Automatic Tuning of the OP-1 Synthesizer Using a Multi-objective Genetic Algorithm

Matthieu Macret
School of Interactive Arts and Technology
Simon Fraser University, Surrey,
B.C., Canada V3T 0A3
mmacret@sfu.ca

Philippe Pasquier
School of Interactive Arts and Technology
Simon Fraser University, Surrey,
B.C., Canada V3T 0A3
pasquier@sfu.ca

ABSTRACT

Calibrating a sound synthesizer to replicate or approximate a given target sound is a complex and time consuming task for musicians and sound designers. In the case of the OP1, a commercial synthesizer developed by Teenage Engineering, the difficulty is multiple. The OP-1 contains several synthesis engines, effects and low frequency oscillators, which make the parameters search space very large and discontinuous. Furthermore, interactions between parameters are common and the OP-1 is not fully deterministic. We address the problem of automatically calibrating the parameters of the OP-1 to approximate a given target sound. We propose and evaluate a solution to this problem using a multi-objective Non-dominated-Sorting-Genetic-Algorithm-II. We show that our approach makes it possible to handle the problem complexity, and returns a small set of presets that best approximate the target sound while covering the Pareto front of this multi-objective optimization problem.

1. INTRODUCTION

Sound re-synthesis is the process of replicating a sound using electronics or software. Tuning a synthesizer for this purpose can be a very unintuitive task, since changes in input parameters can give rise, via nonlinearities and interactions among parameters, to unexpected changes in the output sound. Optimization techniques such as GAs have been successfully used to perform this tuning in the past for a variety of synthesis techniques [1–3] and synthesizers of increasing complexity [4–7]. It is now common for synthesizers to include several synthesis engines, FX and LFOs with large parameter complexity. A research partnership has been started by the authors in collaboration with Teenage Engineering (TE), with the goal of developing a system able to tune their synthesizer, the OP-1, to approximate any given target sound. Tuning this synthesizer presents all the characteristics of a real world problem. In comparison to previously studied synthesizers, the search space is larger, discontinuous and more difficult because of a large number of possible local minima. To the best of our knowledge, there is no system capable of successfully tuning sound synthesizers with the same complexity as the OP-1.

In collaboration with TE, we have developed a new system to select suitable parameters for the OP-1 to approximate a given target sound. In contrast with previously developed systems, we consider that the problem requires a multi-objective approach to be solved. We use a custom Non-dominated Sorting Genetic Algorithm (NSGA-II) to address this problem. An evaluation of the system with contrived and non-contrived target sounds is presented in Section 5. Sound examples are available for public audition [8].

2. RELATED WORKS

The complexity of a re-synthesis problem can vary tremendously according to the number and the nature of the synthesis parameters to search. First efforts focused on optimizing a limited number of synthesis parameters.

Horner et al. [2] used additive synthesis to replicate instrument sounds. A GA was used to better approximate the envelopes to apply to the oscillators. The number of oscillators to use and their frequencies were not determined by the GA but through spectral analysis.

Chan et al. [1] implemented and evaluated a hybrid sampling wavetable model. A GA adjusted some of the wavetable parameters in order to minimize phase cancellations during the crossfade between sampling and wavetable synthesis.

Wakefield and Mrozek [9] used subtractive synthesis to create artificial reverberation. A GA was used to search for low-order filter parameters so that the generated impulse response best matched that of a target room transfer function.

Horner et al. also used GA to optimize several FM parameters: the modulation indices, carrier and modulator frequencies for a variety of carriers [3]. The relative spectral error between the original and matched spectra served as a fitness function in guiding the GA's search for the best FM parameters to mimic instrumental sounds. The FM synthesis model was constrained to limit the complexity of the problem. In a previous work, we used a similar technique to optimize modulation indices, carrier and modulator frequencies for ModFM synthesizers [5].

Automatic optimization schemes were also used to solve another class of more complex re-synthesis problems. These problems involved searching every synthesis parameter. For example, Vuori et al. [6] built a GA system for estimating each parameter of a non-linear physical flute model. Each chromosome of this system represented 8 different parameters of the physical model. The relative spectral error between the original and matched spectra served

as the fitness function. They showed that their algorithm converged smoothly and effectively with the desired sound.

Bozkurt and Yuksel [7] presented automatic parameter tuning experiments with genetic algorithms in application to multiple-modulator FM synthesis. Contrary to the FM synthesis systems previously presented [3, 5], their synthesis model was not constrained and every FM parameter was estimated. Mitchell [10] applied an Evolutionary Strategy to tune the parameters of a FM synthesis model of similar complexity.

GAs were also used to determine the parameters of more complex synthesizers. Yee-King and Roth [4] used a GA to find the parameters of Virtual Studio Technology instruments (VSTi) synthesizers to match a given target sound. They evaluated their system on 2 VSTi synthesizers: the mdaDX10, a single modulator FM synthesizer with 16 parameters, and the mdaJX10, a substrative synthesizer with one noise oscillator and 40 parameters.

