Generating Rhythmic Patterns: a Combined Neural and Evolutionary Approach

Marcus Pearce

MSc in Artificial Intelligence
Division of Informatics
University of Edinburgh
September 2000

Abstract

The objective of this research was to design a creative aid to musical composition: a system that would generate a (user specified) number of drum patterns, within a specified style and showing a sufficient amount of variation, on any one run. In order to achieve these aims, a genetic algorithm was implemented using as its critic a multi-layer perceptron, trained on a set of drum patterns from the style of “drum and bass”. Domain specific knowledge was incorporated into a number of areas of the GA and an island model was used to generate multiple solutions. While some degree of success was achieved, experiments conducted using human subjects questioned the style and comparability of the generated patterns to human generated patterns. These partial failures of the system to achieve the stated aims were attributed to shortcomings of the data used to train the ANN critic. Suggestions for future research are presented in terms of improving the critic, extending the approach and better evaluation of machine compositions.

Acknowledgements

I would first of all like to thank my supervisor, Alan Smaill, for his patient and careful guidance during the course of the research reported here. Thanks is also due to Luke Phillips and Geraint Wiggins for useful discussions during my period of study and for comments on earlier drafts. I am also grateful to my fellow MSc students for taking part in the evaluation experiments and to the University of Edinburgh for supplying funds for required materials. Finally, I thank the EPSRC who provided financial support during my year of study via studentship no: 99407250.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

The motivation for this project was to produce a creative aid to musical composition by developing a system that would generate short rhythmic sequences based on examples taken from a particular style. The generated patterns would form libraries of new musical phrases which the composer would be able to incorporate into his or her own compositions or draw from as a source of rhythmic inspiration. An important feature of the system was that it should be able to generate patterns displaying a degree of variation (within the specified style).

1.2 Aims

The specific aims of the project were to develop a system that would generate a (user specified) number of short drum patterns on a single run. Furthermore, the generated patterns should conform to the following criteria:

1. They should be comparable with human generated patterns
2. They should be within a specified style
3. They should show sufficient variation both between and within runs to make the system a useful tool
4. They should show features that would not naturally be generated by the user

An important supplementary aim was to extend the approach to more than one style of music. Finally, the system should be practical and intuitive to use.

1.3 Strategy

The proposed strategy to achieve these aims was to use a Genetic Algorithm (henceforth GA) with an Artificial Neural Network (henceforth ANN) critic. In the suggested scheme, the GA evolved chromosomes representing drum patterns that were deemed fit by an ANN trained on a set of exemplar patterns from the chosen style. The use of a GA to achieve these aims was motivated by the following factors:

- GAs provide an efficient means of searching large, complex search spaces such as the present problem domain (the space of all possible drum patterns).
- An evolutionary approach was attractive since musical composition often involves the combination and permutation of themes (rhythmic subpatterns in this case).
- The framework of a GA allows a balance of exploitation and exploration of the search space. This would make possible the generation of patterns showing a certain amount of variation while still constraining search to areas of high fitness.
- Evolutionary techniques lend themselves to generating multiple solutions on any one run.

Furthermore, a number of factors influenced the decision to use an ANN critic (see section 3.3.4 for a discussion of these issues):

- The ANN provides a fully automated critic avoiding the problem of the fitness bottleneck associated with interactive human evaluation.
- An ANN can be trained on examples taken from the style of music to be modelled thereby avoiding both the subjective bias involved in human evaluation and the difficulty (and artificial nature) of formulating explicit evaluation criteria.
- The generalisation abilities of the ANN can be employed to alleviate the determinism associated with rule based critics, thereby allowing greater diversity in the generated drum patterns.
- An ANN can be trained on examples from several different styles of music (removing the need to design a entirely new knowledge base in the case of rule based critics).

Only two studies have investigated the application of evolutionary techniques specifically to rhythmic patterns. [Horowitz, 1994] used an Interactive GA to evolve drum patterns and reported successful results. The IGA is viewed as less than satisfactory for a number of reasons outlined in section 3.3.4. This approach was improved by [Burton, 1998] who automated the critic, using an unsupervised ART network to evaluate candidate drum patterns. Although success was again reported, a certain homogeneity was noted in the generated patterns. The present research can be seen as an extension of these studies, attempting specifically to increase the diversity and novelty of the generated drum patterns through the use of a trained Multi-Layer Perceptron as an automated critic.

1.4 Evaluation

The system was evaluated experimentally on a number of objective measures which directly related to the research aims stated above. These measures were:

1. A musical Turing test of the patterns generated (the degree to which system and human patterns could be distinguished)
2. The degree to which the patterns were within the specified style
3. The amount of variation shown by the generated phrases both within runs and between runs

Experiments designed to evaluate the generated patterns on the basis of these measures demonstrated that the patterns were only partially successful in meeting these criteria. Other non-experimental criteria used to evaluate the system were:

- The evaluation of the system by a musician, who uses MIDI drum parts in his compositions, on the basis of its ability to generate patterns that he considered aesthetically pleasing and would not naturally compose himself.
- Practical issues such as ease of use and the time it takes to run

1.5 Overview of the Dissertation

The contents of this dissertation are as follows. This chapter has introduced the motivation, aims, strategy employed to achieve these aims and the evaluation of the system. Chapter

2 provides an overview of the main features of genetic algorithms and neural networks that the reader should be familiar with. Chapter 3 focuses in more detail on the literature relevant to this research, including the application of evolutionary methods and ANNs to musical composition. Chapter 4 describes in detail the design and development of the system including a discussion and justification for design decisions made. In chapter 5, the system is analysed and evaluated on a number of measures, both objective and informal. The final chapter of this dissertation presents a discussion of the research and the conclusions drawn, along with proposals for future work.

Chapter 2

Background on Techniques

2.1 Overview

This chapter presents a brief introduction to the concepts and main features of genetic algorithms and artificial neural networks. These were the two AI techniques used in the developed system and readers familiar with these techniques may proceed directly to chapter 3.

2.2 An Introduction to Genetic Algorithms and Genetic Programming

2.2.1 Overview

Genetic Algorithms are general tools for search and optimisation inspired by the biological theory of natural selection and first proposed by [Holland, 1975]. They have been applied in a wide range of domains such as optimising circuit board layout and timetabling and scheduling problems (see [Ross and Corne, 1995] for a review of applications of GAs) and have been shown to be particularly useful in searching large, unstructured search spaces. The following is a brief summary of the main features of GAs and GP (see e.g., [Goldberg, 1989] or [Heitkotter and Beasley, 2000] for a further details).

A GA typically consists of the following components:

- *A representation scheme*: this describes the domain specific mapping between the real-world features of a solution to the problem (the *phenotype*) and the representation of that solution to be used by the GA (the *genotype*). A candidate problem

solution converted into this representation is known as a *chromosome* (or *individual*).

- *A population*: this is the group of chromosomes which are to be evolved. Chromosomes in turn are said to be made up of *genes* each of which may have a particular value (an *allele*) and these genes make up the genotype of that individual. In a GA, the population is typically initialised with a fixed number of randomly generated chromosomes. The number of individuals in the population is the *population size* and is usually fixed for any one run of the GA.

- *A selection method*: this is a means of selecting individuals from a population by which chromosomes of higher fitness have a greater probability of being chosen.

- *Genetic operators*: crossover is a mechanism by which selected chromosomes are combined to generate new candidate solutions while mutation operators randomly generate variations on existing material.

- *Critic*: the critic or *fitness function* returns a measure of how well the solution represented by a chromosome solves the problem (its *fitness*).

- *Stopping criterion*: this defines when evolution stops and the solutions in the current population are returned. Typical examples include stopping after a fixed number of generations, when a chromosome of a particular fitness is obtained or when all individuals in the population are identical *(convergence*).

2.2.2 The Algorithm

The sequence of steps shown in algorithm 1 describes how these components (which are discussed in more detail below) are combined in [Goldberg, 1989]'s Simple GA (SGA).The algorithm shown is that for a *generational GA* since a whole new population of children is generated which replaces the entire parent generation. A *steady state GA* on the other hand generates children individually which, if they are fitter than the least fit of the parent generation, immediately replace that parent.

Algorithm 1 The Generational GA

1. Create an initial population of individuals.
2. ***Evaluation:*** the critic returns a fitness for each chromosome.
3. ***Selection:*** fitter chromosomes selected for an intermediate population.
4. ***Crossover:*** apply crossover with a certain probability to all chromosomes and a randomly selected mate from the intermediate population to generate two children which are placed in the new population; otherwise the chromosomes are copied straight into the next generation unchanged. Repeat until the new population is full.
5. ***Mutation:*** mutate all genes with a certain probability.
6. Repeat steps 2 to 6 until the stopping criterion is reached (each iteration is known as a *generation*).

2.2.3 The Representation Scheme and Critic

These are the two main areas in which knowledge about the domain is built into the GA. The representation scheme used will depend on the problem that needs to be represented and may be chosen in order to help cut down the search space. As with many areas of AI where knowledge representation is a critical issue, choosing an appropriate chromosome representation is a key problem in the design of a GA ([Goldberg, 1989]). Typically, the chromosome will be a string of characters which, although much of the theory behind GAs is based on binary strings, are often members of non-binary alphabets chosen because of their applicability to particular problems.

The role of the critic can be compared to that of the environment in natural evolution: the fitness of any individual creature is defined by its environment which determines the degree to which the phenotypical characteristics of the individual help or hinder its ability to survive and reproduce. In a GA, knowledge of the problem domain is drawn upon to determine how well a solution represented by a chromosome will solve the problem and thereby calculate its fitness. There are many ways in which a critic can be implemented. For example, it may compare candidate solutions to an ideal solution, or calculate fitness based on a series of formal criteria or rules. Alternatively, a human user may return feedback on the quality of the solutions represented by the chromosomes (this is known as an Interactive GA or IGA).

2.2.4 Selection and Genetic Operators

The purpose of selection is to ensure that the fitter chromosomes have a greater chance of reproducing, thereby propagating their fit genes into the next generation. A number of

selection schemes are commonly used in GAs including *rank based selection*, *tournament selection* and *remainder stochastic sampling.* Tournament selection, for example, randomly selects *N* chromosomes from the population and chooses the winner of this tournament to reproduce. Larger tournaments result in increased *selection pressure*, while smaller tournaments result in higher sampling error (see section 4.6).

If reproduction were asexual, that is genotypes were copied as they are into the next generation there would be no means of exploring new areas of the search space not represented in the current population (besides mutation - see below). Crossover, therefore, is a means of combining individuals with high fitness to potentially discover new high fitness areas of the search space. The basic method of crossover, known as *single point* crossover, involves randomly selecting a point in the chromosome (the *crossover point*) and cutting both parents at this point.

Parent1: 0000000|00000
Parent2: 1111111|11111
Child1: 0000000|11111
Child2: 1111111|00000

Figure 2.1: Single Point Crossover

Two children are then generated by combining sections from each parent (see figure 2.1). Crossover is normally applied to selected chromosomes with a certain probability (the *crossover rate*).

It is quite possible that after a few generations certain *alleles* will have been lost from the population. In fact a general difficulty with genetic techniques is the tendency for one relatively fit individual to saturate the population resulting in a population of identical individuals. When this occurs without the GA having found a global optimum it is said to have converged prematurely. Mutation is a means of introducing new genetic material into the gene pool, thereby guarding against premature convergence. A common mutation operator (using a binary representation) simply flips the value of a gene from 0 to 1 or vice versa. It is important to note, however, that mutation can also disrupt a population away from optimal regions of the search space and therefore is typically applied with a small probability (the *mutation rate*).

Mutation also has an important role as a search operator (some evolutionary techniques rely on mutation alone to guide search). While crossover contributes to the search process by combining fit sections of individuals to generate even fitter children, mutation conducts

a search by hill-climbing (partially exploring the neighbourhood of the current state and moving in directions that improve fitness).

2.2.5 Genetic Programming

Genetic programming ([Koza, 1992]) is a variation on the evolutionary theme in which programs, which generate problem solutions, are evolved. A fit program can be run with a given input to generate the desired output. The basic elements of these programs are problem specific *terminal* and *function* sets. The initial population contains a number of programs consisting of randomly combined elements from the terminal and function sets. The fitness of each program is assessed by running it on a series of inputs called *fitness cases* and then applying the critic to the output of these runs. As in a GA, the fittest individuals (programs) are chosen by some selection scheme (as described above) and then crossover is used to generate two offspring from pairs of the selected parents. Typically, crossover involves the exchange of a randomly chosen subtree of each program. Finally, the children are mutated with a low probability (this might involve, for example, substitution of one terminal or function for another) and placed in the next generation.

A notable difference between GAs and GP is that while the chromosome in a GA is typically (but not always) of a fixed length, in GP the programs evolved can be of any length. Similarly the problem solutions generated by those programs may be of variable length.

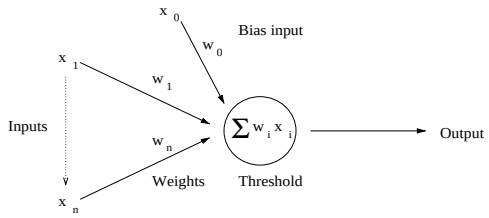
2.3 An Introduction to Artificial Neural Networks

2.3.1 Overview

Artificial Neural Networks constitute a family of AI techniques, inspired by models of biological neural systems, which consist of interconnected networks of simple processing elements called *nodes.* In contrast to the sequential symbolic processing used in other fields of AI, the processing elements of ANNs operate in parallel on numerical data (although this is typically simulated on sequential *von Neumann* style machines). For the purposes of this thesis just the type of ANN used in this research (see section 4.5) is described: the *perceptron* (see [Haykin, 1999] for a general introduction to neural networks and [Sarle, 1997] for an excellent online FAQ).

2.3.2 The Perceptron

The first neuronal model was developed by [McCulloch and Pitts, 1943] and extended by [Rosenblatt, 1959]. The *perceptron* consists of a simple unit which performs thresholding on n inputs $x_1 ... x_n$, each of which may take a value of 0 or 1. There is also a fixed input called the *bias node* whose value is always 1 and which has an associated weight w_0 which serves to shift the decision boundary away from nought.



$$\text{output} = \begin{cases} 1 \text{ if } \sum w_i x_i \geq 0 \\ 0 \text{ if } \sum w_i x_i < 0 \end{cases}$$

Figure 2.2: The Perceptron Unit

As shown in figure 2.2, the threshold unit calculates the weighted sum of the inputs. If this weighted sum is greater than 0 then the output of the unit is 1 else the output is 0. Algorithm 2 shows the sequence of steps used to train the perceptron.

Algorithm 2 The Perceptron Training Algorithm

```
1. Initialisation
        Set the weights, W_i , to small random  values

2. Training
        For each epoch of training
              For each input  pattern
                    Present  pattern
                    Calclulate ouput of unit  (o)_
                    Calculate  adjustment  of weight  associated
                    with input x_i   and adjust weight .
        Until termination criterion
```

There are a number of methods for updating the weights: the *perceptron training rule*, for example, calculates the new weights as

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

where η is the learning rate, a parameter that governs the amount weights are changed on each epoch of training; t is the desired output for a particular input x_i; and o is the actual output of the unit with that input.

In this manner, the weights associated with each input are changed by small amounts on each *epoch* of training (iteration of the algorithm) in such a way as to reduce the error between the actual output and the desired output. Over many epochs a weight vector is found that satisfies all of the input patterns. The perceptron can be trained to classify certain patterns of inputs such as the boolean functions AND, OR and NOT. An architecture with any number of these units can be used and the above algorithm applied to each. However, it has been demonstrated that in order to solve non-linear problems, such as the boolean XOR function, a more complex architecture is needed consisting of multiple layers of perceptron units. Furthermore, the perceptron training rule is inadequate for such an architecture.

2.3.3 Backpropagation and the Multi-Layer Perceptron

The multi-layer perceptron (MLP) is a network of perceptron units consisting of an *input layer*, a *hidden layer* and an *output layer* as shown in figure 2.3. An MLP was used in the system to be described (section 4.5).

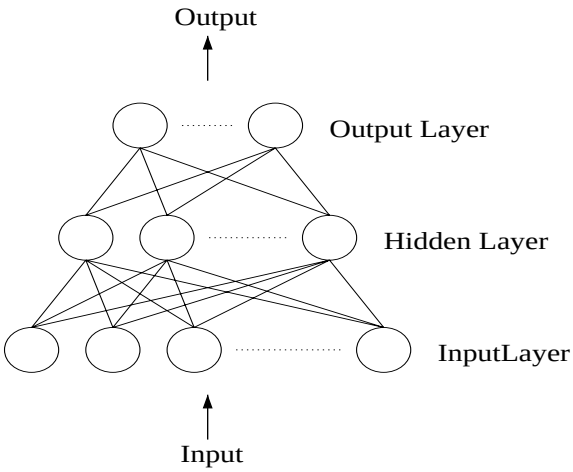


Figure 2.3: The Multi-Layer Perceptron

The backpropagation algorithm (see [Rumelhart and McClelland, 1986]) was developed to learn the weights in a multilayer perceptron. Once again, an input pattern is presented to the network and propagated forward through the layers to generate a pattern on the output units. This output is compared to the desired output for that input pattern and the error propagated back through the network layers and the weights at each layer updated. A feature of back propagation is that it requires a continuously differentiable *activation function* rather than the discontinuous step function used by the original perceptron. One

commonly used solution is the sigmoid function which computes an output o as

$$o = \frac{1}{1+e^{-y}}$$

where y is the output of the weighted sum of the unit's inputs, $\Sigma w_i x_i$. The backpropagation algorithm attempts to minimise the disparity (or error) between the network output values and the target values for these outputs. The error function normally used to compute this disparity is a squared error function which, for one input pattern, is defined as

$$E = \frac{1}{2}\sum_j (t_j - o_j)^2$$

where j is the number of output units of the network t_j and o_j are the target and output values associated with the jth output unit. Backpropagation makes small changes to the network weights on each iteration to reduce this error. The aim is to find a set of weights where the error E is minimised (as close to nought as possible) for all input vectors .

