Norwegian University of
Science and Technology

Using Genetic Algorithms to Find and Evaluate Parameters for Adaptive Digital Audio Effects

Anders Wold Eldhuset

Master of Science in Computer Science
Submission date: June 2016
Supervisor: Gunnar Tufte, IDI
Co-supervisor: Øyvind Brandtsegg, IM

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Cross-adaptive audio effects are audio effects whose parameters are controlled or influenced by features of audio signals other than the one being affected. In other words, there is a mapping from one or several analyzed features of some signal(s) to one or more parameters of an effect processing a different signal. This thesis aims to find and evaluate such mappings using AI methods, specifically a genetic algorithm, with the evaluation of a mapping's fitness being a measure of musical applicability.

A possible design and its implementation are presented, and experiments which show that the idea is feasible are carried out.

Sammendrag

Cross-adaptive lydeffekter, er lydeffekter hvis parametre påvirkes av egenskaper ved andre lydsignal enn det signalet effekten behandler. Med andre ord finnes det en kobling mellom analysen av ett signal og parametrene til en effekt som påverker et annet signal. Denne masteroppgaven tar sikte på å finne og undersøke slike koblinger ved hjelp av metoder fra kunstig intelligens, konkret en genetisk algoritme, og å måle virkningen av en kobling ut fra musikalsk nytteverdi.

Gjennom oppgaven fremlegges et design og en implementasjon av denne, og det fremlegges eksperimenter som viser at oppgavens grunntanke er gjennomførbar.

Acknowledgements

I would like to thank my supervisors, Gunnar Tufte and Øyvind Brandtsegg, for their guidance and for the discussions we had during the course of this project.

I would also like to thank the creators and maintainers of Clojure, Csound, R, and LaTeX—as well as the open source community at large—for making their work freely available to the world.

Anders Wold Eldhuset

Contents

Abstract i

Acknowledgements iii

Contents vi

List of Tables vii

List of Figures x

List of Listings xi

1 Introduction 1
1.1 Motivation and goals 1
1.2 Related work 2
1.3 Experimental approach 2

2 Background 3
2.1 Audio and audio processing 3
2.1.1 Analog and digital audio signals 3
2.1.2 Digital audio processing 5
2.2 Artificial intelligence and genetic algorithms 6
2.3 Adaptive digital audio effects 7

3 Methodology 9
3.1 Specification of the problem and its solution 9
3.2 Genetic algorithm 11
3.2.1 Genotypes, phenotypes and mutation 12
3.2.2 Fitness 13
3.3 Implementation 15
3.3.1 Csound 15
3.3.2 Clojure 15
3.3.3 Communication between Csound and Clojure 16

4 Results 17
4.1 Static processing 17
4.2 Dynamic processing 28

5 Discussion 31
5.0.1 The results and their meaning 31
5.0.2 Further work 32

6 Conclusion 33

Bibliography 35

Appendix: Source code and audio examples 37

List of Tables

4.1 Analysis of the three audio samples used for input and as affecting signals. 30

4.2 Analysis of the three audio samples used for input and as affecting signals. In the interest of preserving space, the weights have been combined into one column; they appear in the order $w_{centroid}$, w_{cps}, w_{rms}, which is the same order as their corresponding features. 30

List of Figures

2.1 Illustration of digital sampling of an analog signal. 4

4.1 Volume control with vocals as input and drums as the affecting signal. Fitness and root mean square (RMS) amplitude values over time for 10 runs of static processing with $w_{rms} = 1$ and $w_{centroid} = w_{cps} = 0$. 18
4.2 Ring modulator and band-pass filter with guitar as input and vocals as the affecting signal. Fitness and RMS amplitude values over time for 10 runs of static processing with $w_{rms} = 1$ and $w_{centroid} = w_{cps} = 0$. 19
4.3 Ring modulator and band-pass filter with drums as input and vocals as the affecting signal. Fitness, centroid, pitch, and RMS amplitude values over time for 10 runs of static processing with $w_{centroid} = w_{cps} = w_{rms} = 1$. . . . 20
4.4 Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness and RMS amplitude values over time for 10 runs of static processing with $w_{rms} = 1$ and $w_{centroid} = w_{cps} = 0$. 21
4.5 Fitness and pitch values over time for 10 runs of static processing with $w_{cps} = 1$ and $w_{centroid} = w_{rms} = 0$. 22
4.6 Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness and centroid values over time for 10 runs of static processing with $w_{centroid} = 1$ and $w_{cps} = w_{rms} = 0$. 23
4.7 Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness, pitch, and RMS amplitude values over time for 10 runs of static processing with $w_{cps} = w_{rms} = 1$ and $w_{centroid} = 0$. 24
4.8 Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness, centroid, and RMS amplitude values over time for 10 runs of static processing with $w_{centroid} = w_{rms} = 1$ and $w_{cps} = 0$. 25
4.9 Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness, centroid, and pitch values over time for 10 runs of static processing with $w_{centroid} = w_{rms} = 1$ and $w_{cps} = 0$. 26
4.10 Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness, centroid, RMS amplitude, and pitch values over time for 10 runs of static processing with $w_{centroid} = w_{cps} = w_{rms} = 1$. 27

4.11 Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness over time for a single frame over 10 runs of dynamic processing with $w_{rms} = 1$ and $w_{centroid} = w_{cps} = 0$. 28
4.12 Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness over time for a single frame over 10 runs of dynamic processing with $w_{rms} = 1$ and $w_{centroid} = w_{cps} = 0$. 29

List of Listings

2.1 Example of a low-pass filter in Csound. 5
3.1 Example of communication between Csound and Clojure through channels. 16

Chapter 1

Introduction

This chapter introduces the subject matter and goal of this thesis. It also acknowledges some of the related work which has been done prior to this project, and briefly introduces the general approach taken in attempting to solve the problem at hand.

1.1 Motivation and goals

Adaptive digital audio effects are audio effects in which the parameters of the effect change in accordance with features extracted from from the affected signal itself, or from another audio signal. It is a relatively recent area of study; special cases of adaptive audio effects have existed for some time, and the idea has been subject to research since at least the late 1990s [1], [2]—however, the general concept was not properly defined until 2006 [3]. It is also a potentially very large area of study, bounded only by the number of features we are able to extract from audio signals, the number of audio effects imaginable, the number of parameters controlling those effects, and the limitations of the computational resources available to us. It is therefore well suited to an automated and experimental approach.

Given such a problem, with many parameters and a potentially large solution space, methods from the field of artificial intelligence (AI) seem a natural fit. One such method is the genetic algorithm, in which many possible solutions are created and evaluated for fitness, leaving the best solution or solutions as the starting point for creating new possible solutions. This process is repeated in an iterative manner until a solution is deemed "good enough", or until the algorithm encounters another criterion for termination and gives up.

It is the goal of this thesis to explore whether or not AI methods—specifically genetic algorithms—are a useful tool in the search for good configurations for adaptive digital audio effects, "good" being defined by some measure of musical applicability. In the interest of keeping the results quantifiable, musical applicability is defined, in this case, by a measure of similarity between the affecting signal and the result of the audio processing.

1.2 Related work

This thesis, while written for the Department of Computer and Information Science at the Norwegian University of Science and Technology (NTNU), was cosupervised by Øyvind Brandtsegg from the Department of Music at the same university. It is motivated and influenced by his work in [4], which introduced a tool for interactive experimentation with adaptive audio effects for musicians and music technologists. It was also preceeded by a preliminary project by the author of this thesis, which investigated the technologies and challenges involved with combining adaptive audio effects and genetic algorithms.

1.3 Experimental approach

In order to determine whether or not our goals have been satisfied—that is, whether or not genetic algorithms are capable of, and well suited to, inform the parameters of an adaptive digital audio effect—experiments will be conducted. In these experiments, a selection of different audio samples will be subjected to a selection of different audio effects. It is the influence, or weighting, of different features of the affecting audio sample upon the processing performed by the audio effect which will be varied by the genetic algorithm. Finally, we shall analyze the resulting audio output and determine whether or not it has taken on similar characteristics as those of the affecting audio sample.

Chapter 2

Background

This section introduces the fundamental concepts of audio and digital audio processing, as well as the fundamentals of genetic algorithms, so that readers with a background in one but not the other may follow the rest of the thesis. Finally, we shall introduce adaptive digital audio effects in particular as this is the class of audio effects treated in this thesis.

2.1 Audio and audio processing

Audio processing—the intentional alteration of auditory signals—is a relatively mature area of study. Ever since audio recording and broadcasting technologies began to appear in the late 19th and early 20th centuries, the desire or need to alter the audio signals being reproduced has arisen for both technical and artistic reasons. Many of the fundamental techniques and apparatuses used for audio processing today have existed for fifty years or more, and some manufacturers of audio equipment deliberately use old technologies such as vacuum tubes—or even old electrical components that were manufactured decades ago—to achieve a certain sound which they deem desirable. Even so, technological invention and improvement has played and continues to play an important role in audio processing, and this is particularly true today in the realm of *digital* audio processing, or digital signal processing (DSP) in general.

