UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

Evolutionary Optimisation and Prediction in Subjective Problem Domains

by

Dan Costelloe

BSc

Supervisor: Dr. Conor Ryan

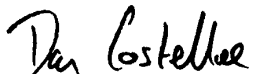
External Examiner: Dr. Maarten Keijzer

Submitted to the University of Limerick in partial fulfillment of the requirements for the award of Doctor of Philosophy, November, 2009

Declaration

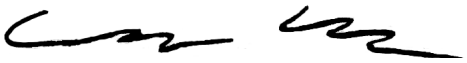
I hereby declare that the work presented in this thesis is original except where an acknowledgement is made or a reference is given to other work. I confirm that I have read and accept the Handbook of Academic Regulations and Procedures.

Student: Dan Costelloe

Signed: 

Date: November, 2009

Supervisor: Dr. Conor Ryan

Signed: 

Date: November, 2009

Abstract

Artificial Evolution is a powerful tool for generating realistic solutions to a large range of computationally difficult problems. It has been applied with great success to many optimisation problems in engineering and science, yet its application is not restricted to problems specific to these fields. The power of evolution can also be coupled with human supervision to tackle problems whose solutions must be (wholly or partly) subjectively evaluated. This thesis describes the design, implementation and use of Evolutionary-based system used for the evolution of such entities whose "goodness" is commonly only subjectively defined.

Additionally, this research investigates and tests formal models of subjective notions for a specific problem: the Interactive Evolution of music. It is demonstrated by this research how various evolutionary techniques can be used to generate and evolve pleasing musical sequences. It is also shown how similar techniques are used to build models of the subjective notions used by human users, when evaluating the goodness of musical pieces. The research presented here also makes it possible to understand what environmental conditions lead to the construction of artificial models that have good predictive power.

Finally, an investigation of the generalisation performance of a specific Evolutionary technique, Genetic Programming, is presented in the context of more recently developed improvement techniques. It is demonstrated that any improvement must take generalisation performance into account in order to be considered a worthy addition to the field. It is also shown how a combination of recent improvement techniques make significant performance improvements on both artificial and real-world symbolic regression problems.

Dedicated to my parents, Phyl and Liam Costelloe

Acknowledgements

There really is no way to fully express how thankful I am to my supervisor Dr. Conor Ryan. Any time that my confidence was dwindling, Conor was there (very often out of office hours) to provide a necessary and welcome boost. It is thanks to Conor's advice, expertise, guidance and experience that this thesis has reached completion.

Getting here has been a long and sometimes quite a difficult process. I am extremely fortunate to have had excellent company along the way. In particular, I have been blessed with the friendship and support of Dr. Miguel Nicolau. Miguel has been a mentor, a critic, a Guinness drinking buddy, and an inspiration.

During my time at University of Limerick, I was also lucky enough to work with some of the warmest and brightest individuals that there are. For their part in providing *research group therapy* I offer sincere thanks to my colleagues and fellow students (past and present): Dr. Atif Azad, Eoin Ryan, Dr. Hammad Majeed, Dr. Michael O'Neill, David Wallin, Miriam O'Riordan, Al Sheahan, Darwin Slattery and Fiacc Larkin.

For their help with viva preparation and the supply of willing subjects during the final stages of this work, I would like to thank some of my former colleagues at NUI Maynooth: Dr. Susan Bergin, Dr. Aidan Mooney, Prof. Ronan Reilly, and Dr. Diarmuid O'Donoghue. I would also like to thank Dr. Adrian Trenaman for convincing me to start with this avenue of research and Dr. Peter Mooney for persuading me to continue with it at UL.

Many thanks are also due to my examiners, Dr. Maarten Keijzer and

Mikael Fernström for the invaluable feedback that has helped to improve this thesis and for providing the proof that a viva examination can be lots fun as well as truly terrifying. Thanks also to Dr. Jim Buckley for taking the time to act as chairperson and for setting me at ease.

The last few years of this PhD were carried out on a part-time basis. Sincere thanks are therefore due to my employer, Andy Dowling, for providing the flexibility for me to see it out until the end.

I owe massive thanks to my friends, for listening, tolerating, encouraging and supporting me over the past few years: Scott, John McCamley, John O'Brien, Conor, Lee, Ca, Evelyn, Paul, Eimear and Keith. I would also like to thank Jane Geraghty for helping me over some of the bumps along the way.

For their love and unquestioning support, I wish to thank my family: Mom, Rob, Suzie, and Nöelle and my grandfather (the *original* Dan Costelloe). Thanks also to John and Anne of the Egan clan and to Marie-Claire for successfully convincing others that my research was all about "coffee drinking robots" — she wasn't far off.