In this work, we apply an automatic optimization scheme to tune the parameters of a current commercial synthesizer: the OP-1. This synthesizer differs from the synthesizers previously presented in the number and the complexity of its parameters and in the number of available synthesis modules.

3. AN ALL-IN-ONE SYNTHESIZER: THE OP-1

The OP-1 is the all-in-one portable synthesizer, sampler and controller developed by Teenage Engineering (TE) [11]. TE provided us with a C++ library that embeds most of the functionalities of the OP-1. We had access to seven different synthesizer engines (FM, Digital, DrWave, String, Cluster, Pulse and Phase), four different FXs (Delay, Grid, Punch, Spring) and three different LFOs (Tremolo, Value, Element). In the following sections, the parameters selecting the engine, FX and LFO will be referred to as *type parameters*. Only one engine, one effect and one LFO can be used at a given time to produce a sound. An ADSR envelope is also always applied to the sound. Once chosen, the synthesizer engine, FX, LFO and ADSR are each controlled by 4 knobs. In the following sections, the parameters controlling the knobs will be referred to as *knob parameters*. The knob parameters are mapped to integers ranging from a minimum of 0 to a maximum of 32767, corresponding to the fine-tuning mode of the OP-1. The OP-1 has 24 physical keys and it is possible to change the octave from -4 to 4. Therefore, 120 different keys ($8 \times 12 + 24$) are available when using the OP-1. We refers as OP-1 presets a set of *knob parameters* and *type parameters*. More details about the functionalities can be found on the Teenage Engineering website [11].

Equation 1 gives the number of possible different combinations.

$$N_{\text{eng}} \times N_{\text{LFO}} \times N_{\text{FX}} \times N_{\text{k}}^{N_{\text{knobs}} \times N_{\text{t}}} \times N_{\text{key}} \quad (1)$$

where N_{eng} is the number of engines type, N_{LFO} the number of LFO types, N_{FX} the number of FX type, N_{t} the number of modules that can be controlled by knobs (engine, LFO, FX and ADSR), N_{k} the number of possible integer values for each knob and N_{key} the number of keys. Their numerical values are given in Table 1. An estimate of the total number of possible combinations for the OP-1 synthesizer is then 10^{76}.

4. AUTOMATIC CALIBRATION WITH A MULTI-OBJECTIVE GENETIC ALGORITHM

Searching the synthesizer parameters space to approximate a given target sound has all the characteristics of a real world problem. First, the search is very large (10^{76} possible different combinations). By comparison, the number of atoms in the observable universe is estimated at about 10^{80}. Second, the synthesizer is not fully determinist. The output sound can be slightly different for the same set of input parameters, which induces noise in the evaluation and can then slow down or even mislead the search. Third, there are discontinuities in the search space. For example, for a given individual, switching from an engine to another completely changes the nature of the output sound. As a result, its fitness objectives values also substantially changes causing a discontinuity in the fitness landscape. It also completely modifies the mapping of the *knob parameters*. For example, the *knob parameters* for a FM engine do not map to the same synthesis parameters than the *knobs parameters* for a Digital engine. Finally, our experiments showed that there is a large number of local minima (see Section 4.3). For instance, it is often possible to get a similar level of sound approximation using two different engines. Given these problem characteristics, it is not conceivable to use a random search or a simple optimization technique such as hill climbing or greedy algorithms to find a good set of parameters to match a given target sound. These techniques are highly dependant on the initial conditions and doesn't scale very well to large and difficult search spaces [12].

GAs are search algorithms that mimic the process of natural evolution. In a GA, a population of strings called chromosomes (which encode candidate solutions to an optimization problem) are evolved toward better solutions. GAs are especially well adapted to the characteristics of our problem. First, GAs scales very well to complex fitness landscapes [13]. Contrary to gradient search methods, they are less susceptible to converge prematurely to a local minima.

Second, GAs perform well in search space where the evaluation is approximative or noisy [14]. Adjustable selection pressure makes it possible to keep diversity in the population. A large number of individuals are evaluated for each generation. Because mutation and crossover are stochastic operators, it is common for an individual to be rediscovered several times during the evolution. The fact that the individual is re-evaluated each time it is rediscovered makes it possible to reduce the effect of the noise in the evaluation.

GAs are complex algorithms with a large set of parameters to tune (population size, stopping criteria, choice of the genetic operators...). In order to find the best configuration for the GA, we explored several options. In the following sections, we adopt the vocabulary developed by Mitchell et al. [15] and refer to the target sounds generated using the OP-1 as *Contrived sounds*. *Contrived sounds* were used as target sounds when exploring different GA configurations.

N_{eng}	N_{LFO}	N_{FX}	N_{k}	N_{knobs}	N_{t}	N_{key}
7	3	4	32767	4	4	120

Table 1. Synthesizer parameters complexity

Using these sounds as target sounds has two advantages. First, it ensures that a solution exists. Second, it is possible to easily track the performance of the algorithm because the target synthesis parameters are known.