2.1.1 Analog and digital audio signals

Sound may be defined in the world of physiology or psychology as a sensation produced by certain types of atmospheric disturbances, or in the world of physics as the disturbances causing that sensation [5]. In more practical terms, we may say that sound consists of fluctuations in air pressure which, when they impact our ears, give us the sensation of hearing. Because fluctuations in air pressure are difficult to work with directly, conventional sound systems instead represent sound using fluctuating voltages. These voltages may be

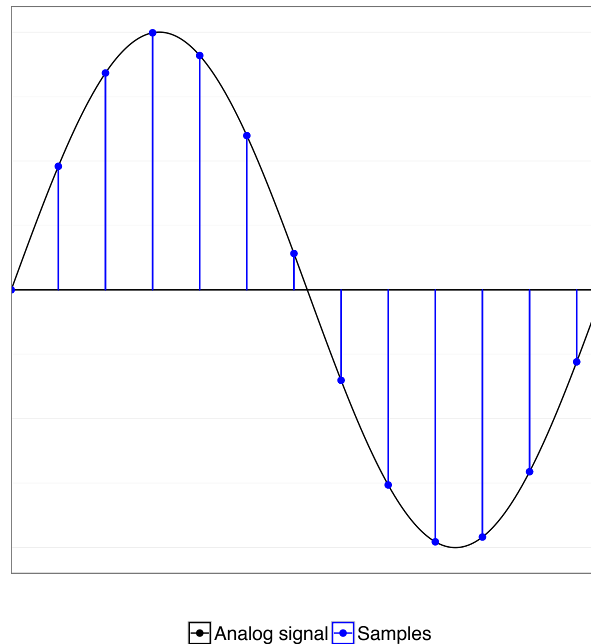

Figure 2.1: Illustration of digital sampling of an analog signal.

induced from air pressure through the use of a microphone, amplified by electrical circuits and converted back into fluctuations in air pressure by the moving cone of a loudspeaker, as is the case with a public address system.

Sound and the voltages used to represent sound are both analog signals, i.e., they are continuously variable. A digital signal, being a series of digits, is by definition discrete. Therefore, in order to treat audio in a digital system, a conversion from the analog to the digital domain needs to take place. This is done by sampling an analog signal at fixed intervals, thus creating an approximation of the original signal. The rate at which this sampling takes place is known as the sampling rate or sample rate. An example of this process is illustrated in figure 2.1. The Nyquist–Shannon sampling theorem states that an analog signal which is bandlimited with bandwidth B can be sampled at a sample rate of $2B$ and accurately reconstructed from the samples [6]. This means that the sample rate needs to be at least twice the value of the highest frequency to be sampled from the analog signal. For example, in order to recreate a sine wave with frequency B, you would need to sample at least the positive and negative peaks of the wave, requiring a sample rate of $2B$. It is considered good practice to use an even higher sample rate because the analog signal may be not be in phase with the sampling points at all times—that is, the sine wave from the previous example may not be at its peak values at the time that it is sampled.

Each sample in a digital audio signal is just a number. Given the limitations of digital computers, that number needs to be stored within a finite number of bits; the number of

bits devoted to each sample of a digital audio signal is known as the bit depth. Given that analog signals do not have these limitations, the magnitude of the analog input signal will be quantized to the nearest possible value by an analog-to-digital converter. Thus, whereas the sample rate of a digital audio system determines the range of frequencies it can reproduce, the bit depth determines the dynamic fidelity of the system.

2.1.2 Digital audio processing

Given that digital audio is represented as a sequence of numbers, digital audio effects are electronics devices or computer software which operate on sequences of numbers. As a simple example, let us consider a low-pass filter. A low-pass filter is an audio effect which attenuates frequencies above a certain threshold known as the *cutoff frequency*. Mathematically, a digital low-pass filter can be expressed as

$$y(n) = \frac{x(n) + x(n-1)}{2} \tag{2.1}$$

where x denotes an input value, y denotes an output value and n represents time measured in samples. This type of filter is also known as a first-order *moving-average filter*. The operation of this filter is quite intuitive; remember that a higher frequency is merely a faster rate of change in values. Averaging subsequent values in this way will have a greater impact if the two values are very different than if they are nearly the same. Therefore, higher frequencies are attenuated whereas lower frequencies remain largely unaffected. "First-order" referes to the fact that we introduce a delay of one sample; if we were to keep two previous samples for processing, it would be a second-order filter, and so on.

Written in the Csound programming language, the filter in (2.1) would look something like this:

```
; A simple low-pass filter.
instr 1                      ; define an instrument identified by the number 1
a1      in                   ; take some mono audio input and store it in a1
a1_1    delay1 a1            ; keep the previous sample and store it in a1_1
aout    = 0.5 * (a1 + a1_1)  ; perform the filtering
out     aout                 ; write the output audio to a file or output device
endin                        ; end instrument definition
```

Listing 2.1: Example of a low-pass filter in Csound.

Csound will be introduced in more detail in section 3.3.

2.2 Artificial intelligence and genetic algorithms

AI may be defined in several different ways. One of the definitions, proposed by R.E. Bellman in 1978 and cited in [7], is:

> [Artifical intelligence is the automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning. . .

This definition seems apt to our problem. The configuration of audio effects for musical applications is a conventionally human activity, carried out by a musician or audio engineer, which we now hope to automate.

The field of AI is wide and varied, and there are many different approaches to decision-making, problem solving, etc. The method employed in this thesis, is a genetic algorithm. Genetic algorithms are heuristic search algorithms which mimic the process of natural selection, to a greater or lesser extent. The general procedure is as follows:

1. Initialize a population of possible, but randomly generated solutions;
2. Evaluate the fitness of each individual in the population;
   (a) If one of the individuals presents a solution which is "good enough", stop and return that solution;
   (b) Otherwise, proceed to step 3;
3. Select one or more individuals as "parent individuals", based on their fitness values;
4. Mutate the parent individuals, creating a new population of possible solutions;
5. Go to step 2 unless a maximum number of generations has been reached.

The two salient points which separate such an algorithm from a random iterative search, are the notions of *fitness* and *mutation*. The fitness of an individual is a measure of how well it solves our problem; in our case, that measure is one of similarity between the affecting audio signal and the output signal. Mutation is what creates the next generation of individuals. By selecting "parents" with high fitness values, and making small changes to each of those parents in order to create the next generation, we have a greater chance of moving towards an acceptable solution than if we were to simply generate an entirely random population for each iteration of the algorithm.

Finally, it should be noted that an individual typically consists of two parts, a *genotype* and a *phenotype*. The genotype is a representation internal to the genetic algorithm, and it is the genotype which is mutated during the algorithm. The phenotype represents the actual solution pertinent to our problem domain, in our case a set of parameters for a digital audio effect. In order to evaluate fitness in a meaningful way, we need to evaluate the phenotype. Therefore, a genetic algorithm requires a function which maps a genotype to the actual solution we are interested in.

2.3 Adaptive digital audio effects

As was mentioned in the introduction, adaptive digital audio effects are audio effects in which the parameters of the effect vary in accordance with features extracted from the signal being processed, or from another signal. In the interest of clarity, a *feature* may be the amplitude of a signal, whereas a *parameter* may for example be the cutoff frequency of a filter. Multiple definitions of adaptive audio effects exist, with one definition given in [2] and another in [3] by the same authors. It is the most recent definition, given in [3], which is used in this thesis. This definition divides adapative audio effects into four different forms:

1. *Auto-adaptive effects* have their features extracted from the input signal being processed by the effect;
2. *Adaptive or external-adaptive effects* have their features extracted from at least one other input signal;
3. *Feedback-adaptive effects* have their features extracted from the output signal;
4. *Cross-adaptive effects* are a combination of at least two external-adaptive effects; their features are extracted from at least two input signals.

It is with adaptive or external-adaptive effects we concern ourselves in this thesis; in the interest of brevity, they are referred to simply as adaptive audio effects.

Chapter 3

Methodology

In this chapter, the specific design and approach which provides the background for this thesis is discussed. The particular genetic algorithm in use will be presented, as well as the decisions and limitations which affect the results. Lastly, some of the technology choices and implementation details will be presented.

3.1 Specification of the problem and its solution

While the study of adaptive audio effects provides room for a great deal of variation and experimentation, restrictions are necessary in order to conduct a practical experiment and achieve meaningful results. In the experiments presented in thesis, those restrictions are:

- There are three audio samples which, in different configurations, function as input and affecting signals to the adaptive audio effects:
  - A vocal track;
  - A guitar track;
  - A drum track.
- Three different audio effects have been implemented:
  - A simple volume control, or envelope follower;
  - A hybrid effect of an assymetrical distortion unit whose output is fed through a resonant low-pass filter;
  - A hybrid effect of a ring modulator whose output is fed through a band-pass filter, as well as a final gain stage to make up for a potentially great loss of energy through the filter.

- Three different features are extracted through audio analysis and considered by the fitness function:
  - The spectral centroid, or simply centroid, of a signal—this is a weighted mean of the frequencies present in the signal, which indicates the perceived "brightness" of the sound;
  - The average pitch, or cycles per second (CPS), of the signal;
  - The RMS amplitude of the signal, giving an indication of perceived loudness.
- In the search for an acceptable solution, each of the analyzed features has a weight associated with it, allowing the different features to be given different levels of importance.

Furthermore, for most of the duration of this project, audio processing was limited to one set of parameters for the entirety of a session. That is to say, the genetic algorithm finds a set of parameter settings to be applied to an effect for the entire duration of an audio sample being processed, such that the result will, *on average*, take on similar characteristics to the affecting signal. The analysis of audio samples also works with averages, producing the statistical means of the centroid, CPS, and RMS amplitude values measured from an audio sample, rather than vectors of each individual measurement. This choice was made in order to investigate whether or not the genetic algorithm would be able to find reasonable effect settings at all, and so that analysis of the results would give a clear picture of the algorithm's performance.

This worked well from an AI perspective, and the results were enlightening. The approach also satisfies the general description of adaptive audio effects given in section 2.3. However, one could argue that an effect whose parameters are adapted to a particular input and affecting signal, but whose settings remain unchanged throughout the processing of a signal is not truly adaptive in the intuitive sense. It is also unlike the work done previously in [4]. Therefore, the software was expanded such that it can also produce an output in which the parameters of an effect change in accordance with the features of the affecting signal over time, at a fixed rate.

The way in which the software was altered to allow for effect parameters to change over time, is that the existing code was reused in an iterative manner. Given an input audio sample, that sample is divided into n frames of equal duration d. The genetic algorithm then processes each frame individually, accumulating a set of parameter settings for each frame. Finally, the entire audio sample is processed, changing the settings of the audio effect for each frame. This is conceptually equivalent to slicing the audio sample into n audio samples of duration d, running the software on each sample individually, and concatenating the results.

