Chapter 3

Searching Sound Synthesis Space

This chapter presents the results of applying various optimisation techniques to the problem of searching the sound synthesis space of various sound synthesis algorithms, including a discussion of error surfaces along the way. The main aim of this chapter is to compare several techniques for automated sound synthesizer programming in their ability to search sound synthesis timbre space. The techniques are a feed forward neural network, a simple hill climber, a genetic algorithm and a data driven approach. Five sound synthesis algorithms are used in the comparison and they are described in detail. They are simple and complex forms of frequency modulation and subtractive synthesis and a variable architecture form of frequency modulation synthesis. A data driven approach is used to examine the effect of parametric variation in each of the fixed architecture synthesis algorithms upon the audio feature vectors they generate. The results are presented in the form of error surfaces, which illustrate the size of movements in feature space from a reference point induced by changes in parameter settings. Parameters with different behaviours such as quantised and continuous variation are compared. Having described the nature of the environment in which the automatic synthesizer programming techniques will operate, the main study is described. Firstly, each technique is used to re-program sounds which were generated by the same synthesis algorithm that is being programmed. These sounds can be produced by the synthesizer given appropriate parameter settings, so this provides an effective assessment of the general abilities of the automated programming techniques in each domain. The second test involves the programming of sounds which are similar to real instrument sounds. This test aims to establish the versatility of the synthesis algorithms as well as the automated programming techniques. The results are presented and it is shown that the genetic algorithm has the best search performance in terms of sound matching. The hill climber and the data driven approach also show decent performance but the neural network performed rather poorly. The complex FM synthesis algorithm is shown to be the best real instrument tone matcher overall but subtractive synthesis offered better matches for some sounds. The speed of the techniques is discussed and hybrid technique is proposed where the genetic algorithm is combined with the data driven approach, showing marginally better performance.

3.1 The Sound Synthesis Algorithms

The sound synthesis algorithms have been designed to be somewhat similar to the algorithms found in typical hardware synthesizers not only in terms of the basic algorithm but also in terms of the behaviour of the parameters throughout their range. For example, the FM synthesis algorithms have parameters with continuous and quantised values, modelled after the Ysmaha DX7 and the subtractive synthesis algorithms have parameters controlling the mode for switchable mode oscillators. The choice of parameter behaviour has a marked effect on the search space, an effect which is discussed in detail in subsection 3.2. One thing that has been omitted from the algorithms which would be found in all real world synthesis algorithms is any kind of envelope generator. Effectively this means the synthesizers have mostly static timbres (except for cases where the oscillators happen to be set up in such a way as to create low speed variations e.g. FM synthesis with two oscillators with very close frequencies). The motivation for this omission centres around the need to create large data sets showing maximal variation in parameter and feature space in order to support error surface analysis. Without envelope generators, it is possible to represent the output generated from a given set of parameter settings using a single frame feature vector since the timbre does not change over time. By contrast to this, synthesizers with envelope generators would need their output to be measured over many frames in order to accurately represent the effects of the envelope generator(s). In the following subsections, the individual sound synthesis algorithms are described in detail.

3.1.1 FM Synthesis

The fixed architecture FM synthesis algorithms are based around the FM7 UGen for SuperCollider which aims to emulate the six oscillator (or in Yamaha-speak, operator) sound synthesis engine found in the Yamaha DX7 synthesizer [64]. The FM7 UGen is controlled via two matrices; the first defines the frequency, phase and amplitude for each oscillator using a three by six grid; the second defines the modulation indices from each oscillator to itself and all other oscillators using a six by six grid. The UGen implements

f_1	p_1	a_1
f_2	p_2	a_1
f_3	p_3	a_1
f_4	p_4	a_1
f_5	p_5	a_1
f_6	p_6	a_1

m_1	m_7	m_{13}	m_{19}	m_{25}	m_{31}
m_2	m_8	m_{14}	m_{20}	m_{26}	m_{32}
m_3	m_9	m_{15}	m_{21}	m_{27}	m_{33}
m_4	m_{10}	m_{16}	m_{22}	m_{28}	m_{34}
m_5	m_{11}	m_{17}	m_{23}	m_{29}	m_{35}
m_6	m_{12}	m_{18}	m_{24}	m_{30}	m_{36}

Table 3.1: The table on the left shows the FM7 oscillator parameter matrix which defines frequency, phase and amplitude for the six sine oscillators. The table on the right shows the parameters from the FM7 phase modulation matrix which defines modulation indices from every oscillator to itself and all other oscillators. E.g. $m_{1..6}$ define the modulation from oscillators 1-6 to oscillator 1.

phase modulation synthesis as opposed to frequency modulation synthesis, as does the DX7 [3]. Using the parameter names defined in table 3.1, the output y_1 of the first oscillator in the FM7 UGen at sample t in relation to the output of the other oscillators $y_{1..6}$ is shown in equation 3.1. The summed output of all oscillators is shown in equation 3.2.

$$y_1[t] = sin(f_1(\sum_{x=1}^{6} y_x[t-1]m_x 2\pi) + p_1)a_1 \tag{3.1}$$

$$y[t] = \sum_{n=1}^{6} sin(f_n(\sum_{x=1}^{6} y_x[t-1]m_{x+n-1} 2\pi) + p_n)a_n \tag{3.2}$$

The basic algorithm does not use all six oscillators; instead is uses two oscillators arranged as a modulator-carrier pair controlled by two parameters. This is akin to a phase modulation version of Chowning's 'Simple FM' synthesis [20]. This algorithm is shown in equation 3.3, where f_1 and f_2 are fixed to the base frequency and m_2 and r_1 are the two adjustable parameters for the algorithm: modulation index and a multiplier on the base frequency. r_1 is quantised to one of 36 possible values: $[0.5, 0.6...1, 2...31]$. This is the same as the 'Frequency Coarse' parameter described in the Yamaha DX7 Manual, [140, p7]. The frequency ratio is biased towards the lower values by using a simple sinh function to map from parameter value to ratio array index. This is musically motivated since higher frequency ratios produce less useful timbres with too many high partials so it was considered desirable to weight the feature space away from such timbres.

$$y[t] = sin(f_1(m_2 sin(f_2 r_1)2\pi) \tag{3.3}$$

The complex algorithm uses three oscillators and is controlled by 22 parameters. Modulation from oscillator to oscillator is controlled by a single parameter per oscillator which decides which of several available modulation routings from that oscillator to other oscillators are chosen. The available routings for each of the three oscillators are $[0,0,0]$, $[1,0,0]$, $[0,1,0]$, $[1,1,0]$, $[0,0,1]$, $[1,0,1]$ and $[0,1,1]$, or 'no modulation', 'modulate oscillator one', 'modulate oscillator two', 'modulate oscillators one and two', 'modulate oscillator three', 'modulate oscillators one and three' and 'modulate oscillators two and three', respectively. This is similar to the feature found on the DX7 which allows the user to choose different algorithms, [140, p24]. A further three parameters per oscillator define the modulation indices to the three other oscillators, the $m_{1,2,3,7,8,9,13,14,15}$ values from table 3.1. Finally, the oscillators are tuned using another three parameters per oscillator: coarse tuning, fine tuning and detune. Coarse tuning is the same as for the basic synthesizer, fine tuning adds up to 1 to the coarse tuning ratio and is continuous, detuning adds or subtracts up to 10% from the final scaled frequency and is continuous. For example, let us consider the case where the coarse tuning parameter is set to 0.5, the fine tuning 0.25 and the detune is 0.75. Within the synthesizer, the coarse ratio r_1 will be 3 (the 8th element from the list of 36, noting the biasing toward the lower end ratios); the fine ratio r_2 will be 0.25; the detune d will be +5%. Using equation 3.4 to calculate the resulting frequency f for this oscillator with a base frequency F of 440Hz yields the result 1440.725Hz.

$$f = F(r_1 + r_2)(1 + \frac{d}{100}) \tag{3.4}$$

A final parameter is used to choose from a set of available settings for switches which will allow one or more oscillators to be routed to the audio out. For example, switch settings of [1,0,1] would allow oscillators 1 and 3 to be heard. The parameters are further described in table 3.2.

3.1.2 Subtractive Synthesis

The basic subtractive synthesis algorithm takes the three parameters listed in table 3.3. The cut off and reciprocal of Q are standard filter parameters but the oscillator mix parameter requires further explanation. The oscillator mix is the amount of each of the four oscillators sent to the audio output, where the available oscillators generate sin, sawtooth, pulse and white noise waveforms. In order to specify the mix using a single parameter, the parameter is used to select from one of many sets of levels for the oscillators, The available levels are restricted so that only two oscillators can be active at a time. For example, $[0.1, 0.5, 0, 0]$ would set the amplitude of noise to 0.1, sawtooth to 0.5, pulse to 0, and sine to 0. Each oscillator's level can take on any value in the range 0-1 in steps of 0.1.

Parameter	Range	Description
Modulation routing (x3)	[0,0,0],[0,0,1],[0,1,0],[0,1,1], [1,0,0],[1,0,1],[1,1,0]	Switches defnining which oscillators this one modulates
Modulation of oscillator one (x3)	0-1	Modulation index
Modulation of oscillator two (x3)	0-1	
Modulation of oscillator three (x3)	0-1	
Frequency coarse tune(x3)	0.5,0.6 ... 1, 2 ... 31	Chosen from one of 36 values, used to scale from base frequency
Frequency fine tune(x3)	0-1	Added to the coarse tune
Frequency detune(x3)	-10% - 10% of scaled frequency	Added to the scaled frequency
Audio mix	[1,0,0],[0,1,0],[1,1,0], [0,0,1], [1,0,1], [0,1,1], [1,1,1]	Switches defining which oscillators are routed to the audio out

Table 3.2: This table lists the parameters for the complex FM synthesis algorithm. The first six parameters are duplicated for each of the three active oscillators.

With the limitation that only two are active, this makes a total of 522 different options. It should be noted that it would be very unusual to find a parameter which behaves in such a way in a real synthesizer, rather the mix would be controlled by four separate parameters.

The complex subtractive synthesis algorithm is controlled with the 17 parameters listed in table 3.4. This algorithm is typical of the sort to be found in a middle of the range analog-type synthesizer, with 2 switchable mode periodic oscillators, a white noise oscillator and 3 resonant filters.

