Evolutionary Computation Applied to Sound Synthesis

James McDermott,[1] Niall J. L. Griffith,[2] and Michael O'Neill[3]

[1] University of Limerick jamesmichaelmcdermott@gmail.com
[2] University of Limerick niall.griffith@ul.ie
[3] University College Dublin m.oneill@ucd.ie

Summary. Sound synthesis is a natural domain in which to apply evolutionary computation (EC). The EC concepts of the genome, the phenotype, and the fitness function map naturally to the synthesis concepts of control parameters, output sound, and comparison with a desired sound. More importantly, sound synthesis can be a very unintuitive technique, since changes in input parameters can give rise, via non-linearities and interactions among parameters, to unexpected changes in output sounds. The novice synthesizer user and the simple hill-climbing search algorithm will both fail to produce a desired sound in this context, whereas an EC technique is well-suited to the task.

In this chapter we introduce and provide motivation for the application of EC to sound synthesis, surveying previous work in this area. We focus on the problem of automatically matching a target sound using a given synthesizer. The ability to mimic a given sound can be used in several ways to augment interactive sound synthesis applications. We report on several sets of experiments run to determine the best EC algorithms, parameters, and fitness functions for this problem.

4.1 Introduction

A typical synthesizer is controlled in two ways. Aspects of performance, such as choice of note or frequency, note length, and note volume, are specified through a device such as a MIDI keyboard or in a saved performance file. Aspects of timbre are controlled by a set of user-variable input parameters. These are generally continuously-variable, though a synthesizer may interpret some parameters discretely.

In applying EC to sound synthesis, we are generally concerned with the problem of setting the input parameters: that is, choosing a point – in the continuous, multi-dimensional space defined by the set of parameters – which corresponds to a desired sound or timbre.

4.1.1 Motivation

There are several reasons why this is a difficult task for users to perform manually. Firstly, parameters are mathematical entities which do not in general relate directly to perceptible attributes of the sound. They are named in a way that is off-putting to non-technical users, and there may be a large number of them – up to 200 or more in some cases, though a range of 20–40 is typical. In many cases, the user does not have a definite target sound in mind, but is rather engaging in simultaneous exploration of possibilities and search for an under-defined goal. The synthesizer can often appear to respond non-linearly to some parameters (a small change in the parameter space can cause a large change in the sound), and often the response to one parameter is dependent on the value of others. In particular, a parameter can in some circumstances have no effect on the sound.

All of this makes synthesis control a difficult and unintuitive process for a beginner; even experienced and technically-oriented users, while composing with a complex synthesizer, sometimes prefer to pursue a desired sound through an intuitive process with immediate feedback rather than switching into analytical, parameter-setting mode. In other situations the ability to automatically match a target sound is required. Evolutionary techniques offer the potential to take away some of the workload involved and make synthesis control more accessible.

4.1.2 EC in the Context of Sound Synthesis

In many EC applications, the fitness function is determined by the problem to be solved, and the choice of representation (the genetic encoding, the genotype–phenotype mapping, and the evolutionary operators) is open for research.

The situation with sound synthesis is somewhat different. Typical synthesizers already possess a natural encoding of parameters as floating-point arrays (some tree-structured exceptions will be seen in Sect. 4.1.3), and the synthesizer itself performs the map from the input parameters (genotype) to a piece of digital audio (phenotype); the choices of evolutionary operators and fitness function to be used are therefore the main areas studied in EC sound synthesis research. Most such research has used the genetic algorithm (GA) [1].

The fitness function in particular is crucial. The basis for defining any fitness function for synthesizer control is the idea of a *distance function* on the sound space: a non-negative real-valued function of two sounds which measures the distance between them. The idea that humans perceive a distance function on the sound space is supported by, for example, Grey [2] and McAdams and Cunibile [3]; however perception of timbre is not fully understood and therefore difficult to model in computational and signal-processing terms. There are therefore two possibilities: we can define automatically-computable distance functions, in the knowledge that at best they approximate human perception, and use them to compare candidate sounds to pre-specified targets (where evolution is towards individuals which closely match the targets); or we can allow a user to rate sounds according to their aesthetic value (where evolution is towards individuals with higher aesthetic value). In this paper we concentrate on the former: see Sect. 4.3.1 for more on the latter.

4.1.3 Literature Review

Several authors have used more or less standard GAs with spectral-comparison (i.e., discrete Fourier transform-, or DFT-based) fitness functions for matching target sounds. In most cases, the individuals in the GA population consist of floating-point arrays, with each element corresponding to a single synthesizer parameter. Each individual can be regarded as a synthesizer preset, and is mapped by the synthesizer to an output sound.

GAs were used by Horner et al. [4] in emulating the spectra of real instruments using FM synthesis. The GA was used to determine the best carrier-to-modulator frequency ratios and (time-invariant) modulation indices. They achieved good results, especially when using several carriers. The fitness function used was a direct comparison of the target and candidate sounds' spectra.

A GA was used by Riionheimo and Välimäki [5] to match target sounds using a plucked-string synthesizer. Here the fitness function used was a comparison of the perceptually-transformed spectra of the candidate and target sounds: the perceptual transformation was motivated by the fact that a comparison of untransformed spectra gives equal weight to all areas of the spectrum, whereas the human ear does not.

Others have used a genetic programming (GP) [6] approach in matching target sounds, evolving the synthesizer itself rather than the parameter settings for a fixed synthesizer. Both Wehn [7] and Garcia [8] defined a small set of synthesis primitives which could be linked together into tree-structures, thus forming complete synthesizers. Again, the fitness functions used in this research were a DFT comparison (in the former case) and a weighted DFT comparison (in the latter).

Spectral-comparison fitness functions tend to lead to rugged fitness landscapes, which can impact on search performance. Mitchell and Pipe [9] used a windowed DFT fitness function, eliminating a proportion of local optima in the fitness landscape. See Sect. 4.2.4 for more on this issue.