For the remainder of this thesis, solutions created using a single set of parameter settings for the duration of the audio processing will be referred to as "static", or statically processed, whereas solutions created by slicing the input into frames and processing each one individually will be referred to as "dynamic".

3.2 Genetic algorithm

In section 2.2, we looked at genetic algorithms in general. The particular variant of genetic algorithm used in this thesis, is the $(\mu + \lambda)$ Evolutionary Strategy [8], with $\mu = 1$ and $\lambda = 4$. The algorithm works as follows:

1. Create a randomly generated individual, the parent, and evaluate its fitness;
2. If the parent's fitness presents an acceptable solution, or if evolution has stagnated, stop and return the parent solution;
3. Generate four new individuals, the children, through mutation of the parent;
4. Evaluate the fitness of the children;
5. If any of the children has a fitness value greater than *or equal to* that of the parent, then the child with the best fitness value becomes the new parent;
6. Go to step 2.

Stagnation, as referred to in step 2, means that the algorithm has run for a given number of generations without (significant) change. It is an alternative to an absolute maximum number of generations, which allows the program to run for a long time if the situation is still improving, but terminates it if the algorithm has run to a halt. This is not a part of the canonical $(\mu + \lambda)$ algorithm, but a variation for this particular application. The number of generations without change that pass before a run is considered stagnant, may be passed to the program as a command line argument; the default value is 100.

Note that the criterion for a child becoming the new parent, is that its fitness value should be greater than or equal to the current parent's fitness value. This allows for some exploration through neutral change which would not occur if the criterion were strictly "greater than". The algorithm is otherwise exploitive, in that it does not pursue solutions worse than the current parent in order to explore the search space.

3.2.1 Genotypes, phenotypes and mutation

The genotype representation used in the implementation is a 64-bit number, or a bit string. This is true of all configurations. However, the mapping from genotype to phenotype necessarily varies depending on which effect is used for the audio processing. This is due to the fact that the different effects not only have different parameters, but also different numbers of parameters, and it is these parameters which affect the audio processing and thus make up the phenotype.

The implemented effects and their parameters are:

1. Volume control / envelope follower
   (a) Gain
2. Asymmetrical distortion / low-pass filter
   (a) Distortion coefficient a
   (b) Distortion coefficient b
   (c) Cutoff frequency
   (d) Resonance
3. Ring modulator / band-pass filter
   (a) Modulation frequency
   (b) Modulation depth
   (c) Center frequency
   (d) Bandwidth
   (e) Post gain / makeup gain

The gain and filter frequency parameters should be fairly self-evident, but a brief note explaining the other parameters is in place. The a and b coefficients of the distortion effect determine at which threshold the input signal begins to clip (i.e., distort) on the positive and negative sides of the signal—it is this feature which makes the effect "asymmetrical". The resonance of the low-pass filter determines to what, if any, degree the signal is boosted at the cutoff frequency, effectively creating a resonant peak around that frequency. Finally, a ring modulator operates on two signals, producing the sum and difference of their frequencies; the modulation frequency of the effect is one of those frequencies, the input signal carrying the other, whereas the modulation depth determines how much the input signal is modulated—it is effectively a "mix" control.

Given that a genotype is always a string of 64 bits, whereas the three effects feature one, four, and five parameters respectively, it is clear that the mapping from genotype to phenotype must be different. What may not be immediately apparent is that it also makes sense for their mutation functions to differ. The simple "volume control" features one parameter whose domain is $[0, 1]$. Mapping an entire 64 bits to such a small range may result in

extremely small changes through mutation, effectively making the search space for the genetic algorithm much larger than it need (should) be. Using and mutating only part of the genotype, 16 bits in this case, solves this problem without requiring different data types for the different effects' genotypes. The ring modulator / band-pass filter, on the other hand, needs to map those same 64 bits to five parameters, some of which have fairly large domains, and so it makes sense to use the entire bit string.

For the reasons given above, both the mappings from genotype to phenotype and the mutation functions are part of each effect's data structure, which is, in turn, defined by a protocol[1] allowing for a generic, polymorphic treatment of effects from the rest of the code. However, although they differ, each of the mutation functions does operate on the notion of "flipping" randomly selected bits; they only differ in how many bits they consider, and how many bits may be flipped.

3.2.2 Fitness

The purpose of a fitness function is to determine how well a given solution solves the problem at hand. As was mentioned in the introduction, the problem considered by this thesis is to find parameter settings for audio effects which satisfy some definition of musical applicability. As has also been mentioned, that definition is, in this case, a measure of similarity. In other words, the fitness function should, given an affecting signal and the result of processing an input signal with the effect parameters of the solution under consideration, determine how similar the two are.

In practice, the fitness function does not operate directly on audio files or signals, but on analyses of these. The affecting signal is analyzed only once, as it does not change, whereas the processing of an audio input with a given set of effect parameter settings must be performed, and subsequently analyzed, for each possible solution. However, these are implementation details; the fitness function operates on the analyses of two different audio signals—each a vector of the average centroid, pitch and amplitude values for that signal—as well as an associated weight for each feature, also a vector of three values. We shall call these vectors $\mathbf{a_a}$, $\mathbf{a_r}$, and $\mathbf{w}$, respectively, for "affector analysis", "result analysis" and "weights".

The weights are normalized in the fitness function, so that their sum is always 1. If we can assume that the difference between the two analyses $\mathbf{a}_{diff}$ is also normalized, then the problem of applying different weights to different features and computing a fitness value f can be solved using the dot product of the weight and difference vectors:

$$f = 1 - \mathbf{w} \cdot \mathbf{a}_{diff} \tag{3.1}$$

This seems like a sensible solution, but it remains for us to compute $\mathbf{a}_{diff}$. Because the RMS amplitude is a value between zero and one, whereas the centroid and pitch frequencies are values which may cover the audio range (roughly 20 Hz to 20 kHz), we also need to

[1]A protocol in Clojure is similar to an interface in Java and some other popular programming languages.

scale the values to avoid the influence of amplitude being “drowned out” by the numerically greater frequency values. At first attempt, this was solved using the relative differences of the elements of each analysis. Given that each of the analysis vectors is of the form

$$\begin{bmatrix} a_{centroid}, & a_{cps}, & a_{rms} \end{bmatrix}$$

then

$$\mathbf{a}_{diff} = \begin{bmatrix} d(a_{a,centroid}, a_{r,centroid}), & d(a_{a,cps}, a_{r,cps}), & d(a_{a,rms}, a_{r,rms}) \end{bmatrix} \tag{3.2}$$

where d is the relative difference function:

$$d(x, y) = \frac{|x - y|}{\left(\frac{|x|+|y|}{2}\right)} \tag{3.3}$$

This doesn’t guarantee that the values lie between zero and one, but it does bring the three different values into the same order of magnitude. This allows us to normalize them, by simply dividing each value by the sum of the values, so that they sum to one.

This approach worked well when one or two of the weights were zero, in other words, when one or two of the features were ignored. However, it has some unfortunate mathematical characteristics which became apparent with experimentation, and lead to it being replaced. Take, for example, the case where $w_{centroid} = w_{cps} = w_{rms} = k$. Recall that the weights will be normalized, so $k = \frac{1}{3}$. Then, because $\mathbf{a}_{diff}$ has also been normalized so that it always sums to one, $\mathbf{w} \cdot \mathbf{a}_{diff} = \frac{1}{3}$. That is, regardless of the information present in the analyses, the fitness value will always be $1 - \frac{1}{3} = \frac{2}{3}$. Clearly, this is unacceptable. However, this approach did yield solid results when focusing on only one or two features, which is the reason it is included in this section.

The approach which has been used to produce the results in this thesis, is to simply divide the absolute difference of each feature pair by the maximum value of their domains. For the RMS amplitude, that value is simply one, as that is the maximum amplitude value a signal can assume. For the centroid and pitch frequencies, that value has been set to 20 000, at the high end of the audible range of frequencies. This approach yields similar results to the one described above, performing slightly worse in some cases, but more reliably across different configurations of weights.

3.3 Implementation

The source code for this project is written in two different languages, Csound and Clojure. Csound handles the audio processing and analysis, whereas Clojure is used for the genetic algorithm and other parts of the program.

3.3.1 Csound

Csound is a powerful domain-specific language for audio processing originally created in the 1980s, inspired by the MUSIC-N family of languages which were developed from the late 1950s onward. It is not, however, a general-purpose programming language, hence it is only used for audio analysis and processing in this project.

Csound is arguably not just a language; it features one language for defining the so-called *orchestra*, and another language for defining the *score*. The orchestra section contains *instruments*, components which may accept audio or control signals as input, as well as emit such signals as output. The score contains instructions for calling the instruments defined in an orchestra, specifying the time at which they should become active, the duration of their activity, as well as other parameters they may accept. Both of these languages are provided as input to the Csound program, which parses and executes the instructions defined in the source code, as well as providing a standard library of audio functionality, and the underlying input and output procedures necessary for handling files, audio devices, MIDI communication, etc. In this project, only the orchestra section is provided as pure Csound code; the score events are created dynamically from the host program, written in Clojure, and passed to Csound at runtime. This allows us to alter effect parameters in real-time, as well as avoid the overhead of starting and stopping the Csound program for each iteration of the genetic algorithm.

Aside from being one of the most popular languages for research into audio processing today, Csound is also the language which was used in [4], and is likely to be the language of choice in any related future work at NTNU. It was therefore decided that it was a sensible choice for this project as well.

3.3.2 Clojure

The code which concerns itself with the genetic algorithm, writing analyses to log files, etc., is all written in Clojure. Personal preferences aside, Clojure is well suited for a project such as this one for a few reasons:

- Clojure is fast. The definition of "fast" will vary depending on the application in question, as well as personal opinion. However, code written in Clojure runs approximately as fast as code written in Java[2], which is quite good by current standards, and certainly adequate for this project.