Parameter	Range	Description
Oscillator mix	Select one of 522 arrays of the form $[a_1, a_2, a_3, a_4]$, with $a_{1...4}$ in the range 0-1 quantised to 0.1	Defines the amplitude level for each of the four oscillators
Filter cut off	Base frequency x 0.01 - base frequenccy x 5	Defines the cut off for the resonany low pass filter
Filter rQ	0.2 - 1.0	The reciprocal of Q, bandwidth / cut off frequency

Table 3.3: This table lists the parameters for the basic subtractive synthesis algorithm.

Parameter	Range	Description
Oscillator 1 waveform	0-3: sawtooth, pulse or sine	4 state switch to select the waveform for oscillator 1
Oscillator 2 tuning ratio	0.25, 0.5, 1, 1.5, 1, 2, 3, 4	Used to scale the frequency of oscillator 2relative to the base frequency
Oscillator 2 waveform	0-3: sawtooth, pulse or sine	4 state switch to select the waveform for oscillator 2
Noise oscillator level	0-1	Amplitude of the noise oscillator
Low pass filter cut off	f/100 to f/100 + (f x 5)	
Low pass filter rQ	0.1 - 1	
Low pass filter level	0-1	Amount of the low pass filter sent to the audio out
High pass filter cut off	f/100 to f/100 + (f x 5)	
High pass filter rQ	0.1 - 1	
High pass filter level	0 - 2	
Band pass filter centre frequency	f/100 to f/100 + (f x 5)	
Band pass filter rQ	0.1 - 1	
Band pass filter level	0 - 2	
Oscillator 2 detune	+/- 5% f	
Ring modulation	0 - 1	Ring modulation from oscillator 1 to oscillator 2
OScillator 1 level	0 - 1	
Oscillator 2 level	0 - 1	

Table 3.4: This table lists the parameters for the complex subtractive synthesis algorithm. Note that f is the base frequency.

Parameter	Range	Description
FM index	0-($f/5$)	Scales the modulation
FM input bus	32-37	FM is read from thie bus
Frequency ratio	one of 0.0625.0.125, 0.25, 0.5, 1,2 ...6	Scales the base frequency
Audio output level	0-1	Scales the audible output of this module
FM output bus	32-37	Write output to this bus
Detune ratio	-0.025-0.025	Added to the frequency ratio

Table 3.5: This table shows the parameters for the variable architecture synthesis algorithm. f is the base frequency.

3.1.3 Variable Architecture Synthesis

The variable architecture synthesizer was initially modelled after an old fashioned analog modular synthesizer, where a variable number of parameterised modules are connected together. However, it was found that the SuperCollider non real time synthesis engine was not suitable for the attempted implementation as it was found to render certain types of sounds inconsistently or not at all. In addition, a non-functional sound would sometimes prevent all subsequent sounds in that batch from rendering correctly. Whilst such a constraint could be viewed as simply another part of the challenge to the optimisers, i.e. to find sounds that SuperCollider can render consistently as well as being close to the target sounds, in practise it slowed down the optimisers to the point of being unusable for the large test sets. This is not to say that the SuperCollider system cannot model a modular synthesizer, but that the modular system designed for this work was not suitable for SuperCollider. It is hoped that in future work, it will be possible to implement an optimisable analog-style modular synthesizer.

The eventual variable architecture synthesizer was therefore made from a single type of module, implementing frequency modulation synthesis using a single sine wave oscillator per module. The synthesizer uses 6 buses to share modulation data between the modules and for the purposes of this experiment has between 5 and 50 modules. The parameters for the module are described in table 3.5. This synthesis architecture can generate a wide range of sounds and provides a highly complex search space.

3.2 Synthesis Feature Space Analysis

During the development and testing of the optimisation algorithms, it became obvious that an analysis of the feature space of the synthesizers would be interesting and that it might feed into the design of the optimisers. For example, the data driven nearest neighbour algorithm (3.3.4) depends on a large data set sampled from feature space; but how should this data set best be sampled? The neural networks depend on training sets and test sets, but what should be in these sets? In the following subsection, the idea of an error surface is introduced and the error surfaces of the various sound synthesis algorithms are examined in detail.

3.2.1 Error Surface Analysis

The error surface is loosely defined here as the variation in the distance from a reference point in feature space as one or more synthesis parameters are adjusted. In this case, the distance metric is the square root of the sum squared Euclidean distance between the reference vector t and the error surface 'reading' e (see equation 3.5). The error surface is similar to the fitness landscape often discussed in genetic algorithm research where the reference point would be taken as the target for the GA and the fitness would be some inverse manipulation of the distance.

$$d = \sqrt{(\sum_{n=1}^{42}(t[n] - e[n])^2)} \tag{3.5}$$

Euclidean distances and error surfaces are precise ways to compare sounds, but what is the perceptual significance of an error of 1.5 vs an error of 2.7, for example? Since the difference between a cello and a French horn is most likely more familiar to the reader than the difference between a mathematically described FM synthesis algorithm with a modulation index of 0.5 or 1.5, that question can be loosely answered by considering the error between the different instruments used in the optimisation task that is the main focus of this chapter. Figure 3.1 shows distance matrices between the ten instruments used to test the optimisers. (The instrument samples were taken from [38]). The value being plotted is the error in MFCC feature space between the different instruments. This value is calculated by taking the 42 dimensional mean of 25 consecutive MFCC feature vectors starting at frame 0 and frame 50 into the sound, for the attack and sustain plots, and generating the error from the other instruments using equation 3.5. Since the instruments are sampled at 44100Hz and the hop size on the feature vector is 512 frames, the time being considered here is about 0.3s. Comparing the error observed between instruments to figure

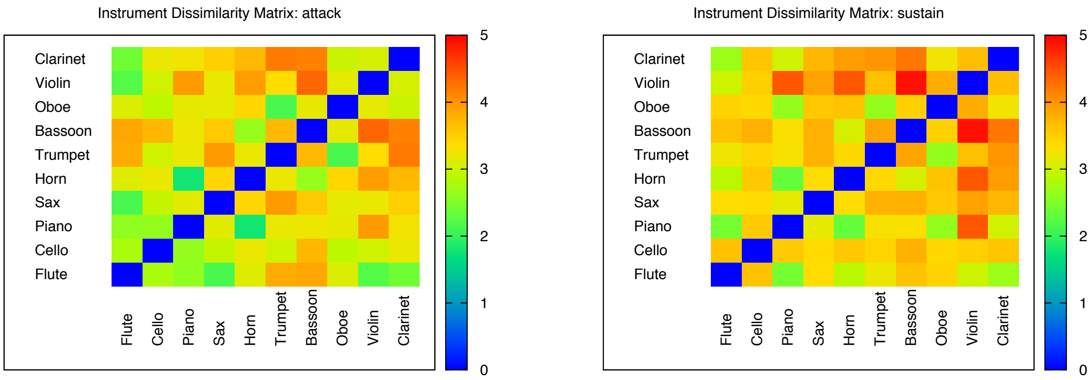

Figure 3.1: This image shows the distances in MFCC feature space between the ten instrument samples used as targets for the optimisers. The higher the value, the greater the distance and therefore the dissimilarity. The left matrix compares the attack portions of the instruments, the right compares the sustain periods, defined as the first 25 feature vectors and feature vectors 50-75, respectively.

3.3 which shows the error observed when a synthesis parameter is varied, it is possible to say that a variation from 0.5 to 0.45 in parameter space causes that synthesizer's output sound to change as much as the difference between the steady state portions of an oboe and a trumpet.

In the following subsections, the error surfaces of the different synthesizers are examined, firstly to establish the resolution required to represent them accurately and then to describe the variation of the terrain observed as parameters are changed.

3.2.2 Error Surface Analysis: FM Synthesis

In figure 3.2, the error surface is plotted with increasing resolution for the basic two parameter FM synthesizer. The figure consists of eight graphs, the four on the left showing the error surface as the first synthesis parameter is varied from 0-1, the four on the right showing the same for the second parameter. The error is the distance in feature space from a reference point which is the feature vector generated with the synthesis parameters both set to 0.5. The distance is calculated using equation 3.5. It is clear that the two parameters have differently shaped error surfaces. The former (in the left column) has a 'Manhattan skyline' surface caused by the parameter being quantised to 36 discrete values within the synthesizer. The latter has a smooth surface caused by it being pseudo continuous within the limits of the 32 bit float with which this value is represented in the synthesis engine (23 bits of precision, [14]). Based on the evidence in the graphs, the former surface seems to take shape at a resolution of 0.001 or 1000 steps in the range 0-1, despite there only being