In general, authors have not compared their methods experimentally with alternative parameters, algorithms, or implementations. The experiments described in Sect. 4.2 begin to address this.

4.2 Experiments with Automatically-Computable Fitness Functions

In this section we discuss experiments run to determine the best EC algorithms, parameters, and fitness functions for the problem of automatically matching a target sound using a particular synthesizer.

4.2.1 Experimental Setup

All software used in this research is included on the accompanying DVD.[4]. It is described next.

Synthesizer

The synthesizer used is a slightly restricted version of the XSynth synthesizer [10], an analog-modular style subtractive synth written in C, featuring two oscillators, two assignable envelopes, one assignable low-frequency oscillator, and a six-mode filter. The assignable features make the parameters very interdependent, increasing the difficulty of the problem. The full version of the synthesizer has 32 input parameters: however to avoid an instability in the filter, and to prevent very large pitch vibrato, a few of the parameter ranges have been restricted, and in three cases closed off altogether. The resulting synthesizer effectively has 29 floating-point parameters, of which four are really integer-valued but encoded as floating-point.

The experimental setup could incorporate other synthesizers as plug-in replacements for XSynth. Search success in this case is likely to depend on the number of synthesizer parameters, and their degree of interdependence. This is an open question for possible future work.

Target Sounds

All sounds (both candidate and target) used in this study were 1.5 seconds long, generated using XSynth by sending a note-on signal with MIDI note number 69 (concert A), followed 1 seconds later by a note-off, after which a “release tail” of 0.5 seconds was recorded.

For each evolution, a new target sound was generated by setting the synthesizer according to a randomly-generated set of parameters. An alternative possibility is to use recorded samples of real sounds as the targets. This is the more likely real-world application of the system. It is likely that searching for a recorded sound will be less successful than searching for a sound originally generated by the synthesizer, since in general a synthesizer cannot exactly reproduce every possible sound. We avoid this complication in our experiments by using sounds known to be achievable using the given synthesizer: again, this is a possible area for future work.

[4] Updated versions may be available for download at `http://www.skynet.ie/~jmmcd/research.html` and via email from `jamesmichaelmcdermott@gmail.com`.

GA Parameters

The EA used here was a steady-state GA over 100 generations with 100 individuals in the population. Each individual genome consisted of 32 floating-point values, one per synthesizer parameter. The synthesizer, in mapping from the parameters to digital audio, performed the genotype–phenotype mapping. The replacement probability was 0.5, one-point crossover had a probability of 0.5, and Gaussian mutation had a per-gene probability of 0.1 (except in Experiment 2, which investigates different values for crossover and mutation). Selection was by the roulette wheel algorithm. This amounts to a fairly typical floating-point GA.

4.2.2 Fitness Functions

A fitness function, in this context, is a measure of similarity between a candidate sound and a target. Several fitness functions were implemented, each returning a fitness value of the form $1/(1 + d(t, c))$ ($\in [1/2, 1]$), for a target sound t and candidate sound c, where d ($\in [0, 1]$) is the *distance* between the two sounds, calculated in a different way for each fitness function.

Timbral, Perceptual, and Statistical Sound Attributes

Distance functions can be defined based on timbral, perceptual, and statistical attributes extracted from the target and candidate sounds. Some attributes, such as attack time, are intended to mimic as closely as possible aspects of human audio perception of audio. Some, such as pitch vibrato rate, are known to be significant determiners of timbre, e.g., in differentiating between the orchestral instruments. Some, such as zero-crossing rate, are statistical in nature. Almost all of these attributes have been used in recent machine learning research. Many are described by, among others, Jensen [11], Eronen and Klapuri [12], and Lu et al. [13]; and our previous work [14] describes our choice of attributes. Table 4.1 lists them together with their ranges and a key name for each.

The attributes we have chosen do not break down neatly into hierarchical subsets: for example, pitch vibrato depth fits in both the partial-domain and the periodic subsets, and neither is a subset of the other. We have chosen to classify attributes into nine overlapping groups, as shown in Table 4.2.

Attribute Differences

We can calculate a measure of the difference between two sounds by comparing their respective attribute values. A few of the attributes used here are known to be perceived logarithmically: for example, the difference between sounds of 220 Hz and 440 Hz is perceived to be the same as the difference between sounds of 440 Hz and 880 Hz (each is an octave jump), even though the

Table 4.1. Attributes and their ranges: log-domain attributes are marked *

Attribute		Key	Min	Max
Attack	*	att	0.0	1.0
Mean RMS	*	rms	0.0	1.0
Zero-crossing Rate		zcr	0.0	22050.0
Crest Factor		crest	0.0	1.0
Mean Centroid		cen	0.0	512.0
Spectral Spread		sprd	0.0	1.0
Spectral Flatness		flat	0.0	1.0
Mean Flux		flx	0.0	1.0
Presence	*	pres	0.0	1.0
Spectral Rolloff		roff	0.0	1.0
Fast Modulation		fastm	0.0	1.0
RMS Vibrato Depth		vdpth.rms	0.0	1.0
RMS Vibrato Rate		vrate.rms	0.0	20.0
Centroid Vibrato Depth		vdpth.cen	0.0	1.0
Centroid Vibrato Rate		vrate.cen	0.0	20.0
RMS Temporal Centroid		tcn.rms	0.0	1.0
Centroid Temporal Centroid		tcn.cen	0.0	1.0
RMS Temporal Peakedness		tpk.rms	0.0	1.0
Centroid Temporal Peakedness		tpk.cen	0.0	1.0
RMS HFVR		hfvr.rms	0.0	1.0
RMS LFVR		lfvr.rms	0.0	1.0
Centroid HFVR		hfvr.cen	0.0	1.0
Centroid LFVR		lfvr.cen	0.0	1.0
Zero-crossing Rate HFVR		hfvr.zcr	0.0	1.0
Zero-crossing Rate LFVR		lfvr.zcr	0.0	1.0
RMS Heuristic Strength	*	hs.rms	1.0	10.0
RMS Delta Ratio	*	dr.rms	0.1	10.0
Centroid Heuristic Strength	*	hs.cen	1.0	10.0
Centroid Delta Ratio	*	dr.cen	0.1	10.0
Pitch	*	pit	20.0	10000.0
TWM Pitch Error		twm.err	0.0	40.0
Pitch Vibrato Depth		vdpth.pit	0.0	1.0
Pitch Vibrato Rate		vrate.pit	0.0	20.0
Inharmonicity		inh	0.0	1.0
Irregularity (Jensen's)		irr	0.0	10.0
Tristimulus 1		tri1	0.0	1.0
Tristimulus 2		tri2	0.0	1.0
Tristimulus 3		tri3	0.0	1.0
Odd Harmonic Ratio		odd	0.0	1.0
Even Harmonic Ratio		evn	0.0	1.0