[2] `https://benchmarksgame.alioth.debian.org/u64q/clojure.html`

- Clojure is a modern addition to the LISP family of languages. LISP was created in the 1950s by John McCarthy, the man who coined the term "artificial intelligence", and it has been closely associated with AI ever since. Influenced by these historical and cultural circumstances, Clojure is well suited for AI applications, providing concise and idiomatic solutions to many of the practical problems which arise in AI code.
- Clojure runs on the Java Virtual Machine, and is able to make use of libraries written for Java. The Csound program features an API which is available for many languages, one of them being the Java. As such, it is possible to communicate with a running Csound instance from Clojure, which was an important feature for this project.

Clojure is also one of the languages the author is most familiar with, and so, combined with the points given above, it seemed a natural choice.

3.3.3 Communication between Csound and Clojure

The Csound and Clojure parts of this project communicate over Csound channels. Channels, in Csound, are a construct which allow both the host (Clojure) and the client (Csound) to read and write to a common reference. This means that the AI code written in Clojure can pass effect parameters to the audio processing code written in Csound, and vice versa for audio analysis. In practice, it looks something like this:

```
; Set the cutoff frequency on a channel in Clojure:
(.SetChannel csound-instance "cutoff" 761)

; Get the cutoff frequency from a channel in Csound:
kcutoff  chnget "cutoff"
```

Listing 3.1: Example of communication between Csound and Clojure through channels.

Note that Csound channels are asynchronous, and that Csound and its host program run as separate processes, each with their own concept of time; this can lead to interesting race conditions if one is careless.

Chapter 4

Results

In this chapter, we shall examine some of the data resulting from the experiments conducted.

4.1 Static processing

The following pages contain graphs illustrating the results of the genetic algorithm in a static context, i.e., with one set of parameters for the duration of the run. First, we shall consider a few different configurations of audio effects and input signals, so that we may assess the overall performance of the genetic algorithm. Following that, each combination of weights, given only zero and one as possible values, are presented for a single combination of audio effect and audio signals. This allows us to study the different biases, if any, exist for a particular feature or set of features.

Note that while the perception of both pitch and loudness is often treated logarithmically, neither of the units involved in these experiments are logarithmic, nor is there any information about these perceptions encoded in the genetic algorithm. Therefore, the graphs are presented with linear scales. The dotted horizontal lines, where present, represent the analyzed values of the affecting signal, i.e., the target values for the genetic algorithm.

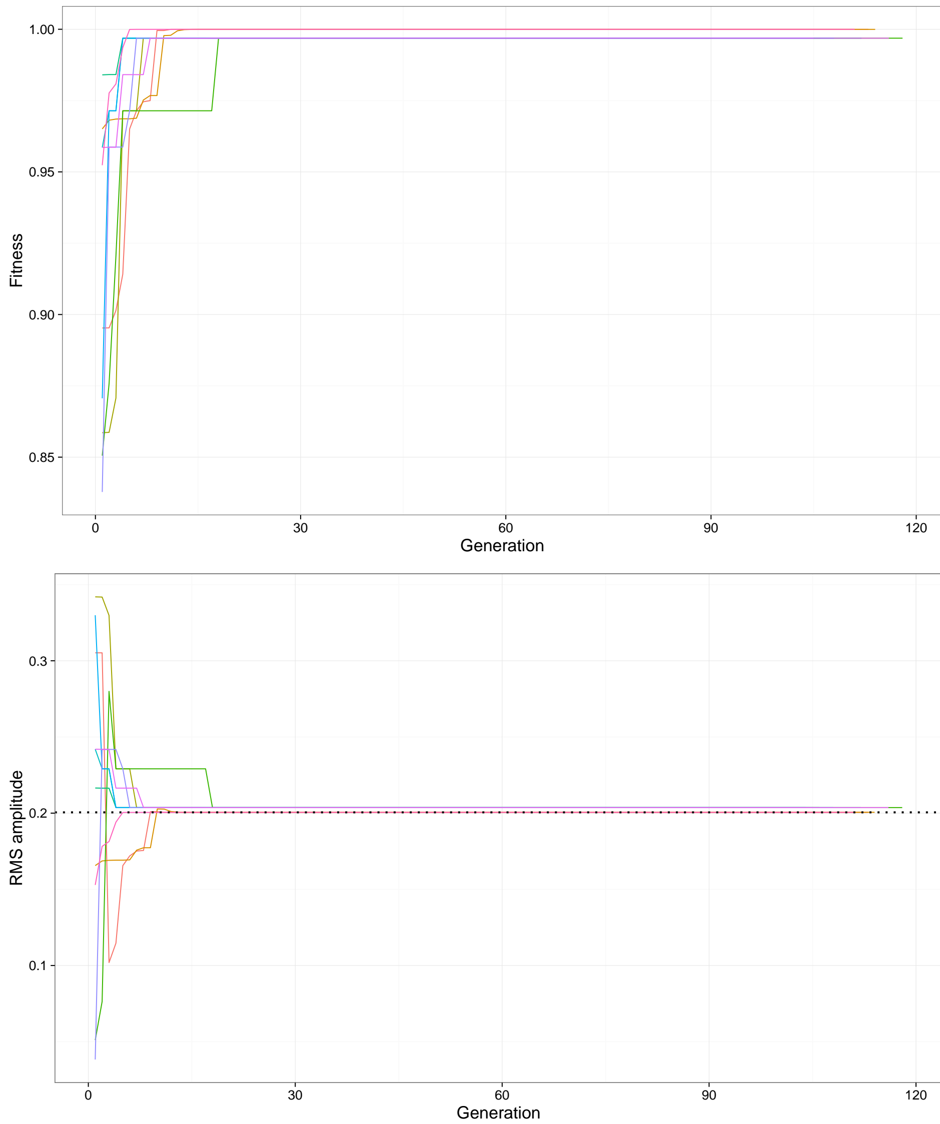

Figure 4.1: Volume control with vocals as input and drums as the affecting signal. Fitness and RMS amplitude values over time for 10 runs of static processing with $w_{rms} = 1$ and $w_{centroid} = w_{cps} = 0$.

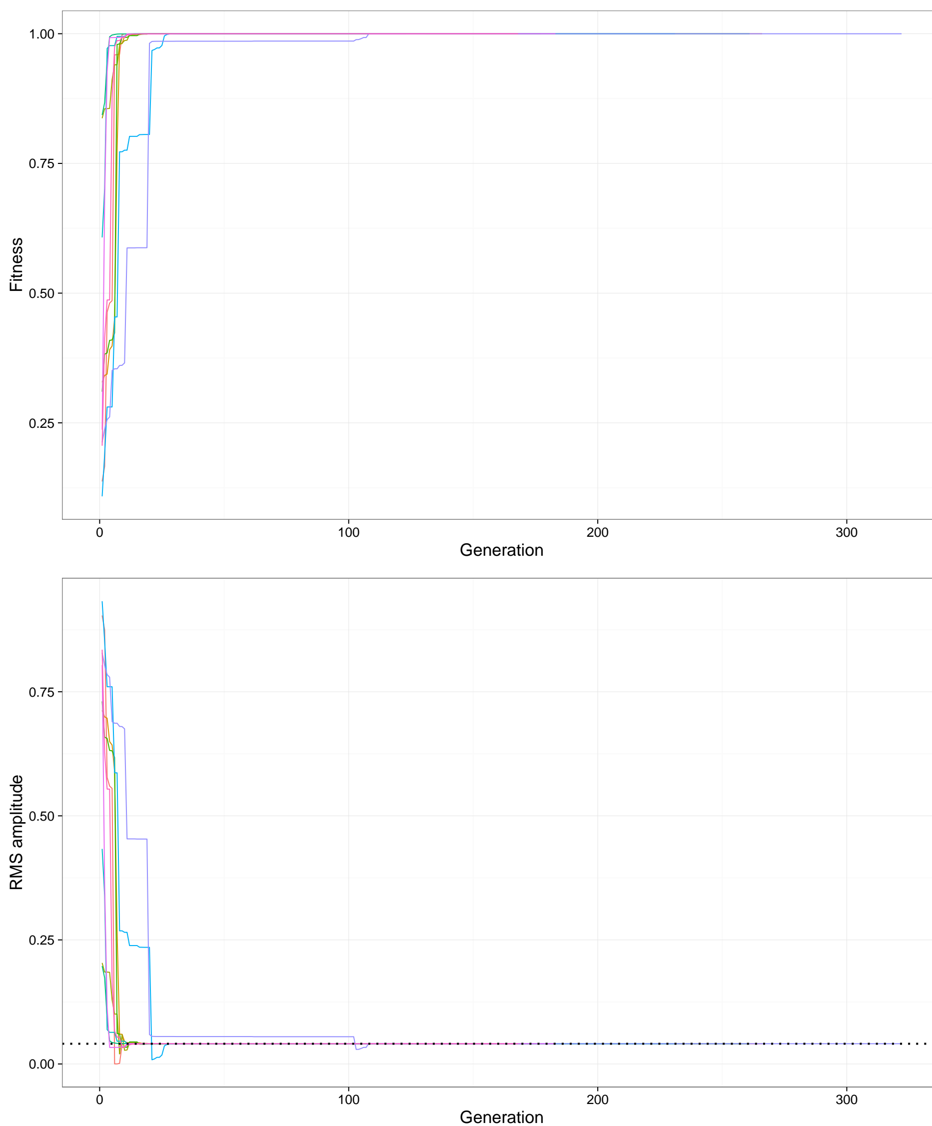

Figure 4.2: Ring modulator and band-pass filter with guitar as input and vocals as the affecting signal. Fitness and RMS amplitude values over time for 10 runs of static processing with $w_{rms} = 1$ and $w_{centroid} = w_{cps} = 0$.