We adopted an iterative design process and considered problems of increasing complexity. Table 2 describes these problems ordered by increasing complexity. From problem 1 to 4, we progressively added the knob parameters. In this first set of problems, we fixed the type parameters to limit the search space discontinuities. Finally, in problem 5, every type and knobs parameters were searched. At first, we limited the search to the 4 knobs controlling the engine parameters (Problem 1). We experimented with different GA configurations until we found one configuration able to either, in the best-case scenario, reverse engineer the target set of OP-1 parameters or, in the worst-case scenario, gave a perceptually satisfying approximation of the target sound. Once a satisfying GA configuration was found, the 4 knobs controlling the ADSR were added (Problem 2). The previous satisfying GA configuration was tested on the new problem. If this configuration was not satisfying anymore, we adjusted the GA parameters again until a satisfying one was found. This process was reiterated with the other problems until we obtained good performance when searching every parameter (Problem 5).

In the following subsections, we describe the final system implementation for searching all the parameters. We explain our design choices given the observations gathered during the different steps of our iterative design process.

4.1 Representation

The parameters in our synthesizer are integers. At first we decided to encode these parameters using a binary representation (which is a common choice in practice). However, our experiments using this representation on Problem 1 (see Table 2) seemed to indicate that the GA was always converging to the same local minima. Investigating further, we realized that, in order to improve the best individual fitness, 12 bits would have to be changed to go from 4095 to 4096 and improve the objective fitness values, which is very unlikely to happen. We then switched from a binary encoding to a Gray code encoding for both *type parameters* and *knob parameters*. The Gray code is based on the idea that two successive values differ by only one bit. Our experiments with this new encoding showed that the GA now converged toward the target set of parameters, thus we chose to keep this representation for our system. Our chromosome is made up of blocks representing the type and knobs parameters. The two first lines of Table 2 in bold letters show the final chromosome design.

4.2 Genetic operators

4.2.1 Crossover

Losing diversity during the evolution is a normal phenomenon given that we apply a selection pressure on the population. However, a lack of diversity can lead to premature convergence because there is not enough genetic material to explore the fitness landscape. One reason for the loss of diversity is the recombination of identical chromosomes. Indeed, when the same individual is selected twice for cross-over, two offsprings identical to this individual are produced. This phenomenon causes the diversity to go down. In order to avoid this situation, we apply a crossover operator that tests the parent chromosomes before recombining them. If they are identical, the first offspring will be a copy of the parents and the second offspring will be a new randomly generated chromosome. This simple technique is shown to be efficient in slowing down the diversity loss and prevent premature convergence [13]. Our system uses a 2-point crossover and a crossover rate of 60 %.

4.2.2 Mutation

The mutation operator participates in both exploration and exploitation (local search). Flipping one bit in a Gray code can either lead to a small change in the coded parameter (local search) or a relatively large change in the coded parameter (exploration: by jumping to another area of the fitness landscape). This flip-bit mutation operator is applied to every individual in the population whether recombined or not. In our system, the probability of flipping k bits in a N_{bits} long chromosome follows a binomial law with $p = \frac{1}{N_{\text{bits}}}$ and $n = N_{\text{bits}}$.

4.3 Fitness function

In our attempt to solve Problem 1, we used the Euclidian distance between the Short-Time Fourier Transform (STFT). The sampling rate was 44100 Hz and we set a window size of 1024 samples (23 ms) and an overlapping of 512 samples (11.5 ms). Our experiments showed that this fitness function worked well when we restricted the optimization to include only the 4 knobs which controlled the engine parameters (Problem 1).

However, when we added the 4 knobs controlling the ADSR parameters (Problem 2), the GA appeared to converge prematurely. A further investigation of this phenomenon showed that the weight of the temporal envelope in the Euclidian distance between the STFTs is significant. For example, consider a target sound T and two candidate sounds A and B. A has a similar spectrum to the spectrum of T for the first short-time windows but not for the last ones because the temporal envelope for A has a shorter release time than the one for T. Globally, the spectrum for B is not as similar to the spectrum for T but their temporal envelope is the same. The weight of these last short-time windows in the STFT distance can make B appear closer to T than A.

In this context, a right set of engine knobs parameters (A) can be discarded because the associated ADSR knobs parameters are not right. It slows down the evolution because some good genetic material is lost. It can even lead to a premature convergence if this set of engine knobs parameters is never recovered again later in the evolution.

It is not surprising that, in previous work [3, 5], the envelope was determined analytically for each individual in the population, however it is not possible to do this in our case because we do not know the mapping between the ADSR knob parameters and the resulting temporal envelope. Indeed, performing a local search to set the ADSR knobs parameters for each individual in the population would be too computationally expensive. Furthermore, given the non deterministic nature of our synthesizer, any classic local search algorithm such as greedy algorithm or hill climbing would likely fail because the noise in the evaluation would mislead the search.