Table 4.2. Attributes listed by group

Group name	Keys
Basic	rms, cen, pit, att
RMS	rms, tcn.rms, tpk.rms, hs.rms, dr.rms, hfvr.rms, lfvr.rms, vdpth.rms, vrate.rms
Centroid	cen, dr.cen, hs.cen, hfvr.cen, lfvr.cen, tcn.cen, tpk.cen, vdpth.cen, vrate.cen
Partial domain	evn, inh, irr, odd, tri1, tri2, tri3, pit, twm.err, vdpth.pit, vrate.pit
Trajectory	dr.rms, dr.cen, hs.rms, hs.cen, tcn.rms, tcn.cen, tpk.rms, tpk.cen
Periodic	vdpth.rms, vrate.rms, vdpth.cen, vrate.cen, vdpth.pit, vrate.pit
Statistical	hfvr.rms, hfvr.cen, lfvr.rms, lfvr.cen, hfvr.zcr, lfvr.zcr
FFT domain	cen, sprd, flat, flx, pres, roff
Time domain	rms, crest, att, zcr, fastm

linear differences between the pairs are not the same. Attack time and RMS energy are also known to be perceived in this way. Attributes seen as log-domain require a different comparison function from those seen as linear-domain. Table 4.1 indicates those attributes measured in the log domain.

For each attribute, a difference function is defined which depends on the attribute's theoretical upper and lower bounds, and on whether the attribute is supposed to have a logarithmic or a linear quality:

$$d_i(x, y) = |f_i(v_i(x)) - f_i(v_i(y))|. \tag{4.1}$$

Here x and y are the two sound signals, and $f_i(v) \in [0, 1]$ is a scaling function:

$$f_i(v) = \frac{v - lb_i}{ub_i - lb_i} \tag{4.2}$$

for linear-domain attributes, and

$$f_i(v) = \log(1 + \frac{v - lb_i}{ub_i - lb_i}(e^k - 1))/k \tag{4.3}$$

for log-domain attributes. $v_i(x)$ is the ith attribute value extracted from a sound x, and ub_i and lb_i are the theoretical upper and lower bounds, respectively, for the ith attribute. k is a constant controlling the shape of the logarithmic mapping: here it is assigned the value 5, as used by, e.g., the Sineshaper synthesizer [15]. Note that $d_i \in [0, 1] \; \forall \; i$.

We make an overall attribute comparison between two sounds by combining the individual attribute differences:

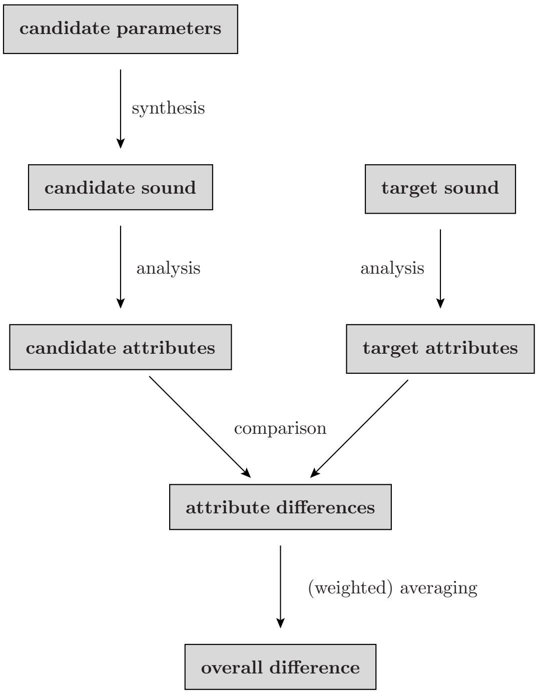

Fig. 4.1. Synthesis, analysis, and comparison using an attribute-difference fitness function

$$d_A(x,y) = \frac{\sum_{i=1}^{n} w_i d_i(x,y)}{\sum_{i=1}^{n} w_i} \tag{4.4}$$

where the weights w_i are taken to be equal to 1 if we require simple averaging, rather than weighting.

This system can be summarized as in Fig. 4.1.

Other Distance Functions

Other types of distance functions can also be defined, such as the *pointwise metric*:

$$d_P(x,y) = \frac{\sum_{t=0}^{T} |x_t - y_t|}{2T} \tag{4.5}$$

where x and y are the sound signals. This is the *DFT metric*:

$$d_F(x,y) = \frac{\sum_{L \in \{256,1024,4096\}} d_{F_L}(x,y)}{3} \tag{4.6}$$

where

$$d_{F_L}(x,y) = \frac{\sum_{j=0}^{N} \left(\sum_{i=0}^{L/2} |X_j(i) - Y_j(i)| \right)}{N} \tag{4.7}$$

where L is the transform length, X_j and Y_j are the normalized outputs from the jth transforms of the sound signals x and y, and N, the number of transforms for each sound, is determined on the basis of 2×-overlapping Hann windows.