Finally, to the person who has had to put up with so much, expecting nothing in return, I thank my wife and best friend, Sarah Egan. Sarah will never know the true extent of the love, help and encouragement that she seamlessly provided to me as I was working towards this PhD. It is with great pleasure, then, that I look forward to spending the rest of my life thanking her.

This research was supported by Science Foundation Ireland.

Contents

Nomenclature

ANN Artificial Neural Network

EA Evolutionary Algorithm

EC Evolutionary Computation

GA Genetic Algorithm

GE Grammatical Evolution

GP Genetic Programming

IEA Interactive Evolutionary Algorithm

LS Linear Scaling

NSM No Same Mates

List of Figures

List of Tables

Chapter 1

Introduction

With advances in computer-aided musical composition and Artificial Evolution, many systems have emerged that allow human users to guide the evolution of melodies, drum patterns, harmonies and musical pieces. However, because of the subjective nature of the way that music is experienced by humans, there is no universally acceptable formal function that can judge the quality of an arbitrary musical phrase. Hence, Evolutionary music systems need constant human guidance in order to progress from one generation to the next.

The ultimate goal of the research presented in this thesis is to build a system for generating pleasing music[1] that has the extra ability to learn users' musical tastes from past experience – using historical data that is collected as the users progressively evaluate and evolve musical phrases. In this research, Evolutionary learning methods are also investigated for this seemingly impossible modelling task.

To achieve the ability to model subjective taste, an artificial system must be able to learn from the aesthetic judgements as made by humans in order to effectively mimic their choices. This task is by no means trivial since the quality metrics that people use when making aesthetic judgements are highly subjective. For example, any two people can have *entirely* different opinions

[1]There are many different forms of music. In the context of this thesis, music refers to note-based, tonal music, traditional European Western art music or popular music.

about a given piece or art. Reasons for these differences may be obvious and easy to quantify, for example, use of colour and perspective could feasibly be the sole basis for a preference. Equally, another basis could be caused by an emotional response to an aspect of an image or scene – such factors are obviously much more difficult (and arguably impossible) to formalise. Unlike the types of formal measurements that can be found elsewhere in science and engineering, a vastly different set of criteria is applied for the types of aesthetic judgements that are used in the creative arts.

The research presented in this thesis describes a process of extracting a formal model of a subjective notion. In particular, the notions of musical "pleasantness" are examined in the context of interactive musical evolution with the goal being to computationally discover a model which is closely correlated with a set of aesthetic judgements made by the human user of the system.

1.1 Driving Force

Introducing a human decision maker into an automated, iterative process introduces an obvious (however necessary) bottleneck. Any formalisation of the subjective functions at work in the mind of the human decision maker offers a distinct advantage in terms of alleviating this bottleneck. This ability would be of significant value to the field Interactive Evolution. Countless examples can be found in academic literature reporting the success of a particular Evolutionary system at solving a previously unsolved or difficult medical or engineering problem. However, when human intervention becomes a necessary part of the evaluation process, the conclusions almost without exception point to the fact that Interactive Evolution is slow, humans can become tired and inconsistent, often leading to "interesting" however not altogether useful results. Having some way to speed up the interactive process without a negative effect on solution quality would be a very desirable utility.

1.1.1 Central Hypothesis

This thesis forwards the hypothesis that Artificial Evolution can play a fundamental role in modelling the subjective choices made by humans taking part in interactive creative processes. Prediction of future behaviours can be achieved using artificially created models built from past experiences.

In this way, the Evolutionary Algorithm is more than just the *engine* that drives interactive systems, but can also be a mechanism for *bootstrapping* fitness allocation.

1.2 Core Questions

To explore the central hypothesis, this research poses the following core questions:

- Can a system be constructed that displays the ability to learn subjective notions from humans?
- How well do artificially created models perform on future (unseen) choices?
- Can the causes of any inconsistent user behaviour be identified and restricted?

Outside of the musical domain, this research also investigates the predictive power of evolutionary learning and prediction methods. In particular, this thesis investigates the generalisation abilities of Genetic Programming and asks if recent advances in the theory have lead to practical improvements on real-world problems.

1.3 Thesis Contributions

The central contribution of the research described in this thesis is that subjective notions *can* be formalised in the context of Interactive Evolution. Two

Evolutionary music systems, the *DrumGA* and the *MelodyGA* were created with the latter used in the world's first Evolutionary music song contest. Results produced from experimentation on human subjects show that artificial models can be found with good predictive power.

