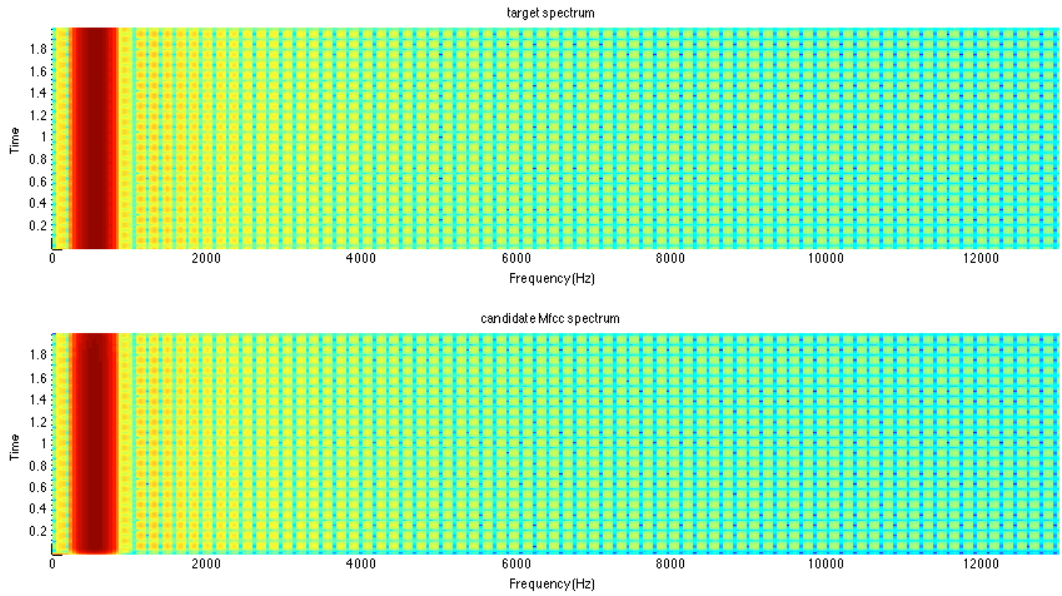
Experiment 1: Retro Engineering on a Fm audio engine

Here is a plot of the best fitness value (in black) versus the mean fitness (in black), per generation. Table 1 gives our algorithm set up.



Our system is converging quite quickly toward the exact same sound after about 50 generations. We had a population of 500 individuals and we stopped the algorithm at a fitness value of 50 because it is enough low and it means that we found a sound very close to our target sound.

Besides, the spectrogram of the two sounds prove that we found a good set of parameters to mimic our target sound:



The next two experiments can be heard at this URL:

http://metacreation.net/ldroguet/Sounds/Retro_Engineering/html_soundSet/index.html

6.2 Recorded Sounds

For our next experiments, we used two kinds of instrument sounds. The first ones are acoustic sounds recorded in anechoic chamber from the MIS database. Each sound is approximately 2 seconds long. The sound set included the following instruments: alto sax, bass, cello, flute, horn, trombone, trumpet, and viola. The second kinds are samples from different types of commercial synthesizers: TB303, Juno 106, Korg Ms20, Korg Ms10 and Native Instrument Massive).

All the sounds can be heard at this URL:

http://metacreation.net/ldroguet/index sound.html

Population	Between 100 and 1000
Max Generation N_{gen}	Between 100 and 5000
Stopping Criteria SC	Weighted change in the fitness $< 10^{-10}$ over 50 generations
Selection Operator S	Binary tournament
Elite Children N_{elite}	2%
Mutation Operator M1	40% Gaussian
Mutation Operator M2	10% customized
Crossover Operator C	Two points
Crossover Proportion	50%

