AUTOMATIC TUNING OF THE OP-1 SYNTHESIZER USING A MULTI-OBJECTIVE GENETIC ALGORITHM

by

Matthieu Macret
M.Eng., Centrale Marseille, 2009

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
School of Interactive Arts and Technology
Faculty of Communication, Art and Technology

APPROVAL

Name:	Matthieu Macret
Degree:	Master of Science
Title of Thesis:	Automatic Tuning of the OP-1 Synthesizer Using a Multi-objective Genetic Algorithm

Examining Committee: Dr. Robert Woodbury
Professor
Chair

____

Dr. Philippe Pasquier
Assistant Professor
Senior Supervisor

____

Dr. Arne Eigenfeldt
Associate Professor
Supervisor

____

Dr James McDermott
Lecturer, School of Business
University College Dublin
External Examiner

Date Approved: July 16, 2013

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.

Keywords: Artificial Intelligence; Genetic Algorithms; Sound Synthesis; Multi-objective Optimization

À la mémoire de mes parents, Jean-Louis et Claudie.

“The richness I achieve comes from nature, the source of my inspiration. ”

— Claude Monet

Acknowledgments

I would like to thank my senior supervisor, Philippe Pasquier, for his enthusiasm and guidance, and for cultivating my interest in evolutionary computation and sound synthesis. Thanks to all the members of the MAMAS lab (`metacreation.net`) who provided valuable feedback on various papers and presentations related to my work.

I would also thank David Möllerstedt and Markus Nilsson from Teenage Engineering for their collaboration and support and for providing me with a software version of their awesome OP-1 synthesizer. Thanks to Westgrid-Compute Canada [5] for providing me with the computation power and the support I needed to complete my project. I would also thank Laurent Droguet from IRCAM for his preliminary work with the OP-1 and Dr Corey Kereliuk from McGill University for his feedback and advice on my work. I would also like to thank my schoolmate, Nicolas Gonzalez Thomas, for assisting me in the quest to find the best coffee shop in Vancouver to write my thesis.

Finally, I would like to thank my family, for their encouragement and support; to my little sister, Marion, that inspired me by her perseverance and courage when life was not easy; and most of all, to my girlfriend, Annie, for her understanding, patience, and optimism.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Audio synthesizers has been and are still an important tool for modern musicians, composers and sound designers, making it possible for them to electronically generate the sound of acoustic instruments or to create new sounds never heard before. A large number of synthesis techniques have been developed to craft musical sounds.

Efficient control and exploration of a synthesizer's sound space requires expert knowledge of the related synthesis technique, which may use theoretical and/or empirical knowledge. It is common that composers have to abandon their task of making music to concentrate on the task of programming a synthesizer, i.e. tuning parameters to create the desired sound. Depending upon the synthesis technique used, a variable number of parameters have to be tuned. Therefore, for a complex synthesizer, the size of the parameter search space can quickly become large and challenging to handle manually by the user. Exploring the parameter search space to tune a synthesizer, even in a principled fashion, can come a time consuming activity.

To musicians who do not have this technical knowledge, the synthesizer interface can present an obstacle between musical ideas and their expression. The synthesizer's parameters used to craft the sound are specific to the particular synthesis technique being used, and rarely reflect the human understanding or perception of sound. This makes the synthesizer interface unintuitive and challenging to handle for someone interested in musical creativity rather than programming a synthesizer.

In order to deal with this issue, synthesizer manufacturers often provide the user with a large number of parameters settings, also called presets, that can be used to produce recognizable or interesting sounds. These presets are also intended as starting points for users to explore the synthesis sound space. However, even if these presets are meant to convey the variety of sounds that the engine is capable of generating, they can still fail to cover the entire synthesis sound space. In this work, we intend to make the synthesizer interface more natural and accessible for the user. The idea would be to let the user interact with the synthesizer without having to deal with obscure parameters setting and also to facilitate the generation of other interesting sounds than the one possible using only the presets provided by the manufacturer.

A first step to achieve this is the development of a process which can efficiently search the synthesizer parameter space to identify presets which approximate given target sounds. This thesis examines the use of evolutionary computation to do just this for a modern commercial synthesizer developed by Teenage Engineering [4]: the OP-1. The OP-1 is the all-in-one portable synthesizer, sampler and controller. Compared to previously studied synthesizers [22, 52, 45, 58, 23, 30, 41, 11], the OP-1 contains several synthesis engines, effects and Low-Frequency-Oscillators, which make the parameter search space larger and more complex. Another particular challenge is that, for a given OP-1 preset, the generated sound can be slightly different. It makes the OP-1 not fully deterministic.

1.1 Evolutionary computation

Evolutionary computation is a subfield of artificial intelligence that involves continuous optimization and combinatorial optimization techniques. These techniques are often inspired by biological mechanisms of evolution. Evolutionary computation has been applied to sound synthesis parameters estimation for various synthesis technique ranging from additive synthesis [22] to frequency modulation (FM) [41, 23, 30] through physical modelling synthesis [45, 58]. Domain-specific knowledge has often been applied to reduce the complexity of the problem, by crafting problem-specific genetic operators or carefully selecting the genetic encoding to make the search space easier to explore. In the OP-1 synthesizer case, we have a limited knowledge about the mapping between the synthesizer's parameters and the embedded synthesis methods, FXs, LFOs or envelope. For example, in the envelope case, the ADSR is controlled by four knobs. We know that the first knob is mapped to the Attack, the second one to the Decay, the third one to the Sustain and the last one to the Release. However, we do not know how the knobs values are precisely mapped the ADSR. It is then not possible to analytically set the knob parameters using domain-specific knowledge. We considered the OP-1 synthesizer as a black box and focused on the application of existing and new evolutionary computation techniques to solve the parameter tuning problem.

Evolutionary techniques, such as genetic algorithms, have shown to present several advantages over more traditional optimization techniques. First, as evolutionary search is a heuristic search, solutions are found more quickly than purely random searches but also more efficiently than brute force search [54]. As a population-based search, evolutionary search is less likely exposed to being caught within local optima than other iterative search such as hill climbing, without the need for domain specific knowledge [54].

However, despite these qualities, evolutionary computation techniques can still struggle to solve specific problems: for example, problems of large complexity and/or presenting a large number of high fitness optima (also known as niches) [54]. Indeed, it can be challenging to balance the exploration phase (phase when the algorithm explores the problem space) and the exploitation phase (phase when the algorithm exploits profitable regions) to make the algorithm converge to the global optimum.

In this thesis, we explore recent evolutionary techniques able to identify several potential solutions instead of only one as for most of the previous systems [40]. This affords more flexibility to the user who receives a set of presets instead of only one presets. The user can then make the final choice. In order to achieve this goal, we develop a new multi-objective fitness function and include a cluster analysis of the final set of solutions in order to make it easy to handle for the synthesizer user.

1.2 Objectives and Contributions

The principal objectives which have directed this research are enumerated below.

1. To explore the potential for evolutionary computation as a mechanism for parameter estimation for a complex commercial synthesizer: the OP-1.

2. To assess and develop optimization algorithms suitable for optimizing multiple sets of OP-1 parameters that approximate given target sounds of a different nature.

3. To evaluate quantitatively and qualitatively the algorithmic performance in application to sound matching problems.

In satisfying the above objectives the following contribution to knowledge are included in this thesis:

- In Chapter 4, Section 4.3, a Non-dominated Sorting Genetic Algorithm-II (NSGA-II) is presented which incorporates a 3 objective fitness function, Gray code encoding and a modified crossover operator to preserve population diversity. It enables the users to receive a set of solutions rather than an unique solution as with previous systems.
- In Chapter 4, Section 4.3.3, a 3 objectives fitness function including FFT, Envelope and SFFT is developed which addresses some of the difficulties associated with the exploration of a multi-modal search space such as the OP-1 parameters space.
- In Chapter 4, Section 4.3.6, a clustering method has been developed to better analyze and explore the set of final solutions. This method is based on k-mean clustering and the silhouette methodology to set the clustering size.
- In Chapter 5, an evaluation is proposed using contrived target sounds (sounds produced by the OP-1) and non-contrived target sounds (sounds produced by other means than the OP-1).

1.3 Project history and methodology

1.3.1 Pure data patches evolution

The project started with Pr Pasquier's idea of using Genetic Programming (GP) to generate Pure Data (Pd) patches in order to match a given target sound. When evolving Pure Data Patches, one is not only searching for the synthesis parameters but also for the synthesizer architecture, i.e. the way the synthesis components (oscillators, LFO, filters, etc) are linked together. The complexity of the search becomes then significantly larger than for problems where only the synthesis parameters are searched, Our first idea was to use a canonical GP with large populations over a great number of generations in the attempt to handle the complexity of the problem. Noemie Perona from the engineering school ENSPS (Strasbourg, France) and Denis Lebel from McGill University (Montreal, Canada) implemented the first prototype of our system that was running only in serial. In order to get enough computation power to solve our problem, the author, assisted by Nicolas Gonzalez Thomas (SIAT/SFU), improved the GP algorithm and focused on distributing it on a supercomputer cluster. Unfortunately, the canonical GP did not give satisfactory solutions even with large populations and a great number of generations; however, it demonstrated the difficult and challenging nature of the problem.

1.3.2 ModFM synthesis

The Pd project was put aside. We moved to a more manageable search space, that of Modified Frequency Modulation (ModFM) is a distortion technique derived from the classic Frequency Modulation (FM) technique that has recently shown potential in multiple applications. We focused on a constrained ModFM model whose parameter are the modulation indices and modulation frequencies for each carrier. We completely automatized the calibration of this model for the reconstruction of harmonic instrument tones using a GA [33]. In this algorithm, we refine parameters and operators such as crossover probability or mutation operator for closer match. As an evaluation, we showed that our GA system automatically generates harmonic musical instrument sounds closely matching the target recordings. Some results can be found in Chapter 3, Section 3.2.6.

1.3.3 OP-1 synthesizer

Compared to the previous ModFM model, the OP-1 synthesizer presented new challenges for us. The OP-1 contains several synthesis engines, effects and Low-Frequency-Oscillators, which make the parameter search space larger and more complex, but also discontinuous. Another particular challenge is that, for a given OP-1 presets, the generated sound can be slightly different. It makes the OP-1 not fully deterministic. Proof and measures of the non-determinism of the OP-1 are given in Chapter 5, Section 5.1.3.

Building on the knowledge we gained from working on the ModFM synthesis problem, we explored how GA could be used to find OP-1 presets to match given target sounds. In this perspective, Laurent Droguet from IRCAM (France) implemented a classic GA, similar to the one we used for the ModFM synthesis model. His results showed that a classic GA was not able to identify OP-1 presets that can successfully approximate given target sounds. A classic GA didn't seem able to handle the complexity of the problem. This thesis describes in detail the challenges raised when searching the OP-1 parameter space and the solutions we developed to overcome them.

1.4 Thesis structure

Chapter 2 provides a review on evolutionary computation especially focused on GA and a variety of GA extensions and alternatives which are intended to enhance performances within difficult, multimodal search domains. Their applications to various sound matching synthesis problems are discussed in Chapter 3. In Chapter 4, we describe the specific problem we are trying to solve in this research. We give a description of the OP-1 synthesizer and point out the challenges raised when searching presets to approximate a given target sound with the OP-1. Our methodology to overcome these challenges as well as our proposed solution and its implementation are also discussed in Chapter 4. Experiments involving contrived and non-contrived target sounds are presented in Chapter 5 as a validation of our system. Chapter 6 concludes with a discussion of the benefits of our approach and outlines future work.

Chapter 2

Background: Evolutionary Computation

Although the work presented throughout the later chapters of this thesis is largely built upon the theoretical framework of the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [15], it is important to first consider the general nature of parameter optimization with Evolutionary Computation. This section provides an introduction to Genetic Algorithms (GAs) and NSGA-II, concluding with a brief summary of their similarities and differences.

2.1 Canonical Genetic Algorithm

GA was originally developed by Holland [20] to study and model the mechanisms of natural adaptive systems. Later, De Jong [28] set out the framework for the application of Holland's adaptive model to the problem of parameter (function) optimization. This application formed the precursor to a plethora of GA-based optimizers designed to improve performance when applied to new and specialized problem domains. GA was recognized as a useful tool for search and optimization problems [54]. This section will provide only a brief summary of the canonical GA.

2.1.1 Fitness function

In this section, we consider an optimization problem P on the search space S. S is the space of all the feasible solutions to P. Each and every point in S represents one possible solution.

Therefore, each possible solution x can be *marked* by its fitness value. This fitness value emphasizes how well the solution satisfies the optimization objectives. A fitness function f models the optimization problem P and assigns to each points in S a positive fitness value (see Equation (2.1)). In order to find (or to approximate) the best solution, this function has to be minimized (or maximized, depending on the function definition).

The fitness function is a critical element in the success of GA. This function, not only, sets the optimization objectives but also induces the shape of the fitness landscape on which the GA is searching.

$$f(x) : S \to \mathbb{R}^+ \tag{2.1}$$

2.1.2 Representation

The canonical GA models the candidate solutions on S using a Boolean representation. In analogy to natural sciences, the induced search space is called ϕ_g, the genotypic search space and S is called the phenotypic search space ϕ_p. A genotype-phenotype mapping d exists between these two spaces (Equation (2.2)).

$$d(x^g) : \phi_g \to \phi_p \tag{2.2}$$

The choice of binary-coded representation is inspired by the way in which biological structures are encoded into the low cardinality alphabet of DNA. Within the GA architecture, individuals in ϕ_g are constructed from a single bitstring (also called *chromosome*) which is divided into segments (*genes*) mapped to each element of an individual in ϕ_p. The genotypic search space ϕ_g is denoted by $\phi_g = \{0,1\}^l$, where l is the length of a binary vector $x_g = (x_{g_1}, \ldots, x_{g_l}) \in \{0,1\}^l$. For example, if $\phi_p = \mathbb{N}^n$, the commonly used genotype-phenotype function mapping d between ϕ_g and ϕ_p is the binary decoding/encoding of natural numbers illustrated in Equation (2.3). For instance, an individual with the genotype 11 would have the phenotype 3 with this particular mapping.

$$d(x^g) : \{0,1\}^l \to \mathbb{N}^n \tag{2.3}$$

2.1.3 Genetic operators

GA considers a population of individuals of size N_{pop} belonging to ϕ_g. The genetic operators are operators used to initialize this population, to select individuals for recombination and/or mutation, and to stop the evolution.

Initialization

The first population is built by randomly generating bitstrings. In order to speed up the optimization, it is possible to seed the population with promising individuals if some are known. The fitness value of each individual is then evaluated using the fitness function f.

Selection

A selection operation is used to select the individual to be recombined and mutated. In the canonical GA, selection is facilitated probabilistically using the so-called *roulette wheel* selection mechanism. Each individual is represented by a sector of a wheel, sized in proportion to its fitness. A spin of the wheel yields a mating candidate, which is copied into a temporary mating pool in preparation for variation by recombination and mutation.

Recombination

The recombination operator is called crossover, and provides the primary source of variation within a GA. The most basic GA recombination technique is known as single point crossover, which operates by simply concatenating the first part of one parent string with the second part of another; where the crossover point is chosen at random. Crossover is responsible for combining useful segments from the gene pool to form fitter solutions. p_{rec} is the crossover rate: the proportion of the population to be recombined.

Mutation

The mutation operator is widely considered to be the background source of variation, The bit flip mutation operator, that is used in the canonical GA, randomly inverts bits at a low probability p_{flip} (usually around 1% per bit). This mutation operator is applied to each of the recombined individuals. It makes possible to perform some local search but also to explore other locations in the search space that it would be possible only with the recombination operator.

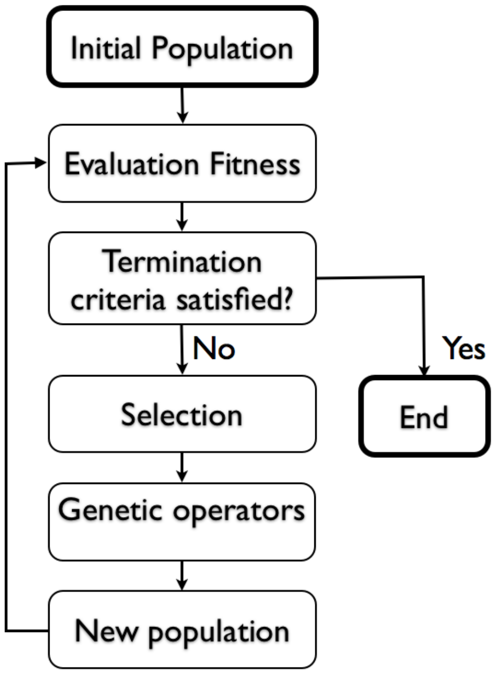


Figure 2.1: Canonical GA

2.1.4 Termination criteria

In the canonical GA, the evolution stops after a fixed number of generations or if a optimal fitness value, fixed before the evolution, is reached.

2.1.5 Algorithm

Figure 2.1 illustrates the GA steps. First, the population is initialized and evaluated. The algorithm checks if the termination criteria is satisfied. If it is satisfied, the algorithm stops and the best individual is returned. If it is not satisfied, some individuals are selected for crossover and mutation to form a new population. This new population is then evaluated and the algorithm checks if the termination criteria is verified. This process is repeated until the termination criteria is satisfied.

GA parameter and Exploration/Exploitation trade-off

Table 2.1 shows the GA parameters. These parameters directly affect the performance of the GA and the trade-off between *exploration* and *exploitation*. *Exploration* is the creation

Table 2.1: GA parameters

Parameter	**Meaning**
N_{pop}	size of the population
p_{rec}	proportion of recombinations
p_{flip}	probability to flip a bit when mutating
N_{gen}	number of generations before stopping

of population diversity by exploring the search space. *Exploitation* is the reduction of the diversity by focusing on the individuals of higher fitness, in other words, exploiting the fitness information represented within the population. Diversity in GA is a general concept that looks at how much of the search space is represented in the population. Diversity can be indirectly measured, for example, by the phenotype or genotype diversity or the standard deviation of fitness in the population. *Exploration* and *exploitation* are strongly related: an increase in exploitation decreases the diversity of the population and therefore the potential for exploration. In other words, strong exploitation can lead to the premature convergence of the genetic search as it can turn the focus of the search to local optima. But, at the same time, exploitation is necessary because it makes it possible to fine tune solutions when they are near optimum. On the other side, overly focused exploration can make the genetic search ineffective and inefficient as it would randomize the search and would limit the improvement of individuals of high fitness. A judicious choice of the GA parameters is the key to balance exploration and exploitation [32].

2.2 Theory

2.2.1 Fitness landscape and problem difficulty

One of the central elements in GA is the fitness function introduced in Section 4.3.3. It directly influences the shape of the fitness landscape and the convergence of the algorithm to an optimal solution [29].

What is a fitness landscape? Consider a minimization problem and imagine the space of all possible individuals that can be generated by a particular GA system applied to a particular problem. Each individual is associated with a fitness, a real number which reflects how well this individual solves the problem. Then, imagine that this space of all possible individuals is mapped into the x-y plane. Finally, imagine that the fitness of each one of these

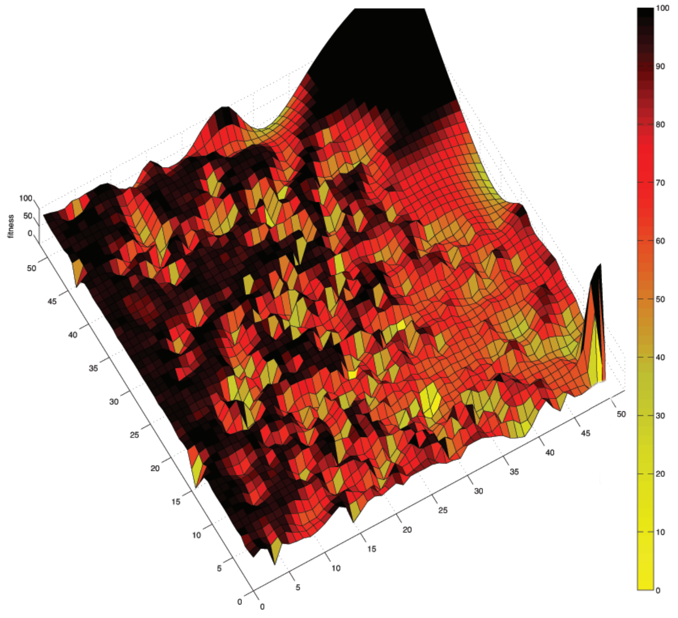


Figure 2.2: Example of a fitness landscape representation

individuals is plotted on the z-axis. This creates a surface where the peaks are the locations of individuals with poor fitness, and the basins show the locations of the individuals with good fitness. Figure 2.2 shows an example of a fitness landscape representation. Discovering the best solution to the problem then becomes equivalent to searching over this landscape for the deepest basin.

The ruggedness of the landscape has a direct bearing on the difficulty of the problem. Evolutionary techniques such as GA will have increasing difficulty in locating the deepest basins on landscapes that display greater ruggedness. It is perhaps easiest to visualize in the opposite case, that of a landscape with a single large basin, the bottom of which is the best solution. In this case, it is relatively easy for the adaptive processes to proceed directly to the bottom of the basin. In the opposite case, where the landscape is quite rugged, perhaps even locally or globally discontinuous, the individuals may get trapped on local minima without every finding the (or a) global minimum, or the population may simply wander aimlessly across the landscape, in which case the adaptive process has degenerated into random search [32].

Phenotype-Fitness Mappings and Problem Difficulty

Jones and Forrest [27] developed a classification of problem difficulty, which is based on the correlation analysis, for describing how the locality of a representation and the choice of the fitness function influences GA performance. The locality of a representation describes how well genotypic neighbours correspond to phenotypic neighbours. For example, in a high-locality representation, genotypic neighbours would remain neighbours in the phenotype space. Contrary to low-locality representation, a high-locality representation preserve the distances. Jones and Forrest showed that the difficulty of an optimization problem can be measured by the correlation coefficient ρ_{FDC} between the fitness distances and the genotypic distances. ρ_{FDC} measures the correlation between the fitnesses of search points and their distances to the global optimum on the genotype space.

Problems are easy for a GA if there is a positive correlation between an individuals distance to the optimal solution and the difference between its fitness and the fitness of the optimal solution.

Problems become much more difficult if there is no correlation between the fitness difference and the distance to the optimal solution. Then, the fitness landscape does not guide a mutation-based search method to the optimal solution. No search heuristics can use information about a problem which was collected in prior search steps to determine the next search step. Therefore, all reasonable search algorithms show the same performance as no useful information (information that indicates where the optimal solution can be found) is available in the problem. Because all search strategies are equivalent, random search is also an appropriate search method for such problems. Random search uses no information and performs well on these types of problems.

Genotype-Phenotype Mappings and Problem Difficulty

When using binary representations the genotype-phenotype mapping d depends on the specific optimization problem that should be solved. The most common encodings for integer are the binary, Gray, and unary encoding [49]. These mappings are critical for the GA success as they directly influence the difficulty of the problem. Table 2.2 shows examples of binary, Gray and unary encoding for integers from 0 to 7. Gray code and unary code have a higher locality than the binary code as only one bit has to be changed when encoding one integer and its following.