The thesis also contributes valuable lessons learned from musical evolution experiments on human subjects; a condition termed here as *fatigued distraction* is identified as a potential threat to consistent behaviour, which can adversely affect the ability to construct accurate, artificial models. Methods to alleviate this condition are introduced including the use of a binary-decision, tournament-style user-interface as opposed to a larger palette of candidate solutions.

It is shown here that consistent user behaviour is the key to the construction of usable artificial models of that behaviour. When the historical data of a human user is replaced with that of a more more controllable, consistent, artificial agent, the resulting models created display good predictive power.

To address some of the drawbacks of preceding work it is shown how the limitations of an artificial learner may be assessed, making it possible to understand the conditions under which good predictive power is possible.

The final contribution of this thesis relates to the generalisation performance of Genetic Programming. It is shown that the practice of producing and reporting generalisation results in GP improvement techniques is not just a desirable, but an absolutely necessary stage that is often omitted from GP research. A combination of two new techniques is recommended for symbolic regression problems based on a set of experiments on artificial problems. It is further shown how the same combination of techniques leads to better results when re-applied to a real-world problem in the musical domain.

1.4 Thesis Organisation

The rest of this thesis is organised as follows;

In Chapter 2, an introduction to Evolutionary Algorithms, together with

details on their implementation and sub-divisions between them is given. The methods are described and compared in the context of a simple problem that involves making a model that best describes a set of data. A brief description of a non-evolutionary method, related in terms of its application to modelling from historical observations is also provided. All of the methods described in Chapter 2 will make a reappearance in subsequent chapters of this work.

Existing research in the fields of Computer Music and Interactive Evolution is reviewed in Chapter 3. The chapter introduces key issues related to the design of Evolutionary music systems and their associated fitness functions.

In Chapter 4, the design and implementation of two systems for producing pleasing rhythms and melodies is described, together with details of experiments carried out on human subjects using these systems. In this chapter it will be shown that in certain successful cases, Genetic Programming can be used to construct models of the fitness functions of human users.

To try to address the less successful cases, Chapter 5 discusses some of the limitations of previous work and outlines some new directions for research in order to address them. A scaled-down re-implementation of a previous system is provided together with a set of "bottom-up" experiments that lead to a better understanding of the learning power of another Evolutionary technique, Grammatical Evolution, for the task of fitness function modelling.

In Chapter 6, direct attention is diverted from the musical domain and instead focused on the generalisation ability of Genetic Programming. Some practical suggestions for improving the utility of Genetic Programming are provided and demonstrated on a set of artificial symbolic regression problems and then later re-applied to a real-world fitness function modelling problem introduced earlier in the thesis.

Finally, some conclusions and some possible future research directions are laid out in Chapter 7.

Chapter 2

Evolutionary Algorithms and Machine Learning

The work presented in this thesis relies heavily on both the use and the application of the biologically inspired methods known as Evolutionary Algorithms. This chapter will give an introduction to these methods, giving an outline of their overall operation together with some worked examples to illustrate their application. Additionally, a brief introduction to a non-evolutionary learning method used later in this work is provided.

2.1 An Introduction to Evolution

Evolutionary Algorithms (Holland, 1975; Goldberg, 1989; Koza, 1992; Back *et al.* , 1997; Ryan *et al.* , 1998) have been demonstrated to be powerful, robust mechanisms for solving a multitude of problems in engineering and science over the past three decades. Building on their successful application to solving complex search and optimisation problems in the aforementioned research fields, they have also found application in the creative arts. The process of searching a vast array of candidate solutions to a problem using biologically inspired methods can be just as useful for composing pleasing musical pieces as it is for optimising the parameters for the design of aircraft

wings or telescope lenses.

The following sections provide an outline of the operation of Evolutionary Algorithms.

2.1.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) is an umbrella term that is used to describe a set of search an optimisation techniques that have their basis in evolutionary biology. These biologically inspired heuristics are often applied to problems that have no known polynomial-time best solutions. To use an evolutionary approach to solving a particular problem, it is necessary to know at least the following:

1. How can a candidate solution be **represented**?
2. How can a candidate solution be **evaluated**?

With this much information at hand, an Evolutionary Algorithm proceeds as follows:

- **Step 0:** An initial **population** of candidate solutions is created (typically at random)
- **Step 1:** Each candidate solution is evaluated using a **fitness function**
- **Step 2:** The best or fittest solutions are **selected** into a breeding pool to reproduce
- **Step 3:** A generation of offspring is produced by combining candidates from the previous step using **genetic operators**
- **Step 4:** If a termination criterion is met, the process stops, otherwise go to Step 1.