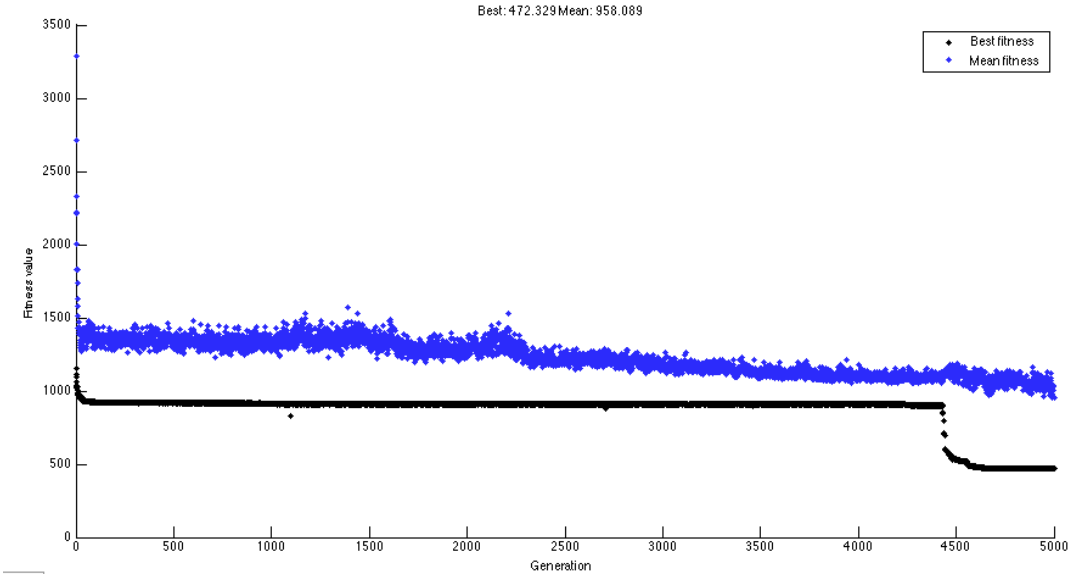
Table 1. Parameters for the GA

6.3 Discussion

The fitness score, which is an euclidian distance between two sounds, is neither an absolute nor perceptual measure and varies between the different target sounds. For example, the best synthesized sound for a Trumpet seems as good as the one for Saxophone but their respective fitness score are very different (Trumpet best fitness: 520 and Sax best fitness: 341). See figure at the end.

Generally, the results are average listening whereas the spectrograms are quite similar for some of them: it is partially due to the fact that we couldn't use the super computer and we made progress very slowly. Or either the fitness function is not enough complex or come from the limitation of the synthesis architecture we are using.

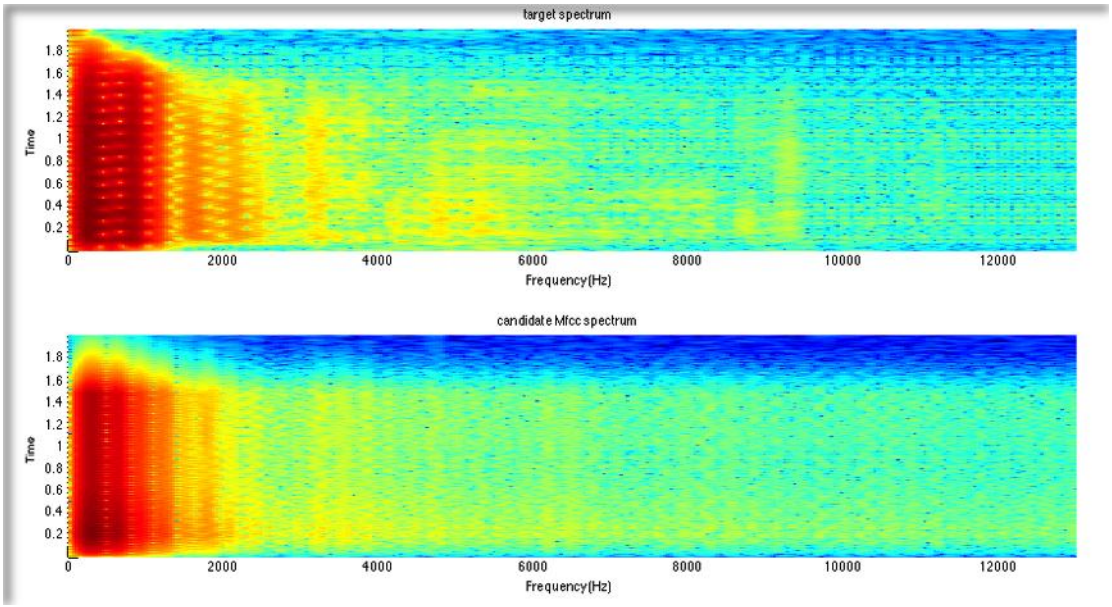
This picture shows us the importance of a lot of generations. Sometime the genetic algorithm can stagnate a long time and suddenly find a good parameter orienting the algorithm toward a better optimum.