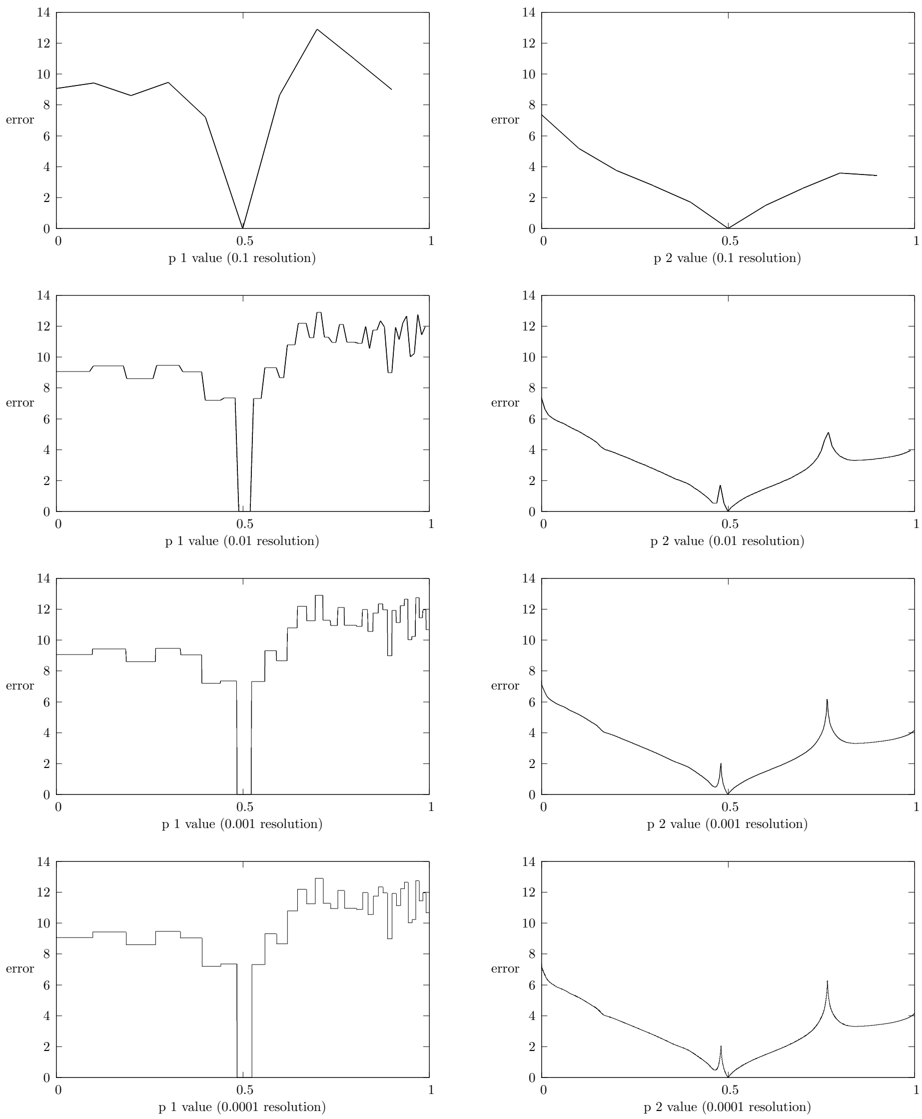

Figure 3.2: The error surface measured from a reference setting of [0.5, 0.5] for the basic two parameter FM synthesis algorithm.

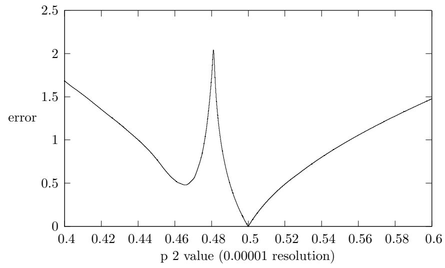

Figure 3.3: Zoomed plot of the error surface in the region between 0.4 and 0.6 for parameter two of the basic FM synthesizer.

36 possible values for this parameter. The graph is more blocky in the lower end of the x axis since the parameter changes more slowly in this range. Closer inspection reveals that in fact the smallest gap that can elicit a change at the high end of the parameter range is 0.01, so this resolution should suffice. The surface for parameter 2 maintains shape after a resolution of 0.001, an assertion further supported by figure 3.3 which shows the error surface for parameter 2 in the region 0.4-0.6 with a sampling resolution of $5{\times}10^{-5}$; no new detail emerges with the increased resolution.

Since the synthesizer has only two parameters, it is possible to map the complete error surface using a heat plot with error represented by colour and the two parameters on the x and y axes. Informed by the earlier observations from figure 3.2 regarding the necessary resolution, parameters one and two are sampled in the range 0-1 at intervals of 0.01 and 0.001, respectively. The results are shown in figure 3.4. In the following text, the variable names from equation 3.3 are used, namely f_1 for parameter 1 and m_2 for parameter 2 i.e. the frequency ratio and modulation index.

The image shows that there is some interdependence between the parameters: at the edges of the strips caused by the quantisation of f_1, a small change in f_1 causes a move to a very differently shaped part of the surface; the shape of the surface of m_2 changes as f_1 is varied but it does retain its two peak structure in many of the strips though the peaks tend to move around. In terms of the timbre, m_2 tends to increase the brightness of the tone by adding additional side bands either side of the base frequency and it continues to do so regardless of f_1's value, in other words f_1 does not inhibit m_2's effect in the feature space. The placement of these side bands will change in response to variation in f_1. In terms of the technicalities of FM synthesis and given that the MFCC provides information about periodicities in the spectrum, in can be said that m_2 increases the number of partials regardless of f_1 and f_1 changes the spacing of the partials, regardless of m_2, as long as m_2 is providing some partials (i.e. is greater than zero). The MFCC will effectively detect

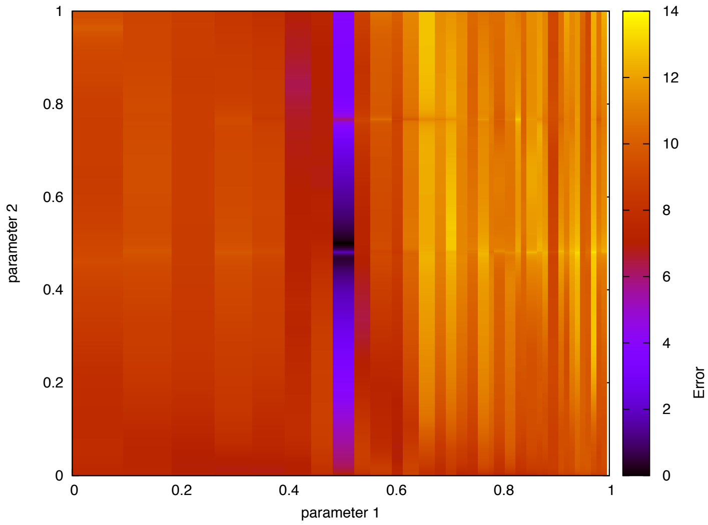

Figure 3.4: The error surface for the basic FM synthesizer. The colour indicates the distance in feature space from audio generated with that parameter setting to a reference feature generated with parameter settings 0.5, 0.5, hence the global minimum error at 0.5,0.5.

such changes. The challenge for the optimisation algorithm here will be not to get stuck in the wrong strip, from whence it might require a highly fortuitous jump to escape.

If we consider the complex FM synthesis algorithm, how does the shape of the error surface change? There are three changes in the algorithms to note here: switching parameters are introduced meaning there are now three types of parameter (more on that below), the frequency is now specified by three parameters instead of one and the overall dimensionality has increased significantly.

Figure 3.5 illustrates the character of the three distinct types of parameter now in operation: the angular surface of the switching parameter, the smooth surface of the continuous parameter and the rocky surface of the quantised parameter. Note that the quantised parameter is the frequency ratio and is therefore not evenly quantised, being weighted toward the lower ratios by the use of a *sinh* function. Figure 3.6 shows the kind of surfaces that are generated when two different types of parameter are varied at the same time. This probably provides the best visual insight into the error surface that will be explored by the optimisers. The three changes from the basic algorithm mentioned above will now be discussed.

How do the switching parameters affect the error surface and at what resolution do they need to be sampled? Looking at the left panel of figure 3.5, the surface takes shape at a resolution of 0.01. Since this parameter can have only seven settings within the synthesizer, it could take shape at 0.1 resolution, but does not since the ten sampled

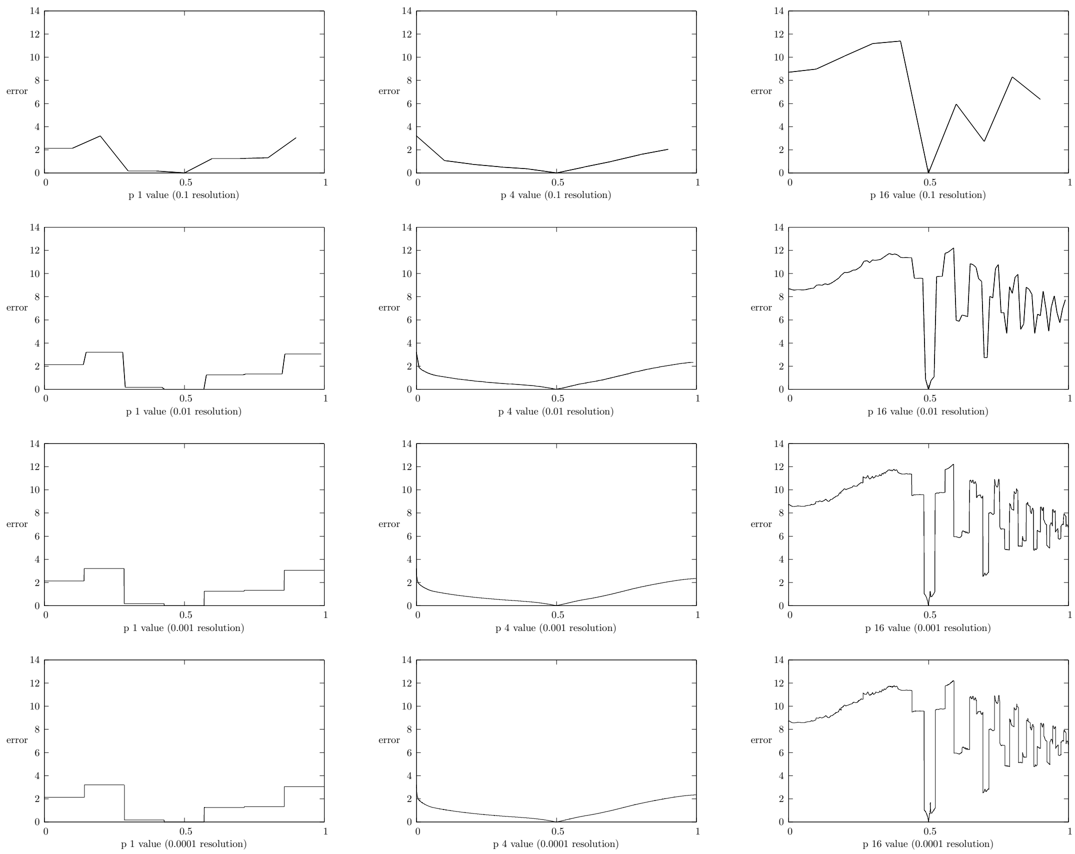

Figure 3.5: The error surface measured from a reference setting of 0.5 for the 22 parameter, complex FM synthesis algorithm. In the left column, a switched-type parameter, modulation routing for oscillator 1; in the centre, a continuous parameter, modulation index from oscillator 1 to 2; on the right, a quantised parameter, coarse frequency ratio for oscillator 1.