The weight update rule used by backpropagation is very similar to that used in the perceptron training rule with the exception that a momentum term is often added. For the weight w_{ji} between units i and j

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$
$$\Delta w_{ji} \leftarrow \eta\, \delta_j x_{ji} + \alpha \Delta w_{ji}$$

The first term in the weight update is the same as the perceptron training rule: the error δ_j on unit j multiplied by the input to unit j (x_{ji}) multiplied by the learning rate (η). The second term adds the previous update for that weight (Δw_{ji}) multiplied by the momentum parameter (α). While the learning rate sets the step size or amount by which weights are changed on each iteration, the momentum parameter governs the degree to which movement over the error surface is maintained in the same direction. These parameters must be set to appropriate values which will depend on the problem being learned. For an explanation of how backpropagation calculates the error term on the hidden units see, for example, [Haykin, 1999]. Backpropagation has been improved in a number of ways resulting in algorithms such as *Extended-Delta-Bar-Delta*, used in the research described here, which heuristically optimises the learning rate and momentum settings.

Multi-layer perceptrons using the backpropagation training rule have been found to

provide a robust approach to learning many types of pattern recognition, function approximation and other classification problems. They exhibit many desirable features such as the ability to generalise beyond the training set and robustness in the presence of noise in the training data. Examples of areas in which they have been successfully applied include learning to recognise handwritten characters, learning to play backgammon and financial prediction.

2.4 Summary of Chapter 2

This chapter has introduced the main features of GAs and the MLP.

Chapter 3

Related Work

3.1 Overview

This chapter contains a review and discussion of research related to the work presented in this dissertation. In section 3.2 the application of AI techniques to the musical domain and the field of algorithmic composition are introduced. Section 3.3 presents a review of the research literature concerning the application of evolutionary techniques to musical composition. A similar but shorter discussion of ANNs to musical problems can found in section 3.4. While the aims of many of the studies to be discussed lie outside the limited domain of this research, the implications of this previous research to the current work are important and are duly discussed.

3.2 AI and Music

There are a number of ways in which AI techniques have been applied to the musical domain. As in much of AI, the motivations involved can be drawn out on a spectrum between Cognitive Science at one end and Engineering at the other [Wiggins and Smaill, 1999]. In the former case the aim is to model and understand such features of human musical cognition as: the perception of music; practice, learning and musical expertise; composition, performance and improvisation; and musical creativity. [Desain and Honing, 1991], for example, have investigated a connectionist approach to modeling human rhythm perception. An intermediate motivation for using applying AI to musical tasks is to aid the field of musicology. As an example, [Desain and de Vos, 1990] describe a system for the analysis of the relation between expression and structure which, for example, facilitates the the study of voice-leading and chord timing.

Finally, at the other end of the scale, many studies aim to produce intelligent systems that will aid or augment the processes of learning, composition or performance of music. For example, [Horner and Ayers, 1995] developed a genetic algorithm for harmonising complex musical progressions and suggested that, as well as providing composers and arrangers with a tool for harmonisation, the system would also provide music theory teachers with an automated means of both testing new progressions to ensure a solution exists and also grading students solutions. In the field of performance [Biles, 1994] has developed a genetic algorithm, called *GenJam* that generates jazz solos in real time and which can "trade fours" with a human soloist; Biles notes that "in many ways, GenJam is the most formidable 'opponent' [he] has encountered."

The research described here concerns the development of a system to aid the composition of music. Within this field of *algorithmic composition*, the aims of researchers range from attempts to produce autonomous composers[1] to developing systems which can create libraries of useful musical phrases that the composer may draw upon in the process of composition (e.g., [Biles, 1994]). This latter has been taken as the goal of the current work.

3.3 Evolutionary Approaches to Algorithmic Composition

3.3.1 Why use Evolutionary Techniques?

A number of AI techniques have been applied to various tasks relating to the composition of music. These range from mathematical models, such as Markov chains, through symbolic methods, such as grammars and knowledge based systems to evolutionary models and sub-symbolic techniques, including ANNs (see [Papadopoulos and Wiggins, 1999] for a summary and review of the advantages and disadvantages of several AI techniques for algorithmic composition). The problem of algorithmic composition as it presents itself to the AI scientist is neatly summarised by [Jacob, 1995]:

> "consider the set of all possible compositions as the solution space, with the problem at hand being 'find a composition that sounds good."'

With this in mind, evolutionary methods such as GAs and GP would seem particularly applicable to the problem on the strength of their ability to efficiently search large, unstruc-

[1] [Spector and Alpern, 1994] discuss attempts to produce what they call "constructed artists."

tured problem domains. These techniques also possess other desirable attributes such as the ability to generate multiple solutions.

It might also be expected that evolutionary techniques would constitute a suitable framework for implementing a compromise between the generation of structured and novel musical phrases since there is a certain amount of flexibility in the degree to which search is guided. This is an important feature since a repeated theme in the literature of algorithmic composition is the tension that lies in the trade-off between structure and novelty. In the words of [Todd and Werner, 1999]:

> "More structure and knowledge built into the system means more reasonably structured musical output, but a also more predictable, unsurprising output; less structure and knowledge in the system means more novel, unexpected output, but also more unstructured musical chaff."

Indeed, genetic algorithms have enjoyed a certain amount of success in this regard in the generation of visual art. William Latham (in [Thywissen, 1996]) has said of his evolutionary systems for evolving 3-d computer sculptures:

> "The machine has given me freedom to explore and create complex ... forms which previously had not been accessible to me, as they had been beyond my imagination."

Finally, [Ralley, 1995] notes that GAs might be particularly appropriate to algorithmic composition since an important part of musical composition is the "processing of ideas through systematic permutation and recombination of themes."

In accordance with these expectations, GAs have been successfully applied to a variety of tasks within the field of musical composition ranging from the evolution of instrumental jazz solos[2] to drum patterns[3]. Genetic Programming has also been employed as an evolutionary technique in a number of studies of algorithmic composition (see e.g., [Johanson and Poli, 1998] and [Spector and Alpern, 1994]). Most of these studies have focussed on a restricted musical domain in an effort to reduce the size of the search space and, therefore, the time taken to converge to a satisfactory solution. Others, however, have imposed a certain amount of structure and worked with larger musical segments (see

[2] See, for example, [Biles, 1994], [Spector and Alpern, 1994] and [Wiggins et al., 1999].
[3] See, for example, [Horowitz, 1994] and [Burton, 1998].

[Jacob, 1995], for example). Appendix A provides brief summaries of the studies discussed in this review.

Since GAs and GP are general methods, the most interesting features of their application to musical tasks lie in those areas in which domain specific knowledge has been exploited. The obvious places in which this typically takes place are the chromosome representation and the critic. However, there are a number of other places such as the initialisation of the population and the genetic operators used which also provide opportunities for incorporating available domain specific heuristics into the GA [Greffenstette, 1987].

The following is a review of previous work which has applied GAs and GP to algorithmic composition. We begin with a discussion of the representations and genetic operators employed and then move on to a more detailed examination of the types of critic used. Finally, approaches taken to evaluating the systems will be reviewed. Although the aims and domains of application of many of these studies range outside the limits of the current research, analysis of the success and failure of the various techniques used was instructive in the design of the system developed here. Furthermore, this study of the literature will provide an academic context for the present research. (See [Burton and Vladimirova, 1997b] or [Todd and Werner, 1999] for more complete reviews of the various applications of evolutionary methods to musical composition).

3.3.2 Representation Schemes

A feature of GAs and GP is that they can achieve efficient heuristic search with domain specific knowledge used only in the representation scheme and critic (the latter is discussed in section 3.3.4). Of particular interest to the current research is the representation of time in the chromosome and, especially, the tradeoff between capturing all the relevant features of a musical phrase and constraining the search space.

In general, the representation schemes in musical applications of GAs and GP have used implicit time structuring: events are represented without explicit temporal and structural relations between events (see [Honing, 1990] for a discussion of the representation of time and structure in music). In a typical scheme, the number of genes in a chromosome corresponds to the quantisation of the bar into discrete timesteps where notes may be placed. Most studies have used a relative time base (e.g., crotchets) rather than representing absolute time (e.g., seconds), in order to remove unpromising regions from the search

space. The metric position of the notes within a bar is, therefore, constrained to those subdivisions represented in the chromosome. The finest metric subdivision represented has varied between crotchets (e.g., [Gibson and Byrne, 1991]) and demi-semiquavers (e.g., [Spector and Alpern, 1995]).

However, a problem with this type of representation is that it is very restrictive. For example, a representation that quantises the bar into semiquaver subdivisions (a chromosome length of 16 if the tempo is 4/4 and one bar is represented) not only excludes the possibility of playing demi-semiquavers but also cannot represent triplets or many time signatures such as 6/8, 5/4 and so on.

One major variation on this theme concerns the primitives used. In the scheme outlined above, the primitives are "points" - the timing of a note event is described by its position in the chromosome. An alternative scheme involves the representation of note onset and duration (see e.g., [Thywissen, 1996], [Phon-Amnuaisuk et al., 1999]). This representation scheme allows for more natural representation of triplets and "alternative" time signatures as well as such features as overlapping notes. A similar scheme suggested by [Urwin, 1997] represents the number of timesteps since the previous note, thereby avoiding the need to explicitly represent the timestep of the note.

An interesting representation scheme was used by [Horowitz, 1994]. One bar rhythmic phrases were represented as sequences of notes and rests occurring on "pulse" (assumed to be semiquavers) subdivisions of the bar. However, also incorporated were values representing the "rhythmic activity" ranging from triplet semiquavers to crotchets, "syncopation" which determined the degree of accentuation of sequential notes with respect to the beat and "accent structure". Unfortunately, [Horowitz, 1994] does not give a very detailed description of how these elements were combined in his representation scheme.

Another important aspect of the current work is the representation of several instruments (the different drums in a drum kit). Both [Burton, 1998] and [Horowitz, 1994] used a 2-d matrix to represent different drums with time along one axis and instrument number along the other[4]. An alternative scheme proposed by [Urwin, 1997] explicitly associates each note event with an instrument number.

Finally, it is worth noting that with the exception of just a few studies[5], non-binary

[4] A similar scheme has been used in the representation of four voice harmonisation ([Phon-Amnuaisuk et al., 1999]).

[5] e.g., [Gibson and Byrne, 1991], [Biles, 1994] and [Burton, 1998]

alphabets (usually integer encodings) have been used in the representation of musical phrases. For example, genes are often integers representing the pitch, velocity or duration of a note on a particular timestep.

We learn from this analysis that, while it is important to represent all the necessary information in the chromosome, the use of constraints can help to remove musically unpromising regions from the search space. The issues discussed above were considered during the design of the representation scheme used in the work described here (see section 4.2).

3.3.3 Genetic Operators

The vast majority of the studies under review have employed the standard crossover operators described in section 2.2. [Horowitz, 1994] used an interesting variation on this theme by restricting crossover to similar drum groups (e.g.,. hi-hat and ride cymbal). This was done in an attempt to provide some continuity, from generation to generation, of instruments which shared similar "roles".

[Burton, 1998] has investigated a number of crossover methods deemed applicable to the domain and their effect on population diversity using a GA which evolved drum patterns. The methods studied were single and multiple point crossover, multiple point crossover between both similar and dissimilar drum groups, pattern interleaving (in which individuals are divided into the instruments represented, bits from these instruments interleaved one at a time and then single point crossover applied on the resulting individuals) and mobius crossover (in which two individuals are concatenated, rotated to a random starting point and then separated at the centre point). It was found that mobius crossover introduced the most disruption into the population (while use of the other operators resulted in low disruption) and [Burton, 1998] concluded that this operator could be used to introduce diversity in the event of premature convergence.

The use of music specific mutation operators has shown much greater popularity in the literature. The following is a discussion of those most relevant to the evolution of rhythmic patterns. Some mutation operators used have operated on entire musical sequences (the entire chromosome or fragments thereof) while others have operated on individual notes. Examples of the former include pattern reversal and rotation (see e.g., [Biles, 1994], [Ralley, 1995], [Thywissen, 1996], [Wiggins et al., 1999]) while some ex-

amples of the latter are note addition, deletion, substitution and exchange between voices (see e.g., [Horner and Goldberg, 1991], [Phon-Amnuaisuk et al., 1999]). Finally, some studies have used an operator that mutates the note durations (e.g., [Wiggins et al., 1999]).

In these cases, the mutation operator is not being used simply as a means of ensuring continuing diversity in the gene pool but also as a search operator (see section 2.2.4). This use of domain specific knowledge to guide search is very important as a means of reducing the problem of the fitness bottleneck in those systems that have used human critics (discussed in section 3.3.4) and also in systems using relatively unconstrained representation schemes.

[Burton, 1998] has studied the effect of six commonly used mutation operators on population diversity, once again in a GA which evolved drum patterns. It was found that pattern inversion, reversal and voice swapping had a more disruptive effect than single or multiple bit (note) substitution or pattern rotation. As with mobius crossover, [Burton, 1998] suggests that these operators could be used to introduce diversity into a population that has converged prematurely. It is interesting to note that rotation of the drum patterns did not cause great disruption. This suggests that a "good" drum pattern, when rotated, may often still be a "good" drum pattern.

This review indicates that the use of mutation operators specific to the musical representation used can be a useful technique for guiding search through complex musical domains. This analysis motivated the inclusion of such operators in the system developed in the course of this research (see section 4.6.5).

3.3.4 The Critic

This section contains a detailed examination of the use of human, rule based and neural network evaluation functions in evolutionary musical systems as well as a section on evolving the critic. The advantages and disadvantages of each method of implementing the critic are discussed using examples from the literature. This analysis of previous work was instructive in choosing and designing the critic in the present system.

Human evaluation

Researchers have reported some success using IGAs for algorithmic composition (see Appendix A.1).

Advantages. IGAs are useful for generating music that satisfies the preferences of a particular user. The advantage of this is that, as noted by [Thywissen, 1996] "people tend to be far more sophisticated in listening than in creating music"; that is people can say whether they like a tune or not but may not be able to compose a tune that they like. The other related benefit is that the developer of the system is relieved of the difficult task of formalising aesthetic merit. Of course the importance of these potential advantages of the IGA depend on the precise aims of the research.

Disadvantages. However, against these potential advantages of an IGA lie some serious drawbacks. The first results from the fact that each chromosome must be listened to and rated by the user which places strict limitations on the population size and number of generations that can be used. [Biles, 1994] has coined the term *fitness bottleneck* to describe this phenomenon.

Researchers have developed a number of means of overcoming this problem including: seeding the initial population with relevant examples ([Ralley, 1995]) to start the evolutionary process in a fit area of the search space; using musically relevant mutation operators to guide search in promising directions ([Biles, 1994]); and using user-defined rules to guide evolution to a certain point at which the user takes over the role of critic ([Horowitz, 1994], [Thywissen, 1996]). These are, in fact, general methods of incorporating domain specific knowledge into the GA and the use of non-random initialisation, music specific mutation and rules in the critic have been investigated in this research (see section 4.6.5).

However, the IGA suffers from a second and potentially more serious problem concerning the subjectivity involved. Al Biles reports some anecdotal findings from the mentoring sessions with GenJam, his IGA for evolving melodic jazz improvisations ([Biles, 1999]). He reports that people who lack enough knowledge about music (in general, and jazz in particular) are unable to form opinions about the generated improvisations and often feel intimidated by having to give an opinion on something they do not understand. Even people who do possess the relevant knowledge soon tire from the intensity of having to actively listen to and criticise music in real time. The inconsistencies introduced due to such factors as concentration span, mood, familiarity with the domain and general musical preferences constitute a potentially very serious problem with the IGA[6] .

[6] see [Biles, 1999] for some methods used to alleviate these problems.

Summary. These problems might not matter if the aim is to produce a tool that will let experienced musicians evolve musical phrases that he or she likes. However, the aim of the present research is to generate drum patterns within a particular style and not simply those that the user likes. The issues of subjectivity associated with human evaluation also make the IGA unattractive for this purpose. Furthermore, the fitness bottleneck associated with the IGA makes it, in general, a less than desirable technique. Finally, from the perspective of AI such a system is not especially interesting; the user is essentially providing most of the knowledge to the GA. It would be more desirable, therefore, to automate the critical evaluation of chromosomes (see the sections on rule based and ANN critics below).

Rule Based evaluation

While [Horowitz, 1994] and [Thywissen, 1996] reduced the fitness bottleneck problem somewhat by allowing the user to define rules that would guide evolution up to a certain point, many researchers have employed purely rule based critics in their evolutionary systems. This method of automating the critic can potentially overcome all the above mentioned problems associated with the IGA.

Advantages. The use of rule-based critics has seen most success in domains where objective critical criteria can be drawn from musical theory. In these areas, the problems of subjectivity and the fitness bottleneck associated with the IGA can be overcome using sets of rules. Evolutionary approaches to thematic bridging ([Horner and Goldberg, 1991]) and four-part harmonisation[7] (see e.g., [McIntyre, 1994], [Phon-Amnuaisuk et al., 1999]), for example, have produced successful results (see Appendix A.2).