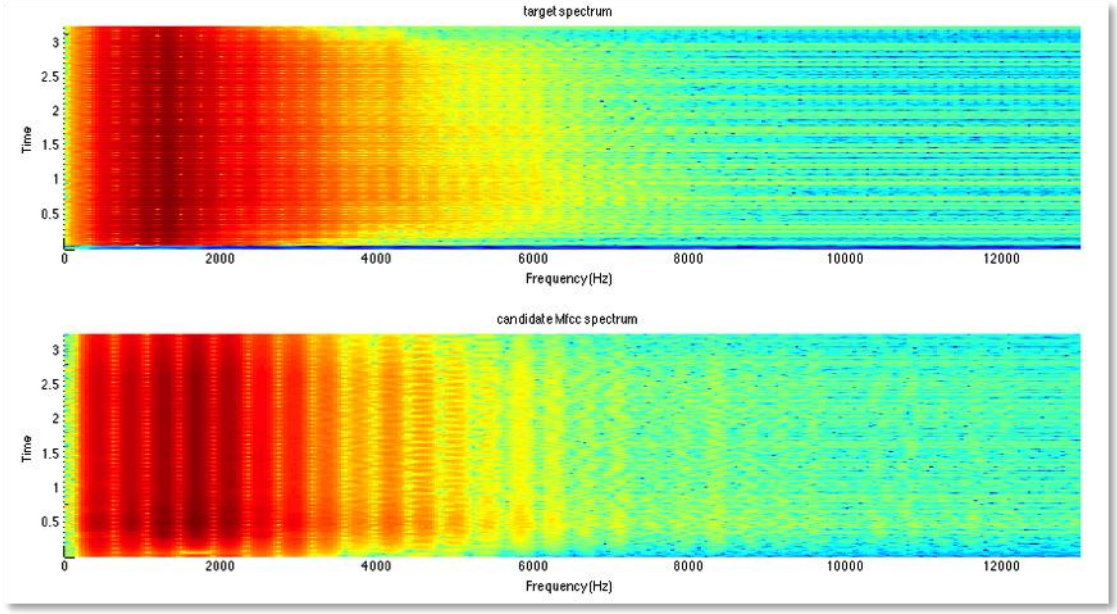


As you can see on all the experiments:

http://metacreation.net/ldroguet/index sound.html

there is no convergence because we needed much more generations and a bigger initial population. Still, we had some interesting and encouraging results at the end. Indeed, some sounds are perceptually close to the target sound it had to mimic as the two examples below (Saxophone and Trumpet):





The spectrograms of the GA best result and the target sounds are generally quite similar in these two cases.

7 Conclusion and future work

We have refined a system to optimize the parameters in a commercial synthesizer to mimic a given target instrument tone. This system is based on a pre-analysis of the target sound, matching through a genetic algorithm. Studies were conducted with various instruments and our results were compared following the set up of the genetic algorithm.
By a lack of time, we couldn't run all the experiments until convergence and we couldn't compare the three fitness functions. Especially I regret the fact that I couldn't use the "clustering fitness" because it was showing good results with the similarity matrix. Next time it will be much more efficient if the algorithm could be parallelized on a super computer. Nevertheless, results with the Mfcc feature are encouraging and show that GA gives generality and efficiency to parameter matching.

8 Bibliography

[DIMI09] D. Tsimeas, E. Mangina, "Dynamic Techniques for Genetic Algorithm-Based Music System", Computer Music Journal - MIT Press, Volume 33 - Number 3, 2009.

[MCDERM08] J. McDermott, Niall J.L. Griffith, M. O'Neill, "Evolutionary Computation Applied to Sound Synthesis", The Art Of Artificial Evolution - Natural Computing Series, 2008.

[YEE11] M. Yee-King, M. Roth, "A comparison of Parametric Optimization Techniques for Musical Instrument Tone Matching", Audio Engineering Society Convention Paper, 2011.