Table 2.2: Common integer encoding

Integer	Binary	Gray	Unary
0	000	000	0000000
1	001	001	0000001, 0000010, . . . , 0100000, 1000000
2	010	011	0000011, 0000101, . . . , 1010000, 1100000
3	011	010	0000111, 0001011, . . . , 1101000, 1110000
4	100	110	0001111, 0010111, . . . , 1110100, 1111000
5	101	111	0011111, 0101111, . . . , 1111010, 1111100
6	110	101	0111111, 1011111, . . . , 1111101, 1111110
7	111	100	1111111

Choosing a non-suitable genotype-phenotype mapping can make a problem more difficult to solve. Using a suitable mapping can preserve the problem difficulty induced by the phenotype-fitness mapping f. It can even make the problem easier if some domain knowledge is included in this mapping. For example, take the optimization problem of tuning a FM synthesizer to match harmonic instrument sounds. A way of producing harmonic spectra with FM synthesis is to set the modulation frequencies as multiples of the carrier frequencies. The naive way of encoding the solution would be to have one gene coding for the modulation frequency value and another gene coding for the carrier frequency value. An encoding that would take advantage of domain knowledge would be encoding the carrier frequency as the multiplication factor instead of the carrier frequency value. It would reduce the problem complexity and make the problem more manageable.

2.2.2 Parallelization / Distribution

GA is a population-based technique that requires the evaluation of each generation of individuals. These evaluations are independent and can be performed in parallel to speed up the execution of the GA. This section presents some theory about master-slave parallel GA, based on the works of Cantu [13], and compares the expected improvements to the ones obtained in practice with our implementation.

Theory

One way to implement GAs on parallel computers is to distribute the evaluation of fitness among several slave nodes while one master executes the genetic operators (selection, crossover, and mutation). Master-slave parallel GAs are important for several reasons:

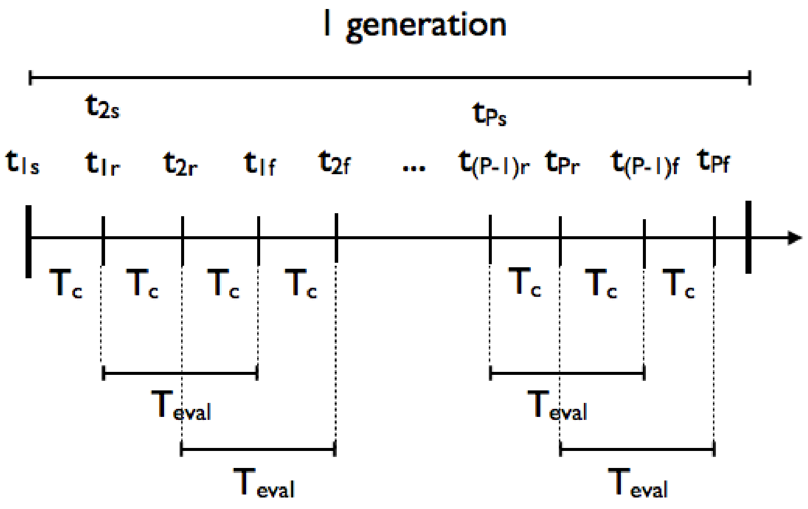


Figure 2.3: One generation in a master-slave parallel GA when the master evaluates a fraction of the population

1. They explore the search space in exactly the same manner as serial GAs, and therefore the parameter settings used for simple GAs are directly applicable;
2. They are relatively easy to implement, which makes them popular with practitioners;
3. In most cases, master-slave GAs result in significant speed-up of the algorithm execution.

The only parameter introduced by this mode of parallelization is the number of nodes. Although at first it may seem best to use all available nodes, the communication time might increase to the point of obliterating the gains obtained by distributing the computing load between the nodes. Focusing our attention on the master node, Figure 2.3 depicts the sequence of events in every generation. First, the master sends a fraction of the population to each of the slaves to evaluate ($t_{is}, \forall i \in [1, P]$), using time T_c to communicate with each. Next, the master evaluates a fraction of the population using time $T_{eval} = \frac{nT_f}{P}$, where T_f is the time required to evaluate one individual, n is the size of the population, and P is the number of nodes or node used. The slaves start evaluating their portion of the population as soon as they receive it ($t_{ir}, \forall i \in [1, P]$), and return the evaluations to the master as soon as they finish ($t_{if}, \forall i \in [1, P]$). We ignore the time consumed by selection, crossover, and mutation because it is usually much shorter than the time used to evaluate and to communicate individuals. In addition, we assume that the same number of individuals are

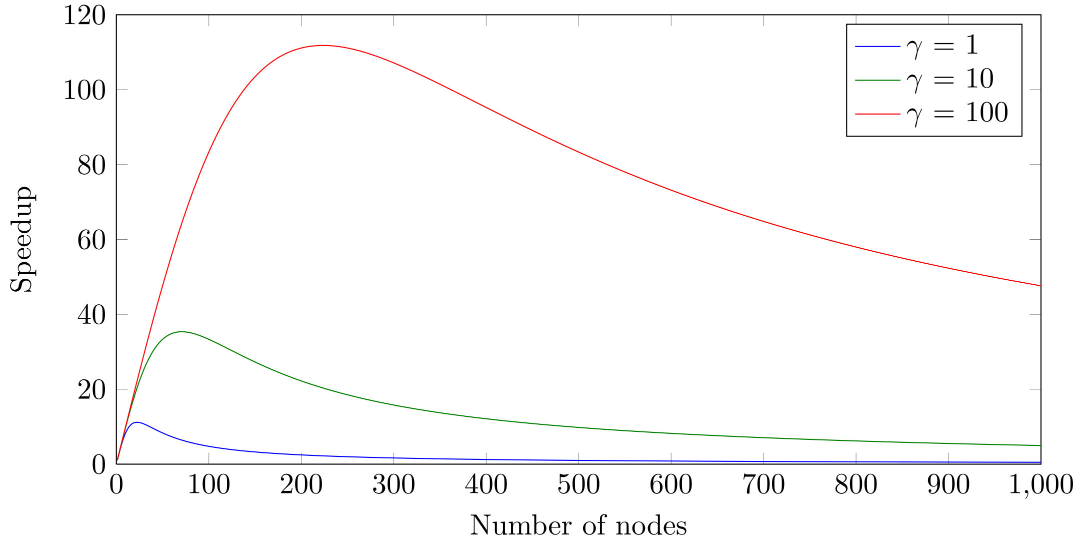


Figure 2.4: Theoretical speedups of a master-slave GA varying the value of γ.

assigned to each slave and that evaluation time is the same for all individuals. With these assumptions, the elapsed time for one generation of the parallel GA may be estimated as the sum of elapsed time spent in computations and time used to communicate information among the nodes:

$$T_p = PT_c + \frac{nT_f}{P} \tag{2.4}$$

As more slaves are used, the computation time decreases as desired, but the communication time increases. The parallel speedup is defined as:

$$\frac{T_s}{T_p} = \frac{nT_f}{\frac{nT_f}{P} + PT_c} = \frac{n\gamma}{\frac{n\gamma}{P} + P} \tag{2.5}$$

T_s is the elapsed time for evaluating one generation of the serial GA. γ is the ratio $\frac{T_f}{T_c}$.

Figure 2.4 shows the theoretical speedups of a master-slave GA varying the ratio $\gamma = \frac{T_f}{T_c}$. As γ increases, more nodes may be used effectively to reduce the execution time. The figure considers a master-slave GA with a population of $n = 500$ individuals, and the speedups are plotted for $\gamma = 1, 10, 100$. In many practical problems, the function evaluation time is much greater than the time of communications, $T_f \gg T_c$ ($\gamma \gg 1$), and master-slave parallel GAs can deliver near-linear speedups for a large range of nodes.

In practice

During our evaluations, we kept track of the evaluation time Tf. For a 1 second long sound, the evaluation time Tf is in average $\mu = 30.4ms$ with standard deviation $SD = 0.1ms$. The communication time T_c was more challenging to measure in practice. T_c varies in function of the size of the population fraction that is sent to the slaves. For example, if you use 2 slaves rather than 6 slaves, the population fraction would be bigger and therefore T_c longer. In this section, we will consider the configuration we use later in our experiments, i.e where the evaluation of 500 individuals is distributed on 100 processors. In this case, the fraction size is 5 individuals. We ran a benchmark to estimate T_c in this particular case. We measure than Tc is in average $\mu = 199.4\mu s$ with standard deviation $SD = 9.2\mu s$.

If we use the average values of Tc and Tf, we can calculate $\gamma = 1525$. Using Equation (2.5), we get a theoretical speedup $\frac{T_s}{T_p} = 98.7$.

However, in practice, we only observe a speedup equals to 43. We also ran a benchmark using our algorithm with a population of 500 individuals distributed on different number of processors. Figure 2.5 shows the speedups we get in practice compared to the theoretical speedups for $\gamma = 1, 10, 100$. Our results in practice seems to indicate a γ around 10.

Numerous factors can limit the speedup in practice. First, contrary to the hypothesis we made in theory, T_c varies in function of the number of processors that is used. Then, in our system, the time to variate the population is relatively long ($\mu = 24.0ms, SD = 4.17ms$) compared to with the canonical GA. We also gather a lot of statistics during the evolution, that can also have an impact on the speedup in practice.

2.2.3 Limitations / Difficulties

Theory vs Practice

In practice, it is not possible to evolve a population of infinite size over an infinite number of generations. These limitations lead to phenomena that can slow down the optimization and even mislead the evolution to a local minima. Because of the selection pressure and successive recombinations, the population tends to become uniform: the diversity decreases. Some useful genetic material can also be lost. This phenomenon is also called *genetic drift*.

These two factors, *loss of diversity* and *genetic drift* can lead to a *premature convergence*, i.e a convergence to a local minimum and not to a global minimum.

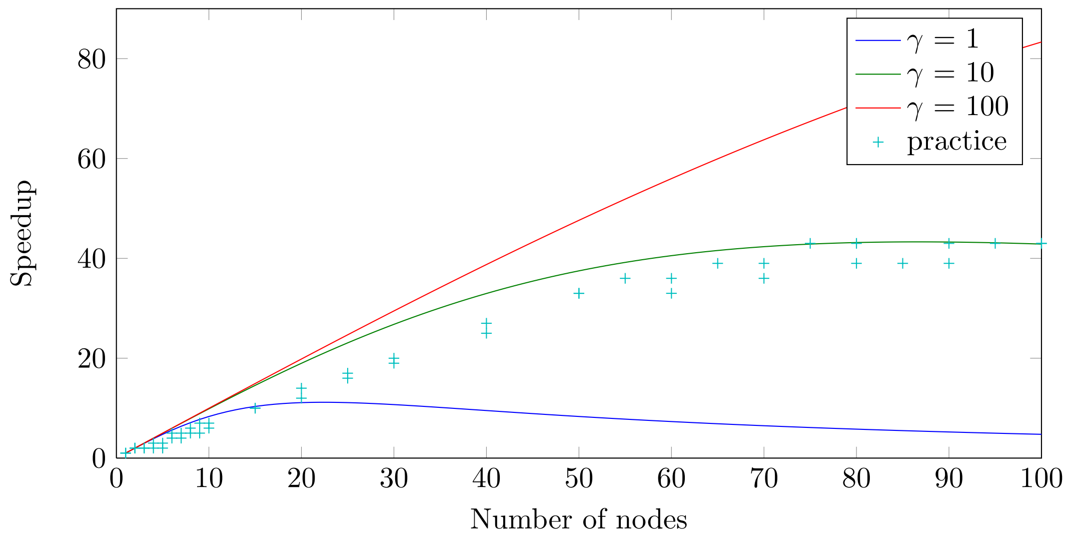


Figure 2.5: Theoretical and practical speedups of a master-slave GA

No Free Lunch Theorem

The No Free Lunch Theorem appears in Wolpert and Macready's work [59] and states that any two optimization algorithms are equivalent when their performance is averaged across all possible problems. This theorem can have important implications for GA development in practice. Indeed, most of the time, applying the canonical GA to solve a given problem does not give good performances right away. It is often necessary to design problem-specific genetic operators, use a relevant genetic representation or fine tune the set of GA parameters to make the GA efficient in practice.

2.3 GA extensions and alternatives

A large number of variants and enhancements to the canonical GA have been proposed to improve its performances for different problem types. In this section, we will focus on the variants we use in our system (see Chapter 4, Section 4.3).

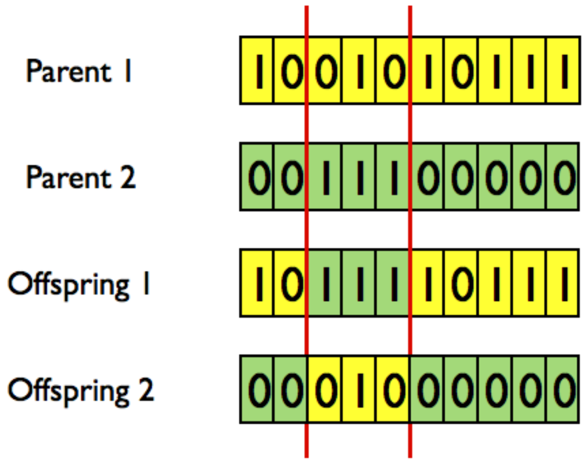


Figure 2.6: 2-point crossover

2.3.1 Elitism

A very successful variant of the general process of constructing a new population is to allow some of the best individuals from the current generation to carry over to the next, unaltered [54].

2.3.2 Two points crossover

Instead of considering only one point for recombination as in the canonical GA, we consider two points. These two points are randomly chosen. This crossover operator makes it possible to exchange blocks of variable sizes between two parents. Figure 2.8 illustrates its functioning.

2.3.3 Tournament selection

An alternative to the roulette-wheel selection operation is the tournament selection. In a tournament selection of size n, n individuals are picked up randomly from the population. Among these n individuals, the individuals with the best fitness value is selected. Setting the tournament size n makes it possible to adjust the selection pressure. The higher n is, the higher the selection pressure is and the more elitist the selection is. A common value for n is 2, in that case, the selection operator is called binary tournament selection.

2.3.4 Gray code

The binary encoding has problems associated with the Hamming cliff [51]. The Hamming cliff describes the effect that some neighbouring phenotypes (the phenotypes that have a distance of one) are represented by completely different binary representations. In order to deal with this limitation, Gray coding has been used instead of binary encoding in GA [7]. The Gray code is based on the idea that two successive values differ by only one bit. Using Gray code has shown to change the problem difficulty for the GA. Therefore, some problems are becoming easier to solve and other more difficult to solve [7]. So far, no method makes it possible to know in advance which representation to adopt to make a given optimization problem the easiest to solve with a GA.

2.3.5 Crowding GA

Crowding was introduced by De Jong [28] as a technique for preserving population diversity and preventing premature convergence. Crowding is applied in the survival selection step of GAs in order to decide which individuals among those in the current population and their offspring will pass to the next generation. Crowding consists of two main phases: pairing and replacement. In the pairing phase, offspring individuals are paired with individuals in the current population according to a similarity metric. In the replacement phase, a decision is made for each pair of individuals as to which of them will remain in the population. A review of crowding approaches for GAs can be found in [39]. The original crowding scheme developed by De Jong [28] consists of randomly selecting, for each offspring, γ individuals from the current population. The offspring, if it has a better fitness, will replace the most similar individual among these γ selected individuals. Parameter γ is known as *crowding factor*, and usually $\gamma = 2$ is used.

2.4 A multi-objective Genetic Algorithm: NSGA-II

Contrary to the canonical GA where a single optimal solution is considered, multi-objective GAs aim to solve optimization problems with multiple objectives and considered a set of optimal solutions. This set of optimal solutions is largely known as *Pareto-optimal* solutions. A solution is said *Pareto-optimal* if none of its objective functions can be improved in value without impairment in some of the other objective values. A solution is *dominated* if some other solution is better for one or more objectives without being worse for the remaining objectives. The *Pareto front* is the set of solutions that are *Pareto-optimal*.

Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [15] has become one of the standard approach to solve multi-objective problems. It embeds most of the GA extensions defined in Section 2.3:

- Non-dominated sorting,
- Diversity preservation,
- Elitism.

In the next subsections, we will define the notion of non-dominated sorting and crowding distance and show how they are used in the main loop of the NSGA-II to converge toward an optimized Pareto front.

2.4.1 A non-dominated sorting approach

The goal is to sort the population in non-dominated fronts. We consider a population of size N and M objectives. In order to identify solutions of the first non-dominated front in the population, each solution can be compared with every other solution in the population to find if it is dominated. At this stage, all individuals in the first non-dominated front are found. In order to find the individuals in the next non-dominated front, the solutions of the first front are discounted temporarily and the above procedure is repeated. The same procedure is used for finding third and higher levels of non-domination. Thus, the worst case is when there are fronts and there exists only one solution in each front. This requires an overall $\mathcal{O}(MN^3)$ computations. The *non-domination level* for a given individual is the rank of the front it belongs to. NSGA-II uses a fast non-dominated sorting approach that requires only $\mathcal{O}(MN^2)$ comparisons [15].

2.4.2 Diversity preservation

Along with convergence to the *Pareto-optimal* set, it is also desired that the GA maintains a good spread of solutions in the obtained set of solutions. NSGA-II assigns to each individual a *crowding distance*. Consider that each individual is plotted as a point in the objectives graph. The *crowding distance* is the average distance of two nearest points on either side

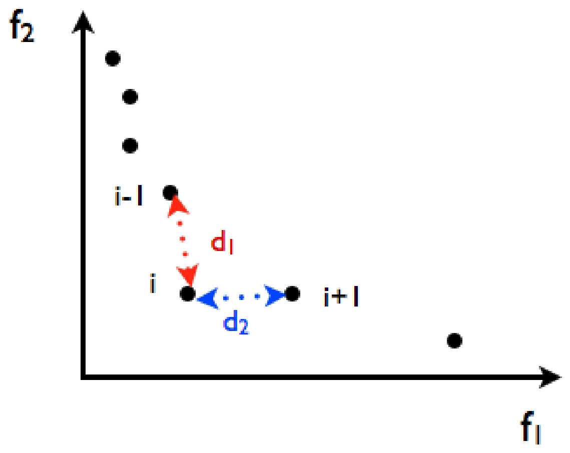


Figure 2.7: Crowding-distance calculation.

of this point along each of the objectives. For example, Figure 2.7 illustrates how these distances are measured. The *crowding distance* for the solution i would be the average of d_1 and d_2.

The crowding-distance computation requires sorting the population according to each objective function value in ascending order of magnitude. Thereafter, for each objective function, the boundary solutions (solutions with smallest and largest function values) are assigned an infinite distance value. This way, the boundary solutions are more likely to be kept in the next generation (see following section for more details). All other intermediate solutions are assigned a distance value equal to the absolute normalized difference in the function values of two adjacent solutions. This calculation is continued with the other objective functions. The overall crowding-distance value is calculated as the sum of individual distance values corresponding to each objective. Each objective function is normalized before calculating the crowding distance.

Equation (2.6) summarizes the calculus of the crowding distance for a solution x.

$$d(x) = \sum_{i=1}^{N_{obj}} |adj_1(i,x) - adj_2(i,x)| \tag{2.6}$$

where N_{obj} is the number of objectives, $adj_1(i,x)$ (resp. $adj_2(i,x)$) gives the normalized objective value of the first (resp. second) adjacent solution when sorted in ascending order of magnitude for the objective i.

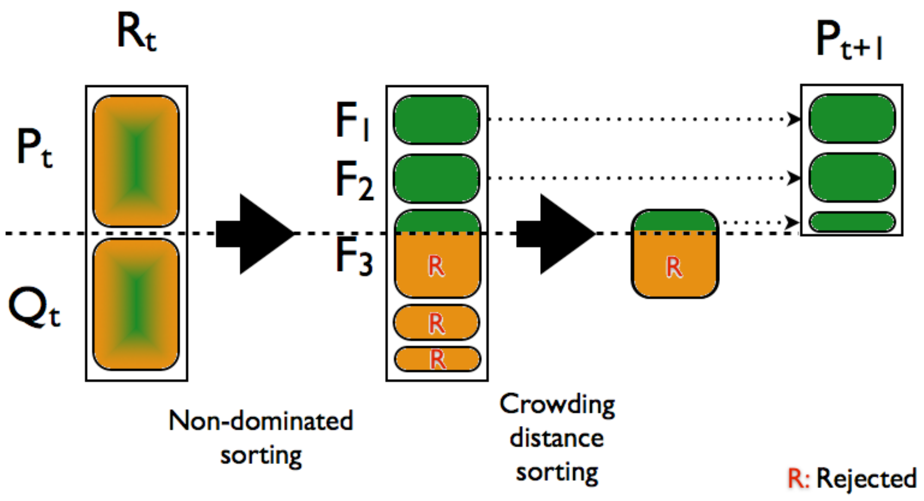


Figure 2.8: NSGA-II procedure

2.4.3 Main loop

First, a population P_0 of size n_{pop} is initialized using the same initialization operator than for the canonical GA in Section 2.1.3. The fitness of each individual is then evaluated using the fitness function. The population is sorted based on non-domination and a rank, equal to its *non-domination level*, is assigned to each individual. Using the algorithm described in Section 2.4.2, a *crowding distance* is also assigned to each individual.

Contrary to the canonical GA, the selection operator used in NSGA-II is a tournament selection operator based on dominance between two individuals. If the two individuals do not inter-dominate the selection is made based on crowding distance. The same recombination and mutation operators than for the canonical GA, described in Section 2.1.3, are used to create an offspring population Q_0 of size n_{pop}. A combined population $R_0 = P_0 \cup Q_0$ is considered. This population R_0 is sorted according to non-domination. Since all previous and current population individuals are included in R_0, *elitism* is ensured. Now, solutions belonging to the best non-dominated set F_1 are of best solutions in the combined population and must be emphasized more than any other solution in the combined population. If the size of F_1 is smaller than n_{pop}, we definitely choose all members of the set for the new population P_1. The remaining members of the population P_1 are chosen from subsequent non-dominated fronts in the order of their ranking. This procedure is continued until no more sets can be accommodated. Say that the set F_l is the last non-dominated set beyond which no other set can be accommodated. In general, the count of solutions in all sets from F_1 to F_I would be larger than the population size n_{pop}. To choose exactly n_{pop} population members, we sort the solutions of the last front by *crowding distance* in increasing order and choose the best solutions needed to fill all population slots. The NSGA-II procedure is shown in Figure 2.8.

Chapter 3

Background: Evolutionary computation for sound synthesis

3.1 Problem description

Replicating the sounds of musical instruments using parametric synthesis techniques is a problem frequently addressed in the field of computer music [21]. The success of any particular synthesis algorithm, is dependent, in part, on the selection of suitable controls and synthesis parameters. Manually estimating parameters for a particular synthesis algorithm can be difficult and time consuming, especially if there is no intuitive relationship between the parameter values and the produced sound. Thus, diverse optimization methods have been used for automatic calibration, such as Particle Swarm [19], HMM [61], Neural Nets [48], Cellular Automata [53] and Genetic Algorithms [21].

It has been suggested that Genetic Algorithms (GA) are well suited to matching musical instrument tones [48]. Indeed, GA have been used extensively for estimating the parameters of various synthesis techniques [22, 48, 17, 23, 33] and more complex synthesizers [60]. Modern synthesizers are now embedding several engines, LFO and FX, often using ad hoc algorithms whose properties are not fully known to the user. For this reason - coupled with the increase of many synthesizers' complexity, it becomes more and more challenging for the user to master all their functionalities. With this rising complexity, automatic synthesizer calibration could help the user in this process. Most of the previous works have been on particular synthesis techniques [22, 52, 48, 17, 25, 58, 45, 30, 11, 33, 23] but few have been applied to real world synthesizers [60]. Trying to solve this problem for the OP-1, a commercial synthesizer, is a first attempt in this direction.

In this chapter, we describe some related works and classify the GA systems in 3 groups given their complexity:

1. Parameter search within a synthesis technique,

2. Parameter search within a multi-engine synthesizer,

3. Automatized synthesis system design and parameter search.

Within these groups, the complexity of the problem to be solved can vary tremendously given the number and the nature of the parameters to be searched. A distinction can also be made between some systems aiming to mimic only harmonic instrument sounds [22, 23, 33, 58] and other aiming to mimic any instrument sounds or any sounds [17, 25]. For all these reasons, it is challenging to compare these systems and to capitalize on them to solve our specific problem. In the next section, we describe a selection of GA systems for each of these 3 groups. At the end of this chapter, Table 3.1, 3.2 and 3.3 summarize and compare most of these systems.