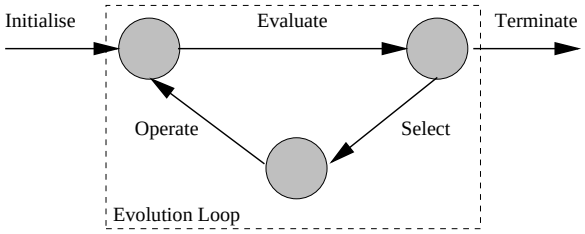


Figure 2.1: Depiction of an Evolutionary Algorithm. In the case of Interactive EAs, the evaluation process is carried out by a human supervisor.

This process is depicted in Figure 2.1. A typical termination criterion is a pre-specified number of iterations; alternatively, the process can be repeated until a threshold for the quality of the best solution produced has been reached.

Assuming that a problem is suitable for the application of an Evolutionary Algorithm (for example, no known polynomial time algorithm exists for solving it), what is typically observed over iterations of the evolution loop is that the overall solution quality increases. The population of evolving solutions is driven towards fitter states due to the repeated application of genetic operators: firstly, the fittest members are **selected** according to a specific selection scheme. The purpose of selection is to produce a breeding pool which is made up from the best genetic material available in the population at each generation. Selection mechanisms vary from one Evolutionary Algorithm to another; two popular mechanisms are *tournament* and *fitness proportionate* selection. In tournament selection, arbitrarily chosen individuals from the population are compared to one another and the *winner* (fittest individual) is placed into the breeding pool. Fitness-proportionate selection operates by assigning a bias to each individual according to its fitness proportion (compared to the fitness of the whole population). This mechanism is often called *roulette wheel* selection since it can be described as assigning each individual a slice of a roulette wheel and spinning it; since the slice of the wheel assigned to each individual is proportional to the fitness, the wheel is more likely to stop on fitter individuals (due to their fitness-proportionate bias).

Although selection is a powerful mechanism for discovering the best solutions in a population, an Evolutionary Algorithm needs more tools at its disposal in order to produce new, genetically different individuals that can lead to fitter individuals than those discovered in previous generations. This is typically accomplished using the genetic operators *crossover* and *mutation*. The purpose of crossover is to exchange genetic material between parent individuals in the hope that their offspring exhibit the best traits of the parents and are therefore fitter as a result. The mutation operator is performed to introduce minor changes to the genetic material of individuals and it is often applied directly after crossover. Crossover can be thought of as an *exploration* method since it provides a means to explore the search space of possible solutions in new ways. Mutation, then, despite its random, undirected implementation, acts as an *exploitation* mechanism by making minor adjustments to individuals produced via crossover with the aim of bringing out the best characteristics of the newly created individuals. An example of the operation of crossover and mutation for two types of Evolutionary Algorithm are shown in Figure 2.2.

Representations

The way in which candidate solutions are represented gives rise to the subdivisions underneath the Evolutionary Algorithms umbrella. If a solution can be represented using a fixed-length binary string and all solutions fit the same fixed-length *schema*, then the process may be better described as a Genetic Algorithm (Holland, 1975; Goldberg, 1989) or GA. In a GA context, the genetic makeup of a solution is referred to as the *genotype*, while the actual solution to the problem at hand is called the *phenotype*. The fitness function acts on the genotype and produces a numeric result indicative of the phenotype solution quality. Parameter search problems are good candidates for this type of approach; for example, the optimisation of telescopic lenses can be reformulated as a search for a fixed-length set of values that produce the best focus when applied to a lens design. In contrast, some other problems

need a variable length solution structure therefore a tree-based representation can be more appropriate. In this case, the process is often called Genetic Programming (GP) (Cramer, 1985; Koza, 1992). A well known example of this kind of problem is the search for mathematical functions that map a set of input values to a set of outputs with minimum error (symbolic regression).

The differences between the operation of GA and GP as a result of the differences in representation are outlined in Figure 2.2.

Grammatical Evolution

In addition to GAs and GP, there are many other evolutionary mechanisms that fit the mould of an Evolutionary Algorithm as described in the previous sections. One such method is Grammatical Evolution (GE) (Ryan *et al.* , 1998; O'Neill & Ryan, 2003). GE combines a genetic algorithm with a grammar so that GP-like programs may be created and evolved. One of the advantages of GE over GP is that programs / expressions can be generated in any language, provided a grammar is specified. This makes GE quite flexible and modular: researchers who wish to use a GP-like system need only be concerned with the formulation of the grammar from which the individuals are produced. One of the first implementations of GP involved the evolution of s-expressions, which would then be interpreted using a Lisp interpreter. Subsequent implementations have used prefix and postfix ordering of symbols, which are parsed and interpreted using a custom-built interpreter. This coupling of representation to interpreter is much looser with GE than with GP – provided an existing compiler / interpreter exists for the language in which the evolving expressions are written.