Pb. Id	Engine		FX		Key / Octave	LFO		ADSR	N_{bits}
	Type	Knobs	Type	Knobs		Type	Knobs	Knobs	
1		X							60
2		X						X	120
3		X		X				X	180
4		X		X			X	X	240
5	X	X	X	X	X	X	X	X	257

Table 2. Problem descriptions

In order to avoid the premature convergence observed with Problem 2, we decided to uncouple the temporal envelope from the spectral components as much as possible. Thus, we chose to extract two separate sound features: 1) the FFT computed on the entire sound; and 2) the temporal envelope. Computing the FFT on the entire sound mitigates, to some extent, the effect of the temporal envelope on the spectrum. We extracted the temporal envelope using the Hilbert transform followed by a low-pass filter.

Our first idea was to put these two sound features in an aggregate fitness function (see Eq. (2)). However, it is difficult to choose the appropriate weights for the temporal envelope a_{env} and the FFT a_{FFT} to make the system converge.

$$f = \frac{a_{\mathrm{env}} f_{\mathrm{env}} + a_{\mathrm{FFT}} f_{\mathrm{FFT}}}{a_{\mathrm{env}} + a_{\mathrm{FFT}}} \tag{2}$$

Therefore, we chose to consider two objectives: FFT and temporal envelope instead of only one: the STFT. We implemented this new 2-objectives fitness function in a multi-objective framework, the Non-dominated-Sorting-Genetic-Algorithm-II. The experiments showed that this new system converged to the target set of parameters for Problem 2.

However, when we added the 4 knobs controlling the LFO or FX (Problem 3-4), our system was converging prematurely again. The explanation was that the addition of a LFO or FX made the spectrum of the target sound non-stationary. The FFT on the entire length of the sound was not able to capture the variation of the spectrum over time. In order to deal with this limitation, we added back the STFT as a third objective. Contrary to simple GA, the NSGA-II uses a selection operator based on non-domination sorting. Therefore, contrary to the simple GA using STFT as fitness function, an individual with a good set of engine knobs parameters would be more likely kept in the population even if it has wrong ADSR knobs parameters. Indeed, this individual would have a high fitness value for the FFT and a low fitness value for the envelope. It would be then kept in the population because it is dominating the population according to the FFT objective. In the simple GA, this individual would likely be discarded because its fitness value would be affected by a wrong temporal envelope.

4.4 Selection

Our system is based on a Non-dominated Sorting Genetic Algorithm II (NSGA-II). Details about this algorithm can be found in Deb's work [16]. The principal features of this algorithm are the following:

- *Elitism:* This property prevents the loss of good solutions once they are found by insuring that the fittest members of the population are kept in future generations.
- *Non-dominated sorting:* An individual is said to be non-dominated if there is no other individual that performs better for at least one of the objectives without performing worse for the remaining objectives. This principle is used to sort the population into non-dominated sets that are then used to form the next generation.
- *Diversity preservation:* The crowding distance is a measure of how close an individual is to its neighbours. A crowded-comparison operator guides the selection process at the various stages of the algorithm toward a uniformly spread-out Pareto front.

Our system uses a population of 500 individuals for each generation. This number of individuals was empirically determined as a good trade-off between performance and computational cost.

4.5 Stopping criteria

The optimization process terminates if the weighted change in the 3 objective fitness, given by Eq. (3), is less than 10^{-30} over 200 generations. δ_{n} is the weighted change at generation n, f_{k} is the best objective fitness score at generation k, N = 200 if $n \geq 200$ otherwise $N = n$. If this condition is never verified, the optimization process stops after 3000 generations.

$$\delta_{\mathrm{n}} = \sum_{i=1}^{N} \left(\frac{1}{2}\right)^{i-1} (f_{\mathrm{n+1-i}} - f_{\mathrm{n-i}}), \tag{3}$$

4.6 Pareto front

The Pareto front is the set of non dominated individuals for the 3 objectives.

4.6.1 Similarity rule

In our first experiments, the Pareto front was very large at the end of the evolution (more than 2000 individuals). Upon closer examination, we realized that strictly identical individuals were present in the Pareto front. This was due to the fact that the synthesizer is not fully deterministic and the non-domination, crowding selection is only made on objective fitnesses. We then added a similarity rule that tests whether the chromosome of an individual is already present before adding it to the Pareto front. This simple rule made it possible to cut the size of the final Pareto front by a factor of more than 2.

However, the size was still too large (around 1000 individuals) to be easily analyzed by a user. Further investigating the Pareto front, we realized that a large number

of individuals sounded perceptually identical even if they were produced using different sets of parameters. In order to limit this kind of duplicate individuals in the Pareto front, we refined the similarity rule. We stated that 2 individuals are considered identical if they have the same engine/LFO/FX types, identical key/octave and the Euclidian distance between the knob parameters is less than 1000 (3 % of the knob parameter range). This value was defined by making tests on a large numbers of Pareto fronts. Depending on the target sound, this last change made it possible to reduce the size of the Pareto front to between 10 and 150 individuals while conserving its quality and diversity.