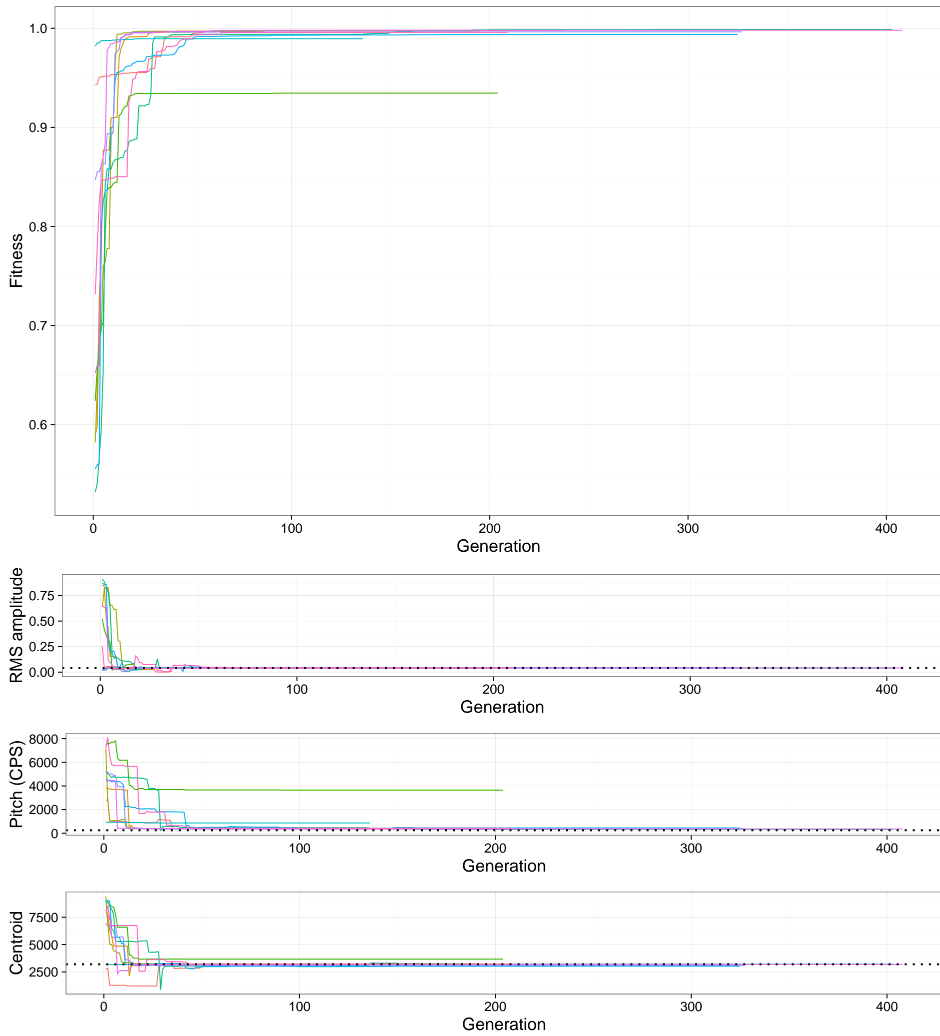

Figure 4.3: Ring modulator and band-pass filter with drums as input and vocals as the affecting signal. Fitness, centroid, pitch, and RMS amplitude values over time for 10 runs of static processing with $w_{centroid} = w_{cps} = w_{rms} = 1$.

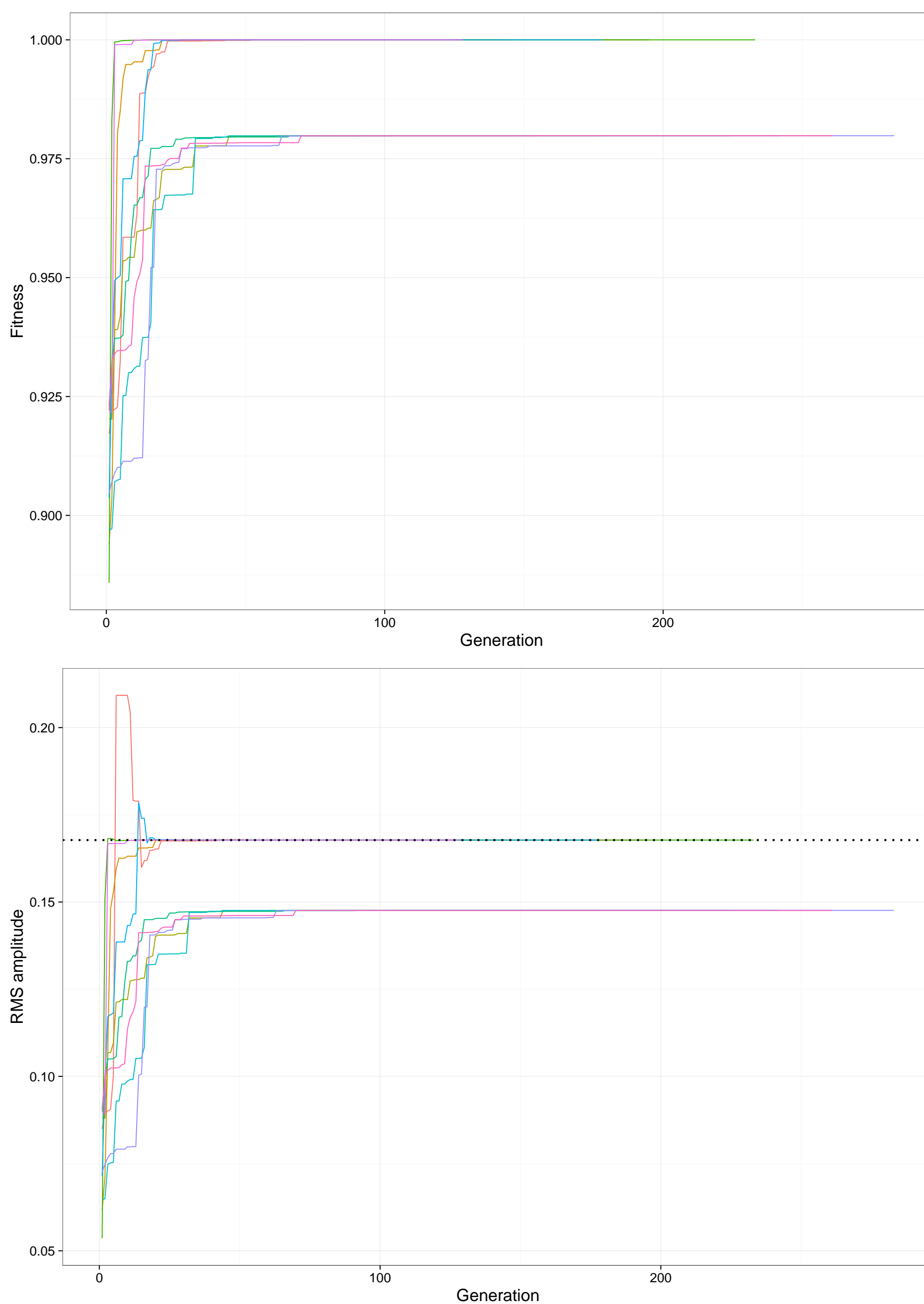

Figure 4.4: Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness and RMS amplitude values over time for 10 runs of static processing with $w_{rms} = 1$ and $w_{centroid} = w_{cps} = 0$.