3.2 GA applied to sound synthesis techniques

GAs, described in Chapter 2, have been successfully used to optimize the parameters of numerous known synthesis methods for a given target sound or simply for exploration. In this section, we present their applications on some of the most common sound synthesis methods [21].

3.2.1 Additive synthesis

One of the very first sound synthesis techniques introduced was additive synthesis [8]. It is based on Fourier's theorem which states that any sound can be constructed from elementary sinusoids. Additive synthesis attempts to apply this theorem to the synthesis of sound by adding a series of sinusoidal oscillators, each having independent amplitude and frequency controls (see Figure 3.1). A short-time Fourier transform of a musical tone gives the amplitude and frequency envelopes over time to match.

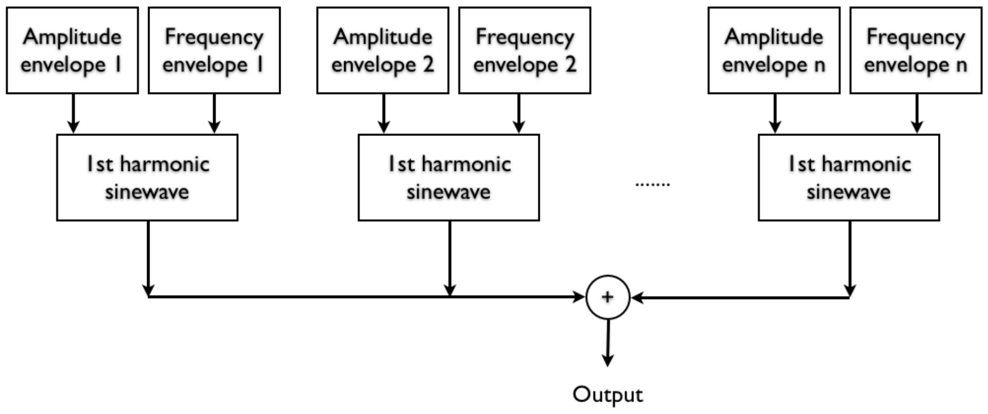


Figure 3.1: Additive synthesis block diagram

This method is one of the most straightforward and powerful synthesis methods and makes it possible to reconstruct the original waveform exactly. However, it needs as many oscillators as the number of harmonics in the spectrum and, thus, necessitates to implement a good configuration for each of these oscillators. To make it computationally less expensive, data reduction is most of the time necessary most of the time.

Piecewise-linear approximation of additive synthesis amplitude and frequency envelopes is one of the most common data reduction technique used [22]. For each envelope, a series of line segments are connected at breakpoints. Finding the optimum breakpoints is not easy because it is a non-linear problem. Indeed, moving one breakpoint changes how well the approximation matches the original envelope over the length of the neighbouring segments.

Horner et al. [22] used a GA system to solve this breakpoint approximation problem. They defined the fitness function as the relative error measure for the harmonics over the frames. They used the binary tournament selection, a one-point crossover operation and a mutation operation with its mutation rate equals to the reciprocal of the number of bits in the encoding. The chromosome had a size equal to the number of breakpoints specified by the user and each gene represented the frame number for the given breakpoint.

Horner et al. [22] made a comparison study between GA and three other methods for choosing the breakpoint. The first equally spaced the breakpoints, the second randomly spaced them and the last one was using a Greedy algorithm instead of GA to solve the breakpoint approximation problem. GA outperformed the others methods and was the best for optimizing solutions for specific numbers of breakpoints, which makes it very appropriate

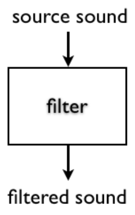


Figure 3.2: Subtractive synthesis block diagram

for hardware synthesis where only a fixed number of points are considered. However, for ten or less breakpoints, the Greedy methods performed as well as GA and faster, which makes it more suitable for software synthesis where speed is more important than having more extra breakpoints.

3.2.2 Subtractive synthesis

In subtractive synthesis, a basic waveform is created, rich in harmonics and is used as a source sound. The harmonic structure is then altered by removing certain harmonics using filters (see Figure 3.2). The main challenge in using this technique is to determine the good set of filters to sculpt the waveform.

Schatter et al. [52] used GA and fuzzy logic to generate substrative synthesis parameters for particular target sounds. They defined their fitness function as the Euclidian distance between spectrums weighted by psychoacoustic factors. They used fitness proportionate selection, a flip mutation operator and a two-points crossover. The chromosomes represented the 23 parameters of the subtractive method. Schatter et al. concluded that their sound-generator is able to approximate a target sound quite well but they pointed out that the quality of the result sounds depends on the sound generation technique as well as the kind of target sound. Complex, in particular impulsive, target sounds did not yield satisfying results.

Yee-King and Roth [48] conducted a study to compare the performance of four parametric optimization techniques in replicating sounds. A Hill-climber algorithm, Genetic Algorithm, Neural Networks and a data-driven approach were considered. In their study, they use a basic and a complex subtractive synthesizer. The basic subtractive synthesizer places sine, sawtooth, pulse and white noise oscillators before a low pass filter and provides three parameters: oscillator mix, filter cut off and resonance. The oscillator mix parameter is set up so that only two oscillators can be heard at a time with a varying mix. As the parameter sweeps through from 0 to 1, different oscillators are switched on and off and/or faded in. The complex subtractive synthesizer has two switchable mode oscillators plus a white noise generator passed through 3 parallel filters which are band, low and high pass. The sound is controlled using 17 parameters for oscillator waveform, oscillator mix and filter control. They designed their GA system with the standard roulette-wheel selection operator, mutation and crossover operators. They defined their fitness function as the Euclidian distance between the Mel Frequency Cepstrum Coefficients (MFCC) for the target sound and the candidate sound. The MFCC measure is an advanced spectral analysis indicator based on a perceptual scale: the mel scale. The results of their study showed that the GA, with the MFCC fitness function outperformed, all the other optimization techniques for every sound.

3.2.3 Granular synthesis

Granular synthesis [46, 56] is an advanced sound synthesis method that operates on the microsound time scale. A complex sound is build up by assembling small pieces of sound. These small pieces are called grains and are approximately 40 ms in length. The grain parameters include amplitude, frequency and duration. The main difficulty with this technique is to manage the parameters of thousands of grains at each instant.

Fuginaga et al. [17] explored the potential of GA for granular synthesis control. In their work, the individuals represent the sound grains and form a population that represents the output sound. They described the standard crossover, mutation and fitness proportionate selection operator. They did not define a precise fitness function but advised the user to design the fitness function to match her taste and explore the granular sound space. The fitness function still applied to the individuals and not to the population. Even though no precise implementation or evaluation was proposed, the authors claimed that their GA system is successful to explore the sound space.

Johnson et al. [25] built a interactive GA system to set the parameters of a granular synthesizer. They designed their system with the roulette wheel selection, crossover and mutation operator. Each individual represented the 7 parameters of a synthesizer configuration: amplitude, fundamental frequency, pitch range, length of grains, octaviation index, decay times and the formant frequency. The fitness of generated sounds was evaluated directly by the user that could interact with the system through the graphical user interface.

Johnson et al. found that it only took a small number of generations to make the population converge to the desired region in the sound space.

3.2.4 Physical modelling synthesis

While the synthesis methods so far presented simulate sounds by reproducing a signal's spectral content (analysis and then resynthesis), physical models are based on mathematical descriptions of acoustic systems themselves. However, these models contain a large number of parameters and can be non-linear. It makes these systems very hard to configure.

Vuori et al. [58] built a GA system for estimating the parameters of a non-linear physical flute model. Each chromosome of this system represented 8 different parameters of the physical model. They designed the fitness function as the Euclidian distance between the candidate sound spectrum and the target sound spectrum. They implemented a roulette-wheel selection, crossover and mutation operator. They showed that the algorithm converges smoothly and effectively to the desired sound.

Later, Riionheimo et al. [45] designed another GA system to control parameters for a plucked string synthesis model. Their system was similar to the Vuori's system except for the fitness function that is defined, here, by the least-squared error of the short-time spectrums. They concluded that the system had good performances that could be improved using a fitness function with perceptual error calculation.

3.2.5 FM synthesis

Frequency modulation synthesis (or FM synthesis) is a form of audio synthesis where the spectrum of a simple waveform is changed by modulating its frequency by one or more additional audio-rate oscillators, resulting in a more complex waveform. The frequency of an oscillator (carrier) is altered or distorted in accordance with the amplitude (modulating index) of a modulating signal (modulator). Dependent of the type of FM synthesis, several carriers and/or several modulators can be used. Figure 3.3 shows a block diagram for a one-carrier, one-modulator FM synthesis system.

Matching arbitrary musical instrument tones with this method is difficult because there is no analytical solution for determining the best set of FM parameters. The greater the complexity of the FM synthesis algorithm, the greater the difficulty in finding the right set of parameters by hand.

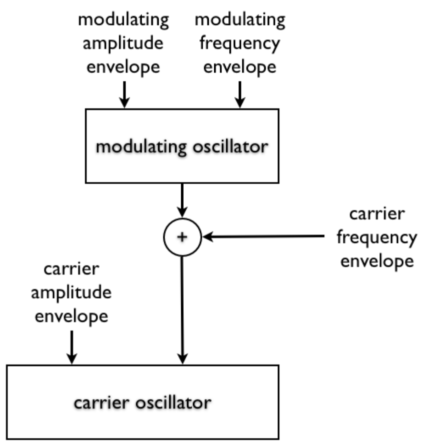

modulating amplitude envelope
modulating frequency envelope
modulating oscillator
+
carrier frequency envelope
carrier amplitude envelope
carrier oscillator

Figure 3.3: FM synthesis block diagram

To deal with this optimization problem, Lai et al. [30] built a GA system to automatize the optimization of the parameters for a single-modulator FM synthesizer for a given target sound. They used a binary tournament selection, 1-point crossover and binary mutation operators. To compute the fitness function, they extracted two spectral features from the target sound and the candidate sound and then calculated the mean sum between these two features:

- the spectral norm that is the Euclidian distance between the two spectra,
- the spectral centroid, that is the centre of gravity of the magnitude spectrum of the short time Fourier transform.

They concluded that GA outperformed other optimization methods in a limited number of generations. They also added that the synthesized sound presents a spectrum close to the target spectrum according to their spectral features but also sounds perceptually similar to the target sound.

Bozkurt and Yuksel [11] presented automatic parameter tuning experiments with genetic algorithms in application to multiple-modulator FM synthesis, where 3 sine oscillators are modulating a single carrier oscillator. Contrary to the FM synthesis systems previously presented [33, 23], no integer multiple relationships in carrier and modulator frequencies is implied in the design. Without this constraint, the search becomes more complex as the search space is not limited anymore to only sounds with harmonic spectrum. Mitchell and Pipe [43] applied a Evolutionary Strategy to a single carrier/modulator FM synthesis model that used a windowed discrete Fourier transform fitness function. The motivation was to achieve a smoothing effect on the fitness landscape. This eliminated a proportion of local optima and improved performance. More recently, Mitchell [41] also applied a Clustering Evolutionary Strategy to tune the parameters of single, double and triple FM synthesis models of similar complexity. He compared his system to Canonical Evolution Strategy [10] and Multi-start Evolution Strategy [34] and showed that his system performed better to approximate any instrument tones. It is also interesting to notice than Mitchell considered the problem as multimodal and therefore his system returned a set of solutions instead of only one with previous systems [40].

3.2.6 Modified FM synthesis

The ModFM technique [31] can be derived from ClassicFM by employing a purely imaginary index of modulation (see Figure 3.3). This small change leads to a different set of scaling functions for the FM spectrum. If FM synthesis is seen as a combination of sinusoids ring-modulated by real sinusoidal waveshaper signals, ModFM is then based on a sinusoid ring-modulated by a complex exponential waveshaper signal. It can be seen that the major differences are in ModFM's monotonically decreasing spectrum. Moreover, the absence of phase-reversed partials allows for a more predictable result when combining several ModFM carriers.

In a previous work [33], we adapted a GA system, initially developed for ClassicFM by Horner et al [23], to work with ModFM synthesis and harmonic instrument tones. Our system used a mixed-integer representation, a binary tournament selection, single point crossover and gaussian mutation. The fitness score for each individual was the accumulated approximation error for the 10 first harmonics. We performed an evaluation of our system with contrived and recorded sounds and showed that we successfully used GA to help find the right parameters using ModFM as synthesis technique to match a specific instrument tone. We also showed that our GA can be applied to ClassicFM without modifying the fitness function or the genetic parameters. Finally, this comparative study also contributed to highlight the differences and similarities between ModFM to FM in terms of parameter tuning. Figure 3.4 shows an example of harmonic approximations we obtained for a trumpet and a viola sound with 2, 4 and 6 carriers.

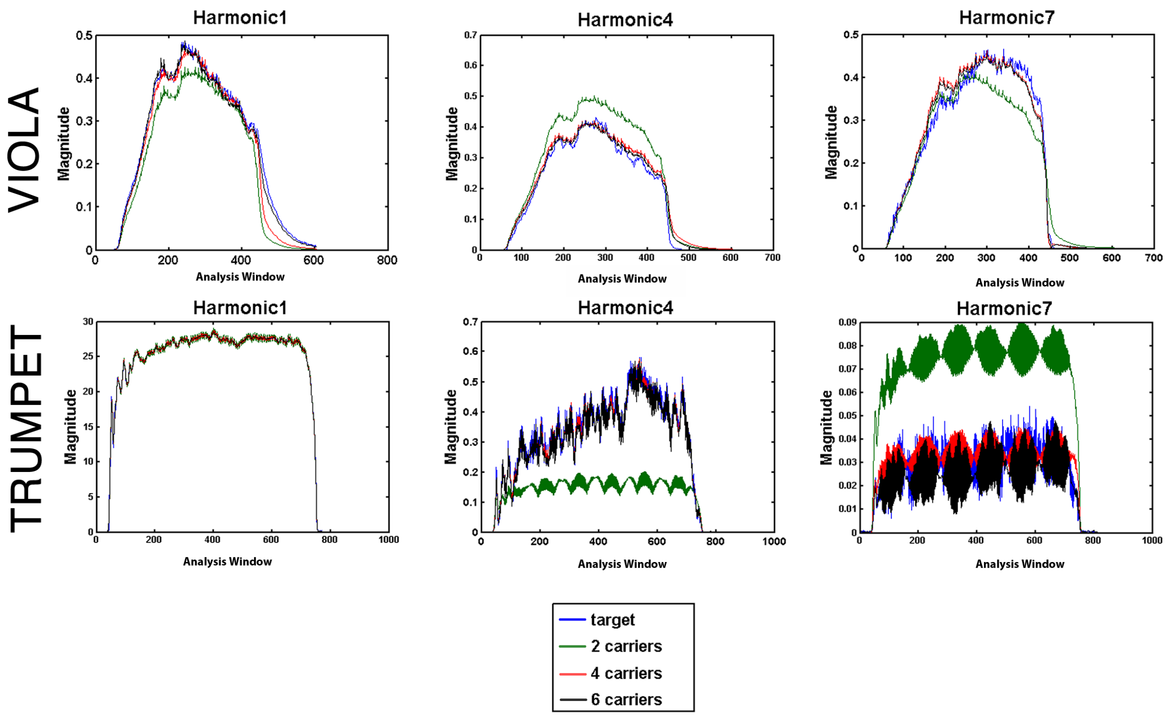


Figure 3.4: modFM harmonic approximation for a trumpet and viola sound

3.3 Parameter search for more complex synthesizers

GAs were also used to determine the parameters of more complex synthesizers. Yee-King and Roth [60] 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 mda DX-10 [2], a single modulator FM synthesizer with 16 parameters, and the mda JX-10 [3], a substrative synthesizer with one noise oscillator and 40 parameters. Yee-King et al. evaluated the performance of their *Synthbot* with a user study [60]. They did a two phase experiment in which the performance of expert human users were compared to that of SynthBot. In phase one, ten expert human users were asked to program two sounds synthesizers using a generic interface to match a real instrument sound and a pitched sound. SynthBot was given the same task. In phase two, the users were asked to rate each of the synthesized sounds for similarity to their respective target sounds. SynthBot then rated the sounds using its MFCC error metric. Finally correlations were sought between the similarity ratings of the human users and those of SynthBot. This experiment was supposed to establish the quality of SynthBots programming performance as well as the usefulness of the MFCC error metric for instrument sounds as fitness function. The first results indicated that SynthBot achieved an MFCC error which is an order of magnitude smaller than the human, in about half of the time. However, Yee-King et al. never published the results of the full user study.

McDermott et al. [36] developed several EC systems to match target sounds with the Xsynth synthesizer [6], an analogue-modular style subtractive synthesizer with 32 input parameters. First, they defined an exhaustive set of sound attributes, based on research in the fields of psychoacoustics and music information retrieval. They derived an *attribute* distance measure between sounds and showed, with a user study, that this distance was better correlated with human judgement of similarity thanother distance measures traditionally used in previous works (short-time spectrum least-squared error and time-domain error).

They used this newly defined *attribute* distance measure in a GA to match target sounds. They compared the performance of their GA, in terms of best achieved fitness score, to the same GA but using traditional distance measures (short-time spectrum least-squared error and time-domain error) and a composite distance measure averaging all 3 distances. They concluded that this evaluation, using the best achieved fitness score, was not successful to make a definite conclusion on which distance measure was performing the best. So, they ran another experiment asking human users to judge and compare the results obtained with the different distance measures. As expected, the time-domain function had been found to give the worst performance, but, again, no statistical difference had been shown between the other three.

They introduced a new interactive EC operator called *sweeping* operator, a user-controlled interpolation between members of the population, to allow fast audition of many individuals. They also introduced the technique of background evolution, running a non-interactive evolution (background) in parallel with the interactive one (foreground). In this technique, the best individuals found in the background evolution are transferred periodically to the foreground evolution for human evaluation. They compared this background evolution technique with a traditional interactive GA (iGA) and an iGA using their newly defined *sweeping* operator. Here again, they combined human listening and quantitative methods based on best achieved fitness scores to perform their study. They concluded that their new *sweeping* method were shown to be competitive with typical interactive EC methods, and to appeal to users as being faster, simpler, and more intuitive.

3.4 Synthesis system design and parameter search

Synthesis system design is a generalization of parameter optimization where the sound synthesis method itself is optimized as a parameter. These systems are supposed to let the algorithm choose the best synthesis method to use for a given target sound. Synthesis system design is even theoretically able to find new synthesis techniques different from the usual methods described in Section 3.2.

Takala et al. [55] evolved what they called *Timbre Trees*. Nodes of the trees were arithmetic operations, analytic functions or noise generators. They defined a roulette-wheel selection operator as well as a crossover and mutation operator for trees. The evaluation of the trees was done interactively by the user that could listen to the sounds generated by these *Timbre Trees*. They used the term Genetic Algorithm to describe their system but the term Genetic Programming (GP) seems more appropriate given the algorithmic process and genetic operators they defined. They did not do any evaluation but they reported that they successfully used their system to produce an entire class of bee-like sounds (ranging from mosquitos to chain saws).

More recently, Garcia [18] designed a Genetic Programming (GP) system for automatic generation of sound synthesis techniques. The data representation used in this system is a tree in which the nodes are synthesizer's basic components (oscillator, addition/multiplication operators, filters, etc) and the terminals are parameters. He defined the fitness function as the least-squared error of the short-time spectrums where the frequencies that are not heard by the average listener were ignored. He implemented the fitness-proportionate selection, crossover and mutation operators. Contrary to other EC systems presented above, two aspect of the system are evolved in parallel during the evolution:

- the form of the trees: it allows the GP system to suggest the most effective synthesis method to use,
- the parameters.

A GA is used to optimize the parameters of each individuals of the population. Garcia tested his system with a FM woodwind instrument and verified that it could generate an expression tree with close similarity to the target FM equation used to design the instrument. However, there were higher energies at high frequencies. He also got a modest result with a sampled piano tone and a resulting synthesized sound that sounded like a *string hit by a hammer*.

3.5 Summary and comparison

Table 3.1, 3.2 and 3.3 summarize and compare all the related works described in the last sections. It is challenging to compare these systems and to capitalize on them to solve new sound matching problems. First, there is not always an evaluation and, if there is one, it does not use the same set of targets sounds or the same performance metric. For this reason, it makes it almost impossible to have a clear idea of the system's performance or to know if it is really solving the problem. It is also not possible to compare the various system's performance. Mitchell et al. [42] described a contrived target test methodology that enables the performance of different optimization techniques to be measured and compared. They applied this methodology to a Frequency Modulation synthesizer, in order to compare the performance of different Evolution Strategy-based algorithms. However, this methodology is limited because they don't provide the set of target sounds they use. Developing a benchmark including target sounds of different natures and performance indicators could be a good contribution to the field of unsupervised sound matching. McDermott and al. [36] and Yee-King and Roth [60] evaluated the performance of their systems with a perceptual user study involving human listening tests. These kinds of evaluations are powerful but require a large number of participants, each performing many repetitions of each task, which it is not easy to get with the constraints of time and the difficulty of attracting volunteers.

Finally, most of the GA systems in the literature are not described well enough to be reproducible often missing a critical parameter or implementation detail. These imprecisions make it impossible to reproduce most of the experiments described in related works.