Disadvantages. However, in less specific domains governed by more amorphous rules (such as is the case for drum patterns) the generation of objective critical criteria becomes a significant problem. [Spector and Alpern, 1994], for example, developed a GP system that evolved programs to transform a four bar BeBop jazz melody into a new improvisation. The critic consisted of five rules gleaned from the literature on jazz improvisation technique. Although, using a case base of 5 Charlie Parker melodies, the system took just 21 generations to evolve programs that pleased the critic, [Spector and Alpern, 1994]

[7] See [Phon-Amnuaisuk and Wiggins, 1999], however, for an experimental comparison of the performance of a GA and a rule based system on the problem of four part harmony and an argument that GAs are not suited to the problem.

conclude that “the response ... does not please us (the authors) particularly well”. Furthermore, it was found that the evolved programs did not generalise well to a new set of Charlie Parker melodies; the system lacked robustness. These failures were not attributed to the evolutionary paradigm used, but rather to the simplicity of the critical criteria.

[Wiggins et al., 1999] describe the application of a GA to the problem of jazz improvisation making extensive use of domain specific knowledge, both in the chromosome representation and the mutation operators, in order to help guide evolution in desirable directions. The critic contained much more detailed criteria than did the fitness function used by [Spector and Alpern, 1994] and the solos generated by the system were considered “quite acceptable” up to a point. However, the authors conclude that the rules still failed to fully capture the relevant features of the domain leading to music that lacked intention and large-scale structure.

These two studies illustrate the problems associated with rule based critics, especially when applied to domains where formal rules are not readily available (as is the case for drum patterns in modern styles of music). The knowledge bases required to describe such domains need to be large and complex with many context sensitive rules and exceptions to rules. The construction of such sets of rules poses, not an impossible task, but certainly a very difficult one. These problems are neatly summarised by [Minsky, 1981]:

> “...it seems that we only know some features that can help - but we know of absolutely no essential features. I do not expect much more to come of a search for a compact set of rules for musical phrases. (The point is not so much about what we mean by ‘rule’, as about how large is the body of knowledge involved.)”

A concise but underspecified knowledge base may suffer from three potential problems. First, the critical criteria will not describe the domain to a satisfactory level. Second, if the system is to generate musical phrases in a specified style (as in the present research), the critic may suffer from the subjective bias of the system designer selecting the rules. A third difficulty is the avoidance of rigidity in the music generated. This would present a problem for the current research in which the generation of patterns showing a degree of diversity is a key issue.

A final problem with rule based critics is that one set of rules will almost certainly only correspond to one style of music or even one set of examples of a style (see discussion of

[Spector and Alpern, 1994]'s system above). A whole new knowledge base will need to be incorporated to cover a different style of music. This is an issue regarding the current research since a supplementary aim was to investigate extending the approach to more than one style of music.

Summary. These difficulties suggest that a purely rule based critic would not have been suitable for achieving the aims of this research (see also section 4.6.2). In particular, the lack of a formal theory of drum patterns in popular music would make the generation of a concise rule base a difficult task, while an underspecified rule base would introduce problems of subjectivity, lack of variation in the output and failure to capture the relevant features of rhythmic patterns. Finally, rule-based critics do not provide a parsimonious means of incorporating more than one style of music into an evolutionary system.

Neural Network evaluation

An ANN critic was used in the research described here to overcome these problems with rule based critics. This section, therefore, gives a more thorough treatment of this method of implementing the fitness function.

Advantages. Neural networks are good at complex pattern recognition and feature detection in multi-dimensional space, they can learn non-linear transformations and they are robust to noise and inconsistency in the training data. These features potentially make them useful as critics in a GA where fitness assignment is heuristic or uncertain and there are many degrees of freedom in the information encoded in the chromosome. This would seem to be the case for many areas of music where, in the words of [Todd and Werner, 1999], "critics based on learning methods such as neural network models ... can generalise their judgments sufficiently to leave (artificial) composers some much-needed rule-breaking 'wiggle room'."

A second advantage of using a neural network as a critic is that the degree of human expertise hard coded into the system is reduced; the network learns by example those features (which may be subtle and context dependent and therefore hard to encapture explicitly in rules) which distinguish good and bad musical phrases. Extracting critical criteria directly from a representative set of examples of the domain using an ANN is, arguably, a more direct (and potentially more accurate) method than constructing sets of rules to describe

those examples.

Finally, a network can, theoretically, be trained to differentiate any number of classes of input. This means that to incorporate a new style of music into the critic, for example, the network can be retrained on a set of training data which includes that style of music. There is no need to incorporate an entire set of new rules to cope with the new style.

An early study, focusing on a simple musical domain, demonstrated that ANNs can be successfully used as an automated critic in musical GAs. An evolutionary system was used by [Gibson and Byrne, 1991] to evolve simple musical phrases using a three stage process in which a rhythm and melody were evolved separately using MLPs and finally combined with other phrases to create a harmony (using simple rules). The complexity of the problem was significantly reduced by considering only certain chords in C major and limiting the metric resolution to quaver notes. The authors concluded that, "genetic algorithms with cooperating neural networks have been shown to produce pseudo-musical composition with some success". However, due to the various musical constraints employed the resulting compositions were not very musically adventurous.

Disadvantages. A major difficulty with the application of ANNs to musical tasks is the generation of an appropriate training set consisting of both positive examples (that is "good" musical sequences) and negative examples ("bad" musical phrases). While it is not too hard to come across examples of aesthetically pleasing music, examples of "bad" musical phrases are not so readily available.[8] [Gibson and Byrne, 1991] tackled this problem by constructing the training set from examples taken from an interactive run of the GA: chromosomes rated as fit become the positive examples while those given low fitness are used as the negative data.

A similar regime was used by [Biles et al., 1996] to train a neural network critic in an attempt to escape the fitness bottleneck associated with the interactive critic originally used in GenJam ([Biles, 1994]). They trained a MLP with a single output node on examples of high and low fitness chromosomes from an interactive run of the GA. It was found, however, after attempts with a number of different input representations that the network failed, in every case, to differentiate the training data without overfitting. Biles and his colleagues note that the human critics, who selected the training data, had extensive experience of

[8] although the author's parents would probably argue that numerous examples of the latter can be found in his record collection.

melody in many contexts, evaluated the melodies in a harmonic context and were sensitive to such factors as variety and novelty. The representation used to train the network lacked such sensitivity to harmonic and structural context and had no access to features such as novelty. Furthermore, there would have been a certain amount of noise in the training data due to subjective issues related to the interactive paradigm. The authors suggest these were the reasons for the inability of the network to learn the human preference data.

Other work has avoided these problems by taking the negative training data from more consistent sources. [Spector and Alpern, 1995] describe extensions to their GP system described above ([Spector and Alpern, 1994]). Disappointed with the results using critical evaluation criteria taken from the jazz literature, they investigated the performance of a trained MLP critic which they expected would extract a deeper structural representation of the desirable features of jazz melodies than was captured by their rules. A training set was constructed from four categories of input: two measures of Charlie Parker melodies (the positive data); one measure of Charlie Parker followed by one measure of silence; a single measure of Charlie Parker followed by a randomly generated melody; and finally a single measure of Charlie Parker followed by a measure of Charlie Parker reversed and randomly manipulated. A three layer MLP was successfully trained on a total of 100 patterns.

The GP system evolved programs which generated a new solo from an existing Charlie Parker melody taken from the case base. Both melodies were passed to the MLP which returned a fitness for each program. During testing of the system, a program in the second generation was assigned maximal fitness. However, the pattern produced by the program was "quite unsatisfactory." The authors suggest that the training set had covered too few types of negative example - "perhaps it would have been a better critic if it had also been trained to reject melodies that are reasonable except for bizarre rhythmic groupings, etc." In addition they note that, "the network had far too small a training set to learn about many of these kinds of errors." As a solution, the output of the network was combined with a subset of the rules employed in their previous experiments ([Spector and Alpern, 1994]). Although this improved matters, the results were still not entirely satisfactory.

[Biles et al., 1996] noted that on many occasions almost identical chromosomes had been assigned maximally opposite fitness values by the human critics. The heuristic bias of the MLP is a smooth interpolation between data points and when this is not the case the network can make ill-advised generalisations between one point in the space and the next.

The central issue would seem to be careful selection of the examples (and in particular the negative instances) making up the training set; in the words of [Todd and Werner, 1999]:

> "We need a system that can be soft when this is useful, but that can still make hard decisions when they are called for. Neural networks can behave this way if they are trained properly, using both positive and negative examples."

This "softness" of a neural network can be a particular problem when it is used as a fitness function in an evolutionary system. [Spector and Alpern, 1995] note that "if there is a simple way to exploit a weakness of a critic then it is likely that GP will find it." Although this problem was solved using a hybrid rule-based/ANN critic, they suggest the use of a larger training set, improved training regime and more sophisticated network architecture. Another way around this problem has been proposed by [Burton and Vladimirova, 1997a] who used an unsupervised ART network to develop clusters corresponding to drum patterns from different styles of music (rock, funk, disco, latin and fusion) from a set of training examples. However, the ART network critic seemed to produce a certain homogeneity in the generated patterns ([Burton, 1998]) making it inappropriate for achieving the aims of the current research.

Summary. The potential advantages of using an ANN critic were attractive given the aims of the present work. The critical criteria are extracted directly from the training data - example drum patterns in the chosen style - thereby avoiding the difficulties and subjective bias involved in designing a rule-base. The generalisation ability of the MLP should allow the generation of diversity and novelty in the evolved drum patterns. Finally, there existed the possibility of classifying a number of different styles of music using the same network.

However, it has been seen that the choice of positive and negative examples and the amount of training data used can have a significant impact on the training of the network and its performance as an evolutionary critic. These issues have been addressed in the research described here (see sections 4.3 and 4.4). Finally, the exploitation of both the loose generalisation abilities of an ANN and the strict classification of rules has also been investigated in the described system (see section 4.6.5).

Evolving the Critic

A final approach to developing critics for musical applications has been to evolve an appropriate set of rules. [Horowitz, 1994], for example, discusses the possibility of evolving populations of parameters (meta-individuals) according to the users preferences – "the user evaluates the families of rhythms evolved by each of the meta-individuals." In a similar manner, [Jacob, 1995] evolved the composer modules and the critics (ear modules) separately through interaction with the user. Since the critics in these studies are evolved according to the preferences of the user, they suffer from the problems associated with the IGA (discussed above).

[Werner and Todd, 1997] (see also [Todd and Werner, 1999]) have developed a rather more complex scheme in which populations of critics and composers undergo a process of co-evolution by sexual selection. Although [Todd and Werner, 1999] admit that the resulting songs are not aesthetically pleasing to the human ear, they cite several advantages of their co-evolutionary scheme over the human, rule-based and ANN critics discussed above. First, it can help reduce the problem, noted by [Spector and Alpern, 1995], of chromosomes finding shortcuts to gaining high fitness; and second, diversity can be maintained through subspeciation of the original population. A problem with this approach is that the evolution of the critic is not bounded by any musical criteria. [Todd and Werner, 1999] suggest that incorporating basic musical criteria into the critic's preferences will keep the songs generated within reasonable musical bounds and that critics who could learn their musical preferences and composers who could learn their songs within a generation would increase the speed of adaptation.

The major problem with such a hybrid co-evolutionary system is one of complexity. Time has not permitted an exploration of this avenue in the current research.

3.3.5 Evaluation of Artificial Composers

The evaluation of artworks is an area of little agreement often coming down to individual subjective opinion and this, as noted by [Spector and Alpern, 1994], "presents a problem for AI scientists wishing to produce computational artists". The appropriate evaluation methods will depend on the aims and techniques used. In the case of an IGA, subjective evaluation by the user may be the only way. [Ralley, 1995], for example, states explicitly that "there is no way to really measure the success or failure of such a system if the solution

the user desires is not predetermined and the acceptability of the final melodic material is entirely up to the user. One means of collecting a more objective aesthetic evaluation has been the organisation of a concert and collecting the aesthetic judgements of each member of the audience ([Biles, 1999] describes audience evaluation of GenJam during concerts).

The use of rule based critics opens up the possibility of a more objective, scientific evaluation of machine generated music. Although most studies have resorted to subjective evaluation by the creator of the system, [Phon-Amnuaisuk et al., 1999] gave the four part harmonies generated by the system to a senior music lecturer at the University of Edinburgh to evaluate according to the criteria he uses for first year undergraduate students' harmony. This type of evaluation is particularly appropriate to those studies working in domains governed by critical criteria derived from music theory. It is less clear how to objectively evaluate system generated music in less formal domains (such as drumming in modern popular styles of music).

This problem concerning the evaluation of computer generated works of art has been discussed in some detail by [Spector and Alpern, 1994] who attempt "to separate those components of an AI system to which aesthetic judgements should apply from those to which scientific judgement should apply." They reject organising a public show and collecting many subjective aesthetic reviews on the grounds of the time involved. Furthermore, they find working in a genre with formalised valuation criteria unsatisfactory on the basis of three problems: the existing formalisations are often "dead" forms; adherence to rules may not be a good indicator of aesthetic value; and finally work such a genre may not generalise well to other areas where criteria for aesthetic judgement are not so uniformly accepted.

Instead, they advocate factoring aesthetic judgement out of the equation by developing systems which take critical criteria and a "cultural context" as parameters and which will work over a wide range of variation of these parameters. In this case, the success of the system can be assessed across cultures and critical criteria. As an example of this approach, critical values (taken from the literature on jazz improvisation technique) and a cultural context (a set of Charlie Parker melodies) were supplied as parameters to their GP system. It was argued that the variations of Charlie Parker melodies generated by their programs could be judged scientifically by within these parameters (that is, by the critic) without raising the issue of aesthetic value.

In spite of this, it is interesting to note that [Spector and Alpern, 1995] found the music

generated by the system was not pleasing to their own ears - thereby resorting to the subjective evaluation typically used - and went on to attempt to extract the critical criteria directly from the case base using an ANN. This approach is favoured since only the case base of examples from the musical domain need be supplied to the system which automatically extracts a critic from those examples.

In the case of the present research, where the goal is to generate musical patterns representative of a specified domain using a set of examples supplied as parameters to the system, an approach analogous to a Turing test for musical phrases is proposed. Aesthetic judgement is once again factored out of the equation and the patterns generated by the system are evaluated on the basis of the degree to which subjects, familiar with the musical domain, can distinguish system generated and human generated rhythmic phrases. This means of evaluation is particularly suited to a system using an trained ANN critic which has extracted its (sub-symbolic) evaluation criteria directly from a set of examples of musical phrases. The subjects can be asked to distinguish the generated patterns from the training data. This method of evaluating machine generated musical phrases has been investigated in the present research (see section 5.2).

3.4 Algorithmic Composition with Neural Networks

3.4.1 Overview

There has been extensive interest in recent years in the application of neural networks to musical tasks ([Todd and Loy, 1991], [Griffith and Todd, 1999]). These can generally by divided into those tasks which attempt to generate music and those which attempt to analyse music. Research has tended to focus on the former, finding it to be a much easier problem to solve. Furthermore, studies of music analysis have tended to focus on relatively low level features such as rhythm perception ([Desain and Honing, 1991]) or the recognition of instruments ([Dolson, 1991]) rather than the higher level problem of classifying musical phrases more relevant to the design of networks for use as critics in evolutionary systems. The following is a discussion of the application of ANNs to musical composition, focusing on the issue of representation and concluding with a discussion of the combination of ANNs with other methods.

3.4.2 Representation Issues

In the field of music generation [Todd, 1989] discusses two possible representations of time in neural networks (see also section 4.2.2). First, the spatial representation of time whereby the input to a network is an entire musical phrase and the target output the next musical phrase. In this scheme, time is represented as the spatial position of a note in a sequence of notes. If each phrase represents a bar length, then the number of fields in the network input represents the granularity of temporal quantisation. This type of representation, although simple, has two major disadvantages: first, the length of the pattern and the granularity of the timesteps are limited by the size and complexity of the resulting network; and second, it might be argued that it provides an unnatural representation of time which is (by definition) a sequential and not a spatial concept (although a spatial representation of time is standard in physics).

In most attempts to use ANNs for the generation of music a very different representation has been used: the sequential representation. In this scheme, the network is fed a sequence of past timesteps one by one and is trained to generate the music to be played at the next timestep. A *recurrent* network[9] is usually required to memorise the timesteps previously presented. In order to generate music, the trained network's output is fed back into the input units and a measure of music presented to start the feedback loop. This type of network has generally been used because of its more natural representation of time and since it allows the input of sequences of arbitrary length and granularity to the network.

It is notable, however, that all of the studies using ANN critics in genetic systems have employed spatial network representations. This is probably due to the natural application of the (generally) spatial chromosome representation to the network inputs and the fact that this type of representation lends itself particularly well to classification problems (where the output of the network represents a classification, not music). Nonetheless, research using sequential representations has important implications for temporal representation in general.