We can also form a *composite metric*:

$$d_C(x,y) = \frac{\sum_{d \in \{d_A, d_P, d_F\}} d(x,y)}{3}. \tag{4.8}$$

4.2.3 Experiment 1: Different Types of Fitness Function

This experiment is intended to compare the performance of four types of fitness function, based on the distance measures (Pointwise, DFT, Attribute, and Composite) described in Sect. 4.2.2.

Each fitness function was used to drive 30 evolutions, each with a different target sound. The best individual found in each of the 30 runs was then evaluated under all four of the fitness functions, to allow their performance to be compared. Figures 4.2 and 4.3 show the results.

The Pointwise function awards a high score to the results of all the other functions. It has a bias towards quiet sounds, in that (by definition) two dissimilar quiet sounds will be judged to be closer together than two dissimilar

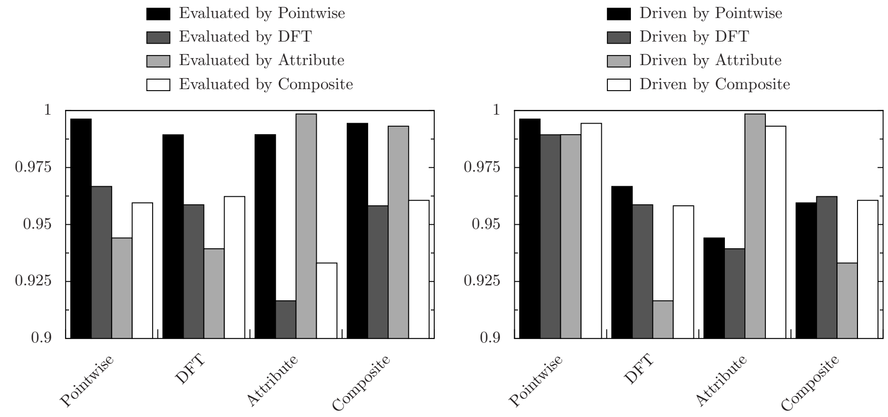

Fig. 4.2. Best fitness averaged over 30 runs driven and then evaluated by each of the four fitness functions, grouped by driving fitness function (left), and the same results, grouped by evaluating fitness function (right). For example, the highest bar indicates the high score of GAs driven by the Attribute fitness function *when evaluated by* the same function; the lowest bar indicates the low score of GAs driven by the Attribute fitness function but evaluated by the DFT function

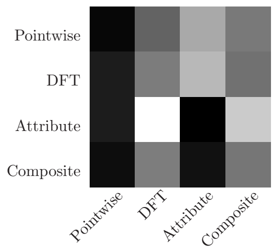

Fig. 4.3. The same results as shown in Fig. 4.2, with scores indicated by intensity, where darker means higher. Driving fitness function is on the y-axis and evaluating fitness function is on the x-axis

loud sounds. In an experiment where the randomly-generated targets were often relatively quiet, this made the Pointwise function very forgiving. The DFT function has a similar bias towards quiet sounds.

Each of the DFT and Attribute functions awards a high score to itself and a low score to its counterpart. The Composite function probably performs best overall. Overall, the lack of an objective method of evaluating performance makes it impossible to draw a definite conclusion.

This experiment is similar to an experiment we have previously reported [16], although that work used a single-modulator FM synthesizer, and a set of simple additively-synthesized target sounds. The Pointwise and DFT fitness functions performed much worse in that experiment, partly because the target sounds were much louder, on average, than those used here.

Relative Improvements in Fitness

Another way to analyze the same results is to consider the relative change in fitness over the course of evolution. For each distance measure, and for each of 30 runs, we calculate the fitness before evolution (i.e., the average fitness of an unevolved population), the best fitness after evolution driven by the corresponding fitness function, and the relative improvement, calculated by dividing the latter by the former. We then average across the 30 runs. These results are shown in Table 4.3.

Table 4.3. Results for four fitness functions, averaged across 30 runs

	Pointwise	DFT	Attribute	Composite
Average best fitness (random search)	0.994	0.953	0.955	0.961
Average fitness before evolution	0.985	0.881	0.884	0.913
Average best fitness after evolution	0.995	0.965	0.974	0.974
Average relative improvement	1.010	1.097	1.102	1.066

The relative improvement for the Attribute fitness function is better than that for the other fitness functions: according to t-tests, it out-performs both the Pointwise and Composite fitness functions with more than 99% confidence, but its advantage over the DFT fitness function is not statistically significant. These results show that the Pointwise fitness function does not perform well. The poor performance of the Composite fitness function is probably due to the influence of the Pointwise function. The DFT and Attribute fitness functions are seen to perform the best: since DFT is the function most commonly used in EC sound synthesis, this justifies further study of the Attribute function.

Also in Table 4.3, we show the best fitness found using a random search algorithm, averaged over the same 30 targets. The random search was over 5000 individuals, the same number processed by our GA, and therefore results for the GA and the random search can be compared. Of the four fitness functions, only the Attribute fitness function drives a GA to perform better than the random search, at a 99% confidence level. Again, this justifies further study of the Attribute fitness function.

4.2.4 Induced Fitness Landscapes

The results of Sect. 4.2.3 can be partly explained with reference to the *fitness landscape*, i.e., the surface corresponding to the (indirect) map from the genome to the fitness value. In our case, for a given target sound, the fitness landscape is the map from the set of synthesizer input parameters to the measured distance between the corresponding candidate sound and the target. In general, the more a fitness landscape exhibits ruggedness and multiple peaks, the more difficult it is to apply any type of machine learning to the problem. Different methods of measuring distance induce different fitness landscapes, and so it is useful to compare the landscapes induced by each of the distance functions discussed in Sect. 4.2.2.