Synthesis methods	**EC type**	**Complexity**	**Target**	**Fitness function**
Additive				
[22]	GA	$N_{harm} \log_2(N_{frames})$	Harmonic target sounds	harmonics error approx.
Subtractive				
[52]	GA	10^{44} pos.	Electronic sounds	spectral distance with psych. weighting
[48]	GA	21 params $\in [0,1]$	Harmonic instrument sounds	MFCC Euclidian distance
Granular				
[17]	GA	N/A	Exploration	user's defined
[25]	GA	112 bits	Exploration	interactive
Physical				
[45]	GA	9 params	Pluck strings sounds	short time spectral least-squared error
[58]	GA	8 params	Flute sounds	spect. dist.
FM				
[30]	GA	4 params	FM sounds	mean sum of spect. norm and centroid
[23]	GA	10^6 pos.	Harmonic instrument sounds	mean sum of spect. norm and centroid
Mod FM				
[33]	GP	12 params $\in [0,20]$	Harmonic instrument sounds	spect. least-squared error
System design				
[18]	GP	10^{73} pos.	Instrument sounds	spect. least-squared error

Table 3.1: Comparison of the EC systems (Table 1)– R-W: roulette wheel

Synthesis methods	**Representation**	**Mutation**		**Crossover**		**Elitism**	**Selection**
		Type	**Prob.**	**Type**	**Prob.**		
Additive							
[22]	Binary	FlipBits	$\frac{1}{N_{bits}}$	1-pt	N/A	No	2-Tourn.
Subtractive							
[52]	Binary	FlipBits	0.03	2-pt	0.9	No	R-W
[48]	Float $\in [0,1]$	Gaussian $SD = 0.1, \mu = 0$	N/A	1 pt	N/A	No	R-W
Granular							
[17]	Binary	FlipBits	dynamic	1 pt	dynamic	No	F-P
[25]	Binary	FlipBits	0.1	1 pt	1	No	R-W
Physical							
[45]	N/A	N/A	N/A	N/A	N/A	No	F-P
[58]	Floats	std	N/A	std	N/A	No	F-P
FM							
[30]	Binary	FlipBits	1 bit	1 pt	0.6	3%	5-Tourn.
[23]	Binary	FlipBit	$\frac{1}{N_{bits}}$	1 pt	0.6	No	2-Tourn.
Mod FM							
[33]	Mixed integers/floats	Gaussian $\mu = 0, SD$ dynamic	0.2	1 pt	0.8	2 best ind	2-tourn.
System design							
[18]	Tree	Nodes/Branches	0.6	1 pt	0.20	No	R-W

Table 3.2: Comparison of the EC systems (Table 2)

Synthesis methods	**Pop. size**	**Stopping criteria**	**Local Search**	**Evaluation**
Additive				
[22]	N/A	N/A	Hill Climbing	Instrument tones
Subtractive				
[52]	70	500	No	Qualitative
[48]	5000	N/A	No	Contrived Recorded
Granular				
[17]	N/A	N/A	No	Qualitative
[25]	9	N/A	No	Qualitative
Physical				
[45]	N/A	N/A	No	Contrived Recorded
[58]	N/A	N/A	No	Contrived
FM				
[30]	100	No		
[23]	N/A	N/A	No	instrument recordings
Mod FM				
[33]	100	weighed fitness change or 300 gen	No	Contrived Recorded
System design				
[18]	40 to 70	200 to 500 pop=1 to 25, gen=1 to 7	GA	instrument synthetized

Table 3.3: Comparison of the EC systems (Table 3)

Chapter 4

OP1 synthesizer calibration using Evolutionary computation

4.1 Problem description

4.1.1 An all-in-one Synthesizer: the OP-1

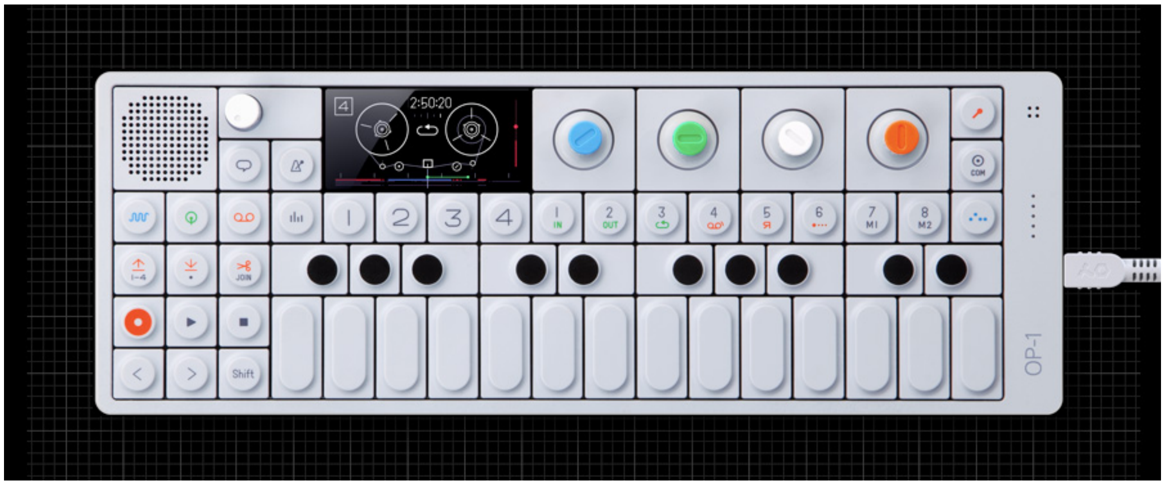


Figure 4.1: OP1 picture

The OP-1 is the all-in-one portable synthesizer, sampler and controller developed by Teenage Engineering (TE) [4] and illustrated in Figure 4.1. TE provided us with a C++ library that embeds most of the functionalities of the OP-1. We had access to seven different

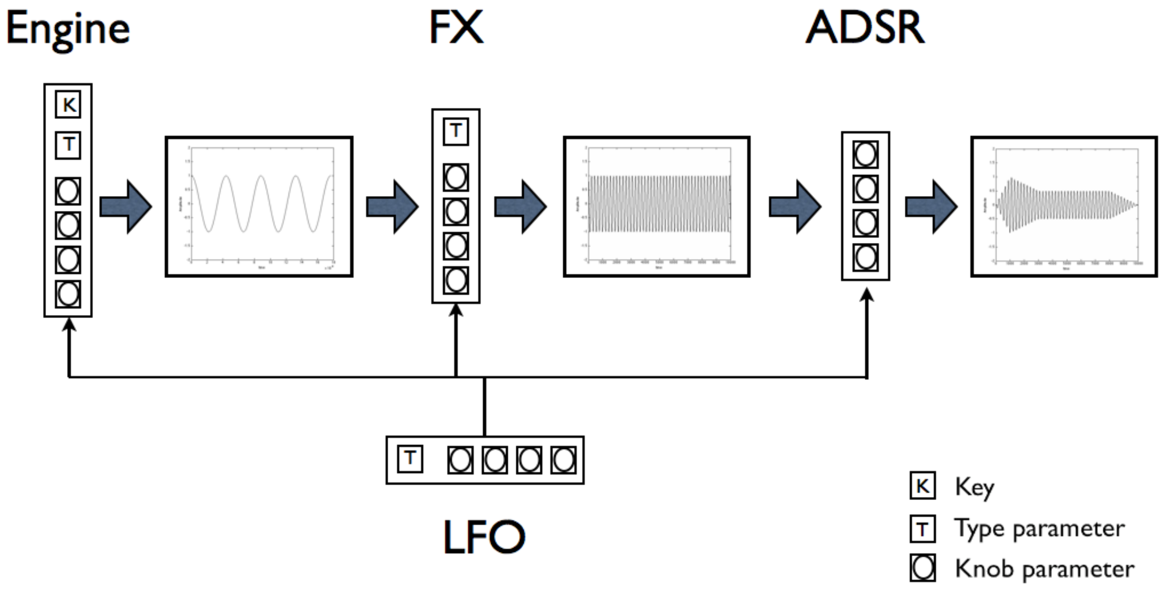


Figure 4.2: The OP-1

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 their type chosen, the synthesizer engine, FX, LFO and ADSR can be each controlled individually by the 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 [4]. Figure 4.2 shows an overview of the OP1 architecture.

Equation 4.1 gives the number of possible different combinations.

$$N_{eng} \times N_{LFO} \times N_{FX} \times N_k^{N_{knobs} \times N_t} \times N_{keys} \tag{4.1}$$

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_{keys} the number of keys. Their numerical values are given in Table 4.1. An estimate of the total number of possible combinations for the OP-1 synthesizer is then 10^{76}.

4.1.2 Challenges

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 deterministic. 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. Proof and measures of the non-determinism of the OP-1 are given in Chapter 5, Section 5.1.3. 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 change 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 are a large number of local minima (see Section 5.2.1). 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 [48].

N_{eng}	N_{LFO}	N_{FX}	N_k	N_{knobs}	N_t	N_{keys}
7	3	4	32767	4	4	120

Table 4.1: Synthesizer parameters complexity

4.2 Methodology

As described in Chapter 2, GAs are search algorithms that mimic the process of natural evolution. GAs are especially well adapted to the characteristics of our problem. First, GAs scale very well to the large and complex search space induced by the OP-1. Contrary to gradient search methods, they are less susceptible to converge prematurely to a local optimum [47].

Second, GAs also perform well in search spaces where the evaluation is approximative or noisy [24], as is the case with the OP-1 and its non-fully deterministic output. 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 refer to the target sounds generated using the OP-1 as *Contrived sounds* [42]. *Contrived sounds* were used as target sounds when exploring different GA configurations. 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 4.2 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 types 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 previously satisfying GA configuration was tested on the new problem. If this configuration was not

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 4.2: Problem description

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).

For the full problem complexity (Problem 5), each OP-1 preset is represented by a string of 257 bits. The number of distinct possible bit strings is then $2^{257} = 10^{257 \log_{10}(2)} \approx 10^{77}$. In Section 4.1.1, we calculated that the number of distinct possible OP-1 presets has an order of magnitude of 10^{76}. The difference of a factor 10 between these two orders of magnitude can be explained by the fact that, when the number of values to encode is not a power of 2, the binary encoding encodes for more values than necessary. For example, we have to encode 120 different keys in our chromosome (see Table 4.1). With 6 bits, it is possible to encode $2^6 = 64$ different keys and with 7 bits, $2^7 = 128$ different keys. We, then, chose to use 7 bits and applied a scaling function to keep the decoded integers between 0 and 119. However, this difference of 9 between the number of keys to encode and the number of possible distinct bit strings when using 7 bits can explain the difference we observed in the orders of magnitude.

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.

4.3 System design

4.3.1 Representation

At first, we decided to encode these parameters using a binary representation (which is a common choice in practice, see Chapter 2, Section 2.1.2). Using a mixed-integer or real-value representation did not make sense in our case because both type and knob parameters in the OP-1 are integers and not real values. Moreover, when using a mixed-integer or real-value representation, the crossover operator loses its ability to explore the genotype space as it is just exchanging integer or real parameters between chromosomes. Whereas, with a binary representation, the crossover operator exchanges bits, which can lead to some changes in the related OP-1 parameters values.

Our experiments using the binary representation on Problem 1 (see Table 4.2) seemed to indicate that the GA was always converging to the same local minima[1]. Investigating further, we realized that, in order to improve the best individual fitness, 11 bits would have to be changed to go from 4095 (0111111111111) to 4096 (1000000000000) 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 4.2 in bold letters show the final chromosome design.

4.3.2 Genetic operators

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 (see Chapter 2, Section 2.1.5). One reason for the loss of diversity is the recombination of identical chromosomes. Indeed, when two strictly identical chromosomes are selected for cross-over, two offsprings identical to their parents 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

[1]`http://metacreation.net/mmacret/thesis/rep`

Table 4.3: Mutation examples

Integer	**Binary**	**Gray**
4095	0111111111111	0100000000000
4096	1000000000000	**1**100000000000
2048	0100000000000	01**1**0000000000

diversity loss and prevent premature convergence [47]. Our system uses a 2-point crossover and a crossover rate of 60 %.

Mutation

The mutation operator participates in both exploration and exploitation (local search) (see Chapter 2, Section 2.1.3). 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). Table 4.3 shows two examples of mutation using the same bit strings than in Section 4.3.1. Flipping one bit in the Gray code can either lead to a single increment in the integer value (4095 to 4096) or lead to a large change in the integer value (4095 to 2048). 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 $n = N_{\text{bits}}$.

4.3.3 Fitness function

In our attempt to solve Problem 1, we used the Euclidian distance between the Short-Time Fourier Transform (STFT): d_{STFT}. Equation (4.2) shows the Euclidian distance between the STFT for the individual t and c.

$$d_{STFT}(t,c) = \sum_{i=1}^{N_w} \sqrt{\sum_{j=1}^{N_s} (t_{i,j} - c_{i,j})^2} \tag{4.2}$$

The sampling rate was 44100 Hz and we set a window size N_s of 1024 samples (23 ms) and an overlapping of 512 samples (11.5 ms). N_w is the total number of windows. 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).

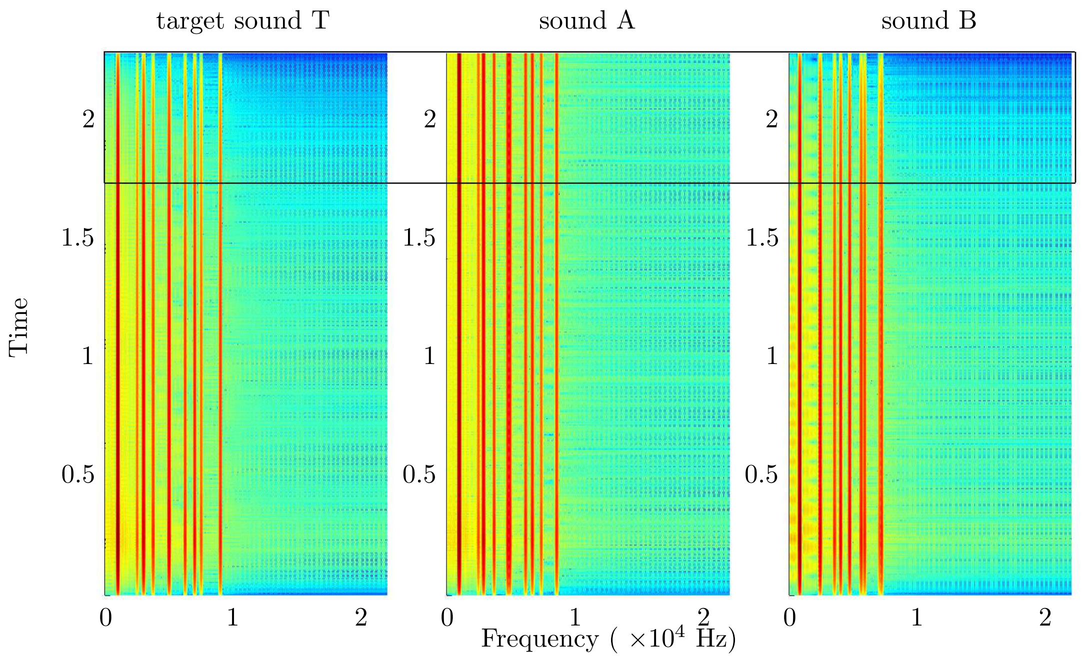


Figure 4.3: Weight of the envelope in the Euclidian distance for the STFT

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 amplitude envelope in the Euclidian distance between the STFTs is significant. For example, consider a target sound T and two candidate sounds A and B (see Figure 4.3). A has a similar spectrum to the spectrum of T for the first short-time windows but not for the last ones because the amplitude 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 than A but their amplitude envelope is the same. The weight of these last short-time windows (shown by the rectangle in Figure 4.3) can make B appear closer to T than A according to the Euclidian distance between the STFT described in Equation 4.2.

In this context, a correct 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 [33, 23], the envelope was determined analytically for each individual in the population, however it is not possible to do this in our case. Indeed, we know that, for the ADSR, the first knob is mapped to the Attack, the second one to the Decay, the third one to the Sustain and the last one to the Release but we do not know the precise value mapping between these knob parameters and the embedded ADSR values. Moreover, an LFO can be use to modulate one or several of these ADSR knobs. Another idea would be to perform a local search to set the ADSR knobs parameters for each individual in the population. However, it would be 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.

In order to avoid the premature convergence observed with Problem 2, we decided to uncouple the amplitude 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 amplitude envelope. Computing the FFT on the entire sound mitigates, to some extent, the effect of the amplitude envelope on the spectrum. We extracted the amplitude 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. (4.3)). However, it is challenging to choose the appropriate weights for the amplitude envelope a_{env} and the FFT a_{FFT} to make the system converge.

$$f = \frac{a_{\text{env}} f_{\text{env}} + a_{\text{FFT}} f_{\text{FFT}}}{a_{\text{env}} + a_{\text{FFT}}} \tag{4.3}$$

Therefore, we chose to consider two objectives: FFT and amplitude 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 amplitude envelope.

4.3.4 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 [15] and in Chapter 2, Section 2.4. 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.

4.3.5 Stopping criteria

The optimization process terminates if the weighted change in the 3 objective fitness, given by Eq. (4.4), is less than 10^{-10} 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 3000th generations.

$$\delta_n = \sum_{i=1}^{N} \left(\frac{1}{2}\right)^{N-i} (f_{n+1-i} - f_{n-i}), \tag{4.4}$$

4.3.6 Pareto front

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

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.

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. [14] 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.

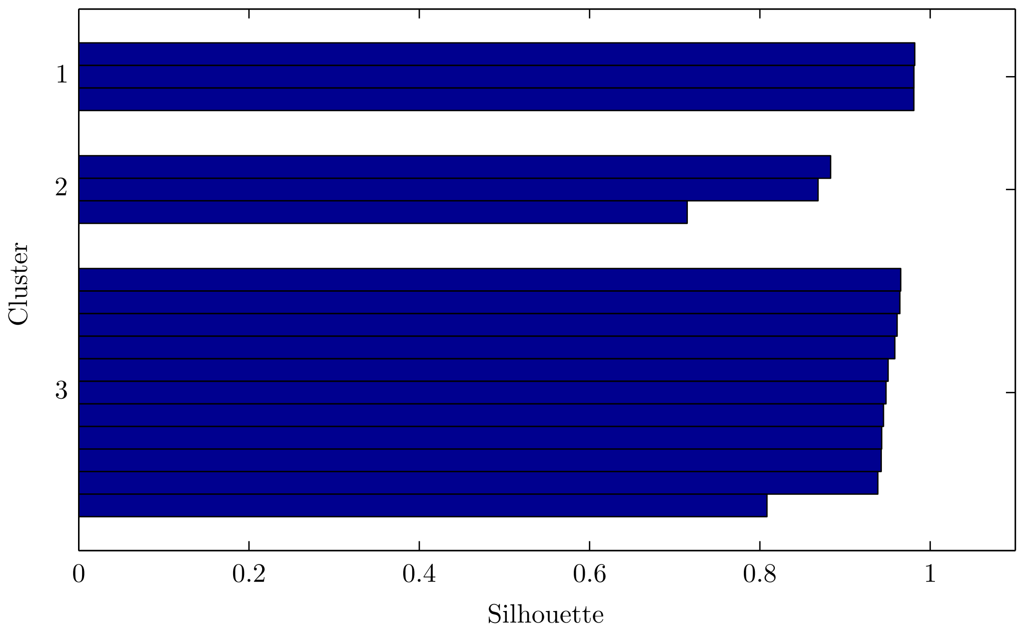


Figure 4.4: Example of a silhouette plot

2. Determine the optimal number of clusters, k. The silhouette [50] 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. Figure 4.4 shows an example of a silhouette plot.

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.

4.3.7 Full problem complexity

In Problem 5, we add the type parameters. These extra parameters to search induces discontinuities in the fitness landscape (see Section 4.1.2). 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 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 4.1.1).

4.4 Implementation

Figure 4.5 gives an overview of the GA implementation. The implementation of the GA is done using the DEAP Python framework [16]. Sound features are extracted using the Python wrapper for Yaafe [35]. 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 to speed up the computation. It also makes it possible to use larger populations than would have been feasible using only one processor for the same running time. We used two clusters alternatively, Grex and Bugaboo, that are part of Westgrid-Compute Canada [5]. 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).

The post processing, including the k-means clustering and the report generation, is done using Matlab.

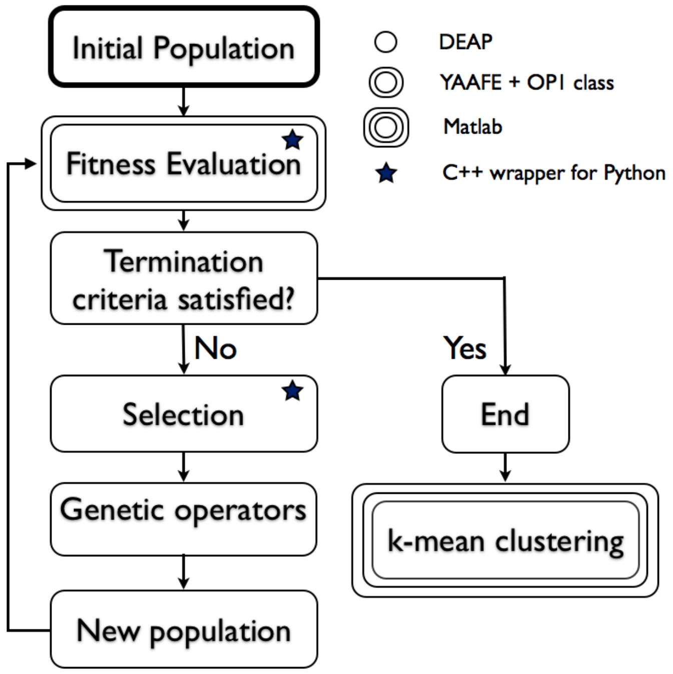


Figure 4.5: Implementation overview

4.4.1 DEAP: Distributed Evolutionary Algorithms in Python

DEAP (Distributed Evolutionary Algorithms in Python) is an evolutionary computation framework written in Python [16]. It also incorporates easy parallelism where users need not concern themselves with implementation details like synchronization and load balancing, only functional decomposition.

As interpreted language, Python is not as fast as C/C++ to execute EAs. Whenever execution time becomes critical, computationally intensive components can always be recoded in C and wrapped in Python code. Many efficient numerical libraries are already available through Python APIs.

Core Architecture

The core architecture of DEAP is built around different components that define the specific parts of what is an evolutionary algorithm.

DEAP's core is composed of three modules: base, creator, and tools. The base module contains objects and data structures frequently used in EC that are not already implemented in the Python standard library. Python providing most of the data structures required, this module actually implements only three classes: a generic fitness class , a basic nary tree class and a toolbox class.

The toolbox is a container for the tools (operators) that the user wants to use in his EA. For instance, if a user's algorithm requires mutation, the user can select one of the built-in mutation designs provided by DEAP and register it into the toolbox with a generic mutation alias. In this way, if later the user decides that some other mutation is better suited, only the toolbox will have to be updated and not the full algorithm implementation.

DEAP implements the Factory design pattern through the creator module. This module is a meta-factory that allows creation of classes via both inheritance and composition using a functional programming paradigm, therefore liberating the user from the burden of class definition. Attributes, both data and functions, can be dynamically added to create new object classes empowered by the user to provide user specific EA functionalities.

The tools module contains frequently used EA operators. It also provides objects that ease the different analysis tasks of EAs such as checkpointing, statistics computation, and genealogy.

Beside these core modules, DEAP also provides the algorithms module that contains four commonly used algorithms in EC: generational, (μ, λ), $(\mu + \lambda)$, and ask-and-tell [16]. However, DEAP is not limited in any way to these four. They are only a starting point for users to develop their own customized EA algorithms. Other modules specific to Genetic Programming (GP) or Covariance Matrix Adaptation Evolution Strategy (CMA-ES) can also be found. The last module of the framework, named DTM for Distributed Task Manager, handles parallelism. It will be described in the next section.

Distribution

In order to allow easy parallelization of specific parts of the user's algorithm, DEAP provides a Distributed Task Manager (DTM) module that can handle parallel sub-task creation and execution on both multi-core computers and clusters of networked nodes by relying on MPI. The DTM interface is composed of two main functions: submit which is used to execute another function as a single parallel subtask; and map which is used to apply in parallel a given function to every element of an iterable object (e.g. a list). Its usage is almost completely transparent. For instance, in the previous examples, the only required changes in order to distribute the fitness evaluation tasks are to, somewhere during the initialization

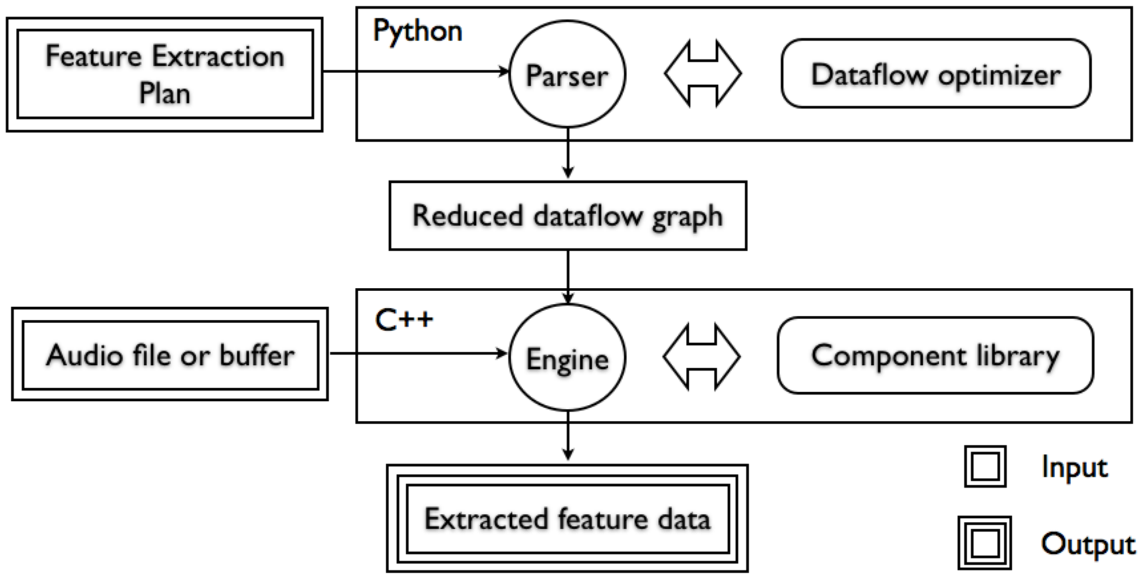


Figure 4.6: YAAFE

steps, import the dtm module and replace the map function in the toolbox with the dtm.map function.