In the scheme used by [Todd, 1989] the duration of a note is proportional to the number of successive timesteps in which that note is presented. The drawback of this type of implicit representation of duration is that, as for spatial representations, time must be divided into very small intervals to allow a complete range of temporal intervals and consequently

[9] The recurrent architecture provides a set of context units which record the state of the network at previous timesteps (with exponential decay).

the the number of timesteps in a piece of music becomes very large. Not only does this place a computational limit on the granularity that can be represented ([Urwin, 1997]) but also the learning of contingencies between notes becomes a difficult task. A final problem is that the generation of notes in one timestep is necessarily limited to a local context of previous notes. In accordance with this analysis, Peter Todd has said of the melodies generated that "the problems ... of lack of higher level structure emerge, and these compositions tend to wander, having no clear direction, and seldom ending up anywhere in particular."

[Mozer, 1994] attempted to address this problem of incorporating higher level organisation by using a more sophisticated representation scheme, motivated by psychological and psychoacoustic considerations. In particular, the duration of each note was explicitly represented in terms of units corresponding to a twelfth of a crotchet length. This allowed the representation of durations as fine as demi-semiquaver triplets and made the representation more compact. In spite of this, Mozer remarked of the generated music that:

> "While the local contours made sense, the pieces were not musically coherent, lacking thematic structure and having minimal phrase structure and rhythmic organisation."

A number of suggestions were made to aid the learning of global musical structure by neural networks (including more sophisticated architectures, representation and training regimes - see [Mozer, 1994] for details).

Given these failures in the melodic and harmonic domain, it is interesting to note that [Urwin, 1997] successfully trained an ANN to generate drum patterns, once again using the sequential paradigm. The representation consisted of velocity values (0 representing the absence of a note) for four instruments (bass drum, snare drum, closed hi-hats and open hi-hats) and the bar was divided into 64 timesteps. The network was trained to predict the velocities of each instrument at the next timestep given a window of velocities at previous timesteps. Given the failures described above, it is interesting that in this purely rhythmic domain the network succeeded in generating patterns that were misclassified as human transcribed MIDI drum patterns in 85% of cases. It was concluded, however, that the use of a small and relatively uniform training set lead to generation of patterns from a limited area of the whole search space.

Inspired by this success, the system designed in the present research used a similar representation scheme to that used by [Urwin, 1997] although translated into a spatial frame-

work (see section 4.2).

3.4.3 Hybrid Approaches

An inability to extract higher level features of music seems to be a problem that has dogged most attempts to compose with recurrent neural networks. This is partly a result of the fact that these studies limit the context (on which the generated sequence of music is to be based) to a small window of previous notes. [Papadopoulos and Wiggins, 1999] suggest that ANNs are less efficient and practical than other techniques "at least as a stand-alone approach." They go on to suggest that the integration of different methods will allow artificial composers to "take advantage of the strengths of each one." ANNs, as proven techniques in machine learning, will almost certainly have a role to play in such integrated systems.

HARMONET ([Hild et al., 1992]) is an example of such a system. The aim of this study was to approximate the function mapping chorale melodies onto their harmonisation using a training set of 400 four-part chorales composed by Bach. They approached the problem by decomposing it into sub-tasks: generating a skeleton structure of the harmony based on local context; generating a chord structure consistent with the harmonic skeleton; and finally adding ornamental quavers to the chord skeleton. Neural networks were used for the first and third tasks and a symbolic constraint satisfaction approach was applied to the second sub-task. The resulting harmonisations were judged by an audience of professional musicians to be on the level of an improvising organist. The authors conclude that:

> "By using a hybrid approach we allow the networks to concentrate on musical essentials instead of on structural constraints which may be hard if learned by a network but easy if expressed symbolically."

The hybrid approach adopted in the research described here (see chapter 4) was inspired by systems such as HARMONET (as well as those described in section 3.3.4) in which neural networks (and other techniques) are applied to those compositional subtasks to which they are best suited.

3.5 Summary of Chapter 3

This chapter has reviewed approaches to musical composition using evolutionary and neural techniques and discussed the implications for the present research.

Chapter 4

System Design and Development

4.1 Overview

The general concept involved in this system is the use of a GA to guide search through the space of drum patterns to find solutions representing drum patterns in a specified style. Candidate patterns generated during the process of evolution were evaluated by an ANN trained (using example patterns from within that style and patterns not within that style) to classify patterns according to the degree to which they represented acceptable patterns within the specified style of music.

For the purposes of this dissertation it will be useful to divide the development of the system into five phases:

1. The representation scheme was designed (section 4.2).
2. The data for training the neural network were collected (section 4.3).
3. The collected data were preprocessed into a suitable format for training the network (section 4.4).
4. The ANN was designed and trained on the collected data(section 4.5).
5. The GA was developed and run using the neural network as its critic (section 4.6).

This chapter presents each of these phases in turn. An overview of the goals and design issues of each stage is presented. The design decisions made with respect to each of these issues is then considered: choices made are justified and steps taken to solve encountered problems are described. Finally, section 4.7 describes an example run of the GA.

4.2 The Representation Scheme

4.2.1 Overview

In order to facilitate interaction between the GA and the trained ANN, when evaluating the fitness of chromosomes in the evolutionary process, the chromosome representation scheme was exactly the same as that used for training the ANN. The chosen scheme therefore had to satisfy the criteria of both roles. It was clearly desirable that the representation of a drum pattern should accurately and completely describe all the relevant information in the data. However, in the case of a GA it was also important to build musical knowledge into the chromosome representation in order to cut down the search space (see section 3.3.2); while in the case of an ANN the use of fewer fields and possible values for those fields can significantly reduce the complexity of the function that must be learned by a neural network.

Noting these points and bearing in mind the failure of previous attempts to train an ANN ([Biles et al., 1996], [Spector and Alpern, 1995]) for use as the critic in evolutionary systems, the representation chosen constrained the problem in the following ways:

1. Only one bar was represented. A bar length was chosen on the basis that this is the smallest musical section that would be useful in composition. Furthermore, the stated aim was to produce a system which generates short rhythmic sequences.

2. Only the bass drum, snare drum, closed hi-hats and open hi-hats were represented. These comprise the central instruments in the modern drum set and are the basic essentials required to play drum patterns from most styles of popular music.

3. Only patterns in the time signature of 4/4 were represented.

4. The metric granularity was limited to demi-semiquaver subdivisions of the bar.

5. The note velocities were quantised to discrete values.

6. The duration of a note was not represented.

7. Tempo was not represented.

The reasons for imposing these constraints are explained in the following discussion of several issues considered during the development of the representation scheme.

4.2.2 Spatial vs. Sequential Representation

Most of the studies of music generation using ANNs, discussed in section 3.4, have employed a sequential representation using a recurrent network architecture (see figure 4.1). While this type of scheme provides a natural means of generating notes based on a previous sequence of notes, it is not necessarily suited to the problem of classifying input sequences. Since this was to be the role of the ANN in the present system a spatial representation, shown in figure 4.2, appeared to be a more natural approach. Furthermore, the concept of sequential representation does not make much sense in the context of the typical GA representation scheme. In a spatial representation, the position and duration of a note and its relation to other notes in the pattern are defined by its spatial position in the input vector (see sections 3.4.2 and 3.3.2).

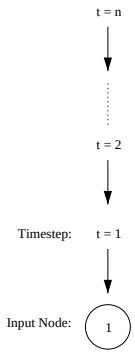


Figure 4.1: Sequential Network Representation

There are, however, several problems with this approach, resulting from limiting the length of a pattern to the number of input units to the network (assuming the pattern is to be a fixed musical section; one bar for example). First, the time signature of the patterns that can be represented is limited. Second, the subdivision of the bar is dependent of the number of input nodes in the network. In order to appreciate these problems, consider the following examples where one bar of a single instrument is to be represented. If there are four input nodes then we can represent notes occurring on crotchet subdivisions in 4/4 (or quavers in 2/4). Time signatures such as 3/4, 7/8, 5/4 etc. cannot be represented.

Furthermore, notes cannot be placed on metric subdivisions finer than crotchet lengths. We might try to solve this second problem by increasing the number of network inputs to 16, for example, so that subdivisions as fine as semiquavers are represented. However, it is still not possible to represent notes occurring on triplet subdivisions of the bar. In fact,

the granularity of the spatial representation would need to be incredibly fine in order to accommodate both triplet and standard subdivisions, resulting in an enormous number of time steps to be represented (and a correspondingly large number of input nodes to the network). This is potentially a very serious problem for the spatial representation as a general scheme for the representation of music, since the triplet subdivision is very important in several styles of music (jazz for example).

In spite of these problems, a spatial representation scheme provides a simple way of representing a musical pattern as the sequence of characters that typically make up the chromosome in GAs. As described in section 3.3.2, this is a common means of representing time in musical applications of GAs and GP. Furthermore, a spatial representation has typically been used in training the ANN in those studies investigating the use of ANN critics (reviewed in section 3.3). This type of representation makes the evaluation of chromosomes simple; the chromosome is presented to the input nodes of a network and the degree to which that chromosome is classified as a member of the positive training set is output. Since the styles of music that were used in this system (d&b and rap - see section 4.3) are generally in the time signature of 4/4 and triplet subdivisions are typically not used (except in rare cases for flourishes) a spatial representation scheme was employed in this system.

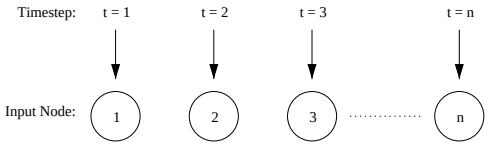


Figure 4.2: Spatial Network Representation

A final question, then, given that a spatial representation was to be used, concerned the granularity of the timesteps to be represented; that is to say the length of the chromosome and consequently the number of fields in the network training data (and the number of network input nodes). The smallest timestep required in order to capture the main features of the training data was a semiquaver subdivision of the bar. In order to capture subtleties such as snare drum rolls, demi-semiquavers were represented (corresponding to a division of the bar into 32 timesteps).

4.2.3 Representation of Velocity

While some studies have used binary representations simply indicating the presence or absence of a note at each timestep represented (e.g., [Burton, 1998]) a better scheme would be to represent the velocity (amplitude) of each note. It was decided to represent velocity on a scale of single decimal place quantisations between nought and one. Therefore, each timestep in the spatial representation could contain a number from this range indicating either that no note is played on that timestep (nought) or a note of a specified velocity is played (0.1 to 1.0). This range of quantised values was chosen to cover a suitable range of velocities and yet also reduce the range of values that each field in the network training data could take, thereby making the network's task somewhat easier.

4.2.4 Representation of Note Durations

A number of possible methods of representing note position and duration in the spatial representation of a musical sequence were considered. These included the following

- Represent explicitly the note on and note off positions for each note in the pattern. This introduces the problem of what to do when one note ends at the same point as another note begins. One solution to this problem would be to create a representation for that type of event, although this was inadvisable due to the increase in complexity involved.

- Represent note on positions and allow any note to be truncated by the presence of a further note-on event. However, the problem of how to cut notes short before the next note-on event remains.

- Represent notes by two fields, one indicating the presence of an event, the second indicating the velocity of that event. In this case, the presence of an event with velocity 0 truncates the previous note without the presence of an audible event. This is the representation scheme used by [Urwin, 1997].

- Represent note-on position and note-duration explicitly as vector pairs (as used, for example, by [Thywissen, 1996]).

Since the instrument events represented in the training data were drum hits of short duration, the representation of note duration was not such an important issue in this system as in

others working in the melodic or harmonic domain. Therefore, each note was fixed at a duration of a semiquaver which was time enough for the note to decay fully, even at the high tempos common in d&b. The only situation in which this representation of duration might prove too restrictive would be if one wanted to cut drum notes or cymbals short after a very short period for special effects. In actual fact, this could be simulated by the positioning of a quiet note on the demi-semiquaver timestep after the note to be cut short.

4.2.5 Representation of Different Instruments

Once again, there were a number of possible means of representing the instruments including:

- Sequentially in the representation (as used by [Burton, 1998]).
- Interleaved by timestep in the overall representation.
- [Urwin, 1997] suggests representing instruments as a parameter included in each note onset representation (in order to make the representation of temporal relationships more concise and hence easier to learn).

The first of these schemes was chosen since it required the least transformation of the events played on each instrument from the MIDI file representation (each instrument was on a separate MIDI track in the training data - see section 4.3).

4.2.6 Representation of Tempo

It is notable that most of the studies reviewed in section 3.3.2 did not explicitly represent tempo in the chromosome. Likewise, tempo has not been represented in the system being described here. Instead the tempo of the MIDI files output by the GA can be set by the user, who may be looking for drum patterns for a particular composition and, therefore, will want to have control over the tempo of the patterns generated.

4.2.7 Summary of the Representation Scheme

One bar patterns of four instruments were represented as a string of 128 numbers between one and nought, quantised to steps of 0.1. The first 32 of these numbers represented the

presence (0.1 to 1.0) or the absence (0.0) of a bass drum on that demi-semiquaver subdivision of the bar, numbers 33 to 64 the presence or absence a snare drum, numbers 65 to 96 the presence or absence of a closed hi-hat and numbers 97 to 128 the presence or absence of an open hi-hat. Tempo was to be set by the user. An example drum pattern from the network training data, shown both in musical notation and in the representation scheme described above, can be found in Appendix B.

4.3 Data Collection

4.3.1 Overview

There were a number of issues to consider when choosing a format for the musical examples used to train the network . These included the following:

- The format should contain all the information that will be required in the network and chromosome representations including information concerning note positions, durations and instruments as well as information concerning tempo and time signature.
- Since ANNs require a large amount of training data a large number of data in this format must be available
- It must be relatively straightforward to convert the format into the representation chosen for training the ANN

Furthermore, both positive and negative examples of the target classifications had to be collected. The two obvious choices of data format were the standard MIDI file format or raw audio format (e.g., .wav files).

4.3.2 Format

The data collected for training the ANN were in the format of MIDI files for the following reasons (see [Rothstein, 1992] for an introduction to MIDI):

- MIDI files contain all the basic information about a rhythmic pattern that will need to be represented.

- MIDI files containing just drum parts are available for sale or via the Internet. It was therefore relatively easy to collect a a large number of them.
- a wide variety of software for the processing and playing of MIDI files is available.

4.3.3 The Training Data

Since the aim of this project was to generate drum patterns within a specified style, the training examples for the neural network were chosen as recognised examples from that style. It would not, for example, have been sensible for the author to sequence the training data himself due to the subjective bias involved. This was considered to be a an important contributory factor in the failure of [Biles et al., 1996] to train an ANN to classify good and bad jazz improvisations. Therefore, the positive examples in the training set were taken from Standard MIDI files containing drum patterns in the style of "Drum & Bass" (henceforth d&b) on disks sold by Keyfax software for use by musicians (see Appendix D for details). These disks contained MIDI files of drum parts which were up to 100 bars in length. Using a MIDI sequencer (*Logic Audio Gold*) the bass, snare and hi-hat tracks were extracted from these MIDI files, quantised to the level of demi-semiquaver subdivisions and finally, each bar saved as a MIDI file. These were taken to be reliable and consistent examples of d&b drum patterns. Patterns in the musical style of "rap" were also obtained from the same source and processed in the same manner to be used as negative data.

The negative set of examples was constructed from four classes of pattern:

1. Randomly generated patterns containing the same rhythmic pattern on each instrument
2. Different randomly generated patterns on each instrument
3. Semi-random patterns generated by randomly combining random tracks and tracks taken from patterns in the musical style rap
4. Rap patterns

This scheme was motivated by the desire to model in the network training data the types of pattern that would be generated by the GA throughout evolution. Initially, since the population is randomly initialised, the majority of the chromosomes generated would correspond to random drum patterns. This motivated the inclusion of classes 1 and 2 in the negative

examples. Since there was a (small) possibility that some of the randomly generated patterns would be reasonable drum patterns, class 1 was included in the training set to ensure that some patterns would be definitively “bad” examples of d&b patterns.

As evolution progresses some structure should begin to emerge in the chromosomes generated and, therefore, a range of semi-random patterns (class 3) was included in the negative examples to ensure that these patterns were not counted as having high fitness.

Finally, evolution might be expected to evolve chromosomes that correspond to well formed drum patterns but lack the features that classify them as d&b patterns. Therefore, a number of well formed patterns from a different style of music (the rap patterns in class 4 above) were incorporated into the set of negative examples. Rap patterns were chosen as being significantly different from d&b patterns within the constraints of the chromosome representation.

A final question concerned the amount of data required to train the network. This was an important issue since [Spector and Alpern, 1995] concluded that the main reason for the failure of their attempts to use a trained ANN as the critic in their GP system was the small number of training examples used (see section 3.3.4 above). A heuristic rule of thumb is to use ten times the number of training instances as there are input nodes to the network ([Sarle, 1997]). Since the representation consisted of 128 input fields (see section 4.2) this gave an estimate of 1280 for the required number training patterns. In order to reach this number a total of 970 one bar MIDI files in the style of d&b were collected and a total of 760 in the style of rap. All three classes of random pattern were generated by an automated procedure (see the following section).

4.4 Data Pre-processing

4.4.1 Overview

This phase involved the reading of a file in the chosen format, extraction of relevant information and conversion of this information into the representational format described in section 4.2.

4.4.2 Conversion Procedures

This was achieved using slightly modified versions of two programs, written by Richard Urwin, called *convert* and *parallelise* (see [Urwin, 1997] for details of these programs). The *convert* program takes a MIDI file as input and generates text files containing numbers representing the velocities of notes on each subdivision of the bar (time and velocity were quantised as described in section 4.2). The *parallelise* program then takes these files (four of them in this case corresponding to the four instruments in the MIDI data) and concatenates them in such a way that the 32 time steps representing bass drum hits are first, followed by the snare drum, closed hi-hats and open hi-hats. In this way, a string of 128 numbers is obtained which represents the note velocities on each demi-semiquaver timestep for each of the four instruments as described above. The positive data (d&b) and the fourth class of negative data (rap patterns - see section 4.3) were generated in this manner. All these procedures were automated using UNIX shell scripts.