2.2 Illustration

To illustrate the operation of GA, GP and GE, the following sections describe the application of each technique to a simple problem. The same problem is used in all cases so that we can focus on the operation of each method.

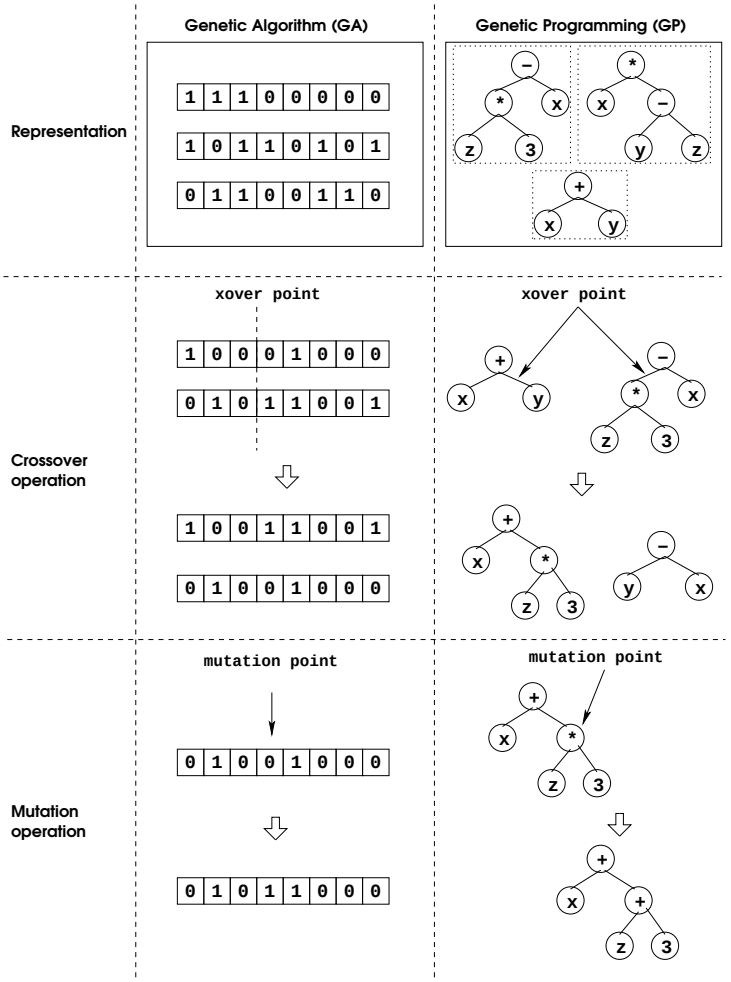


Figure 2.2: Comparison of two Evolutionary Algorithms, Genetic Algorithms and Genetic Programming. The difference in representation is depicted at the top, where we can see the fixed-length binary string representations typically used by GAs on the left and the variable-length, tree based representations that are typically used by GP on the right. While the overall process of evolution for both of these mechanisms remains mostly the same, the difference in representation gives rise to a slight variation in the way in which the genetic operators (crossover and mutation) are carried out.

2.2.1 Problem Description

The problem that will be used in the following examples can be stated as follows: find a function, f, which maps a set of input values onto a set of target values, with minimum error. The problem supposes that we are given a set of data which contains a list of independent observations, x_i, about a phenomenon that we wish to model, t. The task is to produce a function which, when applied to the independent variables, produces the best "fit" when overlaid with the target points. The independent variables and targets are shown in Table 2.1.

Table 2.1: A set of inputs (x) and target values (t) that describe the function f.

x	t
0.1	0.3192
0.2	0.7152
0.3	1.2492
0.4	1.9872
0.5	3.0000

It is worth noting at this point that this type of *model discovery* or *function finding* problem follows the same structure as the problems that will be studied in more detail in later chapters.