4.6.2 Post processing

A number of 150 individuals in the Pareto front is still very large to be handled by a user. We applied a technique developed by Chaudhari et al. [17] to select the most significant individuals in the Pareto front when its size is superior to 10 individuals. This approach consists of the following steps:

1. Apply a k-means clustering algorithm to cluster on the solutions enclosed in the Pareto set. The clustering is done on the OP-1 parameters because our goal is to help the user to identify different good OP-1 presets in the Pareto front, for example using different sound engines.

2. Determine the optimal number of clusters, k. The silhouette of an individual is a measure of how closely it is matched to other individuals within its cluster and how loosely it is matched to individuals of the neighbouring cluster. A silhouette $s(i)$ close to 1 implies that the individual i is in an appropriate cluster, while $s(i)$ close to -1 implies that i is in the wrong cluster. Thus the average $s(i)$ of the entire Pareto Front is a measure of how appropriately the Pareto Front has been clustered. A value of the average silhouette is obtained for several values of k with $k < 10$. The k that gives the highest average silhouette width is selected.

3. For each cluster, select a representative solution. For each cluster, the individual, within the cluster, that encodes the OP-1 presets that is the closest to the cluster centroid presets is selected as the representative solution.

4. Analyze the results. At this point, the user can analyze the k representative solutions of the clusters and then explore the individuals of the cluster that seems the most promising.

4.7 Full problem complexity

In Problem 5, we add the type parameters. These extra parameters to search induces discontinuities in the fitness landscape (see Section 4). However, our experiments show that adding the type parameters to the search (Problem 5) do not diminish the final solution quality when we compare them to final solutions found for Problem 4. The *right* type parameters are determined in early generations and become prominent in later generations. The evolution continues then as if it would be Problem 4 being solved. This phenomenon is induced by the selection pressure and mimics very well the behaviour of a human asked to perform the same task. One would broadly explore the possibilities of the synthesizer and quickly select an engine and key, after which one would fine tune the knob parameters. Another explanation to this phenomenon is that the chromosome size is not very different between problems 4 and 5 (240 bits against 257 bits) because every type parameters has a small range compared to the knobs (see Section 3).

4.8 Implementation

The implementation of the GA is done using the DEAP Python framework [18]. Sound features are extracted using the Python wrapper for Yaafe [19]. Yaafe is coded in C++ and has the advantage of being fast and memory-efficient. In our current implementation, the time to evaluate the 3 objectives for a 1 second-long mono sound sampled at 44100Hz is in average 314 milliseconds (SD= 1ms).

The bottlenecks of our algorithm are the fitness evaluation and the NSGA-II selection operator. The fitness evaluation is distributed between 100 processors on a supercomputer [20] to speed up the computation. It also make it possible to use larger populations than would have been feasible using only one processor for the same running time. We use the DEAP C++ version of the NSGA-II selection operator to further speed up our algorithm. In our current implementation, applying the genetic operator (crossover, mutation, selection) for a population of 500 individuals takes in average 243 milliseconds (SD=4 ms). The total computing time for a run is in average 34 min (SD=4min).

5. EVALUATION

We based our evaluation design on Johnson's recommendation about experimental analysis of algorithms [21]. We especially focused on ensuring reproducibility and comparability.

5.1 Sound collection

5.1.1 Contrived sounds

Using *contrived sounds* as target sounds allows us to validate our system design. It makes possible to show that our system is able to reverse engineer a given target sound generated by the OP-1.

Given the complexity of the algorithm and its running time, we chose to limit our evaluation to 12 contrived sounds. We selected these sounds in order to have a sample of spectrums that was diverse and representative of the the OP-1 possible outputs. We especially focused on having diversity in spectral variation, noisiness and spectral spread. The first half of the sounds has a stationary spectrum and the other half has a non-stationary spectrum, as measured by their respective spectral flux.

We made sure that we used each engine, LFO and FX at least once to generate this set of sounds. These contrived sounds are available to listen online [8].

5.1.2 Other sounds

A second evaluation was performed on 12 non-contrived sounds including synthetic sounds (DX-7 synthesizer, Moog synthesizer, lightsaber sound), instrument sounds (Violin, flute, bassoon, snare), a male voice sound and a natural sound (Cat meow). These sounds were carefully

chosen to have a good diversity in spectral variation, noisiness and spectral spread.

5.2 Measurement

In these evaluations, we evaluated the solution quality and the running time.