No other change is necessary. The map function call for the fitness evaluation of individuals will spawn parallel sub-tasks distributed by DTM across the worker nodes. DTM even loadbalances them if they are not of equal duration. The objective of the load balancer is to evenly distribute the load between worker nodes. The most underloaded workers transmit more often their load estimates to the overloaded ones, so that they inform the later of the relative under utilization of the former. For their part, the most overloaded workers transfer some of their tasks to the underloaded ones in order to reduce their own load.

4.4.2 YAAFE: Yet Another Audio Feature Extractor

YAAFE is a audio feature extraction software [35]. Figure 4.6 describes how YAAFE handles feature extraction. The user has to provide the audio files and a feature extraction plan. The feature extraction plan is a set of bit strings where the user declares the features to extract, their parameters and transformations. To take advantage of feature computation redundancy, YAAFE proceeds in two main stages. In a first stage, a parser analyzes the feature extraction plan in order to find common computational steps (implemented in C++

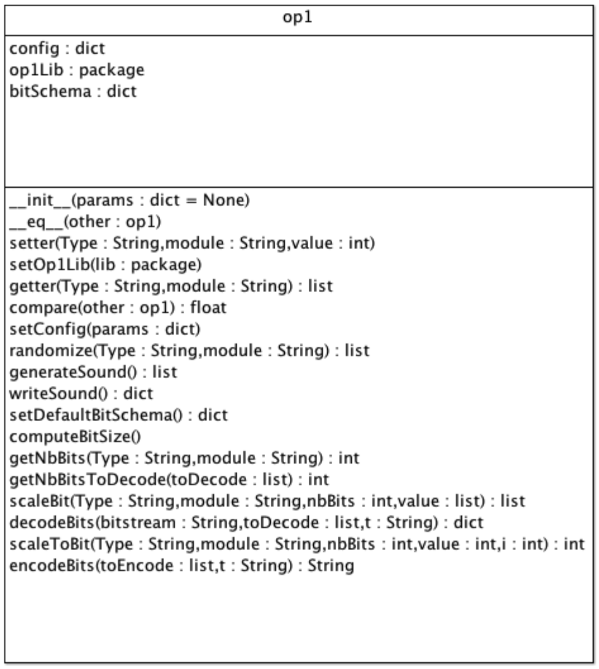

```
op1
config : dict
op1Lib : package
bitSchema : dict
__init__(params : dict = None)
__eq__(other : op1)
setter(Type : String,module : String,value : int)
setOp1Lib(lib : package)
getter(Type : String,module : String) : list
compare(other : op1) : float
setConfig(params : dict)
randomize(Type : String,module : String) : list
generateSound() : list
writeSound() : dict
setDefaultBitSchema() : dict
computeBitSize()
getNbBits(Type : String,module : String) : int
getNbBitsToDecode(toDecode : list) : int
scaleBit(Type : String,module : String,nbBits : int,value : list) : list
decodeBits(bitstream : String,toDecode : list,t : String) : dict
scaleToBit(Type : String,module : String,nbBits : int,value : int,i : int) : int
encodeBits(toEncode : list,t : String) : String
```

Figure 4.7: OP-1 class

components), and a reduced dataflow graph is produced. Then in a second stage, feature extraction is applied to the given audio files according to the reduced dataflow graph and results are stored in a Python array.

Python is used for implementing the parser and the dataflow optimizer because it allows more concise and readable code to be written than C++. The dataflow engine and the component library have been developed in C++ for performance.

Their benchmark [35] showed that Yaafe clearly outperforms its competitors such as Marsyas [57] or Sonic Annotator [12].

4.4.3 OP-1 class

Teenage Engineering provided us with a C++ library of the OP-1. We implemented a Python wrapper for this library and develop a class to interface it with DEAP and YAAFE. Figure 4.7 shows the class diagram for this OP-1 class.

The attribute *config* is a dictionary containing the OP-1 types and knobs parameters, the number of samples to generate and the name of the wav file to write. The attribute *op1Lib* is a reference to the OP-1 Python wrapper. The attribute *bitSchema* is a dictionary that specify for each type and knob parameters the number of bits it has to be encoded with.

The constructor *__init__* initializes the *config* attribute with the argument *params* (if provided) otherwise a *config* is randomly generated using the *randomize* operation. The comparison operator *__eq__* returns true if the *other* OP1 object has the same types and knob parameters, otherwise it returns false. The argument *Type* can take the values: "type" (type parameters) or "params" (knobs parameters) and the argument *Module* can take the values: "Engine", "FX", "LFO", "Key", "Octave" or "ADSR". The operation *generateSound* generates a buffer of samples using the *config* attribute with the OP1 Python wrapper. The operation *writeSound* write a wav file with the specifications of *config* attribute.

The operation *setDefaultBitSchema* sets the attribute *bitSchema* to the default bit schema used when both types and knob parameters are searched. The operation *computeBitSize* computes the optimum numbers of bits to cover the ranges of OP-1 types and knob parameters and assigns them to the attribute *bitSchema*. The operation *decodeBits* decodes a *bitstring* of type *t* ("bin" or "gray") given a list *(*toDecode) of types and knob parameters to decode. This operation uses the operation *scaleBit* to scale the decoded integers to the OP-1 types and knobs parameter values. The operation *encodeBits* encodes in a bit string of type *t* part of the attribute *config* specified by the list *toEncode* of types and knob parameters to encode. This operation uses the operation *scaleToBit* to scale the OP-1 types and knobs parameter values to integers within the admissible range permitted by the number of bits specified in the attribute *BitSchema*.

Chapter 5

Experiments

We based our evaluation design on Johnson's recommendation about experimental analysis of algorithms [26]. We especially focused on ensuring reproducibility and comparability.

5.1 Sound collection

5.1.1 Contrived sounds

Using *contrived sounds* - i.e. sounds actually generated by the OP-1 - as target sounds allows us to validate our system design making it possible to show that it is able to reverse engineer the parameter setting (or presets) of 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 OP-1 possible outputs. We especially focused on having diversity in spectral variation, noisiness and spectral spread. The first half of the sounds had a stationary spectrum and the other half has a dynamic spectrum, as measured by their respective spectral variation. The spectral variation $S_v(t)$ (or spectral flux) represents the amount of variation of the spectrum along time. It is computed from the normalized cross-correlation between two successive amplitude spectra $a(t-1)$ and $a(t)$.

$$\begin{aligned} S_v(t) &= \frac{\sum_k (a_k(t)-a_k(t-1))^2}{\sqrt{\sum_k a_k(t-1)^2}\sqrt{\sum_k a_k(t)^2}} \\ S_v &= \sum_t S_v(t) \end{aligned} \tag{5.1}$$

$a_k(t)$ is the k^{th} bin of the amplitude spectrum at time t.

Conf. Id	Engine	FX	LFO	key	octave	S_v	Stationary
0	FM	Grid	No	17	4	1.252×10^{-4}	N
1	Digital	Delay	No	1	-1	7.641×10^{-6}	Y
2	Cluster	Delay	Value	16	3	6.625×10^{-4}	N
3	Digital	Punch	Tremolo	16	3	1.172×10^{-4}	N
4	Digital	Delay	No	9	2	9.301×10^{-6}	Y
5	FM	No	No	0	2	3.055×10^{-4}	N
6	FM	No	No	11	1	7.128×10^{-5}	Y
7	String	No	No	20	-3	3.055×10^{-4}	N
8	Cluster	No	No	12	0	6.740×10^{-6}	Y
9	Pulse	No	No	9	-1	2.279×10^{-4}	Y
10	Phase	No	No	9	-4	1.154×10^{-4}	N
11	Phase	Punch	Element	16	1	9.4187×10^{-6}	Y

Table 5.1: Contrived sounds

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 [1]. Table 5.1 described these contrived sounds. For each of these sound, we also give its overall spectral variation S_v [44].

We distinguish two groups: the sounds with a stationary spectrum ($S_v < 10^{-5}$) and the sounds with a dynamic spectrum ($S_v \geq 10^{-5}$).

5.1.2 Other sounds

A second evaluation was performed on 12 non-contrived sounds including synthetic sounds, instrument sounds, a male voice sound and a natural sound . These sounds were carefully chosen to have a good diversity in spectral variation, noisiness and spectral spread.

Table 5.2 shows the list of the recorded sounds used for target. As with the contrived sound, we distinguish two groups: the sounds with a stationary spectrum ($S_v < 10^{-5}$) and the sounds with a dynamic spectrum ($S_v \geq 10^{-5}$).

5.1.3 Non-determinism

In order to illustrate the non-determinism of the OP-1, we ran a set of 2 experiments. The first experiment aimed to show that the measure of our 3 objectives distances (Envelope Euclidian distance, FFT Euclidian distance and STFT Euclidian distance) was not flawed. For each non-contrived sound, we measured the 3 objectives distances to itself and verified that they were always equal to zero.

Name	**Type**	S_v	**Stationary**
voice	human voice	5.426×10^{-5}	Y
bassoon	instrument	1.112×10^{-5}	Y
clarinet	instrument	6.140×10^{-5}	Y
violin	instrument	5.811×10^{-5}	Y
cat	cat meow	2.470×10^{-4}	N
dx7_1	synthetic	2.116×10^{-4}	N
dx7_2	synthetic	5.048×10^{-4}	N
lightsaber	FX	2.697×10^{-4}	N
moog_bass04	synthetic	1.449×10^{-5}	Y
moog_vibration05	synthetic	3.164×10^{-4}	N
snare	instrument	4.732×10^{-5}	Y
tb	synthetic	2.534×10^{-4}	N

Table 5.2: Non-contrived sounds

The second experiment involved generating two sounds with the same OP-1 preset and measuring the 3 objectives distances between them. If the OP-1 was fully determinist, we would expect these distances to be always strictly equal to zero. However, it is not what we observe in practice. Figure 5.1 and Figure 5.2 respectively show the average distances we measured for the Envelope and for the FFT and STFT when repeating a hundred times the experiment for each contrived sounds presets. As one can see, the distances are globally small but never equal to zero. There are also differences given which synthesis engine is used. For example, conf2 (Cluster), conf7 (String), conf8 (Cluster), and conf9 (Pulse) have significantly higher distances values than the other OP-1 presets. In some synthesizers' design, noise and randomness are used to add some warmth and realism to the synthesizer's output [9]. However, the sounds still remain perceptually alike from one trigger to another because it would quickly becomes musically unpredictable (in a bad way) if each trigger would sounds too different. This non-deterministic output of the OP-1 induces noise in the evaluation and can then slow down or even mislead the search.

5.2 Statistics for one run

In this section, we will present the statistics gathered during one run of our algorithm. To illustrate these statistics, we will take the example of one typical GA run for the contrived sound: conf4 and one run for the non-contrived sound: cat meow. The statistics and results for all the runs are available online [1].

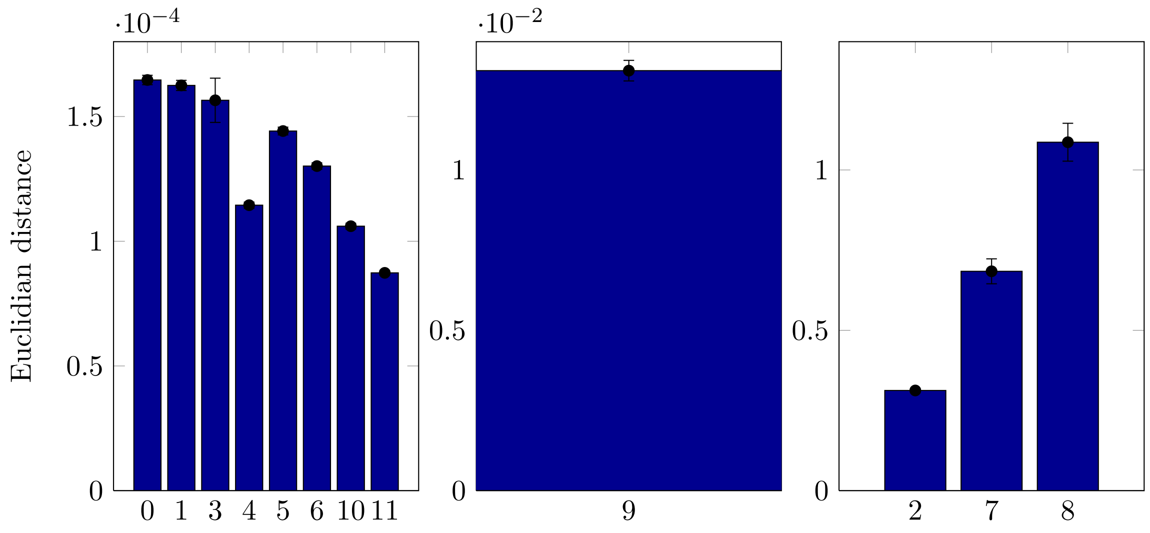


Figure 5.1: Non-determinism: Envelope

5.2.1 Objective fitness

The goal of the GA is to minimize the 3 objectives fitnesses (Envelope Euclidian distance, FFT Euclidian distance and STFT Euclidian distance). Figure 5.3 shows the reduction of the FFT distance over the generations. On both graphs, a viewer can observe plateaus, also called *punctuated equilibriums*, that can be interpreted as periodic improvements in fitness. One can also distinguish on these graph the *exploration* phase and *exploitation* phase. The *exploration* phase happens at the early stage of the evolution with a quick and significant improvement in the fitness. The *exploitation* phase follows with episodic and small improvement in the fitness. The length of these phases can vary depending of the nature of the target sound. For example, a viewer can see that the *exploration* phase is around 50 generations for the contrived sound and around 350 generations for the non-contrived sound. Another interesting point to note is the difference in the range of the final objective fitnesses for the contrived sound experiments and for the non-contrived sound experiments. For example, the final FFT fitnesses vary between 0 and 500 for the contrived sound experiments and between 2000 and 3000 for the non-contrived sounds experiments. This difference illustrates the fact that, as expected, it is more difficult to approximate non-contrived sounds (large final objective fitnesses) than contrived sounds (smaller final objective fitnesses).

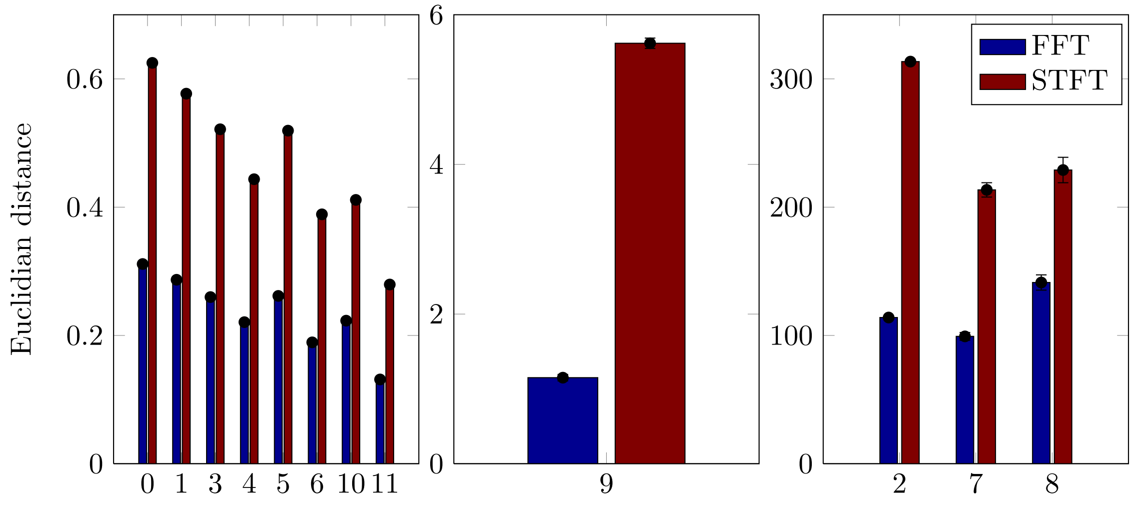


Figure 5.2: Non-determinism: FFT and STFT

5.2.2 Euclidian/Hamming distance

Another way of tracking the improvements of the system for the non-contrived sounds is to look at the parameter distance to the target parameters over the generations. We considered two distances: the Euclidian distance and the Hamming distance.

The Euclidian distance between two individuals is defined in Equation (5.2).

$$d_e(r,s) = \sqrt{\sum_{i=1}^{N}(r(i) - p(i))^2} \tag{5.2}$$

where r and s are two individuals set of parameters. N is the total number of parameters. $r(i)$ (resp. $s(i)$) are the i^{th} parameter of individual r (resp. $s(i)$).

The Hamming distance between the two bitstrings of chromosomes is defined in Equation (5.3).

$$d_h(u,v) = \frac{c_{01} + c_{10}}{n} \tag{5.3}$$

where c_{ij} is the number of occurrences of $u[k] = i$ and $v[k] = j$ for $k < n$, n being the number of bits.

Figure 5.4 shows the minimum of these distances to the target individual/chromosome over the generations for the conf4 sound. Contrary to the objective fitnesses (Figure 5.3), the curves are not monotonic decreasing. The Euclidian distance even appears to increase over the generations. Possible explanations are that there are several local minima or even

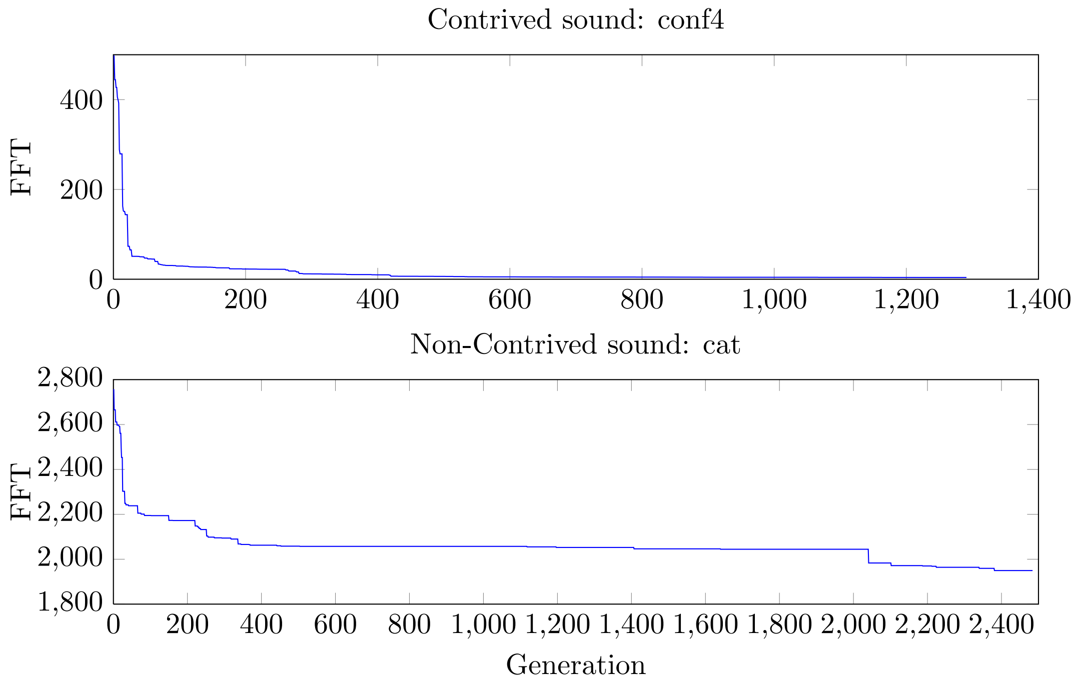


Figure 5.3: Objective fitness: FFT

several possible solutions. Then, it is possible to achieve a good approximation of the target sound using completely different parameters than the ones used initially to produce it. For this reason, tracking these distances is of limited relevance to evaluate the performance of our system. However, they make it possible to estimate the difficulty of our problem observing the correlation between fitness and distances.

Distance/fitness correlation

As described in Chapter 2, Section 2.2.1, the correlation between distance and fitness is often used to evaluate the difficulty of the problem being solved. The Euclidian/Hamming distances presented in section 5.2.2 already give us the intuition that the problem is difficult as the Euclidian distance does not seem correlated at all to the fitness.

To further study the difficulty of our problem, we plot in Figure 5.5 the average fitnesses per generations against their average distances. One can see that, in average, the distances are well correlated to the fitness with a correlation coefficient equals to 0.67 for the Euclidian distance and 0.70 for the Hamming distance. It shows that the fitness lead, in average, the

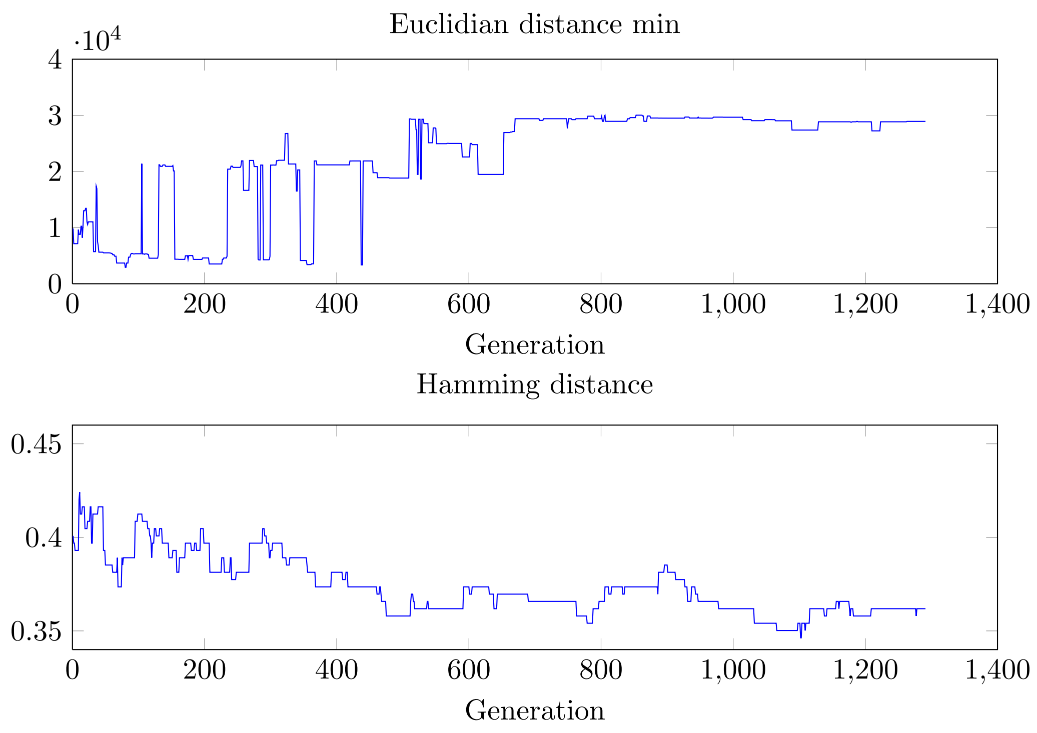


Figure 5.4: Distances (min)

GA to the region in the parameters space where the target is located. We can also observe, with the cluster of points around the coordinates $[3 \times 10^4, 0]$, that a single large optimal plateau exists.