2.2.2 Using a Genetic Algorithm

This example varies slightly from the next two in that it assumes the availability of a preprocessing step which provides the algorithm with one extra piece of information about the problem, the *functional form*:

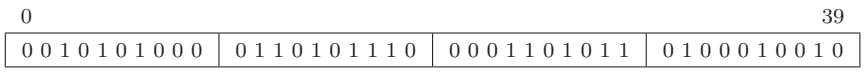
$$f = ax^4 + bx^3 + cx^2 + dx \tag{2.1}$$

The task for the GA, then, is to discover the parameters a, b, c and d (for simplicity, we'll assume that the coefficients a, b, c and d are integers in the

range [0, 1023])[1].

Choosing a representation

To apply a GA to this problem, we first need to define a way to represent candidate solutions. This can be achieved in this scenario by using a 40-bit binary string as follows:



Using the above representation, the bit-string indexed from 0-9 represents parameter a, 10-19 represents b, 20-29 represents c and 30-39 represents d. The example string shown above represents the values (168, 430, 107, 274) for (a, b, c, d).

Evaluating an Individual

To see how well in individual has performed, we need some way to test how close its predicted outcomes are to the target outcomes. For this type of problem it is useful to use a statistical measure of the error as the fitness function. One such measure is the Mean Square Error (MSE) which can be defined as follows:

$$MSE(t, p) = \frac{1}{N} \sum_{i=1}^{N} (p_i - t_i)^2 \tag{2.2}$$

where N is the number of input-target cases, t_i is a target outcome and p_i is a predicted outcome. Using the example individual shown above, we can compute the predicted outcomes by solving the problem described in Equation 2.1 using the decoded integer values. This dataset is shown in Table 2.2.

The mean square error in this instance evaluates to 18106, which is quite *unfit* considering that the goal is to *minimise* the error value[2].

[1]Even though this is never likely to be the case in a real-world problem, we can make this assumption here for illustrative purposes.

[2]In other words, "fitter than" corresponds to "has smaller error than" for this problem.

Table 2.2: Dataset of inputs, targets and predictions.

x	t	p
0.1	0.3192	29.9168
0.2	0.7152	62.7888
0.3	1.2492	104.8008
0.4	1.9872	158.5408
0.5	3.0000	288.0000

Applying Genetic Operators

To show how genetic material is exchanged and updated, the GA genetic operators are now discussed. As mentioned in Section 2.1.1, the *crossover* operation is employed to exchange genetic material between parent individuals. There are many types of crossover implementation for GAs, and the reader is referred to Goldberg's seminal book (Goldberg, 1989) for a detailed discussion of each. In this example, the application of *single point* crossover is illustrated. Using two parent individuals P_1 and P_2, this method works by choosing a random point c_p along the length of the individual and creating a new child individual by copying the genetic material from index 0 to c_p in P_1 and index $c_p + 1$ to the end of P_2. Note that a second child may also be produced using the opposite sections. In the example below the crossover point $c_p = 8$:

P_1	0 0 1 0 1 0 1 0 0	0 0 1 1 0 1 0 1 1 1 0 0 0 0 1 1 0 1 0 1 1 0 1 0 0 0 1 0 0 1 0
P_2	0 0 0 0 0 0 0 0 1	1 1 0 0 1 1 1 0 1 0 0 1 0 1 0 1 1 1 0 1 0 1 1 0 1 0 1 1 0 1 1

By taking copies of the genetic material as described, the following child individual is produced.

C	0 0 0 0 0 0 0 0 0 1 1	0 **1** 1 0 **1** 0 1 1 1 0	0 0 0 **1** 1 0 1 0 1 1	0 **1** 0 0 0 1 0 0 1 0

If necessary, any such minimising problem may be reformulated as a maximising problem by assigning fitness as follows:

$$f(x) = \frac{1}{1 + MSE}$$

In this case, all fitness values fall between 0 and 1 with 1 being the maximum (i.e. perfect) score.

Before this individual is returned to the population, it is given the opportunity to undergo *mutation*. One implementation of this method iterates over the length of the bit string, flipping each bit depending on the outcome of a biased coin toss. This bias is usually set to be quite a low value, causing mutations occur relatively infrequently. This way, the potentially disruptive nature of the operation is minimised. After undergoing mutation, the child individual is depicted below (mutated bits highlighted in bold):

C_m	0 0 0 0 0 0 0 0 1 1	0 **0** 1 0 **0** 0 1 1 1 0	0 0 0 **0** 1 0 1 0 1 1	0 **0** 0 0 0 1 0 0 1 0

Solving Equation 2.1 for the decoded values (2, 142, 51, 18), this individual has an error of 409.61, which is a clear step in the right direction since the goal is to reduce the error to zero, thereby producing a perfect fit. This individual will have a better chance of being selected to take part in the next generation so that it may pass on its genetic material to eventually produce better, fitter progeny.