A slightly different technique was needed for the generation of the random and semi-random negative examples. The program Sinuso (see Appendix D) was used to generate format 0 MIDI files (which contain only one track) with randomly generated note lengths. These were then converted to text files using the *convert* program described above. A small program was written in *Scheme* which made use of the *parallelise* program (described above) to generate data representations that were of the following three types:

- The same randomly generated pattern on each instrument
- Different randomly generated patterns on each instrument
- Each instrument randomly given a randomly generated pattern or a pattern on the same instrument from a rap MIDI file

In this manner, the first three classes of negative data described in the previous section were automatically generated.

Table 4.1 shows a breakdown of the total amounts of data in each class finally obtained for use during training:

Generally, the proportions of positive and negative data for an ANN should match those found under “natural conditions”. In the case of a GA, the initial populations typically contain randomly generated chromosomes with low fitness, while in the later generations,

Class of Data	Number of Examples
d&b (positive)	970
Random (tracks same)	105
Random (tracks different)	105
Semirandom	380
Rsap	380

Table 4.1: Collected Data

nearing convergence, the entire population will be relatively fit. Therefore, equal numbers of positive and negative instances were included in the data set.

4.5 Training the ANN

4.5.1 Goals and Issues

The ANN was designed and trained in such a way that it correctly classified the training data and showed good generalisation to unseen patterns. This phase of development involved completing the following tasks:

- Dividing the data randomly into a *training set*, a *validation set* and a *testing set*. The training set is the body of examples used to train the ANN. To guard against overfitting, the performance of the network during training is measured by its error on the validation set. This ensures that the network continues to generalise well to unseen examples. The validation set is also used to decide between different network architectures and parameters. Finally, the performance of the trained network is tested on the test set of examples.

- Choosing a means of implementing the ANN.

- Choosing a network type, architecture and parameters.

- Choosing an appropriate learning algorithm.

- Training the network.

- Testing the performance of the network on the test set of examples.

4.5.2 Creating the Training, Validation and Testing sets

The data described above were divided at random into three sets, each containing the same proportions of the five different classes of training pattern, as shown in table 4.2.

Set	Proportion of data	Number of patterns
Training	70%	1358
Validation	20%	388
Test	10%	194

Table 4.2: The Training, Validation and Test Sets

This division of the data was chosen since it gave a number of training patterns just over the 1280 required and furthermore, an argument presented in [Haykin, 1999] suggests that 20% of the data should be assigned to the validation set, which left 10% for the test set.

4.5.3 Implementation

There were three implementation alternatives available: using a neural network simulator; coding the neural network myself; and finally, using a neural network construction package. The first option would not have allowed the use of the trained ANN as part of a system, while the second would have taken too much time and these were, therefore, rejected in favour of the third method. The ANN was created and trained using the NeuralWorks^{TM} software package. It was then automatically converted to a C function which could be accessed by means of an input and output array, the sizes of which were defined by the respective sizes of the input and output layers of the network.

4.5.4 Architecture

A MLP was chosen above other types of ANN for the following general reasons (see section 4.6.2 for a justification of the use of an MLP, more specifically, *as the critic* in this evolutionary system):

- It is a general purpose non-linear regression technique which attempts to minimise global error
- Any multi-dimensional function can in theory be synthesised
- It can provide very compact distributed representations of complex data sets

- It is efficient and robust in the presence of errors or inconsistencies in the training data
- It has been shown to provide good results on a number of prediction and classification problems

However, set against these advantages, two serious problems with the MLP are the following:

- Learning is slow, often requiring hundreds of thousands of training epochs.
- Choosing the appropriate network architecture and learning parameters is difficult.

In order to alleviate these problems the network was trained using the *cascade correlation* paradigm and the *Extended-Delta-Bar-Delta* (EDBD) training algorithm.

Cascade Correlation. A problem with the design of an MLP neural network is the selection of an appropriate architecture. This is important since, in general, the more hidden units used the better the performance on the training set but the worse the resulting performance on unseen data (overfitting). There exists a tradeoff between performance and generalisation. While the input and output layers are generally determined by the data representations chosen, the optimal number of hidden layers and the size of these layers are often chosen by trial and error. A number of network architectures are generated and trained and the architecture with best performance on the validation set is chosen. However, given the slow nature of training MLPs this can be a time consuming process.

The cascade correlation paradigm ([Fahlman and Lebiere, 1990]) is designed to solve this problem. This is achieved by training a network with no hidden units and then adding hidden units one at a time. The task of each of these is to predict the current remaining output error in the network. New hidden units receive input from all previous hidden layers and from all input units. This process is repeated until the performance of the network on the validation set no longer shows any sign of improvement. Algorithm 3 shows the cascade correlation procedure.

In this manner the optimal number of network hidden layers and units is determined automatically. This paradigm was used in the training of the MLP.

Algorithm 3 The Cascade Correlation Training Algorithm

1.	Train the direct connections from the input layer and bias to the output layer for a fixed number of iterations or until RMS error stabilises
2. Iterate:	1. Train a new hidden unit so as to maximise a measure of the correlation between its output and the residual error at the output for each training instance until training stabilises when the inputs to that unit are fixed ("tenured")
	2. Connect the newly tenured unit to all output units and randomly initialise these connection weights
	3. Train all connection weights to output layer (from (input units and all tenured hidden units) until RMS error stabilises
3. Until:	error no longer decreases

The EDBD Training Algorithm. A second problem with using a MLP with the back-propagation algorithm (described in section 2.3) is that the learning rate and momentum parameters must be chosen. Once again, the selection of optimal values often involves a process of trial and error and the use of inappropriate values can significantly slow down learning. The EDBD algorithm ([Minai and Williams, 1990]) is an attempt to address this issue of the speed of convergence of MLPs using the following heuristics:

1. Each adjustable network weight should have its own learning rate and momentum parameters since a fixed value may not suit all areas of the error surface.

2. Each learning rate and momentum parameters should be allowed to vary from one epoch to the next since a single region of the error surface may behave differently throughout time.

3. When the current position in the weight space lies on a relatively flat region of the error surface along a particular weight dimension, the learning rate parameter for that particular weight should be increased to speed up convergence.

4. When the error surface in the current region along a particular weight dimension is convoluted the learning rate parameter should be decreased to ensure the avoidance of local minima.

Using these heuristics the EDBD learning algorithm automatically adjusts the learning rate and momentum parameters during training. It has been found to significantly increase the speed of convergence of MLPs to minima in the error surface and was used in the training of the MLP being discussed.

Other Features

The network was trained to minimise the Root Mean Squared (RMS) error on the validation set. The sigmoid function was used as the activation function with thresholds of 0.1 and 0.9 for classification. There was one output unit and positive training examples (the d&b patterns) were associated with a target output of one while the negative examples (all four classes) were associated with a target output of nought.

4.5.5 Results of Training

The cascade correlation training procedure eventually converged on an optimal network with one hidden unit which attained a minimum RMS error of 0.1472 on the validation set. The final RMS values and classification rates (for both the positive and negative training data and the mean) of the network on the training, validation and test sets is shown in table 4.3.

Set	RMS error	Positive	Negative	Mean
Training	0.1284	0.9469	0.9705	0.9587
Validation	0.1472	0.9375	0.9330	0.9352
Test	0.1476	0.9368	0.9278	0.9323

Table 4.3: Results of Network Training

The table shows that the network learned to correctly classify 95% of the training data and, more importantly, its classification performance generalised well to the test set of data which was not used at all during training. Figure 4.3 shows a graphical view of the performance of the MLP on the test set against the desired values (in bold).

4.5.6 Discussion

It is notable that there are a just few instances (especially in the positive data) which are wildly misclassified while most of the data is correctly classified within the limits. In the case of the positive examples, all the drum patterns in the style of d&b were taken

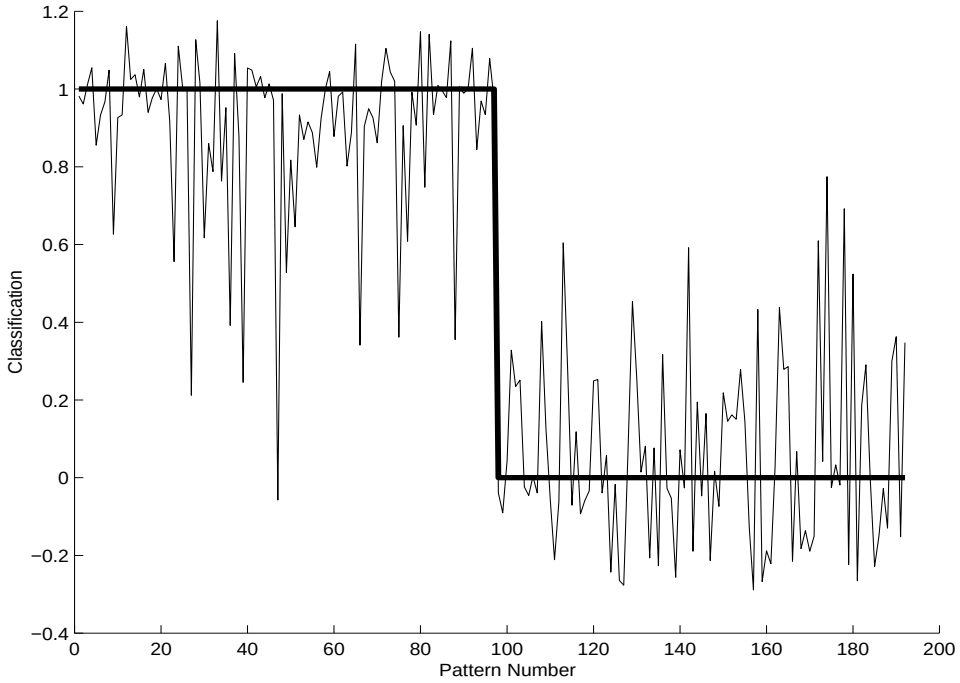


Figure 4.3: Network Performance on the Test Set

from the disks without any filtering by the author. It seemed likely that the misclassified examples would have been "fills", such as snare drum rolls. In the case of the negative examples, the most likely explanation was that some of the randomly generated examples were actually very similar to some of the d&b patterns (especially in the case of the semi-random patterns). An informal analysis of some of these misclassified training instances suggested that these hypotheses were correct.

Removing these outliers from the training data resulted in improved performance of the MLP on the test set (see Appendix E) although time restrictions did not allow the investigation of the performance of the GA using this network as a critic. Furthermore, the cascade correlation procedure found a network with 5 hidden units to produce optimal performance which suggests that a more complex internal representation of the data had been extracted than in the network described above (which had only one hidden unit).

4.6 Developing the Genetic Algorithm

4.6.1 Overview

A number of design decisions were made during the development of the GA:

- Why use a GA rather than some other method?
- Why use an ANN critic?
- How is the GA to be implemented?
- Designing the generic GA features: choosing appropriate techniques for selection, reproduction, crossover and mutation.
- Designing domain specific features of the GA: genetic operators, the initialisation of the population and the integration of the critic into the system.
- Choosing a system for obtaining multiple solutions from the GA.
- Choosing a playable output format for the system considering the issues of ease of conversion from the chromosome representation into the format and the use of a generally accepted and widely used format.

4.6.2 The Choice of Techniques

Why use a GA?

A GA was chosen to achieve the aims specified in section 1.2 of this dissertation. The use of a GA was motivated by several factors (see also section 3.3.1):

- GAs provide an efficient means of searching large, complex search spaces such as the present problem domain (the space of all possible drum patterns). In this case, the size of the search space is 11^{128} (128 genes each with 11 possible values), which is larger than that estimated for Chess (for which exhaustive search is generally considered beyond the capabilities of any computing device).
- The framework of a GA allows a balance of exploitation and exploration of the search space (through the use of varying amounts and forms of domain specific knowledge) and, thereby, of the amount of structure and novelty present in the generated patterns.
- An evolutionary approach is attractive since musical composition often involves the combination and permutation of themes (the recombination of rhythmic subpatterns in the case of drum patterns).

- Evolutionary techniques lend themselves to generating multiple solutions on any one run. A stated aim was to generate a number of patterns on any one run.

The number of studies of algorithmic composition using GAs far outweighs the number using GP. Furthermore, the use of GP has, in general, failed to produce the same degree of success that has been achieved using GAs. For these reasons, a genetic algorithm was chosen above the GP paradigm in the present research project.

Why use an ANN critic?

GAs have achieved some success in searching musical domains for aesthetic musical phrases in previous work (see section 3.3). There are, however, only two studies that have focussed exclusively on the evolution of drum patterns. [Horowitz, 1994] used an IGA (including user specified parameters to initially guide evolution) to evolve drum patterns and reported successful results. However, the IGA was considered a less than satisfactory model for achieving the current aims for the following reasons:

- The fitness bottleneck limits the number and size of generations that can be used
- Subjective bias during the evaluation of chromosomes
- A desire to automate the entire process of drum pattern generation using AI techniques

[Burton, 1998] extended this work by automating the critic, using an unsupervised ART network to evaluate candidate drum patterns bases on their propinquity to existing clusters, corresponding to styles extracted from the training set. The resulting patterns were reported to be similar to the training data (a measure of success) but rather homogeneous. The use of an MLP in this system was motivated by the following factors (see also section 3.3.4):

- The MLP provides a fully automated critic avoiding the problem of the fitness bottleneck associated with interactive human evaluation.
- An MLP can be trained on examples taken from the style of music to be modeled thereby avoiding both the subjective bias involved in human evaluation and the difficulty of formulating explicit evaluation criteria.

- The generalisation abilities of the MLP can be employed to alleviate the determinism associated with rule based critics, thereby allowing greater diversity in the generated drum patterns and also the homogeneity in the output of the GA using an unsupervised ANN

- An MLP can be trained on examples from several different styles of music, removing the need to design a entirely new knowledge base in the case of rule based critics.

4.6.3 Implementation

The GA was implemented in Java since its functional object-oriented paradigm is suited to the design of a GA and it is the language that the author is most familiar with. Java also has the attractive feature of cross-platform compatibility. Interaction with the neural network, a C function output by the NeuralWorks^{TM} program, was achieved by means of the Java Native Interface (JNI).

4.6.4 Design of Generic GA Features

Generational reproduction is generally preferred over steady state reproduction because the replacement of the parent generation with a whole new generation makes the former method less susceptible than the latter to sampling error due to noise in the fitness function. Furthermore, it has been demonstrated that the steady state GA exerts a particularly high selection pressure, mainly because the worst member of the population is always replaced ([Goldberg and Deb, 1991]). In this system a relatively low selection pressure was desired so that individuals would not be driven to find shortcuts to high fitness that the neural network (as a sloppy classifier) might have allowed (this problem was noted by [Spector and Alpern, 1995]).

Selection methods used in GAs vary with respect to two features: the selection pressure they exert on evolution and their sampling error. The purpose of selection is to balance the tradeoff between exploration and exploitation: it must ensure that individuals of higher fitness have a higher chance of reproducing, but also that individuals of lower fitness have some chance of reproduction since they may contain genetic fragments (*schemata*) of high fitness. The degree to which selection favours fit chromosomes is known as the *selection pressure* and an intermediate level is generally required: too much will lead to premature

convergence while too little will cause evolution to occur too slowly. Therefore, selection generally involves some method of stochastic sampling of schemata in the population according to the fitness of the chromosomes containing those schemata. This stochastic process is prone to *sampling error* - that is, the possibility that high fitness schemata are not chosen for reproduction.

A probabilistic form of binary tournament selection was chosen due to the low selection pressure it exerts (for the above reason). Also, however, [Goldberg and Deb, 1991] found, in a comparative analysis of selection schemes, that linear ranking and a probabilistic binary tournament give better expected performance (defined by growth of the fitter chromosomes and time to satisfactory convergence) than proportionate reproduction. They state a preference for binary tournament selection due to its better time complexity. It is also particularly easy to implement since it requires no global information about the relative fitness of a chromosome in the population. However, tournament selection is still prone to sampling error since each tournament is carried out individually ([Hancock, 1994]). Finally, in light of the low selection pressure, due to generational reproduction and a probabilistic binary tournament, a form of elitism was employed whereby the fittest member of the population is automatically copied into the next generation.

Standard single point crossover was used, following most of the studies reviewed in section 3.3. The mutation operator flipped genes (with a probability defined by the mutation rate) to one of the available velocity values between 0.0 and 1.0 with equal probability. Both crossover rate and mutation rate were to be set by the user via command line arguments to the program. The representation scheme was as described in section 4.2 and the initial population was initialised randomly: each gene was assigned a velocity value from the available possibilities between 0.0 and 1.0 with equal probability. The fitness of each chromosome was assessed by the trained ANN described in section 4.4 which took as input an array of 128 values between 0.0 and 1.0 and returned a fitness (between 0.0 and 1.0) based on its generalised classification of the pattern as a member (or otherwise) of the positive data set (d&b patterns).

The stopping criterion of the GA was designed to be set via two command line arguments to the program: the first is a fitness between 0.0 and 1.0; the second is a maximum number of generations. The former specifies that evolution should stop when there is one chromosome in the population of the specified fitness, while the latter specifies that evolu-

tion should stop, and the fittest individual in the population be returned, if the generation count reaches the given limit.