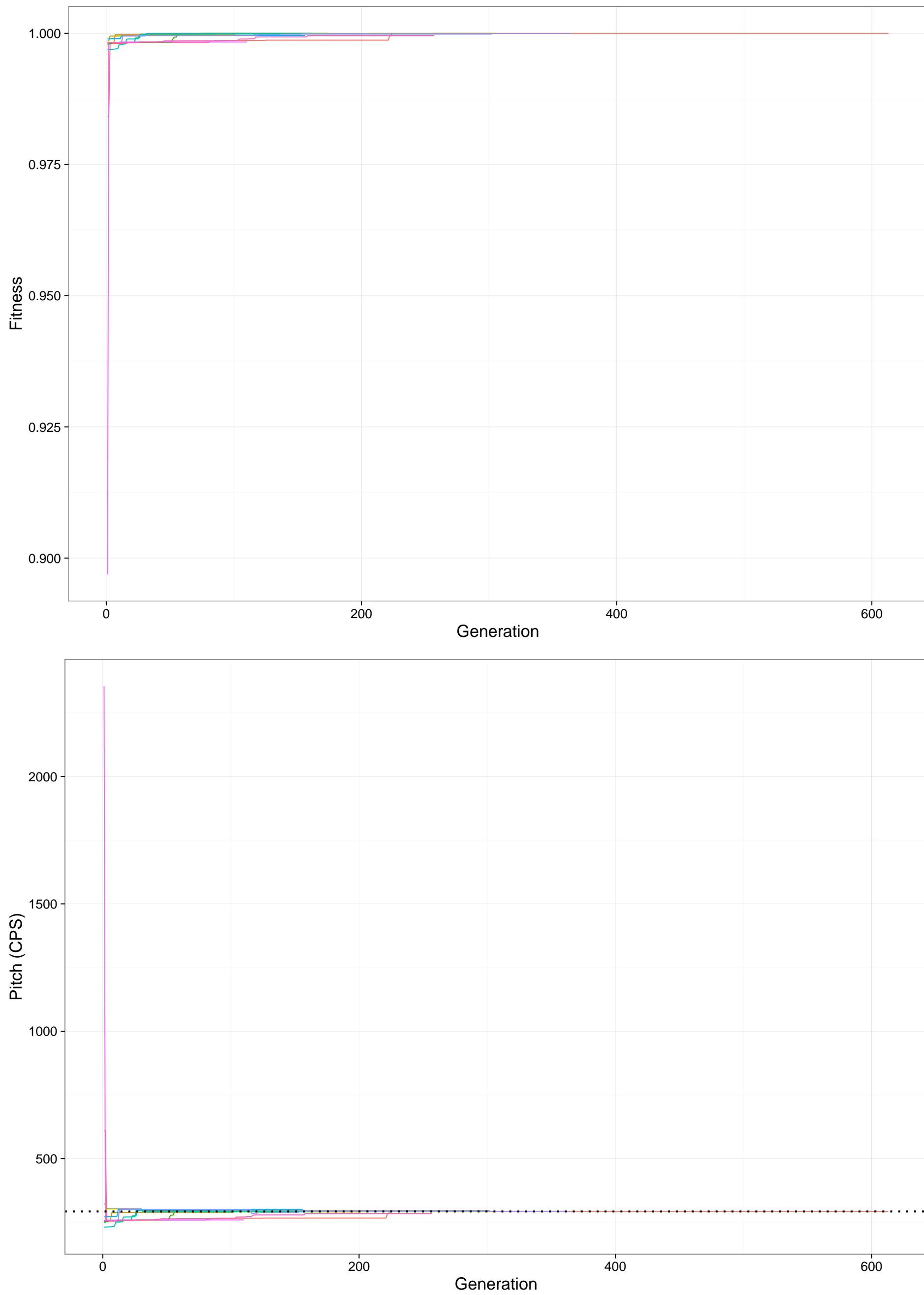

Figure 4.5: Fitness and pitch values over time for 10 runs of static processing with $w_{cps} = 1$ and $w_{centroid} = w_{rms} = 0$.

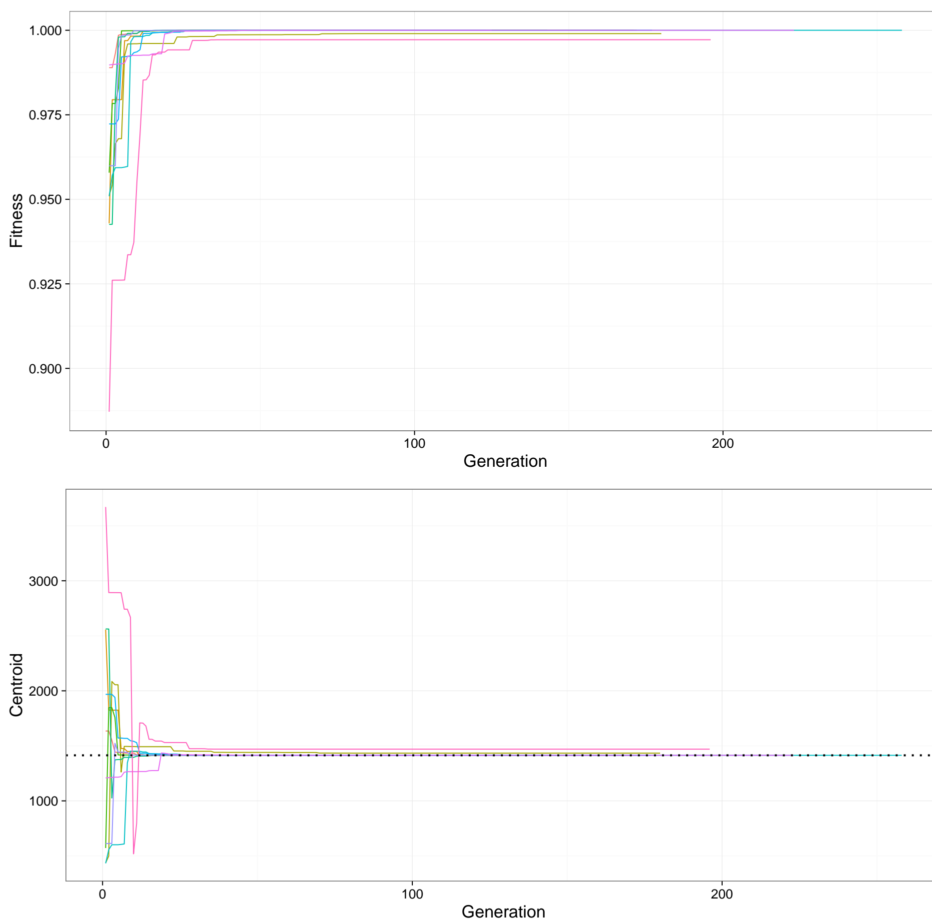

Figure 4.6: Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness and centroid values over time for 10 runs of static processing with $w_{centroid} = 1$ and $w_{cps} = w_{rms} = 0$.

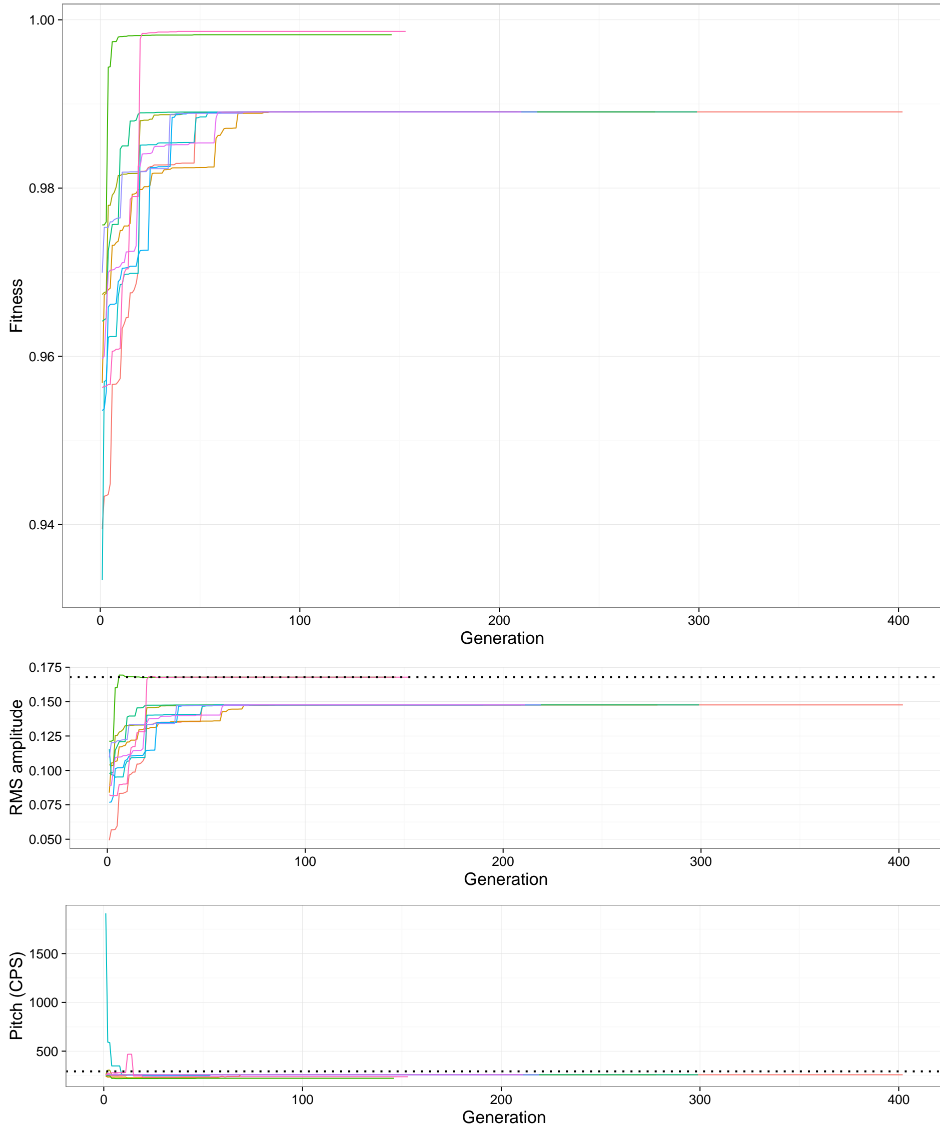

Figure 4.7: Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness, pitch, and RMS amplitude values over time for 10 runs of static processing with $w_{cps} = w_{rms} = 1$ and $w_{centroid} = 0$.