The solution quality was measured differently for the contrived sounds and for the non-contrived sounds. For the contrived sounds, we already know what are the target OP-1 presets. In addition to the target sound, we generate ten other sounds using the target presets and compare them to the target sound. With a determinist synthesizer, their objective fitness values (FFT, envelope and STFT) would be equals to zero but it is not the case with the OP-1. We define the best possible objectives values as the minima of the 3 objectives fitness values over these 10 sounds. We calculate the relative error for each run subtracting these best possible objective values to the best objectives fitness values obtained in the particular run. For the non-contrived sounds, we are only able to measure the final fitness values for the 3 objectives at the end of the evolution. The running times are measured by the number of generations before the GA reaches the stopping criteria ($nbGen$).

5.3 Analysis

We ran the algorithm at least 10 times for each target sound. We used Bootstrapping to obtain estimates of summary statistics [21]. This method involves taking the original data set of size N, and sampling from it with replacement to form a new sample, called a bootstrap sample, that is also of size N and that is not identical to the original sample. This process is repeated a large number of times (1000 in our case) and for each of these bootstrap samples we compute the desired statistic. This provides an estimate of the distribution of the desired statistic. Questions about how this statistic varies or the standard error for this statistic can now be answered. This technique makes possible the extraction of more useful information when the sampling size is small, as is the case in our experiments due to the time complexity of the problem.

For each of the measures described above, we used Bootstrapping to get an estimate of its minimum, maximum, mean and standard deviation. We also used the bootstrap shift method test [21] to assess the significance of every comparison we performed. This test has the advantage of being distribution-free and of scaling well with small sample size.

6. RESULTS

Our experimental results for contrived sounds and non-contrived sounds are available online [8].

6.1 Contrived sounds

6.1.1 Module types selection

Table 3 describes some statistics about the proportion of module types in the population over the various generations. Prop. choice is the proportion of runs where one type was totally taking over in the population. Accuracy is the proportion of runs choosing the correct type when one type was taking over in the population. The Take over gen is the generation as from one type was taking over. Our results suggested that our system performed well at finding the right engine type (90 % prop. choice; 80 % accurate) and the right key/octave (77 % and 91 % prop. choice; 52% and 62% accurate). However, it was not the case for the LFO (43 % prop. choice; 22 % accurate) and FX type (42 % prop. choice; 18 % accurate). A possible interpretation of these results is that the engine type and key/octave have a greater influence on the output sound than the LFO or FX type. The LFO and FX type do not change the nature of the output sound but only alter it. It is then more challenging to determine the right type for the FX and LFO.

6.1.2 Pareto front

The number of different module combinations was very low in the Pareto front (μ = 3.0, SD = 0.2 over 10 080 possible combinations). These findings suggested that the GA successfully identified a limited number of promising locations in the parameter space that dominate all others. When listening to the sounds in the Pareto front, one can distinguish several clusters of perceptually similar sounds. Each of these clusters sounds perceptually similar to the target sound but the OP-1 presets it represents are sensibly different between clusters. A Pareto front affords more flexibility to the user who receives a set of similar sounds rather than a single sound with a simple GA. The user can then make the final choice.

Our system approximates very well the temporal envelope of the target sound as shown by the very low relative errors for the envelope objective (μ = 0.20, SD = 0.02). Figure 1 shows the relative error over the best possible FFT and STFT objectives values. As measured by their respective spectral flux, conf0, conf2, conf3, conf5, conf7 and conf10 are the configurations generating non-stationary spectra. The other configurations are generating stationary spectra. We see that the performances of the GA are not significantly better for the target sounds with stationary spectra than for the target sounds with non-stationary spectra; $p = 0.07$, $p = 0.08$. However, we can still observe differences in the GA performances for different groups of target sounds. A first group with negative relative errors (conf2, conf7, conf9) contains the OP-1 target configurations that are the most non-deterministic. Indeed, in this non-determinist context, the best possible objective fitness values are very difficult to determine precisely. Then, it is possible that the GA finds an OP-1 presets outperforming the best possible objective fitness values, resulting in a negative relative error. These negative relative errors induces a bias when comparing the performances of our system for target sound with stationary spectrum and with non-stationary spectrum. A second group (conf4, conf6, conf8, conf10 and conf11) contains mostly

	Prop. choice		**Take over gen**		**Accu.**
	C	**NC**	**C**	**NC**	**C**
Engine	0.90	0.74	139	129	0.80
FX	0.42	0.44	322	270	0.18
LFO	0.43	0.45	240	317	0.22
Key	0.77	0.38	122	265	0.52
Octave	0.91	0.57	109	174	0.62

Table 3. Statistics about modules types. C: Contrived sounds, NC: Non-contrived sounds, Accu.: Accuracy