4.6.5 Music Specific features of the GA

The output from the generic GA as described above was unsatisfactory for a number of reasons. First, the distribution of notes across metric subdivisions did not match that in the training data. For example, the output patterns generally contained a number of notes on demi-semiquaver subdivisions of the bar (that is notes on timesteps resulting solely from the division of the bar into 32) while the training patterns contained very few notes on these timesteps. In fact, there were in general far too many notes in most of the patterns generated. This problem was addressed by a more sophisticated initialisation of the population .

A second problem was that the patterns generated tended to be fairly similar and this was solved by the use of domain specific mutation operators. Finally, there were some desirable features of the training data lacking in the patterns generated. Particularly noteworthy was the failure to evolve patterns that contained closed hi-hats on the quaver notes of the bar. Problems such as these were addressed by the introduction of rules into the critic.

Incorporation of domain specific knowledge into various areas of the GA improved both the quality of the output and convergence times. The problems and the steps taken to solve them are discussed in more detail below.

Initialisation

It was found that the drum patterns generated contained far too many notes and, in particular, too many on timesteps resulting from the division of the bar into 32 steps. This was partly because the chromosome was initialised with all possible velocities with equal probability resulting in a population of chromosomes 90% of whose timesteps would, on average, contain notes. Since the mutation operator used also changed the velocities to new values chosen with the same probabilities there was little provision for reducing the number of notes in a pattern throughout evolution.

In light of this, an informal analysis was made of the distribution of velocities in the training data and the population was initialised in accordance with this, as shown in table

4.4. In addition, it was noted that demi-semiquaver notes were seldom used in the training data, while in the rare cases where they were it was for ornamental rolls on the snare drum or closed hi-hats. Therefore, timesteps corresponding to these subdivisions were initialised in all cases to 0.0 indicating the absence of a note. The probabilities calculated above took account of this fact. Furthermore, the "bitflip" operator was also adapted so that it changed velocity values according to this probability distribution.

Velocity Value	Probability
0.0	0.5
0.1	0.0
0.2	0.025
0.3	0.025
0.4	0.025
0.5	0.025
0.6	0.05
0.7	0.1
0.8	0.1
0.9	0.1
1.0	0.05

Table 4.4: Initialisation of Note Velocities

It is worth considering why the network did not detect these low fitness features of chromosomes. An informal analysis of the network weights showed that the weights were very small for most of the inputs corresponding to demi-semiquaver timesteps suggesting that these inputs had very little effect on classification. A greater proportion of notes on these timesteps in the negative training data might have resulted in the assignment of low fitness to those chromosomes containing many notes on these steps.

Mutation operators

As noted above, the generated patterns displayed a tendency towards homogeneity. In order to encourage more diversity the following additional mutation operators were implemented:

- a quaver timestep in the bar is randomly selected as a rotation point. Each instrument is then rotated so that that timestep becomes the first timestep. This was inspired by the experiments conducted by [Burton, 1998] suggesting that this operator does not cause a lot of disruption in the population. It should, however, generate useful variations on existing material and exploit the possibility of chromosomes being of low fitness simply because they are slightly "out of phase"

- the genes in each instrument are reversed. This operator was applied with very low probability and was designed to ensure the continuing presence of new genetic material in the population. It was found to be highly disruptive by [Burton, 1998].

The use of these operators resulted in an increased diversity in the drum patterns generated by the system.

Rules

The final problem found with the generated patterns was that they failed to exhibit certain desirable features of the training data. These features were:

- Quaver length hi-hats played on the closed hi-hats and starting on the first timestep.
- Not placing an open hi-hat on the same timestep as a closed hi-hat.
- Not playing more than two open hi-hats in a bar.
- Not placing a bass drum on the same timestep as a snare drum (and vice versa).

Chromosomes lacking these desirable features (and possessing other desirable features) were still given high fitness by the network. It is suggested that this resulted from the "loose" generalisation of the network discussed in section 3.3.4. Another potential reason for the failure to evolve patterns with hi-hat notes on quaver subdivisions of the bar was that these were also present in most rap patterns. This would have resulted in small weights for these timesteps since they failed to distinguish the data. Consequently those inputs to the network would have had little effect on fitness.

In order to solve this problem, four rules were added to the critic (corresponding to the four problems cited above) in the manner of [Spector and Alpern, 1995]. The fitness was defined as the output of the network minus a penalty term proportional to the degree to which an individual violated these four rules. It is worth noting that, with the exception of the presence of quaver length hi-hats, these did not impose any structural constraints defining where notes should be played.

4.6.6 Obtaining Multiple Solutions

A stated aim of the research was to develop a system that would generate a number of drum patterns on each run. One of the attractive features of evolutionary techniques is that whole

populations of solutions are evolved and therefore multiple solutions can be generated. This feature was one reason a GA was chosen to achieve the objectives of this research. However, the fact that a population tends to converge to a point where the entire population is overrun by one very fit chromosome (or small variations thereof) poses a problem: the system must be able to generate a set of patterns exhibiting a certain amount of internal diversity for them to be useful. There exist a number of methods for generating multiple solutions to a problem (excluding the trivial one of re-running the GA).

The most effective of these is a technique called *fitness sharing* which essentially penalises chromosomes for being too similar to others in the population by reducing their fitness (see [Goldberg and Richardson, 1987]). However, this was found to increase significantly the time complexity of fitness evaluation in the current system. A very different approach is to use an *island model* which initialises and evolves several independent populations of chromosomes with periodic migration between subpopulations (see e.g., [Gordon et al. 1992]). An island model has several advantages: first, it increases the parallel nature of search that is the strength of the GA; second, migration represents a continuing source of genetic diversity; and finally, it allows multiple solutions to be evolved. It has been found to improve results over single population GAs even when the total number of chromosomes is the same.

For these reasons, the GA was implemented as an island model with the number of islands set by the user. Migration occurred as a swapping of chromosomes between randomly selected islands. The degree of migration must be balanced since too much will drive local differences between the islands out, while too little will remove the beneficial effects on convergence and diversity. This was achieved via two command line parameters to the system: the migration frequency (how many generations pass before each migration) and the migration rate (the proportion of the population that migrate).

4.6.7 Post-processing of the Generated Patterns

The chromosomes generated by the GA were converted into the format of MIDI files for the following reasons :

- a wide variety of software for the processing and playing of MIDI files is available.
- since the MIDI file is a widely used format the generation of drum patterns in this

format will ensure maximum usability and compatibility.

Each chromosome generated by the GA was written to a text file and then converted into a MIDI file using a slightly modified version of the *convert* program, written by Richard Urwin, and described in section 4.4.2 above. The modifications made involved writing the instrument tracks to the following MIDI note numbers: bass drum track on note number 36; snare drum track on note number 40; closed hi-hat track on note number 46; and open hi-hat track on note number 56. These conform to the General MIDI (GM) format. The user may specify the tempo of the resulting MIDI files which can be played using any MIDI player.

4.7 An Example Run of the GA

In terms of evaluating the system it will be instructive to examine an example run of the GA. The GA is run with a command line call taking the following arguments:

```
$java Rhythms <1> <2> <3> <4> <5> <6> <7> <8> <9>
```

The parameters 1 through 9 are described in table 4.5 (see section 4.6 for further details).

Number	Parameter	Values
1	Generation Size	positive integer
2	Generation Limit	positive integer
3	Fitness Limit	decimal value between 0.0 and 1.0
4	Crossover Rate	decimal value between 0.0 and 1.0
5	Mutation Rate	decimal value between 0.0 and 1.0
6	Number of Islands	positive integer
7	Migration Frequency	positive integer
8	Migration Rate	decimal value between 0.0 and 1.0
9	Output Filename	String with less than 4 characters

Table 4.5: Command Line Arguments Taken by the GA

In this example, the GA was run with the following command line arguments which were found, through a process of trial and error, to produce good results:

```
$ java Rhythms 200 300 0.95 0.6 0.01 3 20 0.1 ex
```

Evolution continued until three patterns (one from each island) attained a fitness of 0.95 (from a maximum of 1.0) after 98 generations (16 minutes on a Pentium 3 600MHz PC running Linux). The three generated patterns (ex0.mid, ex1.mid and ex2.mid) are shown

in figures 4.4, 4.5 and 4.6 in musical notation (the bass drum is on the lowest line of the stave, the snare drum two lines above it, the open hi-hats one line above the snare and the open hi-hats on the top line).

An analysis of the patterns shows that, while they show a certain amount of variation, they tend to be complicated. Although they were not canonical examples of a d&b drum patterns, they were certainly interesting rhythmically and pleased the author by being very different to patterns he tends to generate himself (either using a sequencer or on the drum-set). On the whole, the patterns generated by the GA are "well formed", although some show a tendency to being cluttered and jerky. A more objective analysis of the patterns generated by the system, in relation to the aims stated in section 1.2, can be found in chapter 5.

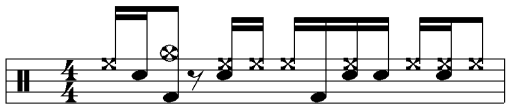


Figure 4.4: Generated Pattern: ex0.mid

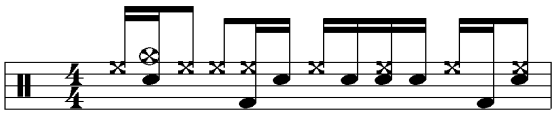


Figure 4.5: Generated Pattern: ex1.mid

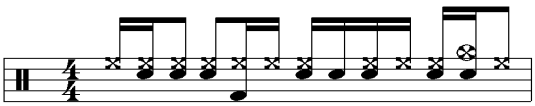


Figure 4.6: Generated Pattern: ex2.mid

4.8 Summary of Chapter 4

This chapter has included a detailed description of the design and development of the system through its various stages. The choice of representation scheme has been described

and justified. The collection and pre/post-processing of the training data was covered. Sections concerning the design and training of the neural network explained the design choices made and described the results of training. Finally, a section covering the design and implementation of the GA described its various features, including the incorporation of domain specific knowledge and the method chosen to obtain multiple solutions. Difficulties and the steps taken to solve them have been presented. An example run of the GA and the drum patterns generated have also been described.

Chapter 5

Evaluation of the Generated Patterns

5.1 Introduction

One area that is often neglected in studies of algorithmic composition is a rigorous evaluation of the compositions generated by the system. Typically, papers will include a sentence suggesting that the author(s) found the output of their own system "pleasing" in some way with a comment such as "while it is difficult to objectively evaluate their quality, I can say with certainty that [the generated musical phrases] rival the carefully prepared demo sequences distributed with most drum machines!" [Horowitz, 1994]!

In the present study an attempt has been made to evaluate in more objective terms whether the system fulfills the specified aims. In fact, aesthetic judgement has been removed from the evaluation of the generated patterns. This chapter describes three experiments: the first was a musical Turing Test of whether subjects could discriminate human from system generated patterns (section 5.2); the second asked subjects to classify the patterns according to style (section 5.3); and the final experiment asked for judgements of the diversity present in groups of three system generated patterns taken from both between and within runs (section 5.4). In a less objective manner a musician was asked to evaluate the system (section 5.5) and finally, the system was informally judged on its usability and practicality (section 5.6).

The experiments were carried out using 19 human subjects from the School of Artificial

Intelligence at Edinburgh University. All experiments were conducted in one session with all 19 subjects present in order to maintain extraneous influences constant across subjects. The questions pertaining to experiments one and two were answered with respect to the same set of drum patterns in an attempt to reduce the amount of listening the subjects would have to do. As noted by [Biles, 1999] subjects find active listening and criticism of music an extremely tiring task. The subjects were asked to state on a scale of between nought and five their knowledge/experience of the musical styles involved.

The patterns used in the experiments were generated using the parameters for the example given in section 4.7. All MIDI drum parts, both human and system generated, were one bar in length and recorded at a tempo of 150 BPM using the GS Roland 909 drum set. It was explained to the subjects that all patterns (both human and system generated) were quantised and recorded using electronic drum sounds. The answer sheets given to the subjects can be found in Appendix C.

All experiments involve testing hypotheses about means and due to the small sample sizes involved the t-test was used. In the case of a one-sample t-test N was calculated as the number of subjects minus one, while in the case of the two sample t-test it was calculated as the number of subjects minus two (for further reading [Cohen, 1995] is an excellent text on experimental methods in AI).

5.2 Experiment 1

This evaluation of the drum patterns attempted to factor aesthetic judgement out of the equation as described in section 3.3.5. The system was designed to generate patterns in the style of d&b using knowledge extracted from a set of example drum patterns in that style. Therefore, a musical Turing Test was designed in which the subjects were asked to discriminate system generated patterns and human generated patterns from the training set. The system was considered to have succeeded if the subjects were unable to distinguish system from human generated patterns.

5.2.1 Experimental Design

A set of drum patterns was constructed containing 10 system generated patterns taken from different runs of the GA and 10 human generated patterns randomly selected from the

ANN training set. These 20 patterns were played in a randomised order to the subjects who were asked to state for each pattern heard whether they thought it was system or human generated. Subjects were also asked to state at the end of the experiment on what basis they were discriminating.

The proportions of system and human generated patterns correctly classified were extracted from the obtained results and the following hypotheses tested with a one sample t-test against the known mean of 0.5 (that expected if subjects were discriminating randomly).

- Null hypothesis one: the mean proportion of human generated patterns correctly classified is the same as that expected if the subjects were answering at random.
- Null hypothesis two: the mean proportion of system generated patterns correctly classified is the same as that expected if the subjects were answering at random.

5.2.2 Results

The results of this experiment are shown in table 5.1.

	Mean	Std. Deviation	Deg. Freedom	t	p
Human	0.516	0.224	18	0.311	0.241
System	0.679	0.181	18	4.241	0.999

Table 5.1: Results of Experiment 1

The results provided two statistical results using 95% confidence intervals. First, we could retain null hypothesis one and second, we could reject null hypothesis two in favour of the following hypothesis:

- Hypothesis two: the sample mean proportion of system generated patterns correctly classified is greater than that expected if the subjects were answering at random.

5.2.3 Discussion

While these results demonstrate that the subjects could distinguish most system generated from human generated patterns it is worth noting that over 20% of the system generated patterns were, on average, misclassified as human generated patterns. This indicates some degree of success. In particular, the only subject to rate his knowledge of the domain as five out of five (and who was the only subject to classify all patterns correctly according

to style in experiment two) misclassified 40% of the system generated patterns as human generated. A score of 50% would have indicated random classification and success on the part of the system. However, the average experience/knowledge professed by the subjects (two out of five) was very low. Given time the experiment would ideally be repeated with more knowledgeable subjects.

It is interesting that the classification performance of the subjects on the human generated patterns was no better than random. This suggests two things: first, that the familiarity of subjects with the domain was low as noted above; and second, a bias towards classifying patterns as system generated. This latter may account, in some part, for the high number of system patterns correctly classified. Some reasons for this bias were suggested by an informal collection of the criteria on the basis of which the subjects distinguished system and human generated patterns.

It seemed that they were, in general, looking out for negative features[1] of the patterns which would classify them as system generated. A sense that they were being asked to "catch the system out" may have lead them to overclassify the patterns as system generated. Those subjects who were looking for features of human generated patterns searched for "smoothness", "coherency", "large scale structure", "subtleties" and such features as whether it qualified as part of a song or similarity to rhythms they had heard in songs. Given that the drum patterns were short, lacking musical context and in an unfamiliar style for most subjects, the use of these criteria may have lead to the bias towards classifying patterns as system generated.

[Urwin, 1997], in a similar experiment, asked subjects to assume that a pattern was human generated if they were unsure (and obtained 85% misclassification of the system generated patterns). However, this may well have produced a bias in the opposite direction. Techniques to counter these kinds of bias warrant further investigation.

5.3 Experiment 2

This experiment was designed to evaluate whether the generated patterns were in the intended style by asking subjects to specify a style for system and human generated patterns.

[1] Examples of these features were lack of originality, randomness (or how chaotic the patterns seemed), predictability and mechanicality lead to classification as system generated. It is interesting to note that both extreme conformity to the prototype of a style and extreme randomness in a pattern classified it as system generated in the eyes (or ears) of the subjects.

If the proportion of system generated patterns correctly classified according to style was equal to or greater than the proportion of human generated patterns correctly classified then the system generated patterns could be considered to be in the correct style.

5.3.1 Experimental Design

A set of drum patterns was constructed containing 10 system generated patterns taken from different runs of the GA, 10 human generated patterns randomly selected from the ANN training set and 10 human generated "techno" drum patterns. Techno was chosen since it is a distinct musical style from d&b but typically has a similar, fast tempo. These 30 patterns were played in a randomised order to the subjects who were asked to state for each pattern heard the style of the pattern from a choice of "drum&bass", "techno" and "other".

The mean proportions of human and system generated patterns correctly classified according to style were collected from the experimental data and the following hypothesis was tested with a two sample t-test. In the case of system generated patterns "correctly classified" refers to classification in the intended style (d&b). The option "other" was counted as an incorrect classification in all cases.

- Null hypothesis: there is no difference in the mean proportions of human and system generated patterns correctly classified according to style.

5.3.2 Results

The results of this experiment are shown in table 5.2.

Human Mean	System Mean	Deg. Freedom	t	p
0.729	0.568	17	2.181	0.978

Table 5.2: Results of Experiment 2.1

Within a confidence interval of 95%, we could reject the null hypothesis in favour of the following hypothesis:

- Hypothesis one: the mean proportion of correctly classified human generated patterns is significantly higher than the mean number of system generated patterns.