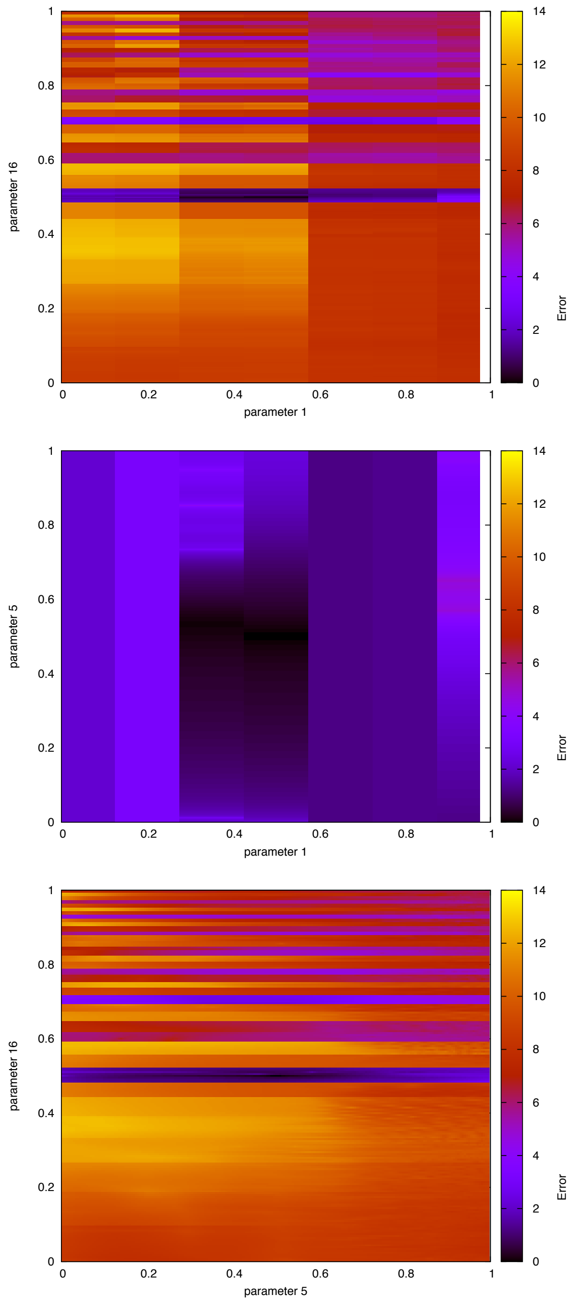

Figure 3.6: Two-parameter error surfaces for the complex FM synthesizer: modulation routing vs frequency ratio, modulation routing vs modulation index, modulation index vs modulation routing.

parameter values do not coincide with the seven variations in the surface. So a suitable resolution is probably 1/14 or about 0.07, to guarantee samples placed either side of every variation in the surface. Not every change in the parameter value has an associated large change in the error surface; this is caused by the interdependence of parameters, e.g. if the oscillator mix switches dictate that oscillator 3 cannot be heard then modulating it will not cause a change in the output sound, assuming oscillator 3 is not modulating an audible oscillator, or indeed is not modulating an oscillator which is modulating an audible oscillator. Clearly, introducing flexible modulation routing and switching parameters in general increases the interdependence of parameters.

How can we measure the effect of the increased dimensionality, informed by the discussion of the new parameters and their required sampling resolution? It was established that a resolution of 100 x 1000 was required to accurately map the error surface of the basic two parameter FM synthesizer, a total of 100,000 points. A simplistic, conservative estimate for the complex FM synthesizer would be 100^{22}, if we sample all parameters at intervals of 0.01. But what of synthesizer redundancy? Some might say that the Yamaha DX7 is a redundant synthesizer, but is it true in terms of the space of possible sounds it can produce, in other words, do different parameter settings produce the same sound? The error surface heat plots show many places with the same error from the reference sound but do these points sound the same or are they just the same distance away? These questions will be explored a little more in the results section.

3.2.3 Error Surface Analysis: Subtractive Synthesis

The basic subtractive synthesizer described in 3.1.2 uses a quantised parameter and two continuous parameters. In figure 3.7, the parameter error surfaces from the reference point of [0.5,0.5,0.5] are plotted with increasing resolution. The first parameter shows smooth stages with periodic jumps. The behaviour of this parameter is described in sub section 3.1.2 where it is stated that there are 522 different states for the oscillator mixes, with either one or two oscillators active at any one time. Figure 3.8 shows what the mix for the oscillators will be for any value of the oscillator mix parameter and it suggests four noticeable stages for the parameter. There seem to be three stages in figure 3.7 but this is caused by the final stage offering an increase of the sine oscillator, which can only add a single partial to the sound, which does not appear to register in the error. Since this parameter can take on 522 distinct states within the synthesizers, a resolution of 0.002 should be sufficient to map this parameter's range of features. The continuous parameters

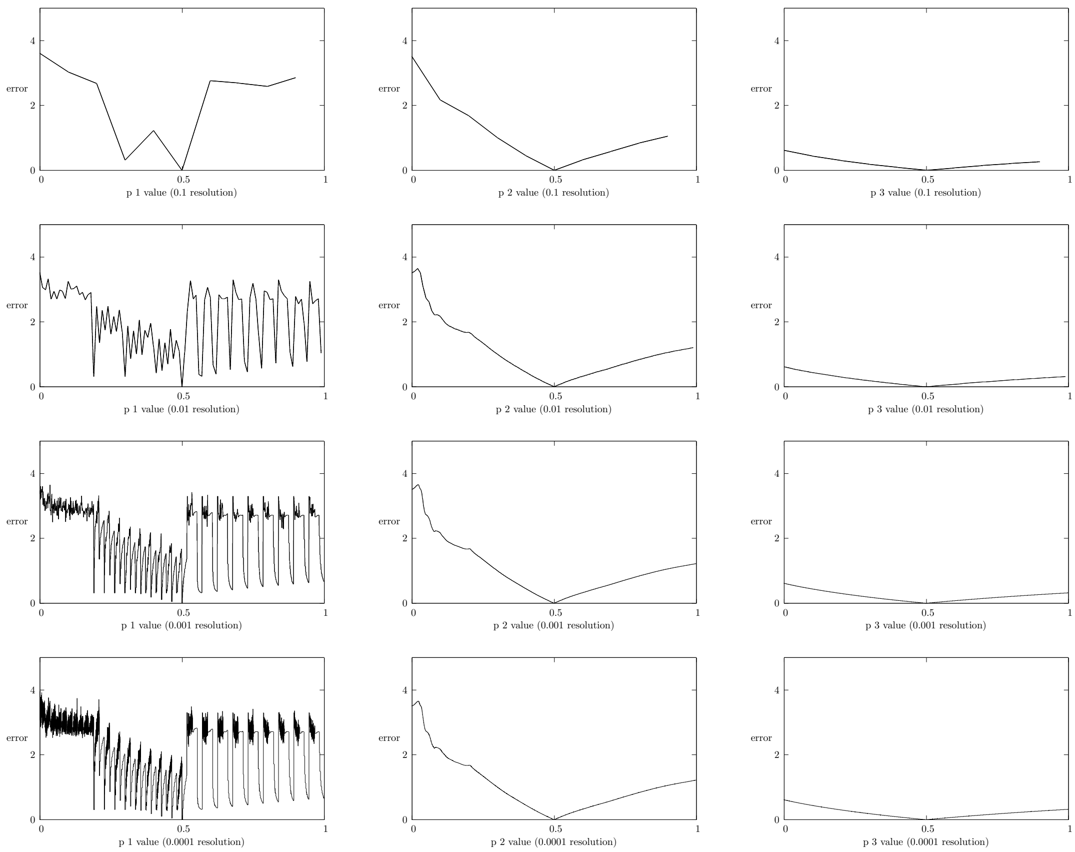

Figure 3.7: The error surface measured from a reference setting of [0.5, 0.5, 0.5] for the basic three parameter subtractive synthesis algorithm.

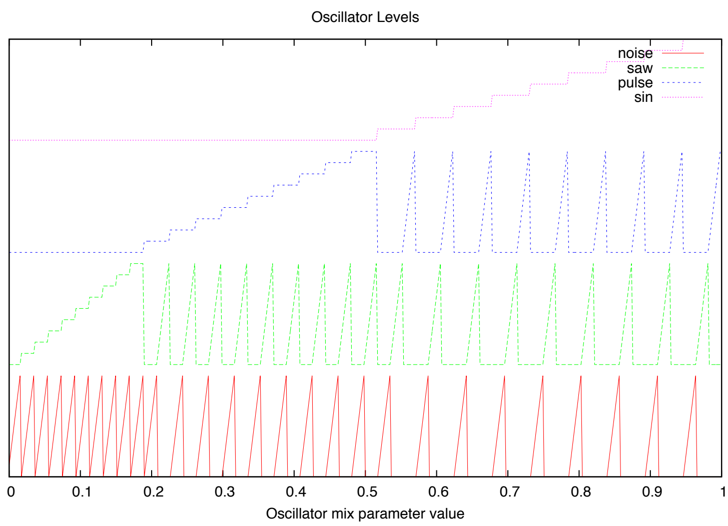

Figure 3.8: The mix levels of each oscillator in the basic subtractive synthesizer as the oscillator mix parameter is varied. Each oscillator can take on one of 10 mix levels.

seem to require a fairly low resolution, taking shape at a resolution of 0.01.

Figure 3.9 contains heat plots showing how the three possible combinations of parameters affect each other. The quantised mix parameter only seems to have a large effect from strip to strip, not within strips. This is because within a strip the oscillators are being faded up, gradually varying the sound whereas between strips, the combination of two oscillators is changed abruptly. It is anticipated that it will be challenging for simpler optimisation algorithms to traverse these strips to find a global minimum, depending on the hop size (in parameter space) of the optimiser, i.e. if the hop size is too low, the optimiser is unlikely to escape a strip. The continuous parameters, i.e. the filter cut off and rQ, show a smooth error surface which should be fairly trivial to search due to its large, single valley of low error. If an optimiser finds itself stuck in a strip a long way off from the target, it will be necessary to traverse several high error strips to reach the target.