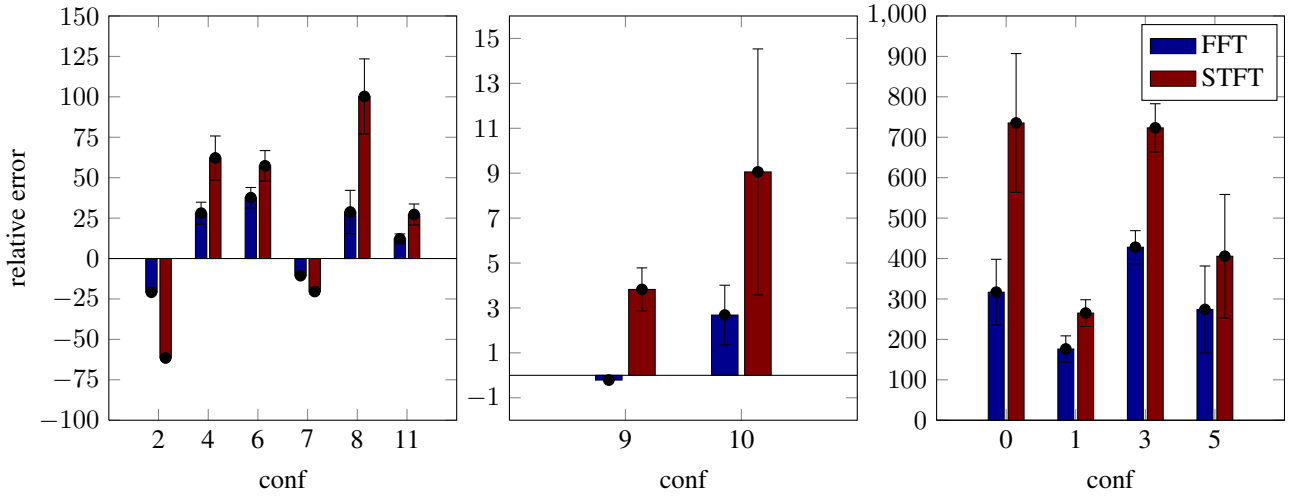

Figure 1. Objectives' relative error

target sounds with stationary spectrum at the exception of conf10. Our system is performing the best for this group as indicated by a very small relative error compared to the other groups. Conf10 presents a STFT under the form of a sawtooth wave over time. This common shape doesn't seem to be difficult to approximate for our system even if the spectrum is non-stationary. The last group (conf0, conf1, conf3, conf5) contains mostly target sounds with non-stationary spectra at the exception of conf1. The relative error for this group is the highest. The spectral energy of conf1 is mainly concentrated in a narrow frequency band. Our system seems to fall into a local minima for this particular kind of spectrum.

6.2 Non-contrived sounds

It is more challenging to evaluate the results of the non-contrived sounds experiments because we do not have any target parameters or parameter distances to refer to. Even if our system has shown good average performances for contrived sounds, it does not automatically mean that it would be good for non-contrived sounds. Indeed, the structure and complexity of the fitness landscape depends largely on the chosen target sound.

6.2.1 Number of generations to converge

The mean of $nbGen$ was significantly superior for the contrived sounds (μ = 1431, SD = 86) than for the non-contrived sounds (μ = 1070, SD = 50); $p < 0.001$. This results may seem surprising as the resynthesis problem for a non-contrived sound is more complex than for a contrived sound.

	FFT		**Env.**		**STFT**	
	μ	**SD**	μ	**SD**	μ	**SD**
C	130.3	17.2	0.20	0.02	249.0	31.2
NC	3163.5	242.7	8.7	0.78	4299.5	262.6

Table 4. Objective best fitness values. C: Contrived sounds, NC: Non-contrived sounds

6.2.2 Module types selection

Table 3 describes some statistics about the proportion of module types in the population through the generations. As with contrived sounds, an engine was quickly taking over. However, contrary to the experiments with contrived sounds, no key was clearly taking over (Prop choice 38 % against 77 % for contrived sounds). It could be explained by the fact that most of the non-contrived sounds do not have a clearly identified pitch (cat meow, DX-7 and Moog synthesizer sounds with pitch modulation). FX and LFO types were, as with contrived sounds, still challenging to set for the GA (FX prop. choice 44 %, LFO prop choice 45 %).

6.2.3 Pareto front

First, we could hear that the Pareto front sounds were perceptually similar to the targets, which is of importance for real world applications. The Pareto fronts were also significantly more diverse (μ = 4.3, SD = 0.3) than the ones we got using contrived target sounds (μ = 3.0, SD = 0.2); $p < 0.001$. They were also significantly more populated (μ = 306, SD = 11; μ = 83, SD = 21); $p < 0.001$. These differences can be explained by a larger problem complexity and also by the fact that, at the difference with contrived target sounds, the existence of a OP-1 presets that would perfectly approximate the target sound is not insured anymore. In other words, there is no guaranteed global optimum, and likely many more local optima. With the concept of Pareto front, the user receive a set of OP-1 presets that produces sounds perceptually similar to the target sound. These OP-1 presets do not involve automatically the use of the same engine, LFO or FX, which gives the users several alternatives of variable quality to approximate a given target sound.

The objective best values for the 3 objectives, shown in Table 4, were, as expected, significantly worse for the non-contrived sounds than for the contrived sounds ($p < 0.001$, $p < 0.001$, $p < 0.001$).