We also plot in Figure 5.6 the fitness of each individual against their distance for the last generation of the GA. One can see that as the distance increases, the fitness decreases. This inverse correlation between distance and fitness makes the convergence toward the exact local or global minima very challenging and also explain why we never get the exact target presets at the end of the optimization.

5.2.3 Diversity

As described in Chapter 2, Section 2.1.5, the diversity in the population is critical for the success of the GA. In a ideal GA, the diversity would be high in the *exploration* phase and low in the *exploitation* phase. However, it is important to keep the some diversity even

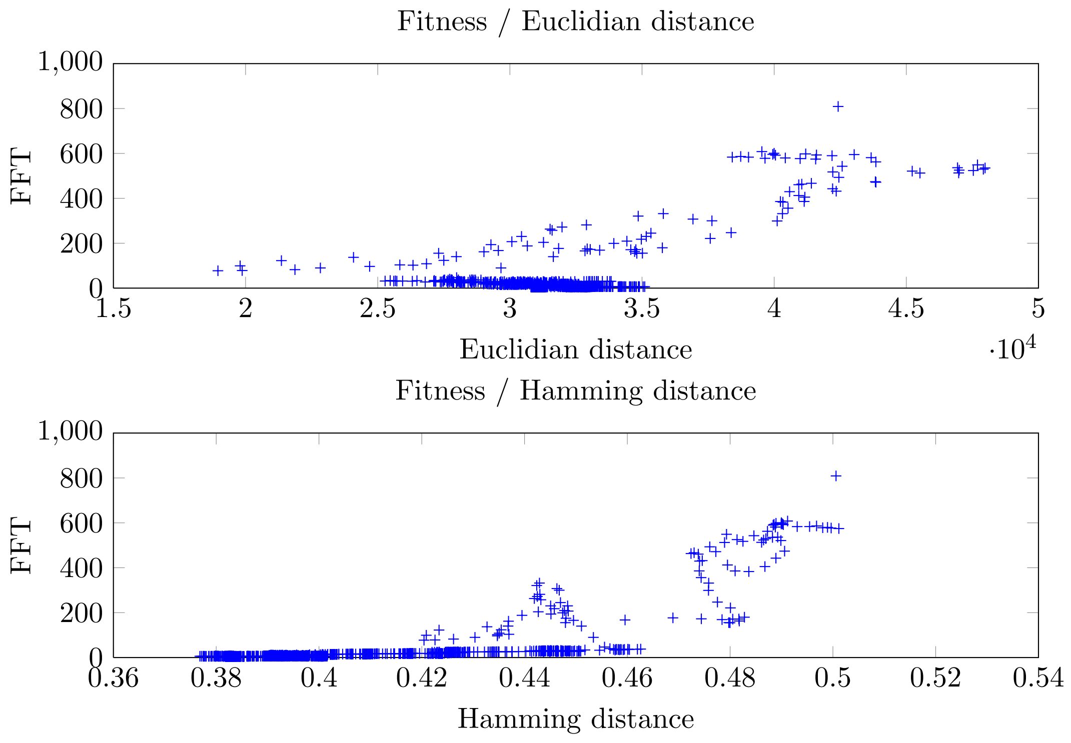


Figure 5.5: Fitness-Distance (global)

in the *exploitation* phase as crossing over a homogeneous population does not yield new solutions and therefore does not bring any gain to the optimization.

We use different common ways of measuring the diversity:

1. measuring the proportion of different individuals for each generation,

2. measuring the numbers of each module types for each generation,

3. measuring the knob parameters standard deviation for each generation,

4. measure the distance standard deviation for each generation.

Proportion of different individuals

For each generation, we measure how many individuals in the population have a different genotype by at least one bit. Figure 5.7 shows the proportion of different individuals over the generations for the conf4 sound and for the cat sound. One can see than the diversity is

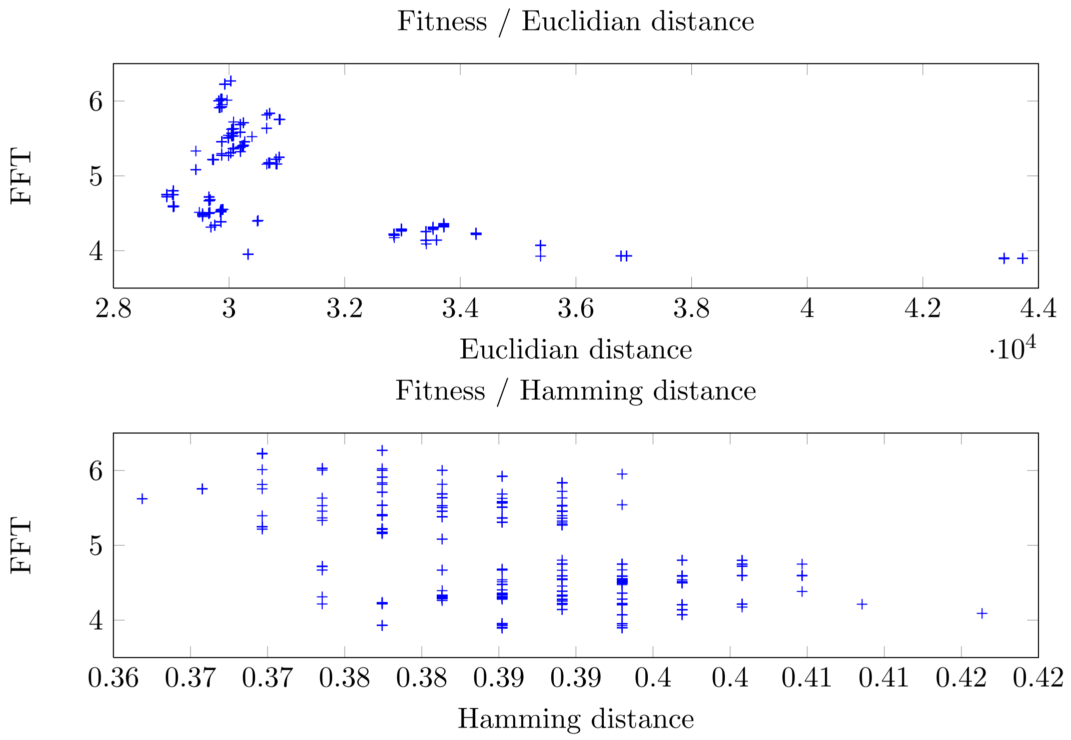


Figure 5.6: Fitness-Distance (last generation)

high at the beginning of the evolution (*exploration* phase) and then stabilizes (*exploitation*). There are also differences between the contrived sound and non-contrived sound. As seen in section 5.2.1, the *exploration* phase is less long for the contrived sound than for the non-contrived sound. Moreover, the diversity stabilizes around 30 % for the contrived sound and 70 % for the non-contrived sound. One explanation is that it is indeed more challenging to identify promising regions in the parameters space for a non-contrived sound than for a contrived sound, as the existence of a solution is not even ensured.

Types

For each generation, we keep track of which module types are used by the individuals in the population. Figure 5.8 shows the repartition of the engine types for each generation for the conf4 sound and for the cat sound. We can observe the same phenomenon than previously observed with the proportion of different individuals. During the *exploration* phase, every type is evenly represented and then one specific type is taking over during the *exploitation*

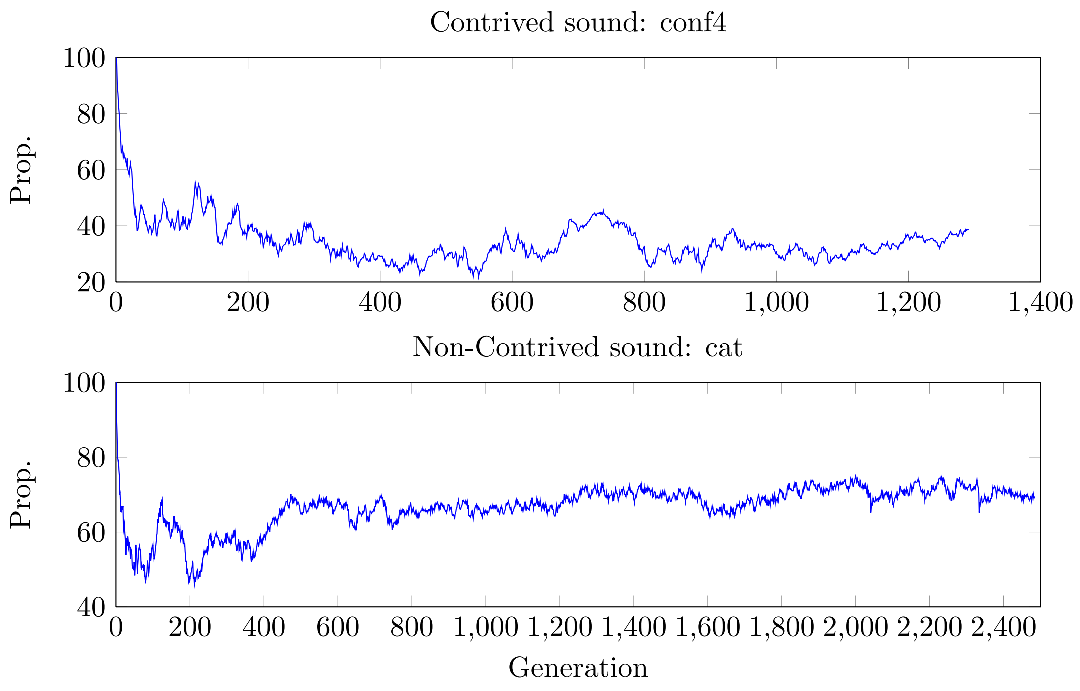


Figure 5.7: Proportion of different individuals

phase. Again, we can observe that the transition between these two phases is happening later in evolution for the non-contrived sound than for the contrived sound.

Knob parameters standard deviation

For each generation, we measure the standard deviation of each knob parameters for the population. Figure 5.9 shows the standard deviation for the ADSR knob parameter number 2 for the conf4 sound and for the cat sound. Here again, a viewer can distinguish the *exploration* phase from the *exploitation* phase, the first being shorter for the non-contrived sound than for the contrived sound.

5.2.4 Pareto front

Analysis, Clustering

Our experiments show that the Pareto front is most of the time too big to be easily handled by the final user. The idea is to give a sample of individuals (less than 10) that is a good representation of the Pareto front. As described in Chapter 4, Section 4.3.6, we apply a

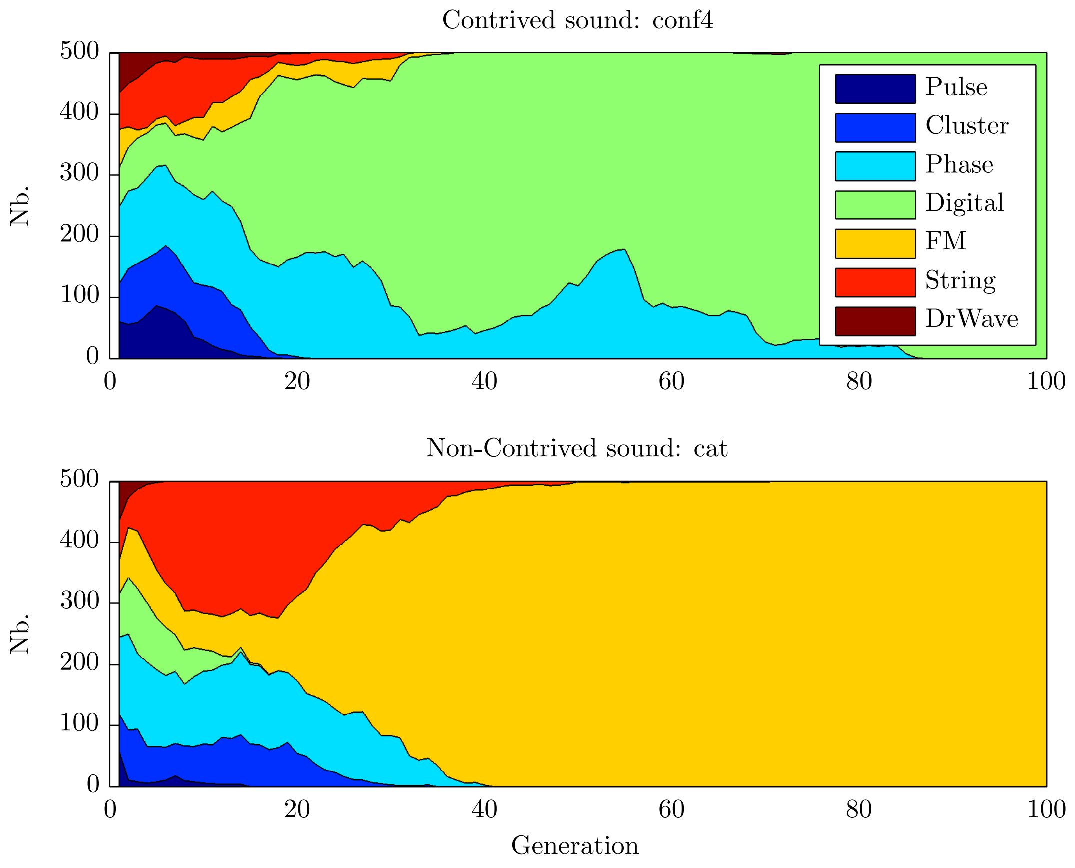


Figure 5.8: Engine Types

k-means clustering algorithm to cluster on the solutions enclosed in the Pareto Front. As the clustering is done on the OP-1 knob parameters, the user can easily retrieve all the individuals enclosed in the given cluster starting with the centroid individual. Indeed, we assume that going from the centroid individual to any individuals in the cluster, the user just has to slightly modify the centroid individual's parameters. Moreover, we also assume that the individuals in a same cluster sound similar as their OP-1 configuration are similar.

The k-means clustering algorithm requires that the number of clusters k to be chosen before running the algorithm. The average silhouette of the data is an useful criterion for assessing the natural number of clusters. The silhouette of a datum is a measure of how closely it is matched to data within its cluster and how loosely it is matched to data of the neighbouring cluster, i.e. the cluster whose average distance from the datum is lowest. A

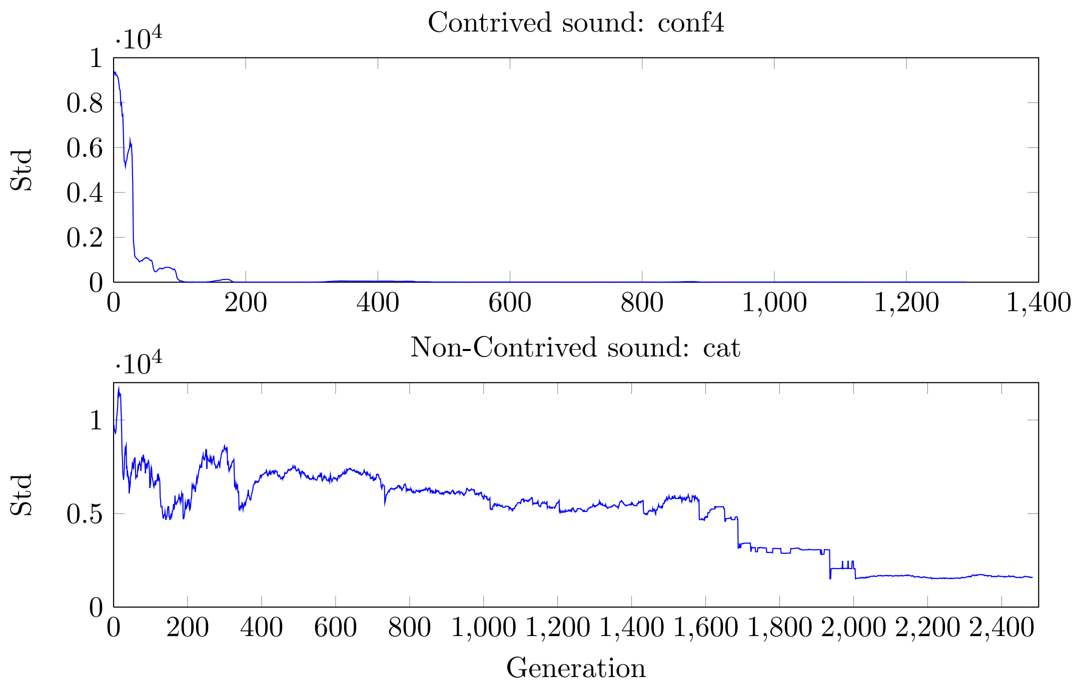
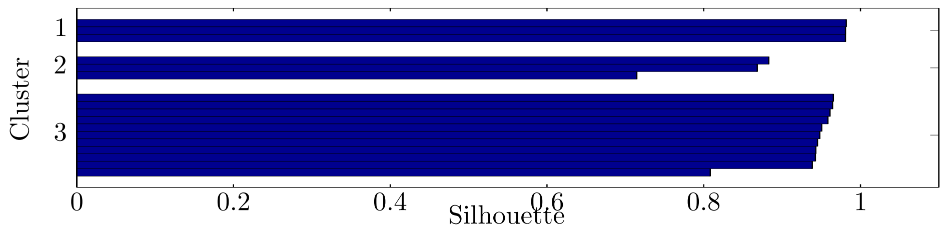


Figure 5.9: ADSR knob parameter number 2 standard deviation

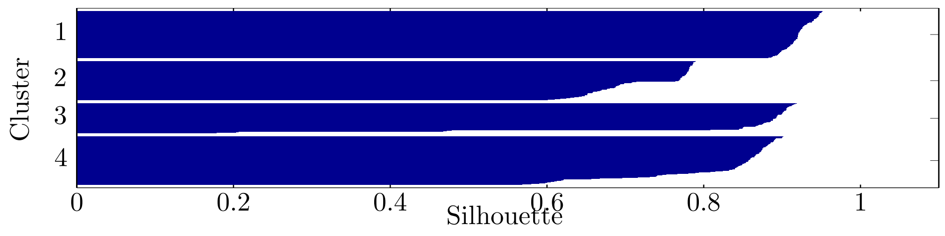
silhouette close to 1 implies the datum is in the appropriate cluster, while a silhouette close to -1 implies the datum is in the wrong cluster. We run the k-means clustering algorithm several times with increasing values of k starting from 1 to a maximum of 10. We select the value of k that gives the best average silhouette. Figure 5.10(a) and Figure 5.10(b) respectively show the silhouettes for the conf4 target and for the cat target sound that give the best average silhouette. Each bar represent the silhouette of an individual. One can observe these bars more easily on Figure 5.10(a) as the Pareto front is small for the conf4 target sound. As one can see on both Figure 5.10(a) and 5.10(b), the average silhouette is close to 1 and there is no individual with a small or negative silhouette. These plots attest that the clustering is effective in both cases.

Cluster representation

Figure 5.11(a) and Figure 5.11(b) give a 3D representation of the Pareto front over the three objectives (FFT, STFT and envelope). Each cluster has its own colour and point marker. As described in the previous section, the clustering is done on the OP-1 knob parameters



(a) Conf4



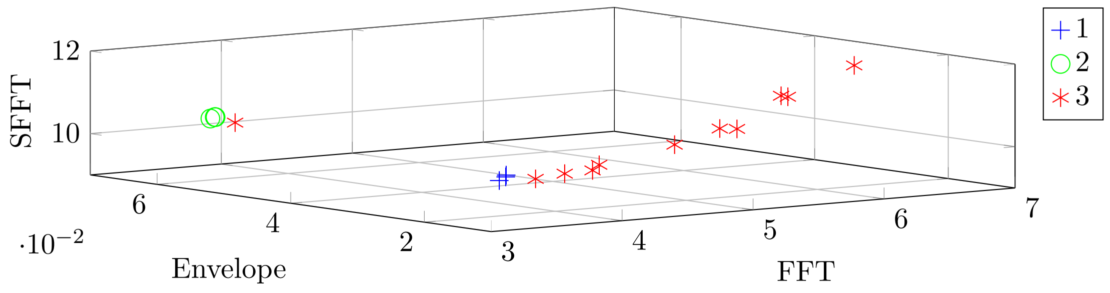
(b) cat

Figure 5.10: Silhouettes

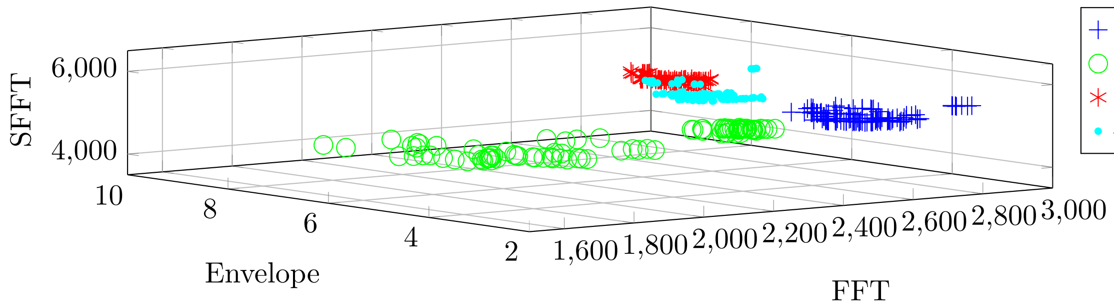
and not on the objective fitness. However, a viewer can observe that the Pareto front is also well clustered over the three objectives. It makes sense because neighbours in the OP-1 parameters space should also be neighbours in the objectives space. However, we can observe some outsiders (for example, one individual belonging to the cluster 3 on Figure 5.11(a)). It can be due to the fact that the clustering is not perfect but it can also illustrate the inverse correlation between parameter distance and objective fitnesses we found in Section 5.2.2.