Because there are a large number of input parameters we cannot picture the entire space: however we can look at general trends in the landscape, and form cross-sections of the landscape using parameter-space interpolation.

For Fig. 4.4, we randomly generate 30 target and candidate points, and for each pair interpolate from target to candidate, so that distance to the target (as measured in the parameter space) increases linearly as we move along the x-axis from left to right: at each point in the interpolation we calculate the distance from the current point to the target using each of the four distance functions used in Experiment 1. Averaging across the 30 runs yields a picture of general trends in the four fitness landscapes.

Figure 4.4 demonstrates some of the strengths and weaknesses of the distance measures. The Pointwise distance measure is shown to have an almost totally flat landscape, with only a tiny area in which evolutionary selection is meaningful. The other landscapes show smooth gradients leading towards the target, and so appear to be relatively "easy": however, the averaging process has smoothed out individual features of these gradients, so in Fig. 4.5 we also

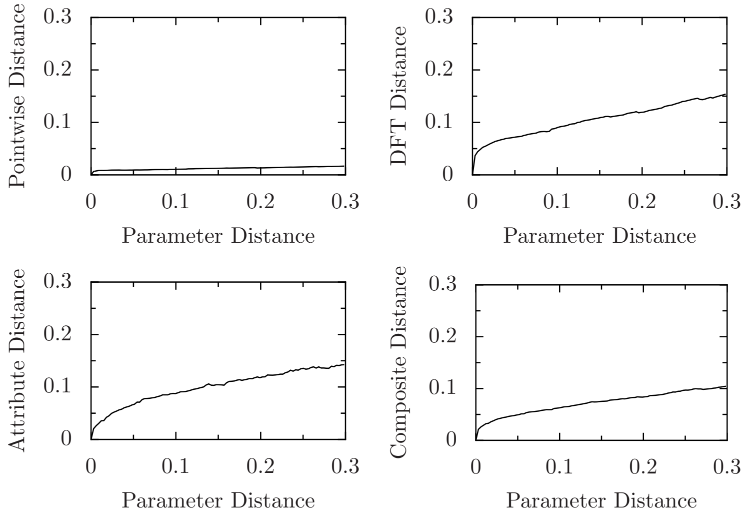

Fig. 4.4. General trends in the fitness landscapes induced by different distance functions. The curves show how each distance measure (indicated on the y-axis) varies with Parameter distance (indicated on the x-axis), averaged over 30 interpolations from target to candidate sounds

examine a single typical cross-section of the landscape. Here, the interpolation was generated in the same way as in Fig. 4.4, but only one interpolation is shown – the same one for each distance function – rather than an average over all 30.

Coupled with the results presented in Fig. 4.4, the cross-sections in Fig. 4.5 provide some additional evidence to suggest the strengths and weaknesses of the various distance measures. The Pointwise measure leads to a very flat landscape with a very small area of decreasing distance: thus evolution becomes a random search for this area. The DFT measure induces quite a smooth landscape, but for much of the interpolation shown the DFT measure is decreasing while Parameter distance is increasing: this presents a difficulty for evolution. The Attribute measure is largely aligned with Parameter distance, except for the addition of many small changes of direction. These, as indicators of local optima, can be detrimental to search performance. Finally, the Composite distance measure combines the strengths and weaknesses of the other measures.

We can also attempt to *quantify* the “difficulty” of a fitness landscape. One method of doing this is to measure the amount of *Monotonicity* in landscape cross-sections such as Fig. 4.5. The larger the number of directional changes the greater the probability of local optima, which can impact performance in many types of search technique. Another method is to use *Fitness Distance Correlation* or FDC [17], which is a measure of to what extent distance – as

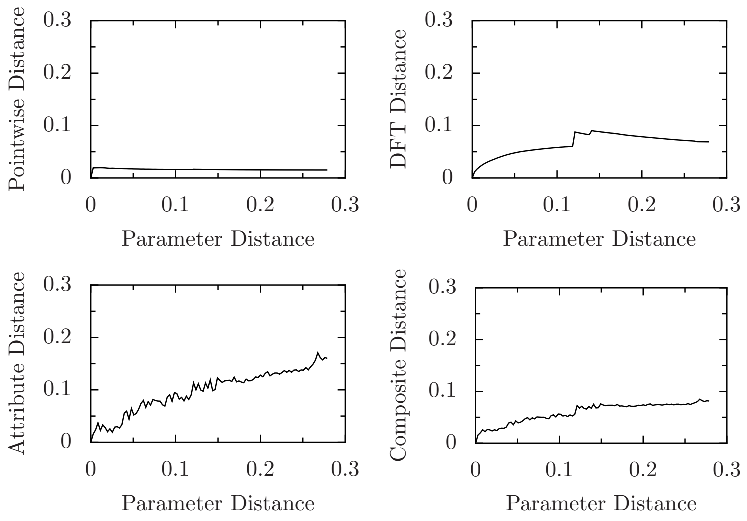

Fig. 4.5. A typical cross-section from the fitness landscapes induced by different distance functions. The curves show how each distance measure (indicated on the y-axis) varies with Parameter distance (indicated on the x-axis)

measured by the fitness function – is correlated with distance – as measured on underlying parameters.

We can estimate the FDC for the fitness landscapes induced by the four distance functions as follows. We take a sample of 30 target points, and for each target, 10 candidate points, both target and candidate points being randomly generated in the parameter space. For each of the 300 pairs, we calculate the underlying parameter distance between the points, and the distance between the pair as calculated by the various distance functions. We then perform a Pearson correlation between the underlying parameter distances and each of the other datasets, to find the FDC in each case. Similarly, we can estimate the Monotonicity by calculating, for each distance measure, 1 minus the average number of changes of direction per point, across all 30 interpolations described for Fig. 4.4.