2.2.3 Using Genetic Programming

In the GA example given in Section 2.2.2, we cheated somewhat by assuming the existence of some pre-processing step that provides the algorithm with the necessary functional form of the data-producing phenomenon. Unfortunately, it is often the case that no such method is known for providing this information, which means that the task of finding the functional form also becomes part of the problem. The variable-length tree based representation that is employed by GP makes this task possible. This type of problem is often studied in the GP literature and is referred to as **Symbolic Regression**.

GP Representation

In GP, individuals are (often) represented as tree structures which vary in shape and size over the duration of the algorithm. The structures are composed of:

- **Terminals**: variables, constants and
- **Non-terminals**: functions or operators

In order to create a valid population of individuals, information about the number of variables, constants and the set of functions that are available must be pre-determined. By examining the example dataset given in Section 2.2.1, we can see that it contains one independent variable, so we add this to the terminal set so that the information is available to the evolving trees. To help with the discovery of coefficients, the *ephemeral random constant*, $\Re$ (Koza, 1992) is also added to the terminal set. When trees are randomly produced at initialisation, this value is used to generate a random constant value. The choice of functions to employ is left entirely up to the user. For the sake of this example we are going to use the arithmetic operators {+, -, *, /}[3] With this information at hand, a population of candidate solutions may be randomly created and evolution proceeds using the process depicted in Figure 2.1.

A Note on Initialisation

For the creation of individuals, Koza describes two methods, *full* and *grow* (Koza, 1992). Building a GP-tree starts with the selection of a non-terminal. Individuals generated using the *full* method have the property that the length of every path between a leaf node and the root is equal to a pre-specified maximum depth value. The *full* method chooses non-terminals for all nodes at depths less than the maximum depth and completes the tree with terminals at the maximum depth level. The *grow* method permits a larger variation in tree sizes by selecting either terminals or non-terminals for every node (except the root node) up to a maximum depth value.

Since the solution structure of the problem (or the shape of the target function) is typically not known in advance, it is important that the initialisation process provides the best platform for the algorithm to start searching

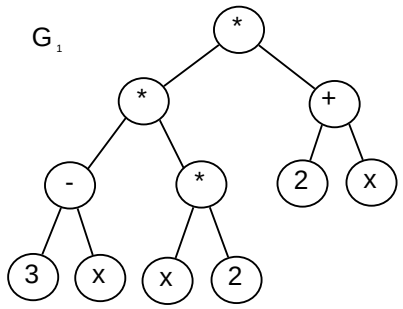
[3]Using a *protected* division operator, which returns one when asked to divide anything by zero.

from. An initialisation method that combines the *full* and *grow* methods called *ramped half-and-half* initialisation permits the creation of an equal-sized sets of trees with depth ranges (from two to the maximum depth) with half of each set created using the grow method and the other half created using the full method.

Evaluating an Individual

Let us assume that during the course of the algorithm, the individual G_1, as depicted in Figure 2.3 is produced:

Figure 2.3: A sample GP-tree.



To evaluate G_1, we first find the set of predicted outcomes by solving G_1 for the independent variables. The results of this operation are shown in Table 2.3. Calculating the MSE using Equation 2.2 results in the value 5.9184.

Applying Genetic Operators

The application of the genetic operators crossover and mutation in this GP context is slightly more complicated than the method described in the previous section and this is mostly due to the structural representation of the GP individuals. To perform crossover in the GA example, it is sufficient to simply choose a random point along the length of the individual and exchange genetic material from the parents about this point. In GP, the crossover

Table 2.3: Solving G_1 for the independent variables (x). The predictions p are shown beside the actual targets (t).

x	t	p
0.1	0.3192	1.218
0.2	0.7152	2.464
0.3	1.2492	3.726
0.4	1.9872	4.992
0.5	3.0000	6.250

must respect the tree representation by ensuring that only valid individuals are produced as a result of the exchange. This property of closure is equally important for the mutation operator. Returning to the example, Figure 2.4 shows the genetic operators at work.