The error surface of the complex subtractive synthesizer is expected to be similar to that for the basic synthesizer. The filter parameters behave in the same way and there are switchable mode oscillators. The difference is that the oscillator volume controls have been changed from a single quantised parameter to several continuous parameters.

The error surface for the variable architecture synthesizer is difficult to plot since it is not clear what should be used as a reference point and what should be varied to create the surface. It can be said that the parameters within a single module will behave similarly to their equivalents in the basic and complex FM synthesizers but that a growth operation which adds a module could cause a large movement in feature space as the new module might modulate the existing modules as well as sending its signal to the audio out.

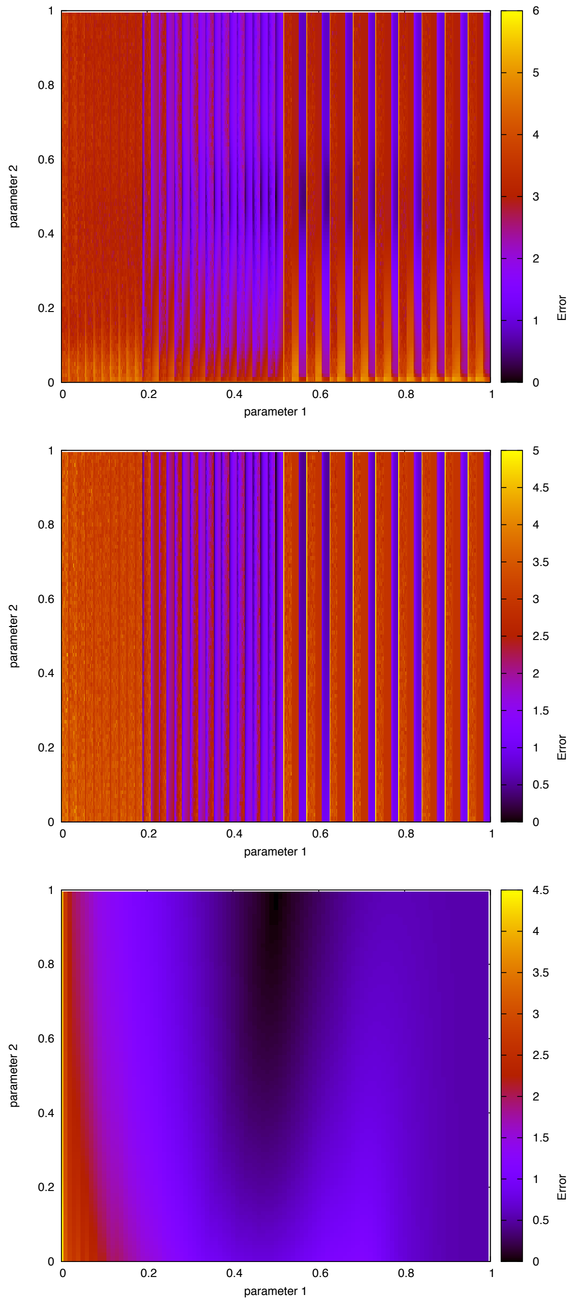

Figure 3.9: Two-parameter error surfaces for the basic subtractive synthesizer: oscillator mix vs cut off, oscillator mix vs rQ, and cut off vs rQ

3.3 The Optimisation Techniques

The optimisation problem is: 'The elucidation of appropriate parameter settings for each sound synthesis algorithm to enable resynthesis of 50 sounds generated by each synthesis algorithm and of 10 real instrument samples, respectively.'. These two test sets are chosen to enable maximum flexibility when comparing different optimisers and different synthesizers. The first set consists of 50 pairs of randomly generated parameter settings and resultant single frame feature vectors, per synthesizer. Since the test set is generated with the same synthesizer that the optimiser is working with in each case, it is possible to gain an error of zero, given a perfect optimiser. This test set enables the comparison of different optimisers and different optimiser settings for a given synthesizer. The second test set consists of two feature vector frames per instrument for the following instruments: Alto Flute, Alto Sax, Bassoon, B♭ Clarinet, Cello, French Horn, Oboe, Piano, Trumpet and Violin. The selection of instruments was chosen based on its coverage of a range of different sound production methods and its successful use in previous research, e.g.: [35] used the same selection, [44] used eight of the ten, having found that musically trained subjects found the sounds highly recognisable. The instrument samples were obtained from the University of Iowa Electronic Music Studio collection [38]. All instruments were playing vibrato-free C4 (middle C). The first frame is the 42 dimensional mean of 25 frames taken from the attack portion at the beginning of the sound; the second frame is the 42 dimensional mean of 25 frames taken from approx 0.5s into the sound. It is unlikely that a synthesizer will be able to perfectly replicate these sounds so the optimiser must find the closest sound available. Since the synthesizers do not include envelope generators to vary their sonic output over time, the optimiser must find the synthesis parameter settings to generate each feature vector frame in turn. This test enables the comparison of different synthesizers in their ability to resynthesize the samples. To summarise, each of the 4 optimisers must find optimal parameter settings for 70 sounds for each of the 5 synthesizers, 50 that the synthesizer can match perfectly and 20 that it can probably only approximate. This is a theoretical total of 1400 tests but only two of the optimisers are appropriate for the variable architecture synthesizer, making the total 1260.

The optimisation techniques will now be described in detail.

3.3.1 Basic Hill Climbing

An introduction to hill climbing can be found in [70]. The hill climbing algorithm here is implemented as follows:

Starting from a random population of 1000 sets of parameter settings, the output of each set is rendered and features are extracted. The error between the target feature and each of the 1000 candidate feature vectors is calculated using the Euclidean distance. (equation 3.5). The score for each is calculated using equation 3.6. The candidate with the highest score is used to seed the next population, which consists of mutated versions of the seed. Mutating involves the addition of a value drawn from a uniform distribution in the range -0.05 to +0.05 to a single parameter setting. A growth operation is also applied if the candidate sound is to be generated with more than one module (e.g. the variable architecture FM synthesizer.). The population is considered to have converged when the mean change in top score over 20 generations is less than 0.01%.

3.3.2 Feed Forward Neural Network

A feed forward neural network is trained using a set of parameter settings and their resultant feature vectors such that the network learns the mapping from feature vector input to parameter setting output. Once trained, the network can be used to elicit the parameter settings required to produce a given, previously unseen feature vector with a particular synthesis algorithm. The implementation and elucidation of optimal properties for the feed forward neural network is described in the following paragraphs.

A training set of data consisting of synthesis parameter settings and the resultant feature vectors is created. The set consists of parameters and resultant features derived by sampling parameter space randomly. A feed forward neural net with a single hidden layer, based on code written by Collins and Kiefer [27], is created. The network has v input nodes and p output nodes, where v is the size of the feature vector and p the number of parameters provided by the synthesis algorithm. The network is trained using the back propagation algorithm. There are several properties associated with this neural network and the training procedure for which reasonable values must first be heuristically obtained. They are learning rate, number of training epochs, number of hidden nodes and size of the training set.These properties must be obtained for each synthesis method since the number of parameters and the mapping varies significantly between them. The procedures and results are summarised in figure 3.10 and discussed in the following subsections.

Learning Rate

The learning rate parameter determines the amount by which the neural network's weights are changed at each iteration of the back propagation algorithm. To establish the optimal

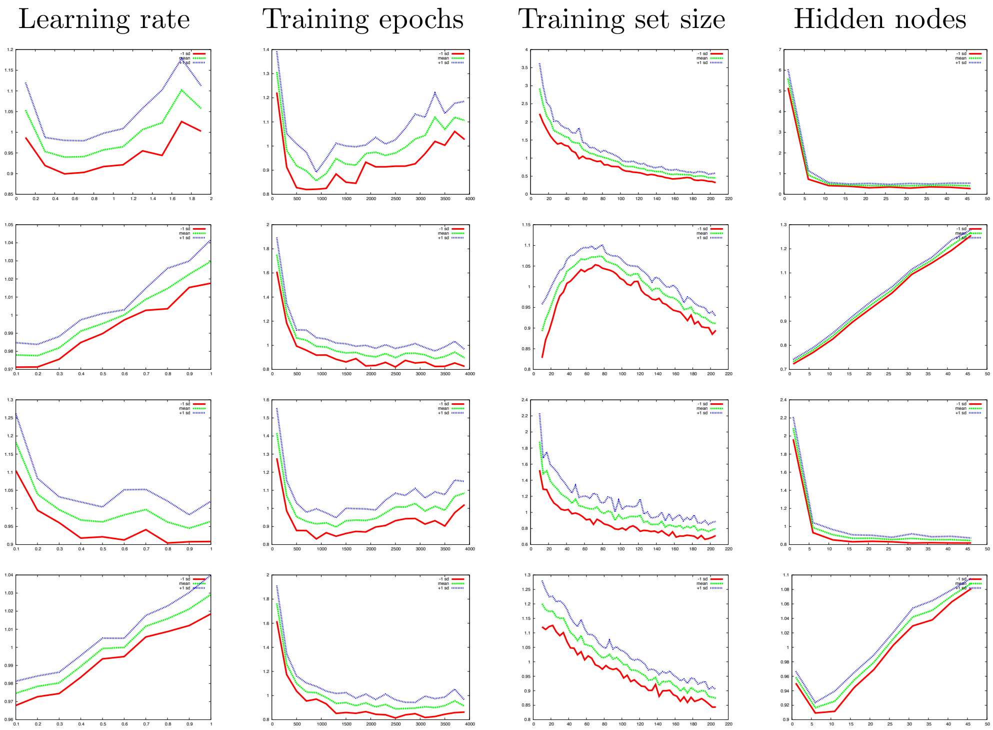

Figure 3.10: Neural network performance for the 4 synthesizers with varying network and training properties. Each row is for a different synthesizer: basic FM, complex FM, basic subtractive, and complex subtractive. Each graph shows the mean and standard deviation either side observed for 25 test runs. The test set error is plotted against each of the 4 network properties apart from the learning rate, where the training set error is used.