Given this result a further one-sample t-test was run against the known mean 0.33 (the expected result assuming the subjects were answering at random) using the null hypothesis:

- Null hypothesis: the mean proportion of correctly classified system patterns is equal to the mean expected if subjects were answering at random.

The result of this test is given in table 5.3.

System Mean	Known Mean	Deg. Freedom	t	p
0.568	0.333	18	3.474	0.999

Table 5.3: Results of Experiment 2.2

We could, therefore, within a confidence interval of 0.99, reject the null hypothesis in favour of the following hypothesis:

- Hypothesis one: the mean proportion of correctly classified system generated patterns is greater than the proportion expected if the subjects were answering randomly.

5.3.3 Discussion

The results clearly demonstrated that fewer system generated patterns than human generated patterns were correctly classified according to style. However, the proportion of correctly classified system generated patterns was significantly greater than that expected with random choice and this indicates a degree of success. Furthermore, the only subject to claim 100% knowledge of the domain, classified **all** the system patterns as being in the style of d&b.

The mean proportion of human generated patterns correctly classified was not very high, once again indicating the low level of expertise of the subjects. The lack of knowledge of the musical styles professed by the subjects might lead one to question the reliability of the judgements made in this experiment. In the light of this, the experiments should ideally be repeated with more knowledgeable subjects. A final point is that, since membership of a stylistic group is probably not a discrete concept, a better experiment might have asked for judgements of the *degree* to which the patterns were considered d&b patterns.

5.4 Experiment 3

This experiment was designed to evaluate the amount of musical variation in the patterns generated both within one run and between runs of the GA compared to the amount of

variation in the training data. This measure of variation was chosen as more musically relevant than an analysis of the patterns themselves (using Hamming distance, for example). Finally, an intermediate degree of variation was desired since too much would take the patterns out of the intended style. The variation in the training data was chosen as a reasonable indication of a desirable amount.

5.4.1 Experimental Design

A set of drum patterns was constructed containing 20 groups of three patterns. Five of these groups of three were constructed from patterns taken from within individual runs of the GA, another five from patterns taken from different runs of the GA and the final ten from patterns randomly selected from the training set. Subjects were played these 20 groups of patterns in a randomised order and asked to indicate on a scale of one to five how much variation they considered there to be within each group. The total amount of variation for the human, the within-run and the between-run groups was calculated for each subject and converted to a fraction between nought and one by dividing it by the maximum possible score. The mean of these values across subjects was then collected.

The mean variation of the within-run and between-run groups was compared to the mean variation of the human groups in a two sample t-test with the following null hypotheses:

- Null hypothesis one: there is no difference between the mean perceived variation of the within-run groups and the human groups.

- Null hypothesis two: there is no difference between the mean perceived variation of the between-run groups and the human groups.

5.4.2 Results

Table 5.4 shows the results of this experiment.

	Human Mean	System Mean	Deg. Freedom	t	p
Within-run	0.601	0.509	17	2.961	0.996
Between-run	0.601	0.502	17	3.055	0.996

Table 5.4: Results of Experiment 3

These statistical results showed that within a 99% confidence interval we could reject both null hypotheses in favour of the following hypotheses:

- Hypothesis one: the mean perceived variation of the human groups of patterns is greater than that of the within-run groups of system generated patterns.
- Hypothesis two: the mean perceived variation of the human groups of patterns is greater than that of the between-run groups of system generated patterns.

5.4.3 Discussion

The level of variation within the system generated patterns (both within and between runs) perceived by the subjects was clearly lower than that perceived in the human generated patterns randomly selected from the training set. However, the variation in both cases corresponds to "quite a bit" on the answer sheets filled in by the subjects (see Appendix C). In this light, the experiment demonstrated that the subjects, on average, perceived a moderate amount of variation in the system generated patterns. This indicates a reasonable degree of success on this evaluation measure.

5.5 Non-experimental Means of Evaluation

An amateur musician who regularly uses MIDI drum parts in his own compositions was given the system to use and asked to answer the following questions (answers are shown in italics):

1. **How would you rate the ability of the system to generate patterns that you would not normally have composed but nevertheless consider "good" patterns and which you would use in your compositions?** *The system produced some technically good and pleasurable-to-the-ear drum patterns most of which tended to be different to those that I sequence myself.*
2. **Are there any reasons you would not use the system?** *As all components of the music being produced must gel rhythmically, a randomly produced drum sequence would mean that all other tracks must be based around that sequence. Also, some of the patterns were slightly busy and chaotic.*

3. **Are there any reasons you would be particularly interested in using the system?** *For the production of open minded and unprejudiced rhythmic sequences, avoiding the need to stick to previously heard and unvaried sequences.*

5.6 Discussion

The degree to which the implemented system achieves the aims stated in section 1.2 can now be evaluated. The aims of this research were to develop a system that would generate a (user specified) number of drum patterns on a single run. The system has achieved this basic goal. Furthermore, the system generated patterns should conform to the following criteria.

1. **They should be comparable with human generated d&b patterns:** the results of experiment one indicate that, on average, 20% of the system generated patterns were misclassified as human generated patterns by the subjects. Furthermore, the only subject to rate his knowledge/familiarity with d&b music as five out five misclassified 40% of the system generated patterns. These results suggest some level of success although the subjects' lack of familiarity with the styles of music involved made interpretation of the results difficult.

2. **They should be within the specified style:** experiment two suggests that 57% of the system generated patterns were on average considered in the correct style (d&b) which is significantly above that expected with random decisions, but significantly lower than the average percentage of system patterns correctly classified (73%). Again the only subject to rate his familiarity with the musical styles as 100% classified all of the system generated patterns as being in the style of d&b. Again, this indicates a modicum of success in satisfying this goal, although the low familiarity of the subjects with the styles involved puts a question mark over the reliability of these results.

3. **They should show sufficient variation both between and within runs to make the system a useful tool:** the subjects perceived a moderate amount of variation in the system generated patterns, both between and within runs. This indicates success in achieving this aim even though the human generated patterns were perceived to contain more variation.

4. **They should show features that would not naturally be generated by the user:** the computer musician evaluated the system favorably in terms of it's ability to generate patterns that were unbiased by human preference and experience and therefore different to those that he naturally produced. It was noted, however, that the patterns were sometimes a little "busy and chaotic."

An important supplementary aim was to extend the approach to more than one style of music. Preliminary investigations into training the network to distinguish rap and d&b patterns failed to produce entirely satisfactory results and a lack of time prevented any further examination of this area. This remains a topic for future work.

A final evaluation measure was the practicality and ease of use of the system. The example run in section 4.7 took 16 minutes to complete (with three MIDI files generated) and this is typical for the system. This is considered (by the author) to be fast enough to make the system practical to use, although the time required increases with the number of solutions (and hence islands) required. While this could be solved by running each island (or indeed individual) on a different processor it is unlikely that the typical musician would have access to multi-processor computers or a network suitable for this purpose.

A second issue, relates to the ease of use of the system. While the command line interface is fairly simple to master, it should be noted that the intended users are musicians who may not be computer literate. In light of this, a GUI would be a more appropriate interface and, furthermore, the expression of the GA parameters in musical, rather than scientific terms, would make the system more intuitive to use.

A few points made by the subjects concerning the experiments are worth noting. First, it was suggested that the short duration of the patterns (just one bar) may have forced subjects to quick and unreliable decisions while the lack of musical context for the drum patterns made the evaluation difficult. Second, the merging of experiments one and two may have lead to unreliable decisions since subjects had to answer two different questions (relating to whether the pattern was system or human generated and what style it was in) about the same pattern. Once again, this may have forced hurried and unreliable responses from the subjects.

Therefore, some suggestions for better designed experiments would be to use separate experiments for each individual test, to use more knowledgeable subjects and to use longer patterns. Finally, the problem of the bias towards classifying patterns as system generated

should be addressed.

5.7 Summary of Chapter 5

This chapter has presented three formal experiments for the evaluation of the system based on the criteria set out in section 1.2. The results of all these experiments have been discussed in relation to the aims of the research . A less objective evaluation of the system by a musician has also been presented. Finally, the system was evaluated in terms of its ease of use and practicality as an aid to composition.

Chapter 6

Conclusions

6.1 Discussion

This research was motivated by a desire to produce a system that would function as a creative aid to the composer, through the automatic generation of libraries of drum patterns within a specified style. The specific objectives were that the system should generate a number of drum patterns, within the specified style and comparable with human generated patterns, on any one run. Furthermore, these patterns should show sufficient variation both within and between runs to make the system a useful tool. A genetic algorithm using an ANN critic was employed to achieve these aims.

The implemented system has been successful to the extent that the system generates a user specified number of drum patterns on any one run, and that these patterns show a degree of variation both within and between runs. Furthermore, the patterns were considered to be different to those typically generated by a human composer. The success of the system generated patterns on the criteria of style and comparability with human generated patterns is more equivocal. The system did not seem to be able to **reliably** generate patterns that would be misclassified as human generated patterns although the fact that 20% were misclassified constitutes some degree of success. Furthermore, it was questionable whether the generated patterns were in the style of d&b, although the inexperience of the subjects with this style of music made the reliability of the results hard to evaluate.

In summary, then, this research has achieved some of the aims set out in section 1.2 and failed (or achieved equivocal success) on others. This is not seen to be a failure of the GA framework, which allowed efficient search of a large and complex search space, the generation of multiple solutions and an elegant means of balancing exploration and exploitation

throughout the search process. Instead, the problems seem to lie with the ANN critic employed in this research in order to avoid the problems associated with human and rule-based critics (see section 3.3.4). A large training set composed of reliable examples of d&b patterns was constructed to avoid the difficulties experienced in past ([Biles et al., 1996], [Spector and Alpern, 1995]) and the classification performance of the trained network was good on the test set. However, the need for a more sophisticated initialisation of the chromosome and the use of constraints in the critic suggest that it provided sloppy evaluation in the context of the GA. The following is an analysis of potential reasons for this.

The reasons for these problems are likely to concern the training data. The set of training instances was constructed from all the d&b patterns on the disks. Analysis of network performance (see figure 4.4) shows that some of the data were wildly misclassified by the trained network. The removal of such instances improved the performance of the network (section 4.5.6). Furthermore, the negative training instances covered only a small region of the space of patterns which do not qualify as d&b patterns. As an example, an informal analysis of the network weights showed that the weights were very small for most of the inputs corresponding to demi-semiquaver timesteps suggesting that these inputs had very little effect on classification. A greater proportion of notes on these timesteps in the negative training data might have resulted in the assignment of low fitness to those chromosomes containing many notes on these steps (this problem is described in section 4.6.3).

In light of problems such as these it would seem that the negative training data were not satisfactory. A better means of generating the negative examples might have been to "invert" each positive example. In this scheme, a negative instance would be generated by taking a positive example and changing all timesteps with non-zero velocity to zero and those timesteps where no note was played to a randomly selected non-zero velocity.

A final problem was that the difference between a rap pattern and a d&b pattern may be one of tempo, percussion sounds used and large scale features (e.g., consistency) rather than structural patterns within a bar. For example, many d&b patterns are simply old funk "breakbeats" recorded at a faster tempo. Similarly, while a rock beat and a rap beat may have the same structural features a rock beat is identified by features such as a deep, loud snare drum and slightly open hi-hats, in contrast to the more crisp electronic sounds typically used in rap.

However, the automation of the critic in evolutionary musical systems is clearly a difficult task[1], as demonstrated by the equivocal success of previous work (see section 3.3.4). The present research has contributed to this body of work by improving on previous attempts to train ANNs for use as critics. While the results have not been absolutely successful, the moderate degree of success achieved warrants further investigation into these techniques for musical composition.

6.2 Areas for Future Research

6.2.1 Improving the Critic

The use of an MLP as the automated critic in musical GAs is attractive due to the possibility of extracting critical criteria directly from a set of example musical phrases, the ability to generalise from these examples and the ability to differentiate different styles of music. However, the research presented here has identified a serious problem with this approach: the generalisation of the network is often too unconstrained. Future research might examine a number of solutions:

1. Improve the training data: the training examples must be carefully chosen and, in particular, the set of negative examples must consider all possible cases of examples which are not within the target classification.

2. Use more complex architecture: the use of a recurrent network would remove the need for a negative training set. In this scheme the network would be trained to predict the drums to be played on the next timestep given those played on the previous timesteps (see section 3.4). The fitness of a chromosome would then be the extent to which the predictions of the network for each timestep correspond to the notes actually played on those timesteps.

3. Use a more complex representation: the use of a more sophisticated representation could have two effects. First, it might make important musical features more accessible to the network. Second, it might be possible to make the representation more compact and therefore facilitate the extraction of temporal relations between notes

[1] Attempts to automate the critic in evolutionary systems for generating visual art have also found this to be a formidable task. See, for example, [Baluja et al., 1994] and [Machado and Cardoso, 2000].

by the ANN. For example, a representation suggested by [Urwin, 1997], represents notes as triples of time since last event, event number (instrument) and event velocity.

4. Use some other learning technique: other learning techniques, such as unsupervised learning or symbolic learning techniques could be used to extract critical criteria from a set of examples. However, an attractive feature of the MLP is its ability to generalise from the data set.

5. Use hybrid critics: as noted in section 3.4 a promising approach to generating music is break the problem down and apply different techniques to those subproblems to which they are suited (e.g., [Hild et al., 1992]). A similar approach might be taken to the problem of evaluating the fitness of candidate musical phrases in an evolutionary system. [Todd and Werner, 1999] have recommended that "musicians intent on creating algorithmic composition systems with the spark of human creativity would do well to adopt [a] combination of co-evolution, learning and rule-following".

6.2.2 Extending the Approach

As noted in chapter 3, this research, although restricted to a limited domain, exists in the wider context of the field of algorithmic composition. To the extent that the system generates musical fragments, the approach described here could be extended in several ways to a play a wider role in this field. First, more than one style of music could be covered; the ANN critic used here could be trained to classify patterns according to style. An exciting possibility would then be to generate patterns that combined elements from different styles of music. This could be achieved using the island model approach used in the present research by assigning each island the task of generating patterns in a different style and using migration to generate populations containing individuals representing different styles of music. The critic could be set up to assign fitness based on different styles of music.

A second interesting area to investigate would be the extension of the approach to generating drum patterns that match the rhythmic properties of melodic sequences. The musician questioned about the system in section 6.5 noted that a drawback of the system is that there is no way of ensuring that the patterns generated will be compatible with the user's compositions.

Finally, the representation scheme should be augmented to cover longer rhythmic phrases

with a more flexible representation of metric time steps and time signature and using more instruments. The ability to represent higher level musical features is also an area worthy of attention when representing larger and more complex musical blocks. A possible way of incorporating higher level features would be to introduce a meta-system which would combine the elements generated by the present system in some way. The appropriate techniques for such a system remain problems for future research.

6.2.3 Evaluation

An attempt has been made in this research to evaluate the success of the system objectively on a number of measures. Specifically, subjective aesthetic judgement has been removed from the evaluation process through the use of a musical Turing test. Several problems have been identified with the experiments made to evaluate the drum patterns generated by the system, on the basis of which, some recommendations can be made for the future work. First, and most important, the subjects should be suitably experienced with the relevant domain. Second, while the experiments should not force subjects to listen to too many musical phrases, the experiments should be limited to one judgement per musical phrase. Third, the patterns should be long enough to allow the subjects to make a considered decision (i.e., longer than one bar). Finally, means of countering the bias shown by subjects towards classifying patterns as system generated should be investigated.

6.3 Conclusions

Although steps were taken to improve the approach in the light of previous failures, this investigation of the application of a GA with an ANN critic to the problem of generating drum patterns has fulfilled its aims with only equivocal success. In particular, the comparability of the generated patterns to human generated patterns and the degree to which they represented the intended style were criteria on which the majority of the generated patterns failed the experimental evaluation. However, the fact that some of the generated patterns fulfilled these criteria was seen as a minor success. Failures have been attributed to shortcomings of the trained ANN in the evolutionary system and, in particular, the training set used. Issues remaining for future research include improving the critic in evolutionary systems, extending the approach to more complex musical phrases and improving the methods

used to evaluate the musical phrases generated by the system.

Bibliography

[Baluja et al., 1994] S. Baluja, D. Pomerleau, T. Jochem. Towards Automated Artificial Evolution for Computer-Generated Images. *Connection Science*, volume 6, pages 325-354, 1994.

[Biles, 1994] J.A. Biles. GenJam: A Genetic Algorithm for Generating Jazz Solos. In *Proceedings of the International Computer Music Conference*, 1994.

[Biles, 1999] J.A. Biles. *Life with GenJam: Interacting with a musical GA.* Presented at the 1999 IEEE Systems, Man and Cybernetics Conference. Tokyo, 1999. Available at http://www.it.rit.edu/jab/

[Biles et al., 1996] J.A. Biles, P.G. Anderson and L.W. Loggi. *Neural Network Fitness Functions for a Musical IGA*. Technical Report, Rochester Institute of Technology 1996. Available at http://www.it.rit.edu/jab/SOCO96/ SOCO.html

[Burton, 1998] A.R. Burton. *A Hybrid Neuro-Genetic Pattern Evolution System applied to Musical Composition*. Unpublished DPhil thesis, University of Surrey, 1998.

[Burton and Vladimirova, 1997a] A.R. Burton and T. Vladimirova. A Genetic Algorithm Utilising Neural Network Fitness Evaluation for Musical Composition. In *Proceedings of the 1997 International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 220-224, 1997.