Table 4.4. Measures of fitness landscape difficulty

Difficulty measure	Pointwise	DFT	Attribute	Composite
FDC	0.103	0.080	0.273	0.153
Monotonicity	0.839	0.907	0.404	0.432

The larger the FDC or Monotonicity value, the easier the fitness landscape. Examining the results we see that these two methods of estimating landscape difficulty give somewhat contradictory evidence in this case: the highest FDC values correspond to the lowest Monotonicity values, and vice versa. The FDC is the difficulty measure endorsed in the literature [17], and so we conclude at this time that the Attribute distance function leads to the "easiest" fitness landscapes – with the caveat that the EA must be designed to avoid premature convergence to local optima, since according to the Monotonicity measure the Attribute distance function leads to more of these.

Considering the relative merits of the fitness functions we have assessed, for subsequent experiments we concentrate on fitness functions based on the Attribute distance measure, and measures derived from it.

4.2.5 Experiment 2: Varying GA Parameters

This experiment compares the performance of GAs, driven by the attribute-based fitness function, in which the mutation and crossover probabilities were varied. Typical values from the GA literature for the two probabilities are compared with more extreme values. We wish to confirm that typical values are appropriate to the particular case of sound synthesis EC.

Figure 4.6 shows the results. According to t-tests, the typical probabilities of (0.5, 0.1) for crossover and (per-gene) mutation perform better than the (0.1, 0.01), (0.1, 0.3), and (0.7, 0.3) combinations, at the 99% confidence level. Their advantage over other combinations is not statistically significant.

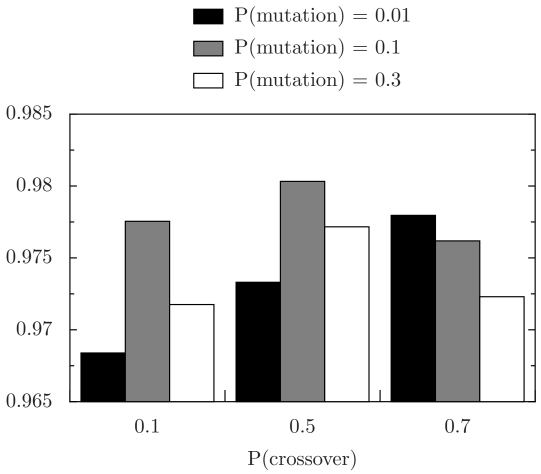

Fig. 4.6. Best fitness averaged over 30 runs for GA with Attribute fitness function and varying values for crossover and mutation probabilities

4.2.6 Experiment 3: Increasingly Discriminating Fitness Functions

We define a set of Increasingly Discriminating Fitness Functions (IDFFs), motivated by the idea that in some search problems, the fitness landscape is characterized by large flat areas of low fitness, and small peaks, with steep sides, of high fitness. When the population is stuck on a flat area, selection becomes meaningless, and evolutionary progress is dependent on chance. An IDFF can reshape the fitness landscape by rewarding minor progress at each stage which is not rewarded by an ordinary fitness function. This idea, "layered learning", has been applied in other areas of EC, such as in robotics control applications [18]. Here, we compare the performance of IDFFs with that of an ordinary GA with an attribute-based fitness function, a GA run with a weighted-attribute fitness function, and a random search algorithm. Eight experiments were defined:

One-stage Here, the fitness function consisted of *all* attributes: in other words, this is an unmodified GA. It was run 30 times with a different randomly-generated target sound for each run.

Two-stage Here, the IDFF consisted of a single attribute for the first 50 generations, and of all attributes for the final 50 generations. Thus there were 40 variations on this experiment, one per attribute: each was run 30 times with a different target sound for each run.

Eight-stage The 100 generations were divided into eight stages of 12 and 13 generations each. The IDFF consisted of five attributes for the first stage, and increased by five attributes for each subsequent stage. The ordering in which attributes were added was randomly generated. Thirty different orderings were used: each was run 30 times with a different target sound for each run.

Nine-stage groups Here, the attributes were divided into the nine groups discussed in Sect. 4.2.2. The 100 generations were divided into nine stages of 11 and 12 generations each. The IDFF consisted of one group of attributes for the first stage, and increased by one group for each subsequent stage. The ordering in which groups were added was randomly generated. Again, 30 different orderings were generated, and each was run 30 times with a different target sound each time.

Twenty-stage This case was similar to that of the eight-stage IDFF evolutions, except that here the 100 generations were divided into 20 stages of five generations each. Evolution began with just two attributes, and two were added for each subsequent stage.

Random search This used a fitness function based on all attributes. In order to compare the performance of different search algorithms, it is only necessary to arrange that they make the same number of calls to the fitness function: hence the random search was conducted over 5000 individuals, the same number as are evaluated by a steady-state GA with the parameters described in Sect. 4.2.1. 5000-individual random search was run 30 times, in each case with a different target sound.

Ordered by difficulty This case was similar to the eight-stage IDFF evolutions, though the ordering, instead of being randomly generated, was chosen to add the most difficult attributes (as reported in Table 4.5) earliest in the evolution. This ordering was run 30 times with a different target sound each time.

Weighted by difficulty This case was a one-stage evolution where the overall fitness function was defined by weighting the attribute differences according to their difficulty, rather than simply averaging them. This evolution was run 30 times with a different target sound each time.

The best individual from each of the 30 runs of the random search was used to calculate an average error for each attribute: these are given in Table 4.5, and were used for the weighting and ordering in the final two experiments.