Synthesizer	Learning rate	Training epochs	Training set size	Hidden nodes
BasicFM	0.6	1000	1810	26
InterFM	0.2	2400	1810	10
BasicSub	0.9	1500	710	28
InterSub	0.1	2500	1810	6

Table 3.6: The best neural network settings found for each synthesizer.

learning rate, the network is trained with varying learning rates and the training set error is calculated. This test is repeated 25 times with different, random training sets. The learning rate which provides a good combination of low error over the training set and lack of oscillation is selected. Oscillation is where the network learns some training sets well and others not very well. In figure 3.10 column one, the results of this test for the four synthesizers are shown. The values chosen for the synthesizers are shown in table 3.6

Training Epochs

The training epochs parameter is the number of iterations of training that are carried out. To establish the appropriate number of training epochs, the test set error is graphed against the number of training epochs. 25 runs are carried out for varying numbers of epochs, each run with a different, random training set and test set. The expectation is that the error will level out when sufficient epochs have been used. The results are graphed in figure 3.10 and tabulated in 3.6. The error is seen to reduce to its lowest point then to increase as over-fitting to the training set increases and generalisation to the test sets decreases.

Hidden Nodes

The neural network has a single, fully connected hidden layer and this property defines the number of nodes in that layer. In order to establish the optimal number of hidden nodes, the test set error is measured against the number of hidden nodes. The expectation is that the error will cease to decrease with increased node count when there are enough nodes. The network is trained and tested with increasing numbers of hidden nodes. This test is repeated 25 times with random test and training sets. The results are graphed in figure 3.10 and tabulated in 3.6. The basic synthesizers behave as expected here, showing a test error which levels once sufficient hidden nodes are in use. The complex synthesizers show a more puzzling result, where the optimal number of hidden nodes seems to be very low, considering the complexity of the synthesizer and therefore of the parameter setting to feature mapping. The error also increases significantly with increasing numbers of hidden nodes. In order to establish if the error would level out and maybe decrease with increasing numbers of hidden nodes, extended tests were carried out with node counts up to 1500 with these two synthesizers. The results are shown in figure 3.11. The error levels out after around 600 nodes but shows no signs of decreasing. Since the mapping for the simple synthesizers is learned with more nodes than 10 and that this low number

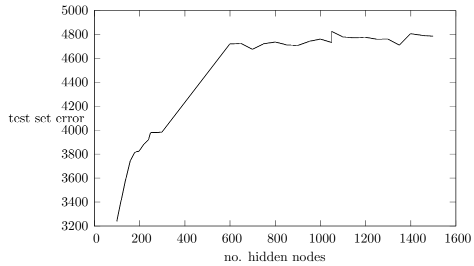

Figure 3.11: This graph shows the results of increasing the number of hidden nodes for the complex FM synthesizer to an unusually high level.

of nodes is therefore not sufficient to model the mapping for a more complex synthesizer, this implies that this network architecture is not capable of modelling the mappings for the complex FM and subtractive synthesizers. More on this in the results section.

Training Set Size

To establish the optimal training set size, the error over a large test set is calculated with increasing training set size. This test is repeated 25 times with random test and training sets. Once the error over the test set is seen to level out, the training set size is judged as sufficient as increasing the size does not decrease the test set error. The results are graphed in figure 3.10 and tabulated in 3.6.

3.3.3 Genetic Algorithm

The genetic algorithm used in this test is a standard model with the following features:

Single population

The genomes exist in a single population and can be bred freely.

Genetic operators

The genomes can undergo point mutation and crossover using two parents and variable numbers of crossover points. The point mutation is controlled by two parameters, one for the per-locus probability of mutation (mutation rate), the other for the size range of the mutation (mutation size). Once a locus is chosen for mutation, the mutation size is chosen from a Guassian distribution with the mean set to zero and the standard deviation set to the value of this second mutation parameter. The mutation rate is set to 1/(synth parameter count), so at least one parameter is mutated per genome, on average. The mutation size is set to 0.1.

Elitism

The best genome each iteration is kept.

Roulette wheel selection

When generating a new population, each new genome is created from two parent genomes. The parent genomes are selected with a probability that is proportional to their fitness relative to the population mean. This is roulette wheel selection. Every member of the population is on the roulette wheel but the fitter they are, they more 'slots' they take up and thus the more probable it is that they will be chosen. Roulette wheel selection maintains diversity in the population by preventing the fittest individuals from taking over in the case where their fitness is only marginally higher than other members of the population. Maintaining diversity is especially useful when it comes to avoiding getting stuck on local maxima. Let us consider the error surface plot shown in 3.4. Maintaining a diverse population makes it possible to search different parts of the error surface in parallel. This is the multi plane sampling Whitley talks about. ([133]). It is possible to further investigate the different properties of the GA and this has been done to a certain extent in the other chapters. For example, in [133] Whitley recommends use of rank based selection of breeding pairs as opposed to fitness proportional selection since it maintains the selective pressure when the population begins to converge. The motivation for the investigation of these finer points of GA implementation in order to produce a more refined, optimised version should be taken from evidence of poor performance of the basic implementation. The preliminary results indicated decent performance for the GA so it was left in its basic state.

General Settings

For the tests, the algorithm was run with a population size of 1000 and a maximum of 500 iterations. The mutation rate was set to 1/(number of synthesis parameters) such that there would on average be a single point mutation per genome. The mutation size range was set to 0.1, i.e. the centre of the Guassian distribution was placed at 0.1.

3.3.4 Basic Data Driven 'Nearest Neighbour' Approach

In this approach, the space of possible parameter settings is randomly sampled to produce a data set of parameter settings and resultant feature vectors. Upon being presented with a target feature vector, the system finds the feature vector in the data set which is the closest to the target and can then provide the associated synthesis parameters. For effective performance, this system is dependent on the ability to store and search a high resolution data set which covers the full timbral range of the synthesizer. The parameters for this optimiser are the data set size and the distance measure. Let us consider the construction of the data set in more detail.

The data set must be sampled at sufficient resolution such that the detail of feature vector space is high enough to represent the timbral range of the synthesizer. Consider 3.2 which shows the error in feature space from a reference setting of 0.5 as a single parameter is varied from 0 to 1 with different levels of resolution. Once the resolution reaches a sufficient level, the graph does not change with further increases: the true shape of the error surface has been ascertained. In the case of the basic FM synthesis algorithm shown in figure 3.2, the graph stops changing at around the 0.005 point, meaning it is necessary to sample this synthesizer at steps of 0.005 for this particular parameter. In the SuperCollider system, the parameters are stored as 32 bit floats, which offer some 23 bits of precision, providing usable resolution in the $\frac{1}{2^{23}}$ range; sampling at intervals of 0.005 is well within these limits. For a synthesizer with two parameters, assuming equal sampling resolution for each, a data set of 40,000 items will be generated ($(1/0.005)^2$). If we consider the complex FM algorithm, it might initially seem that an impracticably large data set of $(1/0.005)^{19}$ or 5.24288e+43 items is required but this does not take into account the quantised parameters or the redundancy of the synthesizer, where different parameter settings generate the same sound.

3.4 Results

In this section, the results of applying the optimisation techniques to the problem defined at the start of section 3.3 will be presented. The scope of the test is quite broad, so the results are presented in summary. There are two main values used to quantify the performance results in this section: the score and the error. The score is the reciprocal of the normalised, sum, squared and rooted Euclidean distance from the target to the candidate feature vector, as shown in equation 3.6. The figure of 42 used to normalise the value is the size of the MFCC feature vector. The other value, the error, is simply the sum, squared and rooted Euclidean distance betwixt target and candidate. The error value is comparable with the values plotted in the graphs and heat plots from the error surface analysis in subsection 3.2.1.

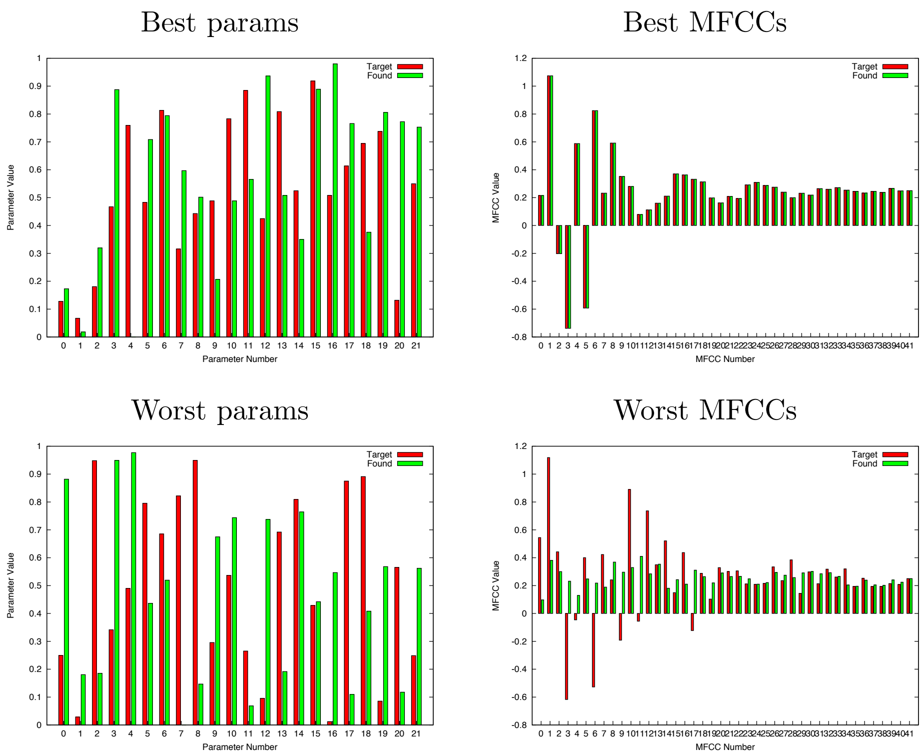

Figure 3.12: Best and worst results for the complex FM synthesizer. The target is in red and the found result in green.

$$s = \frac{\frac{1}{\sqrt{(\sum_{n=1}^{42}(t[n]-e[n])^2)}}}{42} \tag{3.6}$$

3.4.1 Optimiser Test Set Performance

Figure 3.12 provides an easily grasped visualisation of the sort of results that have been achieved. It shows the best and worst matches achieved for the complex FM synthesizer test set alongside the target features and parameters. The most important match here is the feature match, since this defines how similar the sounds are.