[WEHN98] K. Wehn, "Using Ideas from Natural Selection to Evolve Synthesized Sounds", Proceedings of the Digital Audio Effects DAFX workshop, 1998.

[MCDER06] J. McDermott, M. O'Neill, Niall J.L., "Target-Driven Genetic Algorithms for Synthesizer Control", Dept. Computer Science and Information Systems, University of Limerick, Ireland, 2006.

[CAS08] M. A. Casey, R. Veltkamp, M. Goto, M. Leman, C. Rhodes, and M. Slaney. Content Based music information retrieval: Current directions and future challenges. Proceedings of the IEEE, 96(4): 668–696, March 2008.

[LOG00] B. Logan. Mel frequency cepstral coefficients for music modeling, International Symposium on Music Information Retrieval, 2000.

[HOR93] A. Horner, J. Beauchamp, and L. Haken. Genetic Algorithms and Their Application to FM, Matching Synthesis, Computer Music Journal, 17(4): 17–29, 1993.

[GAR02] R. Garcia. Growing sound synthesizers using evolutionary methods. In Proceedings of ALMMA 2002 Workshop on Artificial Models for Musical Applications, pages 99–107, Citeseer, 2001.

[MIT05] Mitchell, T. (2005) Frequency modulation tone matching using a fuzzy clustering evolution strategy. In: Audio Engineering Society 118th Convention, 28-31 May 2005, Barcelona, Spain.

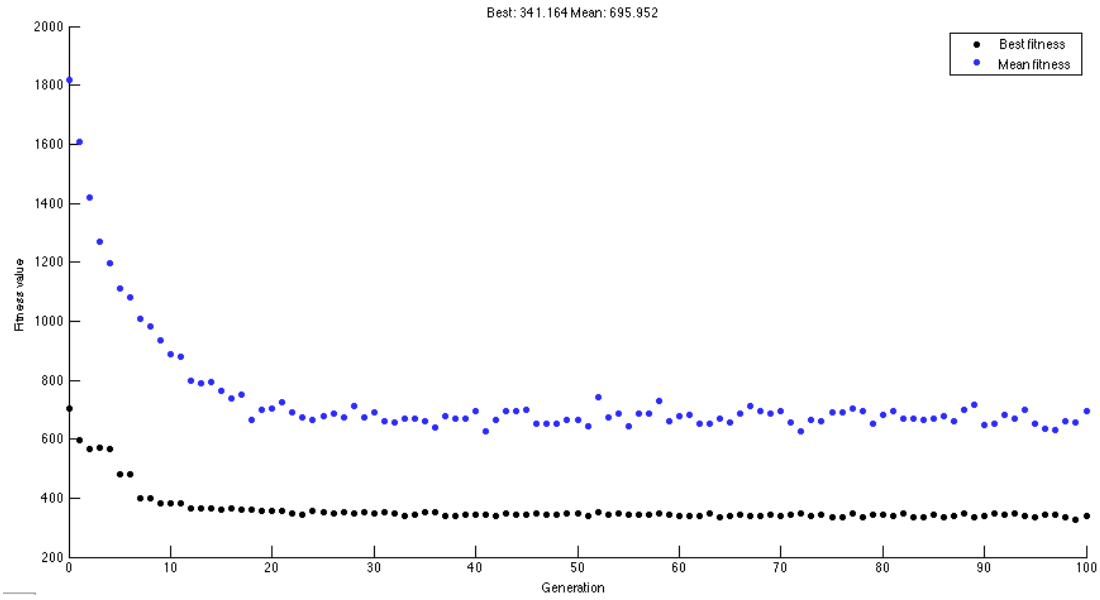
[LAI06] Y. Lai, S.K. Jeng, D.T. Liu, and Y.C. Liu. Automated optimization of parameters for FM sound synthesis with genetic algorithms, International Workshop on Computer Music and Audio Technology, Citeseer, 2006.

[Chi07] M. Chinen and N. Osaka, Genesynth: Noise band-based genetic algorithm analysis/synthesis framework, In Proceedings of the ICMC 2007 International - Computer Music Conference, 2007.

[MM12] Matthieu Macret, T. Smith and P. Pasquier. Using Genetic Algorithms to automatic Modified FM synthesis parameter setting for harmonics sounds, Sound and Music Computing, 2012.

Figures:

Sax



Trumpet