Centroid individual

The individual in each cluster that is the closest to the centroid is chosen to represent its clusters. Figure 5.13(a) (respectively Figure 5.12(a)) shows a comparison between the centroid individual spectrogram and the related target sound spectrogram for the conf4 sound (respectively cat sound). For the contrived sound conf4, the two spectrograms looks perfectly similar. However, the objective valued for the FFT and STFT on Figure 5.11(a) are small but not equals to zero. So, the match is not perfect but it is very close as it is not possible to make the difference perceptually when we listen to the two sounds. For the non-contrived cat, the two spectrograms does not look as similar as it was the case for the contrived sound. With a non-contrived sounds, it is not possible to know in advance if the OP-1 synthesizer is able to replicate it. However, we can see strong similarities



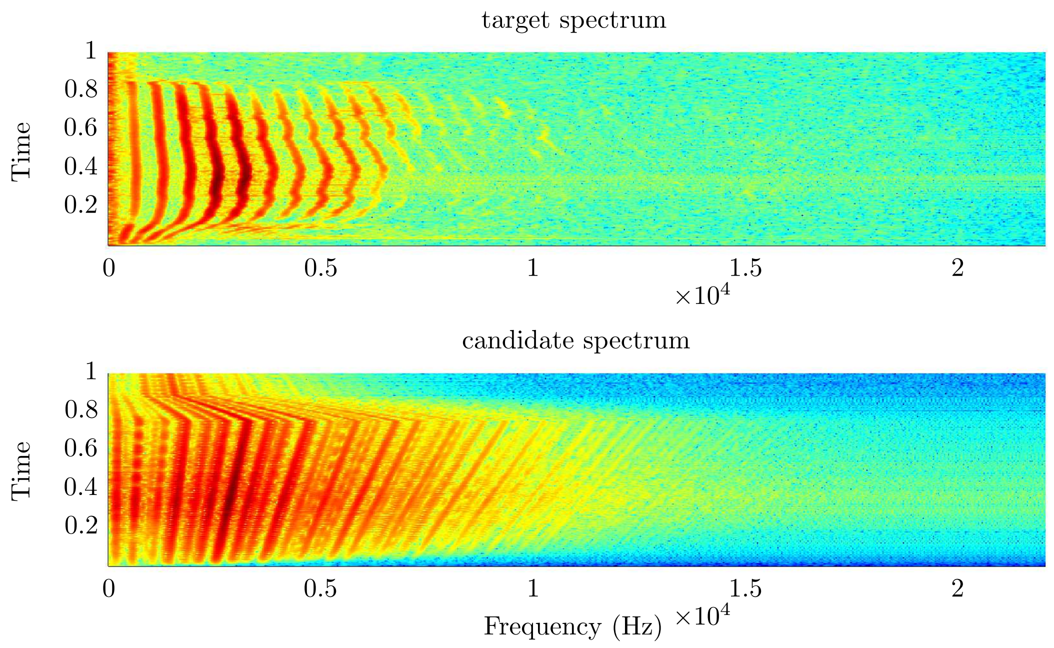
(a) Conf4



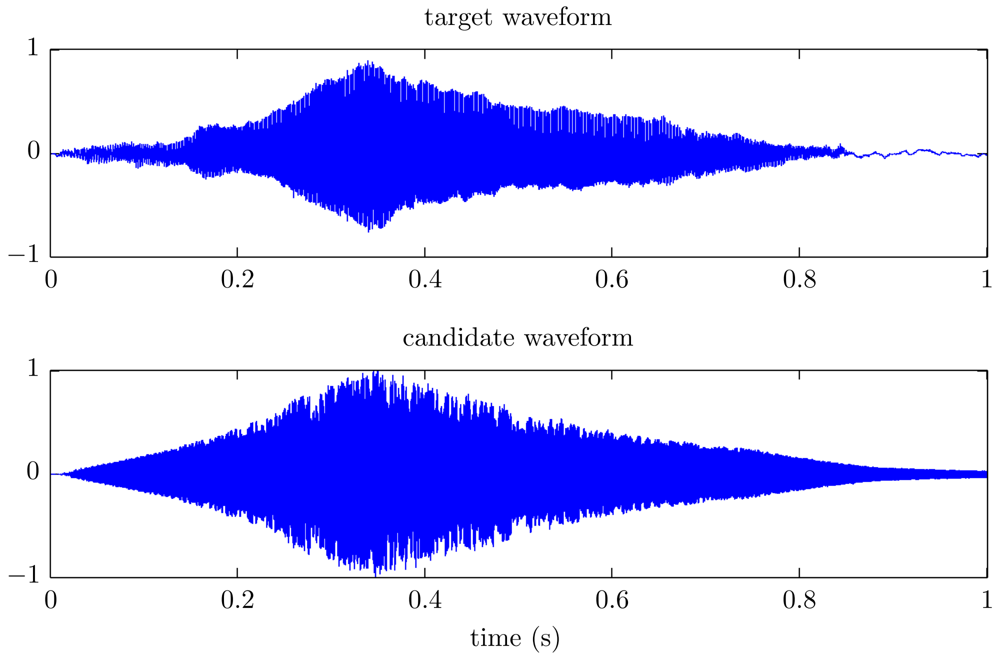
(b) cat

Figure 5.11: Pareto front

in the frequency range in both spectrograms and also in the spectral envelope. When listening to the sounds, you can also find obvious perceptual similarities. Figure 5.13(b) (respectively Figure 5.12(b)) shows a comparison between the centroid individual waveform and the related conf4 (respectively cat) target sound waveform. For both contrived and non-contrived sounds, we can see that the amplitude envelopes are either very close (cat) or looks identical (conf4).

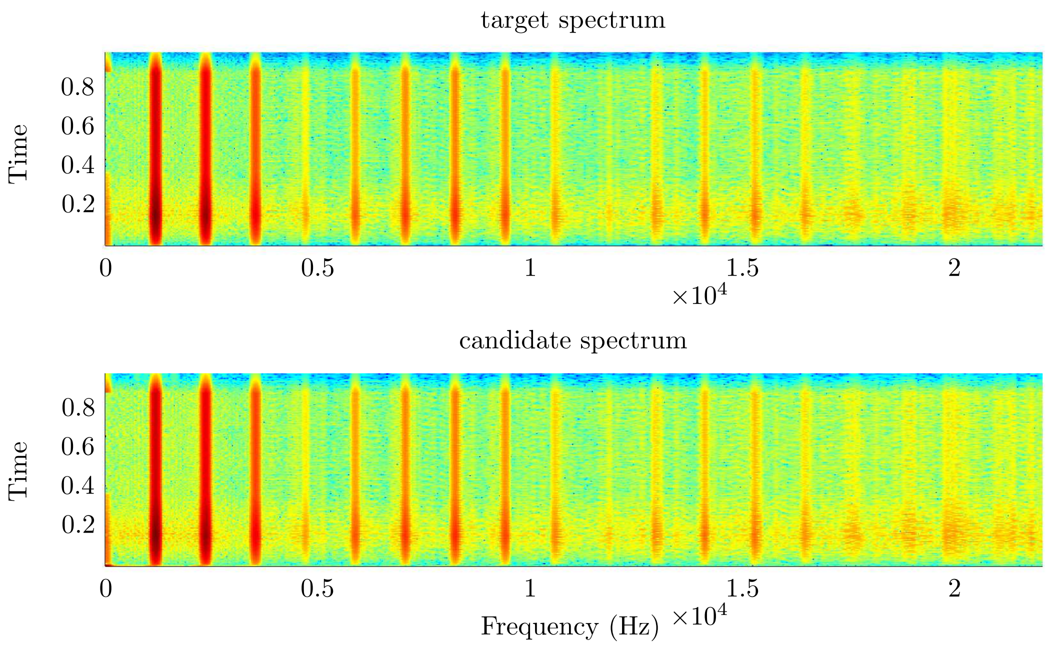


(a) Spectrograms

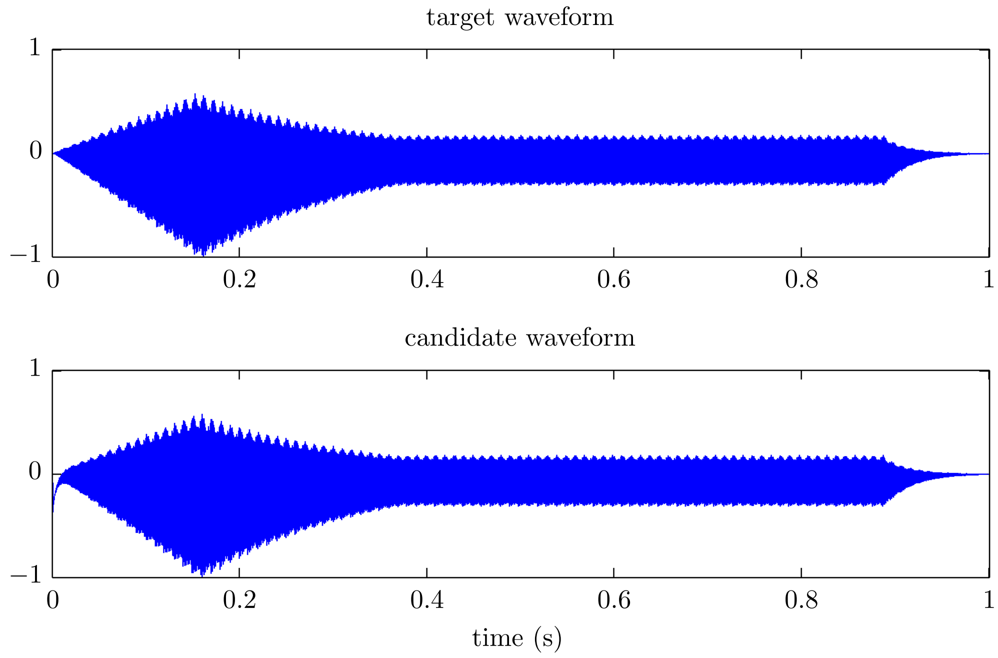


(b) Waveforms

Figure 5.12: Non-contrived sound (cat): Centroid individual for the cluster 1



(a) Spectrograms



(b) Waveforms

Figure 5.13: Contrived sound (conf4): Centroid individual for the cluster 1

5.3 Global statistics

In this section, we will describe the statistics computed on several runs.

5.3.1 Bootstrapping

We ran the algorithm at least 10 times for each target sound. We used Bootstrapping to obtain estimates of summary statistics [26]. 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 [26] 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.

5.3.2 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 10 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. For each objective, we define the best possible objectives value F_b as the minima of the objective fitness value over these 10 sounds. For each objective, we calculate the relative error Δ_r for each run subtracting this best possible objective values F_b to the best objective fitness value obtained in the particular run f_r (see Equation (5.4)).

$$\Delta_r = f_r - F_b \tag{5.4}$$

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*).

Our experimental results for contrived sounds and non-contrived sounds are available online [1].

5.3.3 Contrived sounds

Number of generations to converge

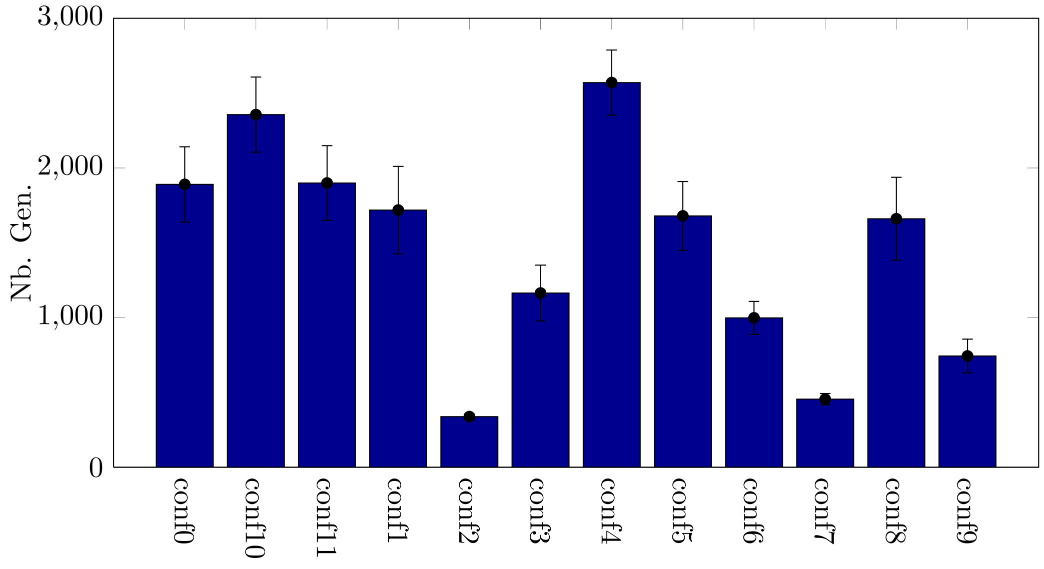


Figure 5.14: Number of generations before stopping: Contrived sounds

Figure 5.14 shows the number of generations before reaching the stopping criteria for each target configuration.

Module types selection

Table 5.3 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. The OP-1 has 24 keys on its keyboard and it is possible to change the octave from -4 to 4. There is an overlapping of 12 keys between two consecutive values of octave. It is then possible to produce the same note using two different combinations of octave and key. The last line of Table 5.3: Note takes into account this particularity and considers the selected note rather than octave and key taken separately.

Our results suggested that our system performed well at finding the right engine type (90 % prop. choice; 80 % accurate) and the right note (74 % prop. choice; 69% 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 the note 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.

Distance-Fitness correlation

The correlation between the Euclidian / Hamming distance and the objective fitnesses are a good indicator of the problem difficulty [27]. We computed the mean of euclidian / hamming distances and the mean for each of the 3 objective fitnesses for each generation. A *global correlation coefficient* was calculated between these average euclidian/hamming distances and each average objective fitnesses. A *local correlation coefficient* between the euclidian / hamming distances and the 3 objective fitnesses of the individuals of the last generation was also computed. Table 5.4 shows these correlation coefficients. The global correlation coefficients were very high for the euclidian and hamming distance. This result explains the tendency of the GA to converge quickly to a punctuated equilibrium of low fitness for the 3 objectives. This tendency is also an illustration of the algorithm ability to converge toward sounds perceptually similar to the target sound. The low local correlation coefficients showed that, once a promising location of the fitness landscape was identified, it was

	Prop. choice		**Take over gen**		**Accuracy**
	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
Note	0.74	0.38	129	273	0.69

Table 5.3: Statistics about modules types. C: Contrived sounds, NC: Non-contrived sounds

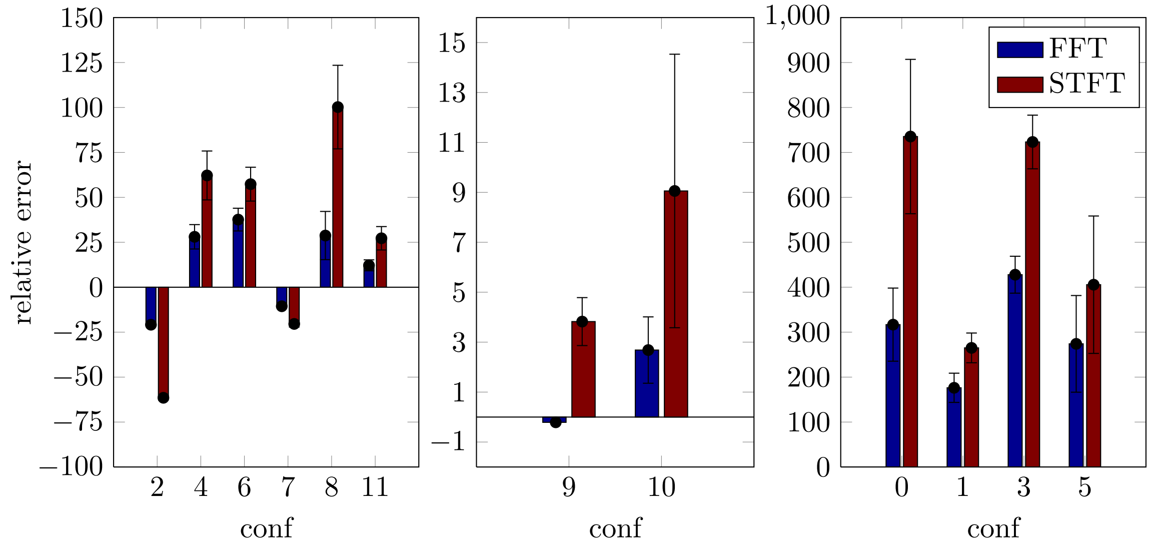


Figure 5.15: Relative error: FFT and STFT - Contrived sounds

challenging to fine tune the input parameters to converge exactly to the target parameters. One explanation is the non-deterministic nature of the synthesizer that causes noise in the evaluation making a fine tuning very challenging. Some knobs also have some inertia and different sensibility: therefore, a change in one knob can either entail a large change in the output or no change at all.

Pareto front

We call module combination, a OP-1 preset without the knob parameters. For example, {FM synthesis engine, FX delay, LFO element, key 22} is an example of module combinations. The number of distinct module combinations was very low in the Pareto front ($\mu = 3.0$, 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

		FFT		**Env**		**STFT**	
		Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**
Euclidian	**Global**	0.35	0.46	0.37	0.45	0.34	0.41
	Local	0.04	0.40	-0.09	0.43	0.09	0.45
Hamming	**Global**	0.54	0.34	0.57	0.27	0.59	0.31
	Local	0.04	0.4	0.006	0.42	0.06	0.38

Table 5.4: Mean and SD for the correlation coefficients

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. We also notice that, as wanted, each cluster contains only one Engine/LFO/FX combination. 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.

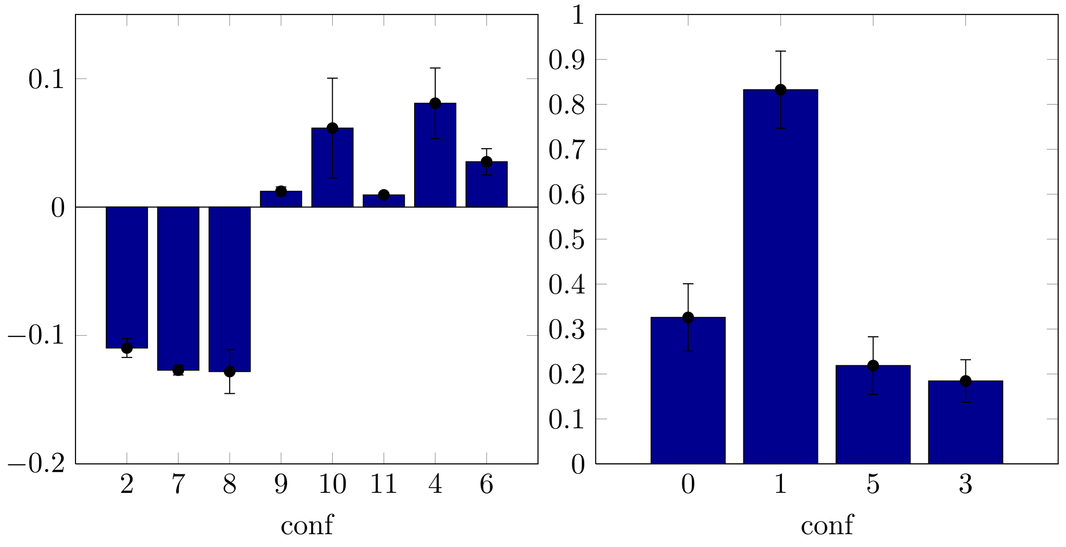


Figure 5.16: Relative error: Envelope - Contrived sounds

Our system approximates very well the amplitude envelope of the target sound as shown by the very low relative errors for the envelope objective ($\mu = 0.20$, SD = 0.02). Figure 5.15 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 dynamic 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 dynamic 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 dynamic spectrum. A second group (conf4, conf6, conf8, conf10 and conf11) contains mostly 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 dynamic. The last group (conf0, conf1, conf3, conf5) contains mostly target sounds with dynamic 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.

5.3.4 Non-contrived sounds

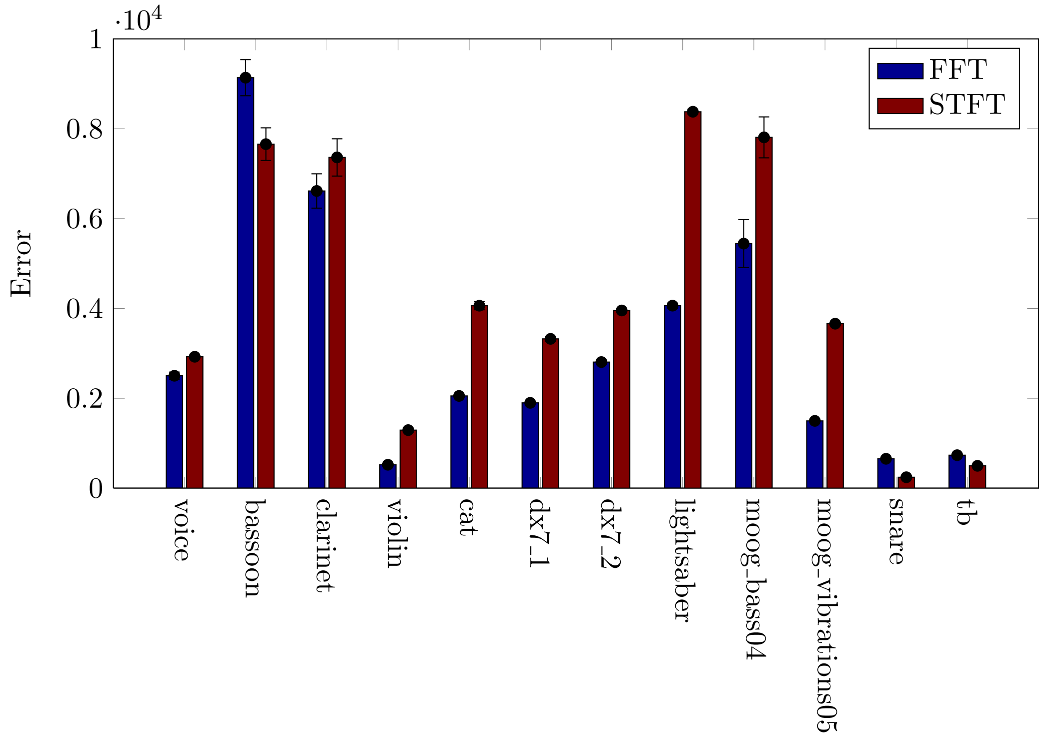


Figure 5.17: Objective fitnesses: FFT and STFT - Non-Contrived sounds

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.

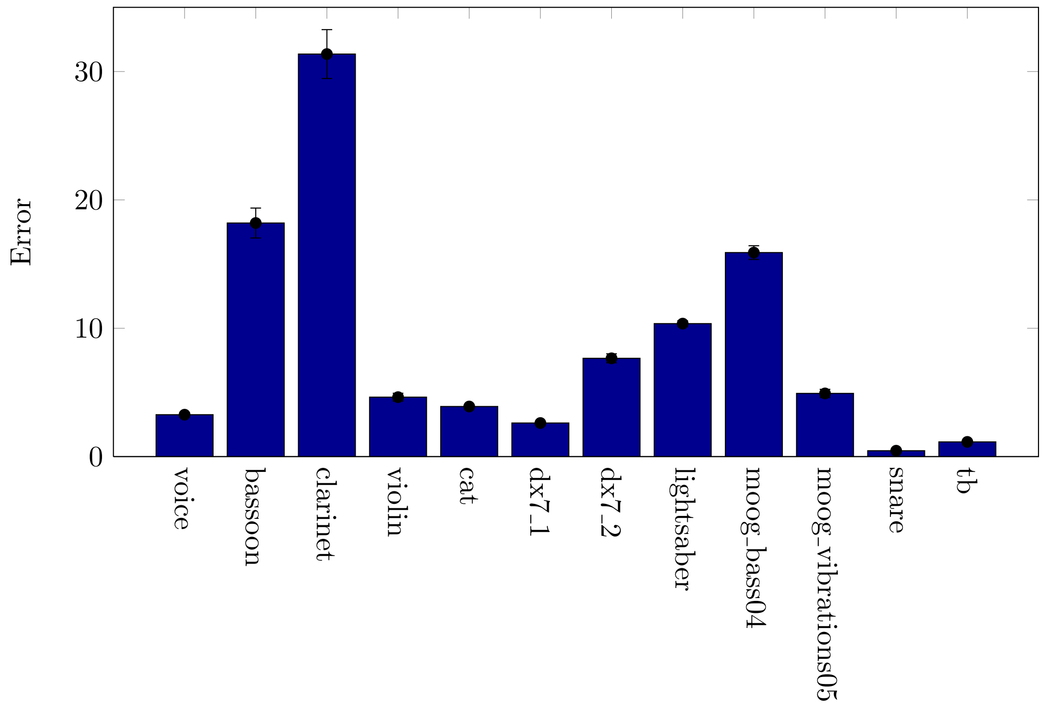


Figure 5.18: Objective fitnesses: Envelope - Non-Contrived sounds

Number of generations before stopping criteria

The mean of *nbGen*, the number of generations before reaching the stopping criteria, was significantly larger for the contrived sounds ($\mu = 1431$, SD = 86) than for the non-contrived sounds ($\mu = 1070$, 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. However, we designed our algorithm to stop if the cumulative fitness improvement is almost null for each objective (see Chapter 4, Section 4.3.5). So, the smaller *nbGen* observed for the non-contrived sounds compared to the contrived sounds can illustrate that our system reaches

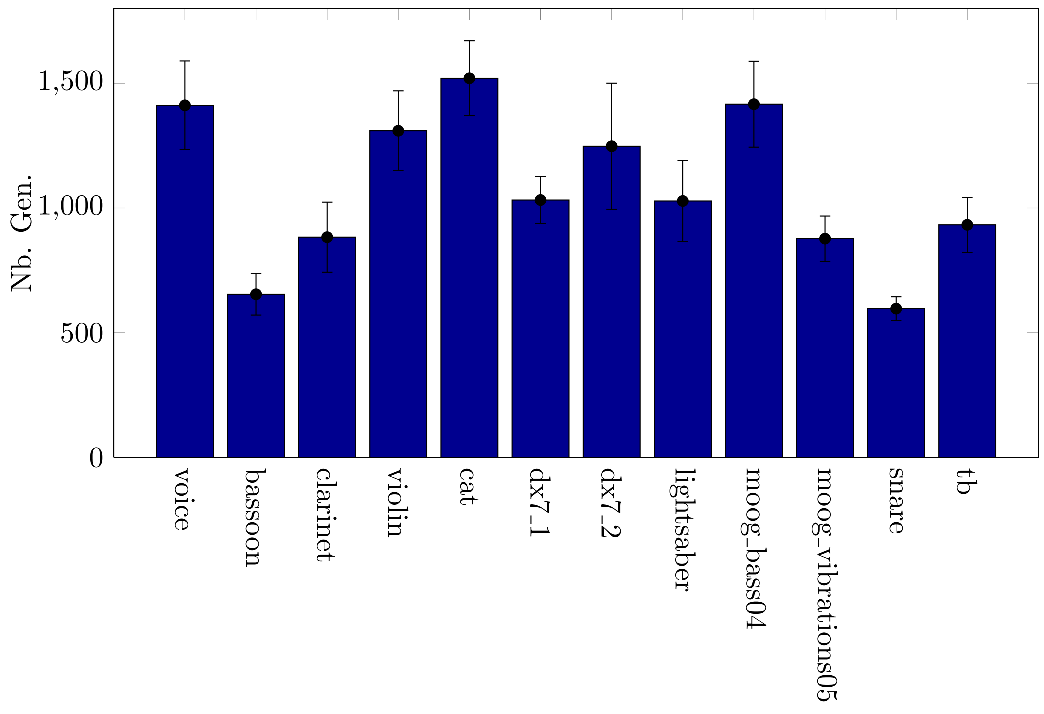


Figure 5.19: Number of generations before stopping: Non-contrived sounds

faster, for the non-contrived sounds than for the contrived sounds, a level of fitness where improvements are more difficult to obtain. Moreover, contrary to contrived sound experiments where an OP-1 preset exists that can replicate perfectly the target sound, we cannot know in advance, for non-contrived sounds experiments, which maximum level of approximation it is possible to achieve with the OP-1. It could also explain why our system reaches faster a plateau of fitness for the non-contrived sounds than for the contrived sounds because the potential improvement is more limited in the first case than in the second.