In table 3.7, an overview of the performance of the 4 optimisers over the 50 test sounds for each synthesizer is shown. The figures shown in the table are the mean score achieved over the test set, the standard deviation over the test set, the more intelligible percentage standard deviation and finally the error.

Noting that the synthesizer should be able to create a perfect rendition of the test sounds as long as the correct parameters can be found, this test should show which optimisers can effectively search or map the space of possible sounds for the synthesizers. The genetic algorithm performs best overall, followed by the hill climber, the data driven

Optimiser	Synthesizer	Mean	Standard deviation	SD %	error
GeneticAlgorithmSO	InterFMOS	1.63e+13	7.97+13	490.16	1.96e-08
GeneticAlgorithmSO	BasicFMOS	1171254.21	2975605.86	254.05	3.59e-05
GeneticAlgorithmSO	BasicSubOS	294349.66	425419.75	144.53	0.0001
HillClimberBestOption	BasicFMOS	157637.41	688455.7	436.73	0.0003
HillClimberBestOption	BasicSubOS	53977.4	178831.26	331.31	0.0008
DataDrivenNearestNeighbour	BasicFMOS	40842.15	41508.45	101.63	0.001
HillClimberBestOption	InterFMOS	5090.84	20429.7	401.3	0.01
DataDrivenNearestNeighbour	InterFMOS	3389.37	19505.18	575.48	0.01
DataDrivenNearestNeighbour	BasicSubOS	885.78	1569.31	177.17	0.05
HillClimberBestOption	GrowableOS	192.3	243.88	126.82	0.22
GeneticAlgorithmSO	InterSubOS	98.76	113.81	115.24	0.43
GeneticAlgorithmSO	GrowableOS	80.39	82.85	103.07	0.52
DataDrivenNearestNeighbour	InterSubOS	57.56	30.68	53.31	0.73
HillClimberBestOption	InterSubOS	44.43	15.7	35.34	0.95
FFNeuralNetSO	BasicSubOS	22.87	15.63	68.32	1.84
FFNeuralNetSO	InterSubOS	14.57	3.54	24.3	2.88
FFNeuralNetSO	InterFMOS	7.13	2.92	41.03	5.89
FFNeuralNetSO	BasicFMOS	5.04	2.09	41.35	8.33

Table 3.7: This table shows the mean performance per optimiser per synthesizer across the 50 sounds in each synthesizer's test set. The score is the reciprocal of the distance between the error and the target normalised by the feature vector size. The SD% column is the standard deviation as a percentage of the mean and the final column is the non-normalised error, comparable to the values in the error surface plots.

search and the neural network. The genetic algorithm is the most effective optimiser for all of the fixed architecture synthesizers but the hill climber out-performs it for the variable architecture FM synthesizer (GrowableOS in the table). What do these figures mean in terms of the observations from the error surface analysis (section 3.2) and the real instrument distance matrix (figure 3.1)?

The worst score is a mean of 5.04 over the test set, gained by the neural net working with the basic FM synthesizer. It is surprising that this combination has a worse score than the complex FM synthesizer with the neural net, given the problems with finding the right settings for the latter neural net. Still, the complex FM synthesizer score is not far off, at 7.13. The final column in the results table shows the error surface figures which can be used to relate the scores to the error surface analysis. The worst score equates to a distance of 8.33 in the error surface analysis graphs. This is off the scale for the instrument similarity matrix (figure 3.1), suggesting that this optimiser and synthesizer combination is unlikely to work for real instrument sound matching. In terms of the error surface plot for the basic FM synthesizer (figure 3.4) which shows the error observed from a reference point as the parameters are varied, a similar error is generated when the parameter settings are up to 0.5 away on the second parameter or almost impossibly far off on the first parameter. Since the standard deviation is quite low, the performance is consistently poor. If one accepts the methodology used to derive the settings for the neural nets, it must be concluded that a feed forward, single hidden layer neural network trained with back propagation is not an appropriate tool with which to automatically program synthesizers.

Having illustrated ineffective performance, how large a score would constitute effective performance? If effective performance is defined as consistently retrieving the correct parameters or at least a close feature vector match from the space of possible sounds, a very low error would be expected, given that this space contains the target sound and the optimiser simply needs to find it. The top 5 mean results certainly show very low errors, errors which equate to minimal changes in parameter space. The best performing optimiser is the genetic algorithm, followed by the hill climber; the data driven search also shows strong performance. But the top results also have very large standard deviations, an observation which warrants a little explanation. Let us consider the results for the genetic algorithm across the 50 target test set for the complex FM synthesizer. The standard deviation is so large since there is a large variation in the scores across the test set. Essentially, the top 15 out of the 50 scores have an almost negligible error (<0.001) and

	1	2	3	4	5	6	7	8	9	10	Mean	SD
Worst	5.3	2.91	4.09	1.35	3.97	4.92	5.28	2.64	4.8	4.09	3.94	1.29
Middle	1.06	0.6	0.91	0.7	0.43	0.07	0.06	0	0.86	1.04	0.57	0.41
Best	0	0	0	0	0	0	0	0	0	0	0	0

Table 3.8: This table shows the errors achieved by repeated runs of the GA against the targets from the test set for which it achieved the best, worst and middle results.

the others range from an error of 0.02 up to 5.26. To qualify the size of these errors, one can refer back to the error surface analysis for the complex FM synthesizer, specifically column 1 of figure 3.5 which shows the error generated by changing a quantised parameter's value. This graph shows that changing this parameter setting enough to move it to a different quantisation point causes a feature space error of between 0.1 and 1. Therefore, for this quantised parameter at least, an error of less than 0.1 implies a close parametric match. It is more difficult to convincingly concoct a figure for an acceptably low error for the continuous parameters but the central column in figure 3.5 shows a maximum error of 2 when the parameter setting is as far off as it can be. Again, an error of less than 0.1 seems reasonable. It is worth noting that typically a GA will be run several times against a given problem to establish the variation in performance caused by the stochastic nature of the algorithm. That would equate to running the GA several times against the same target sound in this case. To satisfy this requirement, the best, middle and worst results from the complex FM test set for the GA were selected (errors of < 0.000001, 0.68 and 5.28, respectively) and the GA was run 10 times against each target. The results are shown in table 3.8. Note that the error shown in the table for the best result is zero but the real error was $\frac{42}{419811201219720}$ which is not far from zero. The figures are reasonably consistent, indicating that the GA is consistently searching feature space and that the non-repeated results are reliable. It would be interesting to figure out what makes the target for the worst result so hard to find but that is beyond the scope of this study.

3.4.2 Instrument Timbre Matching Performance

In table 3.9, an overview of the performance of the 4 optimisers and 5 synthesizers over the 20 real instrument sounds is shown. The table is sorted by score so the synthesizer/optimiser combination which achieved the highest average score over all 20 sounds is at the top. The final column of the table, as before, shows an error comparable to the metric used in the earlier error surface analysis (subsection 3.2.1). Table 3.10 shows the best result achieved for each instrument sound along with the error. Again, the genetic

Optimiser	Synthesizer	Mean	Standard deviation	SD%	Error
GeneticAlgorithmSO	InterFMOS	22.9702	6.2617	27.2601	1.828
GeneticAlgorithmSO	InterSubOS	21.0570	4.4694	21.2255	1.995
DataDrivenNearestNeighbour	BasicSubOS	20.9126	3.5917	17.1749	2.008
DataDrivenNearestNeighbour	InterFMOS	20.7831	5.1740	24.8950	2.021
DataDrivenNearestNeighbour	InterSubOS	20.0827	3.5742	17.7974	2.091
GeneticAlgorithmSO	BasicSubOS	19.7633	3.6499	18.4679	2.125
HillClimberBestOption	BasicSubOS	19.3550	3.6726	18.9750	2.170
HillClimberBestOption	GrowableOS	15.1654	3.4224	22.5669	2.769
HillClimberBestOption	InterSubOS	14.7632	3.4241	23.1937	2.845
GeneticAlgorithmSO	GrowableOS	12.7828	2.1272	16.6414	3.286
HillClimberBestOption	InterFMOS	12.4737	3.6924	29.6018	3.367
DataDrivenNearestNeighbour	BasicFMOS	10.5969	1.7464	16.4808	3.963
GeneticAlgorithmSO	BasicFMOS	10.5942	1.7434	16.4561	3.964
FFNeuralNetSO	BasicSubOS	10.3134	2.1984	21.3157	4.072
FFNeuralNetSO	InterSubOS	9.9958	1.7921	17.9287	4.202
HillClimberBestOption	BasicFMOS	9.1044	2.0576	22.5998	4.613
FFNeuralNetSO	InterFMOS	7.9552	1.8834	23.6752	5.280
FFNeuralNetSO	BasicFMOS	6.0991	1.9064	31.2576	6.886

Table 3.9: This table shows the mean performance per optimiser, per synthesizer across the 20 real instrument sounds. The data is sorted by score so the best performing synthesizer/ optimiser combinations appear at the top.