Table 4.5. Average error by attribute for random search

Attribute	Error	Attribute	Error
att	0.1134	rms	0.0590
zcr	0.0233	crest	0.0280
cen	0.0120	sprd	0.0128
flat	0.0053	flx	0.0104
pres	0.0319	roff	0.0590
fastm	0.0481	vdpth.rms	0.0260
vrate.rms	0.0890	vdpth.cen	0.0339
vrate.cen	0.1167	tcn.rms	0.0223
tcn.cen	0.0314	tpk.rms	0.0192
tpk.cen	0.0182	hfvr.rms	0.0439
lfvr.rms	0.0307	hfvr.cen	0.0005
lfvr.cen	0.0000	hfvr.zcr	0.0724
lfvr.zcr	0.1121	hs.rms	0.0563
dr.rms	0.0485	hs.cen	0.1304
dr.cen	0.0239	pit	0.0303
twm.err	0.0919	vdpth.pit	0.0910
vrate.pit	0.1096	inh	0.0548
irr	0.0163	tri1	0.0589
tri2	0.0437	tri3	0.0508
odd	0.0483	evn	0.0402

Results

The final fitness values reported by each evolution are calculated in terms of *all* attributes, and the **weighted by difficulty** results are evaluated, after evolution has finished, *without* weightings. Therefore it is possible to directly

Table 4.6. Results for eight search techniques, averaged over 30 or more runs (see text for details)

Experiment	Mean	Max	StdDev
Random search	0.955	0.985	0.013
One-stage GA	0.980	0.995	0.010
Two-stage GA	0.968	1.000	0.012
Eight-stage GA	0.973	0.999	0.013
Nine-stage GA	0.973	1.000	0.012
Twenty-stage GA	0.970	1.000	0.013
Ordered by difficulty	0.969	0.992	0.013
Weighted by difficulty	0.972	0.992	0.012

compare results from the weighted evolutions, the IDFF evolutions, the random search, and the unmodified GA.

Table 4.6 shows the results for the random search, the unmodified (i.e., one-stage) GA, the IDFF variations, and the weighted one-stage GA. For each technique, the dataset consists of the highest fitness achieved in each of 30 evolutionary runs (1200 in the case of two-stage, and 900 in the cases of eight-, nine- and 20-stage: see below). For each dataset we give the mean, maximum, and standard deviation.

T-tests show that all of the GA techniques perform better than the random search, at a 99% confidence level. Also, t-tests show that the unmodified GA (one-stage GA) outperforms the modified versions at a 99% confidence level, or higher. However the unmodified version's advantage is not large.

However, this is not the full story: the dataset for each of the two, eight, nine, and 20-stage modified versions is composed of results for 30 (40 for two-stage) *orderings*, each repeated 30 times. Several of the modified versions show high best scores, though means are low. Since we want to test whether some orderings perform better than others, we also look at means and t-tests for individual orderings.

However Table 4.7 shows that these high "best" scores do not come from correspondingly high datasets. In fact, every one of the 30 repetitions for each of the 30 orderings of the eight, nine and 20-stage experiments, and for each of the 40 possible two-stage experiments, performs worse than the one-stage experiment, at the 99% confidence level or higher. This leads us to conclude that the unmodified GA performs better than any of the tested orderings.

The (eight-stage) evolution in which the addition of attributes was *ordered* according to their difficulty does not show improvement over the comparable results (the other eight-stage orderings). Similarly, the technique of *weighting* the attributes according to their difficulty shows a disimprovement in performance, against the comparable results (the unmodified one-stage evolution).

Table 4.7. Results for selected orderings, averaged over 30 runs: the label associated with each two-stage GA indicates the single attribute used to drive evolution for the first of the two stages

Experiment	Mean	Max	StdDev
Two-stage GA, irr	0.981	0.997	0.010
Two-stage GA, sprd	0.982	0.997	0.011
Two-stage GA, lfvr.cen	0.980	0.994	0.010
Nine-stage GA, ordering 0	0.973	1.000	0.014
Nine-stage GA, ordering 8	0.978	1.000	0.010
Nine-stage GA, ordering 16	0.972	1.000	0.015
Twenty-stage GA, ordering 1	0.970	0.997	0.011
Twenty-stage GA, ordering 2	0.975	1.000	0.010
Twenty-stage GA, ordering 6	0.970	0.996	0.012

4.3 Conclusions and Future Work

We have compared the performance of several different types of fitness functions, different values for GA parameters, weightings for timbral attributes, and various increasingly discriminating fitness functions.

The results from the first experiment (Sect. 4.2.3) are hardest to interpret, since the performance of each fitness function can only be described in terms of the others. No clear-cut best function emerges. An alternative analysis, in terms of relative improvement over the course of evolution, may indicate that the Attribute distance fitness function performs slightly better than the others: certainly its performance is competitive, and therefore further work on this method is justified. One thing that is clear is that a fitness function based on timbral, perceptual, and statistical attributes has the potential to be used for constructing sounds in the abstract, perhaps by allowing a user to "sculpt out" desired areas of the attribute space. This is one reason why we have focused on this type of fitness function for later work.

The results on fitness landscapes (Sect. 4.2.4) give contradictory evidence on the question of which fitness functions give the easiest fitness landscapes. The two methods of comparing fitness landscape difficulty (Fitness Distance Correlation and Monotonicity) do not agree. However, comparing these results with those of Sect. 4.2.3 may indicate that Fitness Distance Correlation is the better method of measuring landscape difficulty.

The results on varying GA parameters (Sect. 4.2.5) are not surprising: they confirm that the typical parameters used in the GA literature are applicable to the problem of EC sound synthesis.

The final experiment (Sect. 4.2.6) fails to uncover any technique which can be used to improve on the performance of the unmodified GA. The failure of the IDFFs can perhaps be explained by noting that the fitness landscape for an attribute-based fitness function does not conform to the picture, described in Sect. 4.2.6, of large flat areas of low fitness with small islands of high fitness. In such a situation selection is often effectively random, so evolutionary search is unsuccessful, and a layered technique such as IDFFs can be useful. Instead, the fitness landscape is as shown in Sect. 4.2.4: an individual randomly generated in the parameter space will often have several attributes at least somewhat close to their desired values, and small decrements in parameter distance to the target tend to lead to small increments in fitness. Therefore, selection becomes meaningful and the evolutionary operators make progress. Since the IDFF technique decreases the number of generations available to evolve under the true fitness function, it turns out to be a hindrance rather than a help.