	FFT		**Env.**		**STFT**	
	Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**
C	130.34	17.15	0.20	0.02	248.92	31.24
NC	3163.5	242.67	8.66	0.78	4299.5	262.62

Table 5.5: Objective best fitness values. C: Contrived sounds, NC: Non-contrived sounds

Module types selection

Table 5.3 describes some statistics about the proportion of module types in the population through the generations. As with contrived sounds, a single 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 %).

Pareto front

Figure 5.17 and Figure 5.18 respectively show the relative errors for the FFT/STFT and for the Envelope.

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 in terms of number of type combinations (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 5.5, were, as expected, significantly worse for the non-contrived sounds than for the contrived sounds ($p < 0.001$, $p < 0.001$, $p < 0.001$).

Chapter 6

Conclusions

In this work, we focused on the application of evolutionary computation to automatize the task of tuning the parameters of the OP-1, a complex commercial synthesizer developed by Teenage Engineering, to replicate or approximate given target sounds.

6.1 Summary

In Chapter 1, we introduced the context, set the objectives and formulated the research question. We also highlighted our contributions and gave an overview of the project history as well as the methodology we adopted in this work.

A general background about evolutionary computation was presented in Chapter 3. We started by describing the canonical genetic algorithm (GA) and then discussed some theory about the notion of fitness landscape and its relationship to problem difficulty. We highlighted the limitations and challenges that can arise when using a GA to solve a particular optimization problem. Extensions and algorithmic alternatives commonly used to deal with these challenges and limitations were then described. Finally, we presented a variation of the canonical GA, that was developed to optimize multiple objective functions simultaneously. This variation - the Non-dominated Sorting Genetic Algorithm- II (NSGA-II) - was later selected in the design of our final system.

Chapter 2 provided a literature review on previous works using evolutionary computation to automatically search the parameters of synthesizers embedding various synthesis techniques. This review started with particular optimization problems of relatively low complexity to more general problems of higher complexity involving modern synthesizers that embed multiple complex synthesis engines. The optimization problem involving the OP-1 was situated among these related works.

In Chapter 4, we described the OP-1 synthesizer and analyzed the problem that results when attempting to automatize the search of its parameters to replicate or approximate a given target sound. We highlighted the main challenges raised by this problem. First, the OP-1 contains several synthesis engines, effects and Low-Frequency-Oscillators, which make the parameters search space large but also discontinuous. Interactions between parameters are common and can make the OP-1 output difficult to predict for the user. Another particular challenge was that the produced sounds are not fully deterministic. In the rest of Chapter 4, we exposed the iterative methodology we adopted in order to solve the problem. We considered several sub-problems of increasing complexity when adding more and more parameters to the search. We showed how we modified a standard GA step by step to solve these sub-problems and how it lead us to switch to using a Non-dominated Sorting Genetic Algorithm-II which incorporates a 3 objective fitness function and a Gray code encoding to handle the full complexity. Our complete solution design and its implementation was detailed at the end of this chapter.

An evaluation of our system was proposed in Chapter 5. Contrived target sounds were first used to assess the ability of our system to retrieve the target synthesizer's parameters for contrived target sounds. Our results show that our system is able 8 times over 10 to find the right synthesis engine and the right key and octave. The results are not as convincing for the FX and LFO but the Pareto Front, in most cases, sounds perceptually similar to the target sound. This observation is also emphasized by low relative errors for the FFT and STFT. The system performs also very well to approximate the amplitude envelopes as shown by low relative errors in the envelope distance. Results also show discrepancies in performance depending of the nature of the target sound, some inducing an harder optimization problem than others. The non-determinist nature of the OP-1 synthesizer made impossible the convergence to the exact target parameter values. The second part of the Chapter 5 proposed an evaluation of our system using non-contrived sounds. Our results show that the resulting Pareto fronts sounds perceptually similar to the target sounds. As expected, our experiments show that the optimization problems induced by non-contrived sounds are more difficult than the one induced by contrived sounds, as indicated by higher distance values for the 3 objectives (Envelope distance, FFT distance and STFT distance), and also a lower tendency of seeing a single synthesis engine, LFO, FX or key/octave taking over in the population.

6.2 Contributions

This thesis provides several contributions to the field of automatic parameter tuning for synthesizer.

- In Chapter 4, Section 4.3, a Non-dominated Sorting Genetic Algorithm-II (NSGA-II) is presented which incorporates a 3 objective fitness function, Gray code encoding and a modified crossover operator to preserve population diversity and enable the users to receive a set of solutions rather than a unique solution as with previous systems.
- In Chapter 4, Section 4.3.3, a 3 objectives fitness function including FFT, Envelope and STFT is developed which addresses some of the difficulties associated with the exploration of a multi-modal search space such as the OP-1 parameters space.
- In Chapter 4, Section 4.3.6, a clustering method has been developed to better analyze and explore the set of final solutions. This method is based on k-mean clustering and the silhouette methodology to set the clustering size.
- In Chapter 5, an evaluation is proposed using contrived and non-contrived sounds. The experiments revealed the capabilities of our system to optimize the parameters of the OP-1 synthesizer to approximate both kind of target sounds.

This applied work contributes to the field of sound synthesis using an evolutionary system to find OP-1 synthesizer presets to reproduce given targets sounds. Our evaluation, especially the one using contrived target sounds, will make possible to easily compare the performances of our system to the performances of future systems developed for the same purpose. In the evaluation with contrived target sounds in Chapter 5, we define the notion of global fitness/distance correlation and local fitness/distance correlation. The high global fitness/distance correlation shows that our algorithm converges, in average, to the region in the preset space where the target preset is located and the low local fitness/distance correlation explains why our system is not able to converge exactly to the target preset at the end of the optimization.

The GA system described in this thesis also contributes to the field of applications in evolutionary computation. Our system is based on the NSGA-II, a multi-objective genetic algorithm. This multi-objective approach, one that is not common in the field of automatic parameter tuning for synthesizer, has been shown to produce particularly robust results when used to find OP-1 presets to match given target sounds. Using 3 objectives (FFT, envelope and STFT) instead on only one in previous works made possible to better solve the complex, multimodal and multidimensional optimization problem that induces the OP-1. These 3 objectives combined with the intrinsic mechanisms of the NSGA-II (Non-domination sorting, diversity preservation and elitism) made possible to preserve multiple solutions located at diverse regions of the search space and therefore to avoid a premature convergence. A modified crossover operator, that prevents the recombination of two individuals with the same genotype, has also been introduced to preserve the diversity. This multi-objective approach also enable the users to receive a set of solutions (that can use different synthesis engines, LFO or FX) rather than a unique solution as with previous systems.

Our system can be used with other complex commercial synthesizers without any changes except, of course, the genotype encoding of the parameters.

6.3 Future works

This work presents an exploration of evolutionary computation applied to automatic sound matching with the OP-1 synthesizer.

6.3.1 Improvements to the system

At present, the evolution needs a large amount of computational power to determine good presets to match a given target sound. We are currently evolving population of 500 individuals over thousands of generations. A single run requires approximatively 35 min on a super computer with our algorithm distributed on 100 processors. However, as execution time was not a priority in this work, there are numerous optimizations that can be done to the system. For example, the population size and other GA parameters such as mutation and crossover probabilities or the stopping criteria could be adjusted experimentally. Another idea would be to reduce the time complexity of extracting the 3 objective fitness values for each individual of the population. For instance, an optimized temporal segmentation of the STFT could reduce the time computation but also give a more suitable measure for this objective. Finally, as the GA is revisiting some of the same OP-1 presets over the different runs, keeping a history of these presets and their related spectral and temporal informations could also reduce subsequently the computation time.

FX and LFO module types were also challenging to identify for our system. Adding another objective could be a good idea to better take into account the nature of these modules. It would be also interesting to separate the optimization of the knob parameters form the type parameters. In this perspective, using a co-evolution genetic algorithm, where one population representing the types is co-evolved with an other population representing the knob parameters, seems promising.

6.3.2 Improvements to the evaluation

Our evaluation identified discrepancies in performances using target sounds of different natures. A more in-depth study would make possible to explain these differences and would allows us to improve our system. For example, we suspect that target sounds with dynamic spectrum are more difficult to match than stationary spectrum. If it is the case, adding a objective related to this characteristic of the sound could improve the overall system performances. Our target sounds were limited all limited to 2 seconds. Increasing their lengths in a new set of experiments would be interesting to study the performance of our system for longer sounds.

It was also challenging to compare the performances of our system to other systems in the literature. Indeed, different target sounds and performance indicators are used in the different previous works. Developing a benchmarks including target sounds of different natures and performances indicators could make possible to easily compare similar systems but also to build on previous works.

Even if our experimentations during the designing phase of our system had lead us to switch from the canonical GA to a multi-objective GA (NSGA-II), it has not being proved formally that the NSGA-II out-performs the canonical GA. In this perspective, it would be interesting to do a direct comparison experiment to compare formally the performance of our NSGA-II system to a canonical GA.

An empirical evaluation could also make possible to answer the following question: can a human user do better than our system for approximating contrived (resp. non-contrived) sounds? The quality of the results could be compared qualitatively using human listenings or quantitatively using our objective fitness functions (FFT, Envelope, STFT). The time for a human to tune the OP-1 parameters could also be compared to the execution time of our system.

The relationship between our objective fitness functions and human perception could also be evaluated using perceptual listening tests. A panel of expert listeners would be asked to evaluate the perceived similarity of evolved sound matches compared to their target sounds.

6.3.3 Applications

A practical application of our system would be an online platform in which a user could upload target sounds. The presets search would be done offline using our system and the resulting OP-1 presets would be sent back to the user by email.

An adapted version of our GA system could also be integrated in an online OP-1 patch randomizer [1]. The idea here would be to use an interactive GA instead of a NSGA-II. The user would be asked to rank by preference the individuals in the population. These rankings would be used to select the individuals for mutation and crossover. It would also be possible to use our NSGA-II system for *background evolution* [37]. Here, a target sound would be loaded before any user interaction takes place. Our NSGA-II algorithm would then run in the background, attempting to match the target sound. Meanwhile, the user would interact with the system using a GUI in the *foreground*. For each generation, the individuals in the Pareto Front would migrate from *background evolution* to the *foreground evolution*.

Working on the OP-1 optimization problem gave us numerous insights on how to solve the harder Pd patches optimization problem. This last problem presents a lot of similarities to the OP-1 problem. With the OP-1, a limited number of synthesis engines, LFO and FX are accessible. With Pd, as our building blocks are fundamental synthesis components, such as oscillators or filters, it is virtually possible to generate any synthesis engines, LFO and FX. It makes the search space for the synthesis architecture subsequently more complex for Pd than for the OP-1. The number of input parameters is fixed in the OP-1 case but it can vary in the Pd case, making the search even more complex. Given the complexity difference, using the same optimization system for Pd than for the OP-1 does not look promising. However, the idea of considering a multi-objective fitness function instead of a single fitness function could be a step toward handling the complexity of the Pd optimization problem. The idea of using co-evolution to separate the optimization of the synthesis architecture from the synthesis parameters also seems promising. Limiting the number of inputs parameters for Pd makes sense in an usability perspective and would scale well with new promising

[1] `http://op-rand1.appspot.com/welcome.jsf`

Evolutionary techniques such as Cartesian Genetic Programming that is able to evolve graphs: the natural representation for Pd patches.

We hope this work will bring us closer to making synthesizers more accessible to novice practitioners and help free the musician or composer from tedious calibrations, so that they can focus on producing meaningful sounds and music.

Bibliography

[1] Experimental results. `http://metacreation.net/mmacret/SMC2013/`. Last accessed: July 2013.

[2] mda DX-10 VST instrument download webpage and documentation. `http://freemusicsoftware.org/category/free-vst/yamaha-emulator`. Last accessed: July 2013.

[3] mda JX-10 VST instrument download webpage and documentation. `http://freemusicsoftware.org/1245`. Last accessed: July 2013.

[4] Teenage Engineering website. `http://www.teenageengineering.com/`. Last accessed: July 2013.

[5] Westgrid - Compute Canada. `http://www.westgrid.ca/`. Last accessed: July 2013.

[6] XSynth-DSSI. `http://dssi.sourceforge.net/`. Last accessed: July 2013.

[7] Barbulescu, L. and Watson, J-P. and Whitley, L. Dynamic representations and escaping local optima: Improving genetic algorithms and local search. In *Proceedings of the national conference on Artificial Intelligence*, pages 879–884, 2000.

[8] Beauchamp, J. Additive synthesis of harmonic musical tones. *Journal of Audio Engineering Society*, 14(4):332–342, 1966.

[9] Beauchamp, J. Analysis and synthesis of musical instrument sounds. In *Analysis, Synthesis, and Perception of Musical Sounds*, Modern Acoustics and Signal Processing, pages 1–89. Springer New York, 2007.

[10] Beyer, H. and Schwefel, H. Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.

[11] Bozkurt, B. and Yüksel, K. Parallel evolutionary optimization of digital sound synthesis parameters. In *Proceedings of the conference on Applications of evolutionary computation*, pages 194–203, 2011.

[12] Cannam, C. and Landone, C. and Sandler, M. and Bello, Juan Pablo. The sonic visualiser: A visualisation platform for semantic descriptors from musical signals. In

Proceedings of the International Conference on Music Information Retrieval, pages 324–327, 2006.

[13] Cantú-Paz, E. Parameter setting in parallel genetic algorithms. In *Parameter Setting in Evolutionary Algorithms*, pages 259–276. Springer, 2007.

[14] Chaudhari, M. and VDharaskar, R. and Thakare, V. Computing the most significant solution from Pareto front obtained in multi-objective evolutionary. *International Journal of Advanced Computer Science and Applications*, pages 63–68, 2010.

[15] Deb, K. and Pratap, A. and Agarwal, S. and Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Evolutionary Computation*, 6(2):182–197, 2002.

[16] Fortin F. and De Rainville F. and Gardner M. and Parizeau M. and Gagné C. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, pages 2171–2175, 2012.

[17] Fujinaga, I. and Vantomme, J. Genetic algorithms as a method for granular synthesis regulation. In *Proceedings of the International Computer Music Conference*, pages 138–138, 1994.

[18] Garcia, R.A. *Automatic generation of sound synthesis techniques.* PhD thesis, MIT, 2001.

[19] Heise S. and Hlatky M. and Loviscach J. Automatic cloning of recorded sounds by software synthesizers. In *Proceedings of the Audio Engineering Society Convention*, 10 2009.

[20] Holland, J.H. Adaptation in natural and artificial systems. *Ann Arbor, MI: University of Michigan Press and*, 1992.

[21] Horner, A. *Evolution in digital audio technology.* Springer London, 2007.

[22] Horner, A. and Beauchamp, J. Piecewise-linear approximation of additive synthesis envelopes: a comparison of various methods. *Computer Music Journal*, 20(2):72–95, 1996.

[23] Horner A. and Beauchamp J. and Haken L. Machine tongues XVI: Genetic algorithms and their application to FM matching synthesis. *Computer Music Journal*, 17(4):17–29, 1993.

[24] Jin, Y. and Branke, J. Evolutionary optimization in uncertain environments-a survey. *Journal on Evolutionary Computation*, 9(3):303–317, 2005.

[25] Johnson, C. G. Exploring the sound-space of synthesis algorithms using interactive genetic algorithms. In *Proceedings of the Society for the Study of Artificial Intelligence and Simulation of Behaviour Symposium on Musical Creativity*, pages 20–27, 1999.

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

[27] Jones, T. and Forrest, S. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the international conference on Genetic algorithms*, pages 184–192, 1995.

[28] Kenneth, A D. *Jong: An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.

[29] Kinnear Jr, K.E. Fitness landscapes and difficulty in genetic programming. In *Proceedings of the IEEE conference on Evolutionary Computation*, pages 142–147. IEEE, 1994.

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

[31] Lazzarini V. and Timoney J. Theory and practice of modified frequency modulation synthesis. *Journal of the Audio Engineering Society*, 58(6):459–471, 2010.

[32] Lobo, F. and Lima, C. and Michalewicz, Z. *Parameter setting in evolutionary algorithms*, volume 54. Springer Verlag, 2007.

[33] Macret, M. and Pasquier, P. and Smyth, T. Automatic Calibration of Modified FM Synthesis to Harmonic Sounds using Genetic Algorithms. In *Proceedings of Sound and Music Computing Conference*, pages 387–394, 2012.

[34] R. Martí, M. Resende, and C. Ribeiro. Multi-start methods for combinatorial optimization. *European Journal of Operational Research*, 2012.

[35] Mathieu, B. and Essid, S. and Fillon, T. and Prado, J. and Richard, G. Yaafe, an easy to use and efficient audio feature extraction software. In *Proceedings of the International Society for Muisc Information Retrieval*, 2010.

[36] McDermott, J. *Automatic generation of sound synthesis techniques.* PhD thesis, University of Limerick, 2008.

[37] McDermott, J. and Griffith, N. and O'Neill, M. Evolutionary GUIs for sound synthesis. In *International Conference on Applications of Evolutionary Computing*, pages 547–556, 2007.

[38] McDermott, J. and O'Neill, M. and Griffith, N. Ec control of sound synthesis. *Evolutionary Computation Journal*, 18(2):277–303, 2010.

[39] Mengshoel, O. and Goldberg, D. The crowding approach to niching in genetic algorithms. *Evolutionary computation*, 16(3):315–354, 2008.

[40] Mitchell, J. M. *An Exploration of Evolutionary Computation Applied to Frequency Modulation Audio Synthesis Parameter Optimisation.* PhD thesis, University of the West of England, Bristol, 2010.

[41] Mitchell, T. Automated evolutionary synthesis matching. *Journal on Soft Computing*, pages 1–14, 2012.

[42] Mitchell, T. and Creasey, D. Evolutionary sound matching: A test methodology and comparative study. In *International Conference on Machine Learning and Applications*, pages 229–234, 2007.

[43] Mitchell, T. and Pipe, A. Convergence Synthesis of Dynamic Frequency Modulation Tones Using an Evolution Strategy. In *Applications of Evolutionary Computing*, volume 3449 of *Lecture Notes in Computer Science*, pages 533–538. Springer, 2005.

[44] Peeters, G. A large set of audio features for sound description (similarity and classification) in the CUIDADO project. Technical report, IRCAM, 2004.

[45] Riionheimo, J. and Välimäki, V. Parameter estimation of a plucked string synthesis model using a genetic algorithm with perceptual fitness calculation. *Journal on Advances in Signal Processing*, pages 791–805, 2003.

[46] Roads, C. Introduction to granular synthesis. *Computer Music Journal*, 12(2):11–13, 1988.

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

[48] Roth, M. and Yee-King, M. A comparison of parametric optimization techniques for musical instrument tone matching. In *Proceedings of the Audio Engineering Society Convention*, pages 972–980, 2011.

[49] Rothlauf, F. *Representations for genetic and evolutionary algorithms.* Springer, 2006.

[50] Rousseeuw, P. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[51] Schaffer, J. and Caruana, R. and Eshelman, L. and Das, R. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the international conference on Genetic algorithms*, pages 51–60, 1989.

[52] Schatter, G. and Züger, E. and Nitschke, C. A synaesthetic approach for a synthesizer interface based on genetic algorithms and fuzzy sets. In *Proceedings of the International Computer Music Conference*, pages 664–667, 2005.

[53] Serquera, J. and Miranda, E. Evolutionary sound synthesis: rendering spectrograms from cellular automata histograms. *Applications of Evolutionary Computation*, pages 381–390, 2010.

[54] Sivanandam, S.N. and Deepa, S.N. Genetic algorithms. In *Introduction to Genetic Algorithms*, pages 15–37. Springer, 2008.

[55] Takala, T., Hahn, J., Gritz, L., Geigel, J., and Lee, J. . Using physicallybased models and genetic algorithms for functional composition of sound signals, synchronized to animated motion. In *Proceedings of the International Computer Music Conference*, pages 180–185, 1993.

[56] Truax, B. Real-time granular synthesis with the DMX-1000. In *Proceedings of the International Computer Music Conference*, pages 138–45, 1986.

[57] Tzanetakis, G. and Cook, P. Marsyas: A framework for audio analysis. *Organised sound*, 4(3):169–175, 2000.

[58] Vuori, J. and Välimäki, V. Parameter estimation of non-linear physical models by simulated evolution-application to the flute model. In *Proceedings of the International Computer Music Conference*, pages 402–402, 1993.

[59] Wolpert, D. and Macready, W. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.

[60] Yee-King, M. and Roth, M. Synthbot: An unsupervised software synthesizer programmer. In *Proceedings of the International Conference Music Conference*, pages 1–6, 2008.

[61] Yoshimura, T. and Tokuda, K. and Masuko, T. and Kobayashi T. and Kitamura T. Simultaneous modeling of spectrum, pitch and duration in HMM-based speech synthesis. In *Proceedings of the conference on Speech Communication and Technology*, pages 1315–1318, 1999.