Optimiser	Synthesizer	Target	Score	Error
GeneticAlgorithmSO	InterFMOS	AltoFlute.mf.C4.wav_attack	40.9549	1.0255
GeneticAlgorithmSO	InterFMOS	AltoSax.NoVib.mf.C4.wav_sustain	35.2891	1.1902
GeneticAlgorithmSO	InterFMOS	AltoSax.NoVib.mf.C4.wav_attack	34.5376	1.2161
HillClimberBestOption	BasicSubOS	Cello.arco.mf.sulC.C4.wav_attack	32.1244	1.3074
GeneticAlgorithmSO	InterSubOS	Horn.mf.C4.wav_attack	27.0503	1.5527
DataDrivenNearestNeighbour	BasicSubOS	Piano.mf.C4.wav_attack	25.5118	1.6463
GeneticAlgorithmSO	InterFMOS	Violin.arco.mf.sulG.C4.wav_attack	25.0713	1.6752
DataDrivenNearestNeighbour	BasicSubOS	Cello.arco.mf.sulC.C4.wav_sustain	24.2392	1.7327
DataDrivenNearestNeighbour	BasicSubOS	Piano.mf.C4.wav_sustain	22.9333	1.8314
GeneticAlgorithmSO	InterSubOS	Horn.mf.C4.wav_sustain	22.0686	1.9032
GeneticAlgorithmSO	InterFMOS	AltoFlute.mf.C4.wav_sustain	22.0323	1.9063
GeneticAlgorithmSO	InterFMOS	BbClar.mf.C4.wav_attack	21.9440	1.9140
DataDrivenNearestNeighbour	BasicSubOS	oboe.mf.C4.wav_attack	21.0050	1.9995
HillClimberBestOption	InterFMOS	Bassoon.mf.C4.wav_sustain	20.8823	2.0113
GeneticAlgorithmSO	InterFMOS	Violin.arco.mf.sulG.C4.wav_sustain	19.7671	2.1247
DataDrivenNearestNeighbour	BasicSubOS	Trumpet.novib.mf.C4.wav_attack	19.0931	2.1997
GeneticAlgorithmSO	InterFMOS	BbClar.mf.C4.wav_sustain	19.0656	2.2029
DataDrivenNearestNeighbour	BasicSubOS	oboe.mf.C4.wav_sustain	18.7581	2.2390
DataDrivenNearestNeighbour	BasicSubOS	Bassoon.mf.C4.wav_attack	18.6131	2.2565
GeneticAlgorithmSO	InterSubOS	Trumpet.novib.mf.C4.wav_sustain	18.0972	2.3208

Table 3.10: This table shows the best matches achieved for each of the 20 real instrument sounds.

algorithm shows the best performance, performance which is also consistent across the instruments. The complex FM synthesizer does the best matching over the test set, but not by a great margin, being closely followed by the complex subtractive synthesizer. When the instruments were compared to each other in the instrument distance matrix shown in figure 3.1, the closest match was between the French Horn and the Piano with an error of 2.3157. The closest match achieved to an instrument was quite close indeed, an error of 1.0255. The mean error achieved over the 20 instrument sounds in the best case was significantly lower than the closest match between any of the instruments themselves, an error of 1.828. In other words, on average the optimisers found sounds closer to the real instruments that were significantly closer than any of the real instruments were to each-other.

The data driven nearest neighbour optimiser achieves impressive performance in table 3.10, beating more sophisticated algorithms on many of the sounds. It seems to have more consistent matching performance than the GA, with a smaller standard deviation. The data driven search is also significantly faster than the marginally better performing GA and hill climber, taking a consistent minute to search a database of 100,000 sounds for the closest match. Since the 'database' is simply a text file containing feature data which is not optimised or indexed in any way, this time could be cut down by orders of magnitude through the implementation of a real database. The database could also be increased in size significantly, which could increase the consistency and quality of its matching performance. To investigate this suggestion, a further test was carried out where the database was increased from 100,000 to 200,000 points for the best performing synthesizer (the complex FM synthesizer) and the instrument matching tests were re-run. Did the matching consistency and quality really increase?

In table 3.11, the results of the further study are presented. The original 100,000 point database was supplemented with a further 100,000 points and in over half of the real instrument tests, the error decreased. In the discussion of the matching performance across the test set earlier, an error figure of around 0.1 was said to be an acceptably low error when matching sounds the synthesizer was capable of copying perfectly. The improvements observed here are in the range 0.01 to 0.24, so are significant by this measure.

The data driven search was successful but the genetic algorithm was more so. How might the genetic algorithm be improved? It typically converges after around 50-100 generations, which involves the extraction of 200,000 feature frames in 200 batches as well as the machinations of the algorithm itself. This typically takes around 3 seconds

Target	200K score	200K error	100k score	100k error	Improvement
AltoFlute.mf.C4.wav_attack	35.30	1.19	34.03	1.23	0.04
AltoFlute.mf.C4.wav_sustain	20.13	2.09	18.03	2.33	0.24
AltoSax.NoVib.mf.C4.wav_attack	32.65	1.29	32.65	1.29	0
AltoSax.NoVib.mf.C4.wav_sustain	29.53	1.42	29.53	1.42	0
Bassoon.mf.C4.wav_attack	18.61	2.26	17.78	2.36	0.11
Bassoon.mf.C4.wav_sustain	17.29	2.43	16.27	2.58	0.15
BbClar.mf.C4.wav_attack	19.67	2.14	19.10	2.20	0.06
BbClar.mf.C4.wav_sustain	18.03	2.33	18.03	2.33	0
Cello.arco.mf.sulC.C4.wav_attack	19.74	2.13	19.74	2.13	0
Cello.arco.mf.sulC.C4.wav_sustain	17.79	2.36	17.79	2.36	0
Horn.mf.C4.wav_attack	22.96	1.83	22.35	1.88	0.05
Horn.mf.C4.wav_sustain	20.29	2.07	19.80	2.12	0.05
oboe.mf.C4.wav_attack	18.16	2.31	16.94	2.48	0.17
oboe.mf.C4.wav_sustain	16.10	2.61	16.01	2.62	0.01
Piano.mf.C4.wav_attack	18.87	2.23	18.74	2.24	0.02
Piano.mf.C4.wav_sustain	20.40	2.06	20.40	2.06	0
Trumpet.novib.mf.C4.wav_attack	17.00	2.47	16.40	2.56	0.09
Trumpet.novib.mf.C4.wav_sustain	18.09	2.32	18.09	2.32	0
Violin.arco.mf.sulG.C4.wav_attack	24.30	1.73	24.30	1.73	0
Violin.arco.mf.sulG.C4.wav_sustain	19.67	2.14	19.67	2.14	0

Table 3.11: This table compares the timbre matching performance of the data driven nearest neighbour search with 100,000 and 200,000 point data sets from the complex FM synthesizer. The final column shows the reduction in error observed with the larger data set.

Target	Standard GA score	Hybrid GA score	Standard GA error	Hybrid GA error	Error difference
AltoFlute.mf.C4.wav_attack	40.9549	35.3057	1.0255	1.1896	0.1641
AltoFlute.mf.C4.wav_sustain	22.0323	20.5850	1.9063	2.0403	0.1340
AltoSax.NoVib.mf.C4.wav_attack	34.5376	32.6473	1.2161	1.2865	0.0704
AltoSax.NoVib.mf.C4.wav_sustain	35.2891	32.5919	1.1902	1.2887	0.0985
Bassoon.mf.C4.wav_attack	18.4056	23.0549	2.2819	1.8217	-0.4602
Bassoon.mf.C4.wav_sustain	17.4296	19.5440	2.4097	2.1490	-0.2607
BbClar.mf.C4.wav_attack	21.9440	19.6708	1.9140	2.1351	0.2212
BbClar.mf.C4.wav_sustain	19.0656	19.6745	2.2029	2.1347	-0.0682
Cello.arco.mf.sulC.C4.wav_attack	21.0522	23.4298	1.9950	1.7926	-0.2025
Cello.arco.mf.sulC.C4.wav_sustain	21.9489	18.8694	1.9135	2.2258	0.3123
Horn.mf.C4.wav_attack	22.9452	27.2126	1.8304	1.5434	-0.2870
Horn.mf.C4.wav_sustain	21.2800	22.8407	1.9737	1.8388	-0.1349
oboe.mf.C4.wav_attack	19.5410	18.1563	2.1493	2.3132	0.1639
oboe.mf.C4.wav_sustain	18.6582	16.8630	2.2510	2.4907	0.2396
Piano.mf.C4.wav_attack	21.2228	19.1744	1.9790	2.1904	0.2114
Piano.mf.C4.wav_sustain	22.3547	22.6377	1.8788	1.8553	-0.0235
Trumpet.novib.mf.C4.wav_attack	18.1214	19.4593	2.3177	2.1583	-0.1594
Trumpet.novib.mf.C4.wav_sustain	17.7829	18.0937	2.3618	2.3212	-0.0406
Violin.arco.mf.sulG.C4.wav_attack	25.0713	24.3025	1.6752	1.7282	0.0530
Violin.arco.mf.sulG.C4.wav_sustain	19.7671	19.6673	2.1247	2.1355	0.0108

Table 3.12: This table compares the standard GA which starts with a random population to the hybrid GA which starts with a population derived from a data driven search.

per batch or more if the synthesizer is more complex. This could be reduced with the implementation of a multi-threaded algorithm, probably to $\frac{1}{numberofCPUcores}$ but it will never match the speed of the data driven approach, even with the latter in its unoptimised form. Therefore a hybrid approach is proposed, where the initial population for the genetic algorithm is generated by finding the 1000 closest matches to the target using the data driven search. This system was implemented and it was run against the 20 instrument sounds using the complex FM synthesizer. The results of the standard GA and the hybrid GA are compared in table 3.12. The results do not show the hybrid GA to be superior, it performs worse about as many times as it performs better.

3.4.3 Conclusion

Five sound synthesis algorithms have been described and their behaviour in feature space as their parameter settings are varied has been investigated. Five optimisation techniques have been described and their performance at the problem of eliciting appropriate parameter settings for the sound synthesis algorithms has been extensively tested. The best performing optimisation technique was the genetic algorithm, which searches well and with reasonable consistency over all of the fixed architecture sound synthesizers. It was not the best performer in the search of the space of sounds that can be generated by a variable architecture synthesizer, however, being beaten by the more conservative hill climber. The feed forward neural network failed nearly completely at the task. The greedy search offered by the data driven approach was shown to be more efficient than the genetic algorithm as well as having nearly comparable search performance in several cases. Increasing the greed of this search by using a bigger data set was shown to be effective in an initial test. The hybrid form of the genetic algorithm and the greedy search was not shown to have better performance in an initial test. Regarding the eliciting of optimal parameter settings for the synthesizers for the matching of real instrument timbres, the complex FM synthesis algorithm, similar to that found in the Yamaha DX7, was found to be superior to analog-style subtractive algorithms overall. However, the subtractive algorithms still performed well and better than FM in some cases such as the Violin and French Horn sounds. There was no apparent pattern at work here, e.g. the subtractive synthesizers being better at making string sounds.

The next chapter is about SynthBot. SynthBot is an automatic VSTi sound synthesizer programmer which is implemented based partly on the results of this study.