The same applies to another modification, that of weighting the values of the attributes in an attribute-based fitness function. The general conclusion is that, at least in these cases, the longer evolution is allowed to proceed with the "true" fitness function (i.e., the eventual evaluator), the more successful it will be. However the search remains open for a combination of timbral, perceptual and statistical attributes which both reflect true similarity between sounds and lead to good EC performance.

4.3.1 Future Work

Our experiments on automatically-computable fitness functions leave some work remaining to be done, including comparing the performance of other automatic EC search techniques, such as the particle swarm, differential evolution, and evolutionary Strategies; comparing other synthesizers; and using non-synthesized target sounds. The evaluation of EC performance using subjective listening tests is another very important area for future work.

The area of interactive EC (IEC) for sound synthesis has also been explored by several authors [19, 20, 21]. Much work remains to be done both in exploring new IEC ideas and in quantitatively comparing innovations with standard IEC and non-EC methods of interacting with synthesizers.

We have implemented two new techniques:

- *Background evolution* works by allowing the user to specify a target sound for automatic ("background") evolution, and to continue to work on interactive ("foreground") evolution. The best individuals from the background are periodically added to the foreground population. This technique can thus be seen as a way of combining the strengths of human and machine.
- *Sweeping* is a new population interface, which takes the place of explicit fitness evaluation and also functions as a genetic operator. The user controls an interpolation (at the genetic level) between individuals of the population, thus hearing a great variety of sounds, quickly eliminating unsuitable sounds, and focusing in more closely on interesting areas.

Usability studies comparing these techniques with standard EC and non-EC synthesizer interfaces are ongoing.

4.4 Acknowledgements

Co-author James McDermott gratefully acknowledges the guidance of his co-authors and supervisors, and is supported by IRCSET grant no. RS/2003/68.

References

1. Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley. Reading, MA
2. Grey, J.M. (1976). Multidimensional perceptual scaling of musical timbres. *J. Acoust. Soc. Am.*, **61**(5): 1270–1277
3. McAdams, S., Cunibile, J.C. (1992). Perception of timbral analogies. *Philosophical Transactions of the Royal Society*, **336**(Series B): 383–389
4. Horner, A., Beauchamp, J., Haken, L. (1993). Machine tongues XVI: Genetic algorithms and their application to FM matching synthesis. *Computer Music Journal*, **17**(4): 17–29
5. Riionheimo, J., Välimäki, V. (2003). Parameter estimation of a plucked string synthesis model using a genetic algorithm with perceptual fitness calculation. *EURASIP Journal on Applied Signal Processing*, **8**: 791–805
6. Koza, J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press
7. Wehn, K. (1998). Using ideas from natural selection to evolve synthesized sounds. In: *Digital Audio Effects (DAFX)*
8. Garcia, R.A. (2001). Growing sound synthesizers using evolutionary methods. In Bilotta, E., Miranda, E.R., Pantano, P., Todd, P., eds.: *Proceedings ALMMA 2001: Artificial Life Models for Musical Applications Workshop (ECAL 2001)*
9. Mitchell, T.J., Pipe, A.G. (2005). Convergence synthesis of dynamic frequency modulation tones using an evolution strategy. In Rothlauf, F., et al., eds.: *EvoWorkshops 2005*. Berlin Heidelberg. Springer, 533–538
10. Bolton, S. (2005). XSynth-DSSI. `http://dssi.sourceforge.net/` Last viewed 2 March 2006.
11. Jensen, K. (1999). *Timbre Models of Musical Sounds*. PhD thesis. Dept. of Computer Science, University of Copenhagen
12. Eronen, A., Klapuri, A. (2000). Musical instrument recognition using cepstral coefficients and temporal features. In: *Proceedings of the 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 753–756
13. Lu, L., Zhang, H.J., Jiang, H. (2002). Content analysis for audio classification and segmentation. *IEEE Transactions on Speech and Audio Processing*, **10**(7): 504–516
14. McDermott, J., Griffith, N.J., O'Neill, M. (2006). Timbral, perceptual, and statistical attributes for synthesized sound. In: *Proceedings of the International Computer Music Conference 2006*. International Computer Music Association
15. Luthman, L. (2005). Sineshaper. `http://ll-plugins.sourceforge.net` Last viewed 1 September 2006.
16. McDermott, J., Griffith, N.J., O'Neill, M. (2005). Toward user-directed evolution of sound synthesis parameters. In Rothlauf, F., et al., eds.: *EvoWorkshops 2005*. Berlin. Springer

17. Jones, T., Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: *Proceedings of the 6th International Conference on Genetic Algorithms*. San Francisco, CA, USA. Morgan Kaufmann Publishers Inc., 184–192
18. Gustafson, S.M., Hsu, W.H. (2001). Layered learning in genetic programming for a cooperative robot soccer problem. In Miller, J.F., Tomassini, M., Lanzi, P.L., Ryan, C., Tettamanzi, A., Langdon, W.B., eds.: *Proceedings of EuroGP 2001*. Springer, 291–301
19. Johnson, C.G. (2003). Exploring sound-space with interactive genetic algorithms. *Leonardo*, **36**(1): 51–54
20. Mandelis, J. (2001). Genophone: An evolutionary approach to sound synthesis and performance. In Bilotta, E., Miranda, E.R., Pantano, P., Todd, P., eds.: *Proceedings ALMMA 2001: Artificial Life Models for Musical Applications Workshop*
21. Dahlstedt, P. (2001). Creating and exploring huge parameter spaces: Interactive evolution as a tool for sound generation. In: *Proceedings of the International Computer Music Conference 2001*