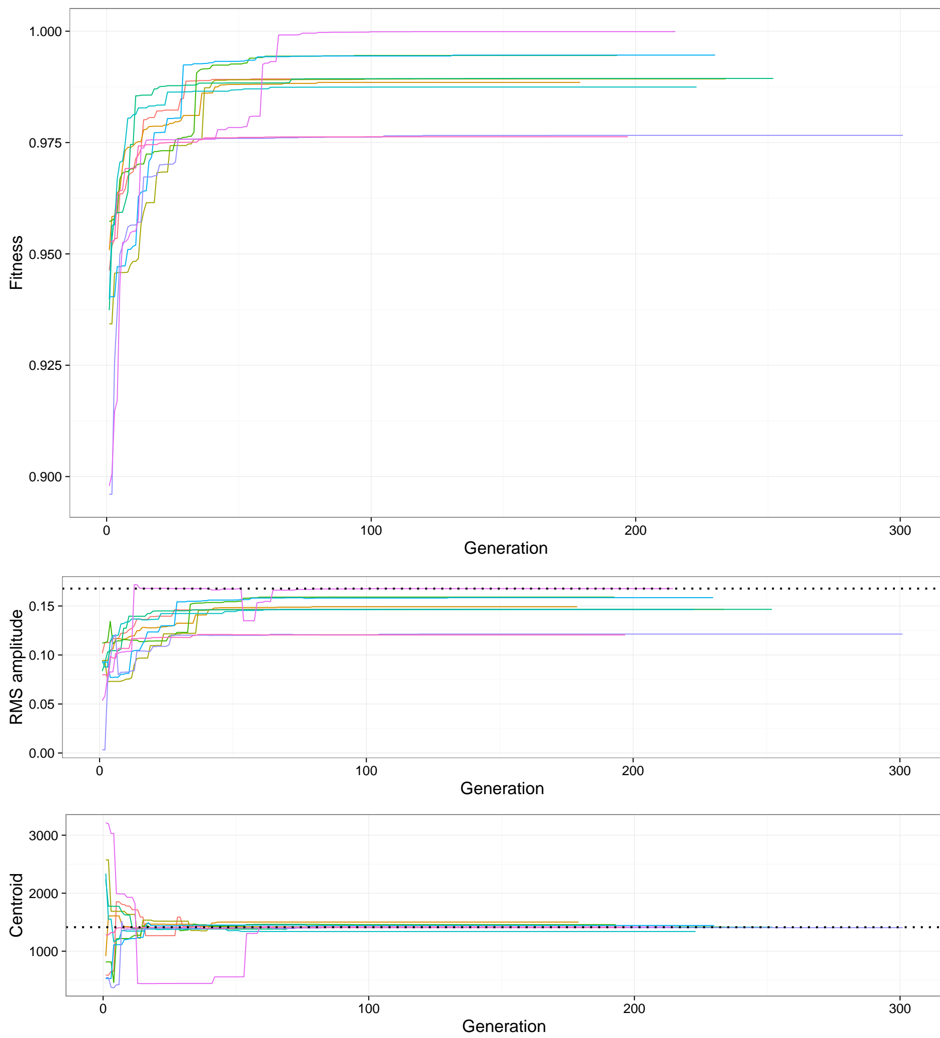

Figure 4.8: Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness, centroid, and RMS amplitude values over time for 10 runs of static processing with $w_{centroid} = w_{rms} = 1$ and $w_{cps} = 0$.

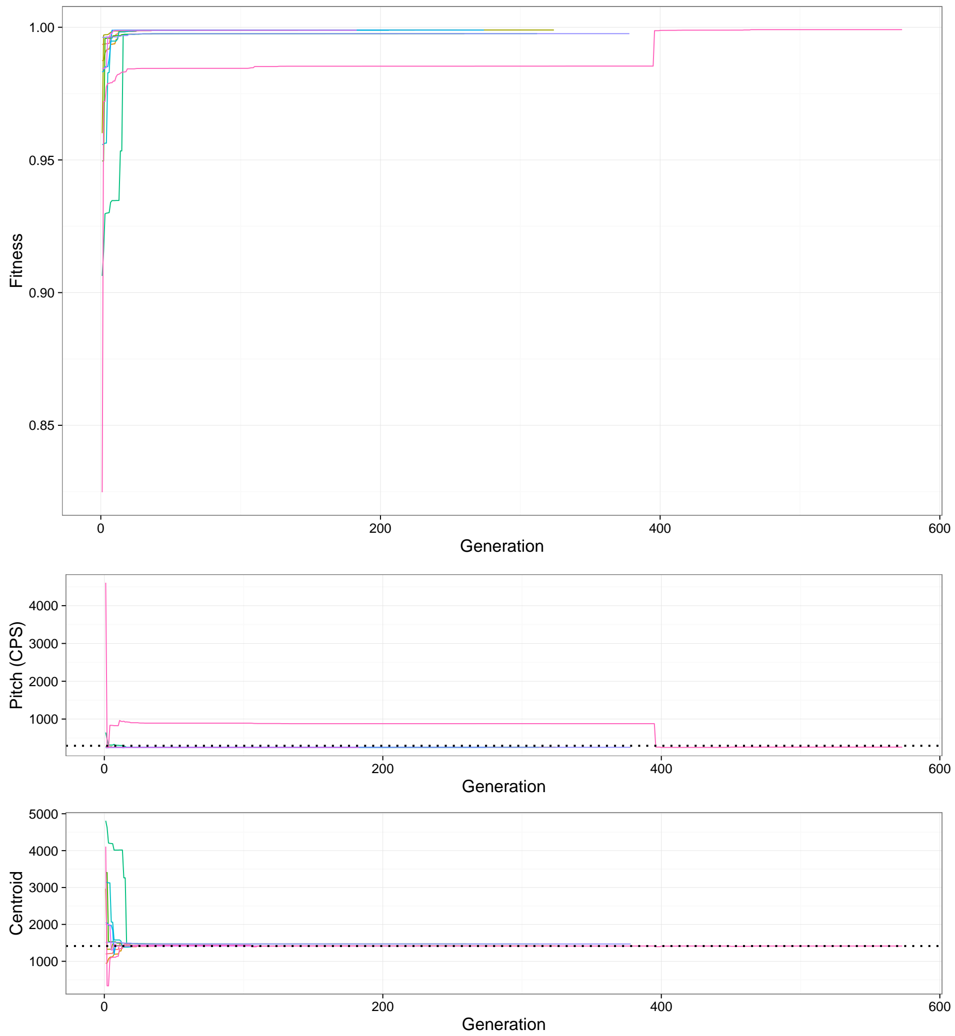

Figure 4.9: Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness, centroid, and pitch values over time for 10 runs of static processing with $w_{centroid} = w_{rms} = 1$ and $w_{cps} = 0$.

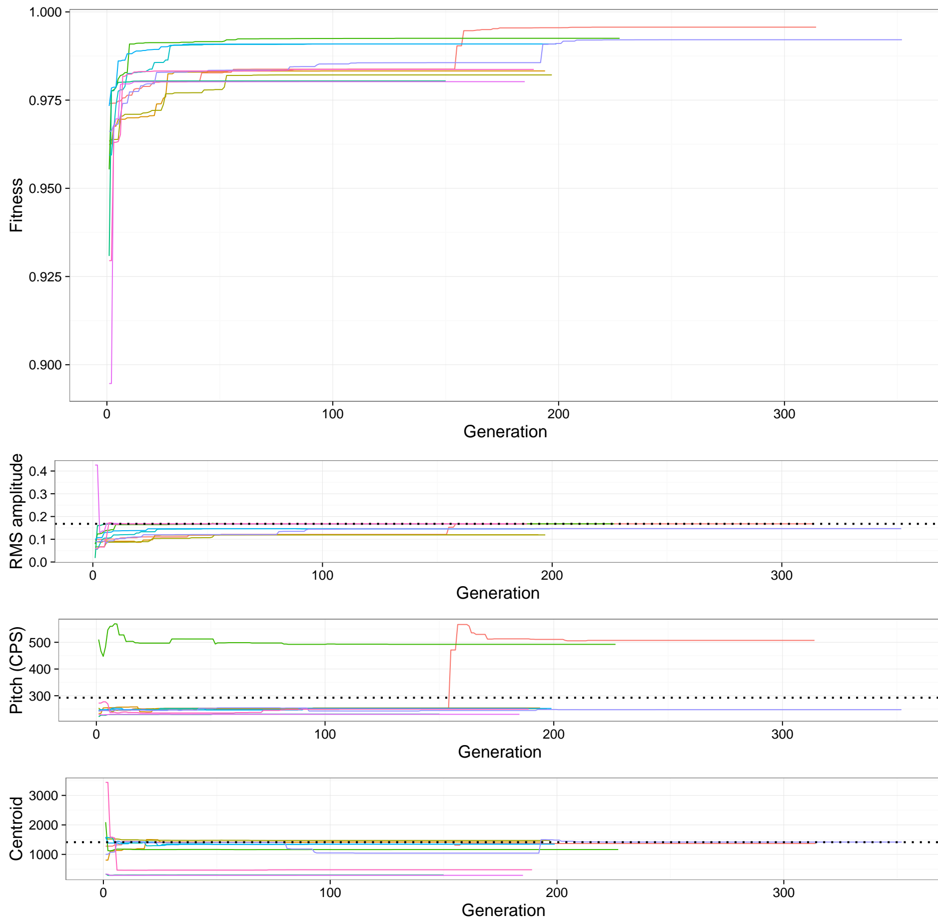

Figure 4.10: Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness, centroid, RMS amplitude, and pitch values over time for 10 runs of static processing with $w_{centroid} = w_{cps} = w_{rms} = 1$.

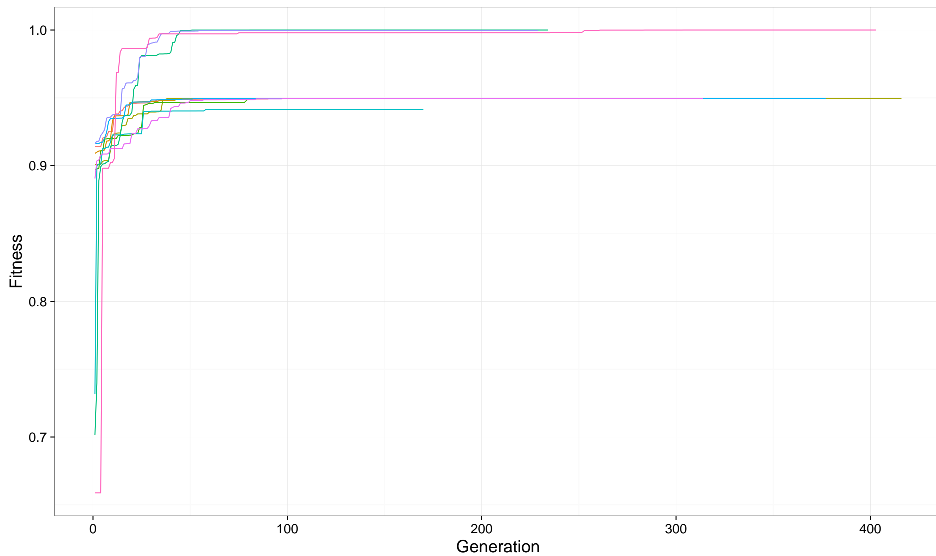

Figure 4.11: Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness over time for a single frame over 10 runs of dynamic processing with $w_{rms} = 1$ and $w_{centroid} = w_{cps} = 0$.