[Burton and Vladimirova, 1997b] A.R Burton and T. Vladimirova. Applications of Genetic Techniques to Musical Composition. Submitted to the *Computer Music Journal*. 1997.

[Cohen, 1995] P. Cohen. *Empirical Methods for AI*. Cambridge, The MIT Press. 1995.

[Desain and de Vos, 1990] P. Desain and S. de Vos. Autocorrelation and the study of musical expression. In *Proceedings of the 1990 International Computer Music Conference*. San Francisco: Computer Music Association, 1990.

[Desain and Honing, 1991] P. Desain and H. Honing. Quantisation of musical time: a connectionist approach. In P.M Todd and D. G. Loy (Eds.) *Music and Connectionism*. Cambridge, Mass.: MIT Press, 1991.

[Dolson, 1991] M. Dolson. Machine tongues XII: Neural Networks. In P.M. Todd and D.G. Loy, (Eds.), *Music and connectionism*. MIT press, 1991.

[Fahlman and Lebiere, 1990] S. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretsky, ed., Advances in *Neural Information Processing Systems 2*, Morgan Kauffmann, San Mateo, CA, 1990.

[Gibson and Byrne, 1991] P.M Gibson and J.A. Byrne. Neurogen: Musical composition using Genetic Algorithms and Cooperating Neural Networks. In *Proceedings of the Second International Conference of Artificial Neural Networks*, pages 309-313, 1991.

[Goldberg and Deb, 1991] D.E Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In G.J.E Rawlins, (Ed.), *Foundations of Genetic Algorithms*, pages 69-93. Morgan Kaufmann, 1991.

[Gordon et al. 1992] V.Gordon, D. Whitley and A. Bohn. Dataflow parallelism in Genetic Algorithm. In R. Manner and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 553-542, Amsterdam. Elsevier Science, 1992.

[Goldberg, 1989] D.E Goldberg. *Genetic Algorithms in Search, Optimisation and Machine Learning*. Addison-Wesley, 1989.

[Goldberg and Richardson, 1987] D.E Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimisation. In J.J Greffenstette (ed.), *Proceedings*

of the Second International Conference on Genetic Algorithms, LEA, Hillsdale, NJ, 1987.

[Greffenstette, 1987] J. Greffenstette. Incorporating problem-specific knowledge into Genetic Algorithms. In L. Davi (ed.), *Genetic Algorithms and Simulated Annealing*, chapter 4, pages 42-60. Pitman, 1987.

[Griffith and Todd, 1999] N. Griffith and P. M. Todd (Eds.) *Musical Networks: Parallel distributed perception and performance*, Cambridge, MA: MIT Press/Bradford Books, 1999.

[Hild et al., 1992] H. Hild, J. Feulner and W. Menzel. HARMONET: a neural network for harmonising chorales in the style of J.S. Bach. In R.P Lippman, J.E Moody and D.S. Touretsky (Eds.) *Advances in Neural Information Processing Systems 4 (NIPS 4)*, pages 267-274, Morgan Kaufmann, 1992.

[Honing, 1990] H. Honing. The representation of time and structure in music. In *Proceedings of the 1990 Music and the Cognitive Sciences Conference*, edited by I. Cross and I. Delieg. Contemporary Music Review. London: Harwood Press, 1992.

[Hancock, 1994] P.J.B Hancock. An empirical comparison of selection methods in evolutionary algorithms. In *Proceedings of the AISB workshop on Evolutionary Computation*. 1994.

[Haykin, 1999] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall International, 1999.

[Heitkotter and Beasley, 2000] J. Heitkotter and D. Beasley (Eds.). *The Hitch-Hiker's Guide to Evolutionary Computation: a list of Frequently Asked Questions*, USENET: comp.ai.genetic. 2000. Available at ftp://rtfm.mit.edu/pub/usenet/news.ans wers/ai-faq/genetic

[Holland, 1975] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[Horner and Goldberg, 1991] A. Horner and D.E. Goldberg. Genetic Algorithms and computer assisted music. In *Proceedings of the 1991 International Computer Music conference*, pages 479-482, 1991.

[Horner and Ayers, 1995] A. Horner and L. Ayers. Harmonisation of Musical Progressions with Genetic Algorithms. In *Proceedings of the 1995 International Computer Music Conference*, pages 483-484, 1995.

[Horowitz, 1994] D. Horowitz. Generating Rhythms with Genetic Algorithms. In *Proceedings of the 1994 International Computer Music Conference*, 1994.

[Jacob, 1995] B.L. Jacob. Composing with Genetic Algorithms. In *Proceedings of the 1995 International Computer Music Conference*, 1995.

[Jacob, 1996] B.L. Jacob. Algorithmic Composition as a model of Creativity. *Organised Sound*, vol. 1, no. 3, pages 157-165. Cambridge University Press, 1996.

[Koza, 1992] J.R Koza. *Genetic Programming*. MIT Press, 1992.

[Johanson and Poli, 1998] B. Johanson and R. Poli. GP-Music: an interactive genetic programming system for music generation with automated fitness raters. In *Proceedings of the third International conference on Genetic Programming*, GP '98. MIT Press. 1998.

[Machado and Cardoso, 2000] P. Machado and A. Cardoso. The Assessment of an Evolutionary Art Tool. In G. Wiggins (ed.), *Proceedings of the AISB 2000 Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science*, Birmingham, UK, 2000.

[McCulloch and Pitts, 1943] W.S. McCulloch and W. Pitts. A logical calculus of the ideal imminent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, pages 115 - 133. 1943.

[McIntyre, 1994] R.A McIntyre. Bach-in-a-Box: the evolution of four-part Baroque harmony. In *Proceedings of the IEEE conference on Evolutionary Computation*, pages 852-857, 1994.

[Minai and Williams, 1990] A. Minai and R. Williams. Acceleration of Back-Propagation through learning rate and momentum adaptation. *International Joint Conference on Neural Networks*, Vol. 1, pages 676-679. 1990.

[Minsky, 1981] M. Minsky. Music, mind and meaning. *Computer Music Journal.* 5(3):28-44, 1981.

[Mozer, 1994] M.C. Mozer. Neural Network Architectures for temporal pattern processing: Exploring the benefits of psychophysical constraints and multiscale processing. *Connection Science*, 6, pages 247-280, 1994.

[Papadopoulos and Wiggins, 1999] G. Papadopoulos and G. Wiggins. AI methods for algorithmic composition: a survey, a critical view and future prospects. In *Proceedings of the AISB'99 Symposium on Musical Creativity*, Edinburgh, UK, 1999.

[Phon-Amnuaisuk et al., 1999] S. Phon-Amnuaisuk, A. Tuson and G. Wiggins. Evolving Music Harmonisation. In *ICANNGA '99*, Slovenia, 1999.

[Phon-Amnuaisuk and Wiggins, 1999] S. Phon-Amnuaisuk and G. Wiggins. The four-part harmonisation problem: a comparison between genetic algorithms and a rule-based system. In *Proceedings of the AISB'99 Symposium on Musical Creativity*, Edinburgh, UK, 1999.

[Putnam, 1994] J.B. Putnam. *Genetic Programming of music*. Technical Report, New Mexico institute of mining and technology, 1994. Available at http://www.nmt.edu/jefu/notes/ep.ps

[Ralley, 1995] D. Ralley. Genetic algorithms as a tool for Melodic Development. In *Proceedings of the International Computer Music Conference*, pages 501-502, 1995.

[Rosenblatt, 1959] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organisation in the brain. *Psychological Review*, 65, pages 386-408. 1959.

[Ross and Corne, 1995] P.M. Ross and D. Corne. Applications of Genetic Algorithms. *AISB Quarterly*, 89:32-30, 1995.

[Rumelhart and McClelland, 1986] D.E. Rumelhart and J.L. McClelland. *Parallel distributed processing: exploration in the microstructure of cognition* (vols. 1 and 2). Cambridge, MA: MIT Press. 1986.

[Rothstein, 1992] J. Rothstein. *MIDI: a comprehensive introduction*. Oxford University Press, Oxford, 1992.

[Sarle, 1997] W. Sarle (ed.). *Usenet Neural Network FAQ*, comp.ai.neural-nets, 1997. Available at ftp://ftp.sas.com/pub/neural/FAQ.html

[Spector and Alpern, 1994] L. Spector and A. Alpern. Criticism, Culture and the Automatic Generation of Artworks. In *Proceedings of the twelfth national conference on Artificial Intelligence, AAAI-94*, 1994.

[Spector and Alpern, 1995] L. Spector and A. Alpern. Induction and recapitulation of deep musical structure. In *proceedings of the IFCAI-95 workshop on artificial intelligence and music*, 1995.

[Thywissen, 1996] K. Thywissen. Genotator: An Environment for Investigating the Application of Genetic Algorithms in Computer Assisted composition. In *Proceedings of the 1996 International Computer Music Conference*, pages 274-277, 1996.

[Todd, 1989] P.M Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, Vol. 13, No. 4, pages 27-43, 1989.

[Todd and Loy, 1991] P.M. Todd and D.G. Loy, editors. *Music and connectionism*. MIT press, 1991.

[Todd and Werner, 1999] P.M. Todd and G. Werner. Frankensteinian approaches to Evolutionary Music composition. In N. Griffith and P. M. Todd (Eds.) *Musical Networks: Parallel distributed perception and performance*, Cambridge, MA: MIT Press/Bradford Books, 1999.

[Urwin, 1997] R. Urwin. *Connectionist Methods for Musical Rhythm Composition*. Unpublished Undergraduate dissertation, University of Edinburgh, 1997.

[Werner and Todd, 1997] G. Werner and P.M Todd. Too many love songs: sexual selection and the evolution of communication. In P. Husbands and I. Harvey (Eds.), *Proceedings of the Fourth European Conference on Artificial Life*, pages 434-443, 1997.

[Wiggins and Smaill, 1999] G. Wiggins and A. Smaill. Musical Knowledge: What can Artificial Intelligence bring to the Musician? In Miranda (ed.), *Readings in Music and Ar-*

tificial Intelligence, Harwood Academic Publishers, 1999.

[Wiggins et al., 1999] G. Wiggins, G. Papadopoulos, S. Phon-Amnuaisuk and A. Tuson. Evolutionary methods for Musical Composition. *International Journal of Computing Anticipatory Systems*, 1999.

Appendix A

Summary of Related Work

A.1 Interactive Human Evaluation

Author(s)	Description
[Biles, 1994]	GenJam is an IGA which evolves melodic jazz improvisations, including improvised responses to human played phrases. It's performance is described as "competent with some nice moments."
[Horowitz, 1994]	Evolves multitimbral drum patterns.The user sets rhythmic parameters which guide evolution to a point where the user takes over evaluation. The patterns "rival the carefully prepared demo sequences distributed with most drum machines."
[Putnam, 1994]	A GP system evolved programs to generate melodic. waveforms. Evaluation was via feedback from a WWW interface. The generated music was described as "unpleasant and irritating."
[Ralley, 1995]	A GA was seeded with stochastically generated variations on a user supplied melody. The system produced "a large number of interesting variations on initial material" but showed "a propensity towards homogeneity."
[Jacob, 1995]	GAs were used to evolve a stochastic music generator and critic in tandem. Generated melodic phrases were arranged into larger pieces for user evaluation.
[Thywissen, 1996]	Evolved musical phrases or combinations of phrases. Phrases were evaluated by rhythmic, harmonic and melodic rules as well as a subjective evaluation procedure. The system generated "interesting music."

A.2 Rule based Evaluation

Author(s)	Description
[Horner and Goldberg, 1991]	Used a GA to search for sequences of operations that would transform an initial note set into a final note set using a deterministic critic. Results were “musically pleasing.”
[McIntyre, 1994]	Used a GA to generate four-part Baroque harmonies based on an input melody in the key of C major using a three tiered critic.
[Spector and Alpern, 1994]	A GP system to evolve programs that generated a four bar continuation of a four-bar Charlie Parker melody. Results did not please the authors.
[Horner and Ayers, 1995]	A GA for evolving four-voice harmonisation of chord progressions using rules derived from musical theory.
[Werner and Todd, 1997]	A system for evolving simple melodies using co-evolution between the music generators and the critic, thereby producing increased diversity.
[Wiggins et al., 1999]	A GA for the generation of instrumental jazz solos using rules from jazz literature and musical mutation operators. The results were “quite acceptable” but lacked high-level structure.
[Phon-Amnuaisuk et al., 1999]	A GA for evolving 4-part harmonies for user specified melodies using music theoretic knowledge. The results were given 30% on average by a music lecturer.

A.3 Neural Network Evaluation

Author(s)	Description
[Gibson and Byrne, 1991]	Used a GA to generate simple musical phrases using MLPs for rhythm and melody evaluation, and simple rules for harmonisation.Produced "pseudo music with some success."
[Spector and Alpern, 1995]	Extended their GP approach described in A.2 above by training a MLP on Charlie Parker melodies. Results were "unsatisfactory" and a hybfid ANN/rule based critic used.
[Biles et al., 1996]	Trained an ANN on good and bad jazz solos taken from interactive runs (see [Biles, 1994] above). The network failed to learn the training data.
[Johanson and Poli, 1998]	A GP system to generate short musical sequences using an MLP trained on data from interactive runs. Generated "interesting and pleasant sequences". But not with any consistency.
[Burton, 1998]	Trained an ART network to cluster drum patterns from different styles of music and used propinquity to a cluster as the fitness function in a GA. Generated patterns were similar to the training data but rather homogeneous.

Appendix B

Example from Training Data

B.1 An Example Drum Pattern

An example drum pattern from the training set in musical notation (the bass drum is on the lowest line of the stave, the snare drum two lines above it, the closed hi-hats one line higher and the open hi-hats on the top-line of the stave):

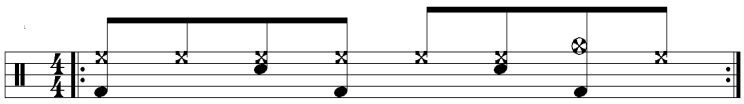


Figure B.1: Example Drum Pattern

B.2 An Example Chromosome

The same drum pattern is shown as a chromosome using the representation scheme described in section 5.2). The first column corresponds to the bass drum, the second to the snare drum, the third to closed hihats and the fourth to open hihats. Each row represents a semiquaver subdivision of the bar on which a note may fall.

1.0 0.0 0.7 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.5 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 1.0 0.5 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
1.0 0.0 0.5 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.8 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 1.0 0.5 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
1.0 0.0 0.0 0.5
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.4 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0

Figure B.2: Example Chromosome

Appendix C

Answer Forms

C.1 Experiments 1 and 2

You will be played a series of drum patterns. For each drum pattern indicate in the table below whether you think the pattern is human generated (with an “H”) or system generated (with an “S”) and also which musical style you think the drum pattern is in (from a choice of “techno” , “drum and bass” or “other”) by ticking the appropriate box.

pattern no.	Human/System	Techno	Drum&Bass	Other style
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
. . . .				
25				
26				
27				
28				
29				
30				

C.2 Experiment 3

You will be played groups of three patterns. For each group indicate how much variation there is within a group (i.e., how different the patterns seem to you) by ticking the relevant box in the table below.

Variation	none	a little	quite a bit	a lot	loads
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

Appendix D

Sources of Training Data

D.1 KeyFax Software

Examples of drum patterns in the following styles were taken from the following floppy discs:

Drum and Bass:	Dangerous Drums MIDI Breakbeats
Rap:	MIDI Breakbeats L.A Riot

available from: KeyFax Software

PO Box 4408

Henley-on-Thames

Oxon RG9 1FS

www.keyfax.com

D.2 Sinuso

A program used to generate type 0 Standard MIDI files, with a randomly generated distribution of notes. This program is available from:

http://www.iwaynet.net/~ryand/sinuso

Calling:

```
sinuso -ch 10 -m 1 -nocc -f example
```

will generate a MIDI file called example.mid containing a pattern one bar long on channel 10 with randomly generated note lengths.

Appendix E

Network Performance

Outliers were removed from the network training data and the network retrained. Figure E1 shows the performance of the trained network on the test data against the target outputs (in bold). As the figure shows, the performance was greatly improved by the removal of outliers from the training set. There were, however, still several misclassified data points in the positive training data. Once again it is suggested that these are drum fills although time constraints have not allowed a verification of this conjecture.

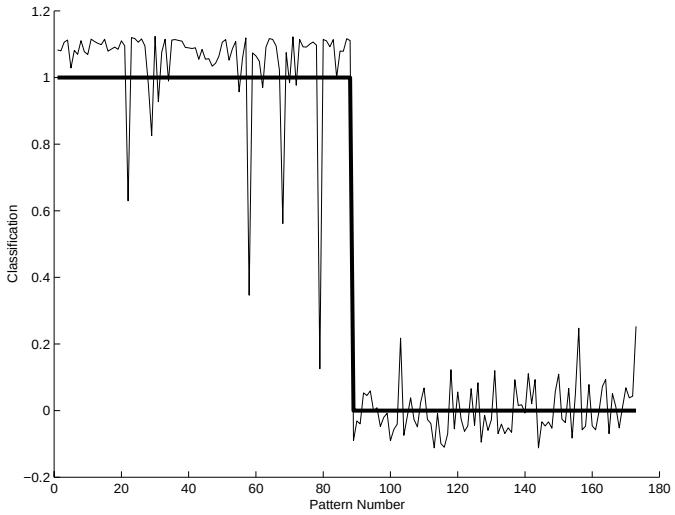


Figure E.1: Network Performance on Test Set