7. CONCLUSIONS AND FUTURE WORKS

We developed an algorithm to automatically tune the parameters of a multi-engine synthesizer in order to repro-

duce a given target sound. We described the iterative design process that lead us to consider a multi-objective approach and to use a custom Non-dominated Sorting Genetic Algorithm-II to tackle this problem. This new approach gives more flexibility to the user who receives a set of presets rather than only one preset as with previous systems. An experimental study has been conducted to assess the performances and limitations of our system. Contrived and non-contrived sounds have been considered in this study. We showed that our system is able to explore a complex parameter space in order to find configurations producing sounds perceptually similar to the target. Disparities in performances have been highlighted with the system performing better for target sounds with stationary spectrum than for sounds with non-stationary spectrum. The comparisons between the system performances for non-contrived sounds and contrived sounds confirmed even more that the complexity of the re-synthesis problem are highly variable given the nature of the target sound. Future works will entail taking into account the characteristic of the target sound to adapt the parameters of our GA system using, for example, some sound pre-processing analysis and machine learning.

Acknowledgments

This research was funded by a grant from the Canada Council for the Arts, and the Natural Sciences and Engineering Research Council of Canada. We would like to thank Teenage Engineering for their collaboration and support. Special thanks goes to Laurent Droguet from IRCAM and Dr. Corey Kereliuk from McGill university.

8. REFERENCES

[1] S. Chan, J. Yuen, and A. Horner, "Discrete summation synthesis and hybrid sampling-wavetable synthesis of acoustic instruments with genetic algorithms," in *Proc. of the International Computer Music Conference (ICMC)*, 1996, pp. 49–51.

[2] A. Horner and J. Beauchamp, "Piecewise-linear approximation of additive synthesis envelopes: a comparison of various methods," *Computer Music Journal*, vol. 20, no. 2, pp. 72–95, 1996.

[3] A. Horner, J. Beauchamp, and L. Haken, "Machine Tongues XVI: Genetic Algorithms and Their Application to FM Matching Synthesis," *Computer Music Journal*, vol. 17, no. 4, pp. 17–29, 1993.

[4] M. Yee-King and M. Roth, "Synthbot: An unsupervised software synthesizer programmer," *Proc. of ICMC, Belfast, N. Ireland*, 2008.

[5] M. Macret, P. Pasquier, and T. Smyth, "Automatic Calibration of Modified FM Synthesis to Harmonic Sounds using Genetic Algorithms," in *Proc. of Sound and Music Computing Conference (SMC)*, 2012, pp. 387–394.

[6] J. Vuori and V. Välimäki, "Parameter estimation of non-linear physical models by simulated evolution-application to the flute model," in *Proc. of ICMC*, 1993, pp. 402–410.

[7] B. Bozkurt and K. Yüksel, "Parallel evolutionary optimization of digital sound synthesis parameters," *Appli. of Evolutionary Computation*, pp. 194–203, 2011.

[8] "Experimental results," last accessed: June 2013. [Online]. Available: http://metacreation.net/mmacret/SMC2013/

[9] G. Wakefield and E. Mrozek, "Perceptual matching of low-order models to room transfer functions," in *Proc. of ICMC*, 1996, pp. 111–113.

[10] T. Mitchell, "Automated evolutionary synthesis matching," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, pp. 1–14, 2012.

[11] "Teenage engineering website," last accessed: March 2013. [Online]. Available: http://www.teenageengineering.com/

[12] M. Roth and M. Yee-King, "A comparison of parametric optimization techniques for musical instrument tone matching," in *Proc. of Audio Engineering Society Convention*, 2011, pp. 18–26.

[13] M. Rocha and J. Neves, "Preventing premature convergence to local optima in genetic algorithms via random offspring generation," in *Multiple Approaches to Intelligent Systems*. Springer, 1999, pp. 127–136.

[14] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments: a survey," *IEEE Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, 2005.

[15] T. J. Mitchell and D. P. Creasey, "Evolutionary sound matching: A test methodology and comparative study," in *International Conference on Machine Learning and Applications*. IEEE, 2007, pp. 229–234.

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[17] M. P. Chaudhari, R. VDharaskar, and V. Thakare, "Computing the Most Significant Solution from Pareto Front obtained in Multi-objective Evolutionary," *International Journal of Advanced Computer Science and Applications*, pp. 63–68, 2010.

[18] F.-A. Fortin, F.-M. D. Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[19] B. Mathieu, S. Essid, T. Fillon, J. Prado, and G. Richard, "Yaafe, an easy to use and efficient audio feature extraction software," in *ISMIR*, 2010.

[20] "Westgrid - Compute Canada," last accessed: June 2013. [Online]. Available: http://www.westgrid.ca/

[21] D. S. Johnson, "A theoreticians guide to the experimental analysis of algorithms," *Data structures, near neighbor searches, and methodology: 5th and 6th dimacs implementation challenges*, vol. 59, pp. 215–250, 2002.