4.2 Dynamic processing

Having seen all combinations of feature weights for static processing, let us now compare some of the same configurations with the dynamic approach. However, because the result of a dynamic run contains data equivalent to that of a static run for each frame, eight frames of 1 s each in all in this case, the data are data are not directly comparable. Figure 4.11 shows the fitness value for a single frame; unsurprisingly, it looks very much like an entire static run, as that is effectively what it is.

Figure 4.12 displays the fitness value over time for all of the frames of a dynamic run superimposed. As we can see, the performance is similar, although with a slightly wider spread that what we find in the static versions. Note that the lowest fitness values typically appear in the last frame, as one of the signals goes silent and the audio effect can no longer assimilate the two.

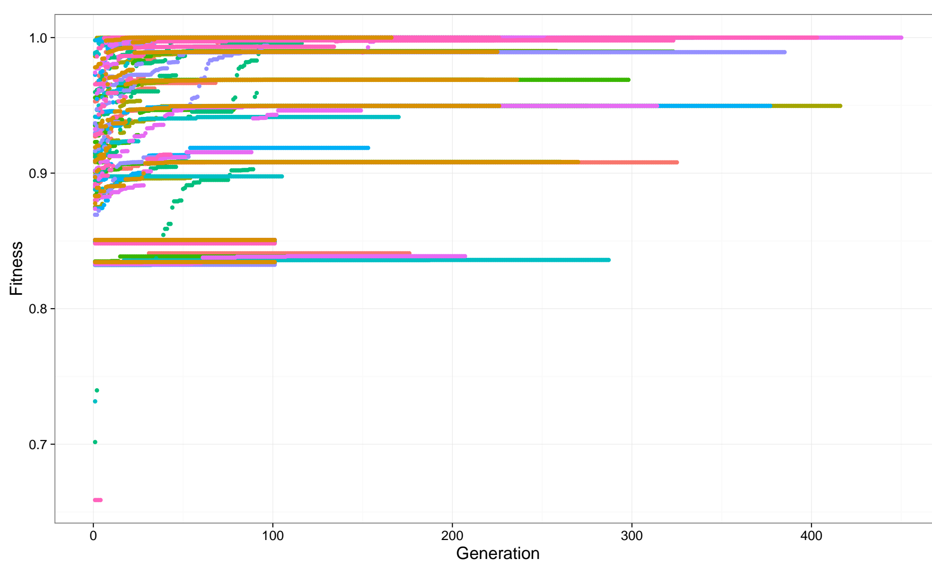

Figure 4.12: Distortion and low-pass filter with vocals as input and guitar as the affecting signal. Fitness over time for a single frame over 10 runs of dynamic processing with $w_{rms} = 1$ and $w_{centroid} = w_{cps} = 0$.

Sample	Centroid	Pitch	RMS amplitude
Vocals	3196 Hz	250 Hz	0.040 726 6
Guitar	1414 Hz	293 Hz	0.167 750 29
Drums	1512 Hz	178 Hz	0.200 518 5

Table 4.1: Analysis of the three audio samples used for input and as affecting signals.

While it is difficult to compare the results of static and dynamic processing from their runtime data, as they differ in structure, we are able to compare the results of processing. An analysis of the unprocessed samples is given in table 4.1. A random selection of a few different configurations and their results is given in table 4.2. Clearly, this is not an exhaustive collection of experiments; rather, it is meant to give a rough indication of the performance of different types of effects and configurations.

Method	Effect	Input	Affector	**w**	Centroid	Pitch	RMS
Static	BP / r. mod.	Guitar	Vocals	[0, 1, 1]	3722 Hz	565 Hz	0.041 046 7
Static	Gain	Vocals	Drums	[0, 0, 1]	3200 Hz	250 Hz	0.200 554 2
Static	LP / dist.	Vocals	Guitar	[1, 1, 1]	1371 Hz	506 Hz	0.167 373 3
Dynamic	BP / r. mod.	Guitar	Drums	[1, 1, 1]	1656 Hz	302 Hz	0.165 770 9
Dynamic	Gain	Vocals	Guitar	[0, 0, 1]	3197 Hz	250 Hz	0.040 839 2
Dynamic	LP / dist.	Vocals	Drums	[1, 1, 1]	1272 Hz	298 Hz	0.123 178 9

Table 4.2: Analysis of the three audio samples used for input and as affecting signals. In the interest of preserving space, the weights have been combined into one column; they appear in the order $w_{centroid}$, w_{cps}, w_{rms}, which is the same order as their corresponding features.

Chapter 5

Discussion

In this chapter chapter, we discuss the results of the experiments conducted, and their consequences. We shall also touch briefly on what further work may be sensible based on the experiences of this thesis.

5.0.1 The results and their meaning

The previous chapter introduced a number of graphs displaying different characteristics of the runtime behavior of the genetic algorithm, as well as the results of the audio processing which followed. A common thread among the illustrations, is a fairly quick convergence, on the order of 30 generations or so, towards a solution or a small area of similar solutions. We also see that the feature or features being considered by the genetic algorithm do in most cases take on a strong similarity to the corresponding features of the affecting signal.

However, we may note some variations. In the case of a single feature being the focus of the fitness function, the algorithm finds solutions consistently well. It appears to perform slightly worse in terms of finding the appropriate amplitude than in approaching the centroid or average pitch of the affecting signal. This is shown in figure 4.4, and it also seems to be the case when combined with one other feature, as in figures 4.7 and 4.8. This may be because of differences in the size of the search space when comparing amplitude to, say, the centroid. If the centroid value is within 4 Hz of its target frequency, that is an excellent result, whereas if the amplitud is off by 0.4 that is very poor. If this is indeed the case, then a change in the mutation function may remedy the problem. That said, the performance of the amplitude measure is still quite good; although it may look like some of the iterations went quite poorly in figure 4.4, they are only off by approximately 0.02—a difference so minute that it is probably impossible, or at least quite difficult to hear.

We also observe that the performance with respect to each individual feature is somewhat reduced when the weights are evenly distributed among them, such as in figure 4.10. However, some change is to be expected; it makes intuitive sense that optimizing for three,

potentially conflicting variables, is more difficult than optimizing for one. Still, most of the attempts produce reasonable solutions, with only a few outliers.

Dynamic processing is apparently less consistent than the static variant, looking at the distribution in figure 4.12. This is, perhaps, unsurprising as the algorithm was initially written for the static case exclusively. Additionally, each dynamic frame has less “space” to work with when converging towards an average value than a longer sample. But even so, the algorithm does “work”.

5.0.2 Further work

While the genetic algorithm implemented for this thesis works quite well in numeric terms, the resulting audio files, while interesting, are certainly not what you would consider “musical” in a conventional way. While there may be an interest in computer-generated, avant-garde remixes of various audio snippets, it does not have universal appeal. Any further work related to this project, would do well to research other measures of musical applicability than simple similarity.

Furthermore, the dynamic processing of this implementation is quite lacking, in its strictly frame-oriented approach. The changes in parameters may be made more subtle through interpolation or low-pass filtering of the stream of parameter settings; however, a change in, for example, cutoff frequency from 100 Hz to 3000 Hz in a frame of a second or less, will necessarily be quite noticeable. It would therefore make sense to look into a more “real-time” approach to the problem, not only if it were to be used in live situations, but also if continuously changing parameters are considered important.

Chapter 6

Conclusion

The goal of this thesis was to investigate whether or not AI methods, and genetic algorithms in particular, may be useful tools for configuring adaptive digital audio effects for musical applications. The background material and considerations have been discussed, and, throughout the text, an implementation of a genetic algorithm for solving that particular problem has been presented. Experiments have been conducted, and they have shown that genetic algorithms are indeed capable of solving problems of this nature, given a quantifiable measure of musical applicability. In this respect, the project has been a success.

However, there is still much room for improvement. The musical results of the algorithm are interesting, but unconventional and unpredictable. The implementation is also rather lacking when it comes to handling an audio signal as an input stream rather than as a pre-recorded file. This thesis has shown that using AI methods to find and evaluate parameters for adaptive digital audio effects is possible; it remains to be seen if it is musical.

Bibliography

[1] D. Arfib, *Recherches et applications en informatique musicale*. 1998, ch. 19. Des courbes et des sons, pp. 277–286.

[2] V. Verfaille and D. Arfib, "A-DAFx: Adaptive digital audio effects," *Proc. of the COST-G6 Workshop on Digital Audio Effects*, pp. 10–14, 2001.

[3] V. Verfaille, U. Zölzer, and D. Arfib, "Adaptive digital audio effects (a-dafx): A new class of sound transformations," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 5, pp. 1817–1831, 2006.

[4] Ø. Brandtsegg, "A toolkit for experimentation with signal interaction," in *Proc. of the 18th Int. Conference on Digital Audio Effects (DAFx-15)*, 2015, pp. 42–48.

[5] *Fundamentals of telephone communication systems*, Western Electric Co., Inc., 1969, p. 2.1.

[6] P. R. Cook, *Real Sound Synthesis for Interactive Applications*. A K Peters, Ltd., 2002, p. 3.

[7] S. J. Russell and P. Norvig, *Artificial intelligence: A modern approach (International Edition)*, 3rd ed. Pearson Education, Inc., 2010, p. 2.

[8] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–a comprehensive introduction," *Natural computing*, vol. 1, no. 1, pp. 3–52, 2002.

Appendix: Source code and audio examples

The complete source code of this project, as well as a selection of audio examples, along with the three audio samples from which they emerged, will be delivered as a digital attachment during the submission of this master's thesis. For any potential readers who do not have access to this attachment, the same resources can be found online at `https://github.com/andereld/interprocessing`.

Note that the audio examples included in the submission were selected to be a concise representation of the span of the audio output produced during this project. That is, some examples were included because they were typical, others because they were interesting.