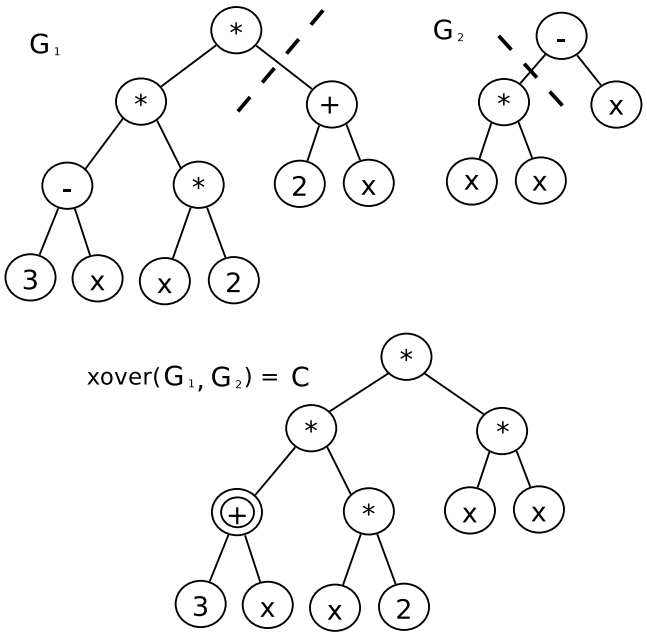
Note that Figure 2.4 also shows the application of the point-mutation operator (double-circled node). Computing the MSE for this new tree results in a value of 1.7720.

2.2.4 Using Grammatical Evolution

We now examine the workings of GE for this example problem. As was the case with the GP example in Section 2.2.3, no pre-processing step exists for this method to produce the functional form so the coefficients must also be discovered. GE works by using a GA as a search engine and mapping binary strings onto programs with the help of a Backus-Naur Form (BNF) grammar.

A BNF grammar is a syntax used to define the rules for constructing sentences in a formal language. It comprises a set of production rules made from *terminals* and *non-terminals*. Terminals refer top indivisible units of the language that do not require further expansion. Non-terminals are symbols that must be expanded to produce one or more terminals and non-terminals. More formally, a BNF grammar can be represented as the tuple $< N, T, P, S >$, where N is the set of non-terminals, T is the set of terminals, P is a set of production rules for expanding elements of N and S is the start symbol

Figure 2.4: Depiction of crossover in GP. The dashed lines show the chosen crossover points. The child tree is produced by swapping out the subtree below the dashed line in G_1 and inserting the subtree taken from the section below the dashed line in G_2, producing the child tree C.



which must be used to initiate the mapping process.

Representation

Since GE uses a GA as a search engine, the representation used can also be binary strings. These are converted to strings of integer values prior to the mapping process. In the GA example of section 2.2.2, a representation was chosen which is tailored to the problem. With GE, this tailoring is performed by providing a BNF grammar which defines how solutions can be constructed. Since the goal of this example is to produce a mathematical function, we can construct the BNF grammar as follows:

```
<expr> ::= <expr> <op> <expr> | <var>
<op>   ::= + | - | * | /
<var>  ::= x
```

Mapping Process

The process by which a binary string is converted into a candidate solution with GE is now described. Assuming a resolution of 8-bits per integer, the binary string is first converted into a string of integer values so that the production rules of the grammar can be indexed. Using the following individual (with values converted to integers) as an example:

index	0	1	2	3	4	5	6	7
	48	29	32	60	33	28	15	46

GE uses modulo arithmetic to index the production options when mapping an individual. The mapping is started by expanding `<expr>`, the start symbol. As shown by the grammar, this symbol has two production options. The production to use for the expansion is chosen by taking the first integer from the string (48) and dividing it by the number of options (2 in this case). The remainder (0) is used to index the chosen production[4]. This produces the (unmapped) string `<expr> <op> <expr>`. To map the next symbol, the next integer value is taken from the integer string (29). The remainder when this divided by the production options (4) is 1, which means that the second production option is chosen. This produces the string `<var> <op> <expr>`. Since there is no production option for `<var>`, a straight substitution can take place leaving the string `x <op> <expr>`. This process continues until the individual is eventually mapped to the solution `x + x * x`. The whole process is illustrated in Table 2.4.

The MSE for the mapped individual in this example evaluates to 1.6214. Note that in this instance, the individual was successfully mapped without the need to use all of the integers in the string. In the event that the individual is still not completely mapped when the end of the integer string is

[4]Option indexes start at zero.

Table 2.4: Overview of the GE mapping process.

Symbol		Value		Mod rule	State
<expr>	becomes	<expr><op><expr>	since	48 % 2 = 0	<expr><op><expr>
<expr>	becomes	x	since	29 % 2 = 1	x <op><expr>
<op>	becomes	+	since	32 % 4 = 0	x + <expr>
<expr>	becomes	<expr><op><expr>	since	60 % 2 = 0	x + <expr><op><expr>
<expr>	becomes	x <op><expr>	since	33 % 2 = 1	x + x <op><expr>
<op>	becomes	*	since	28 % 4 = 2	x + x * <expr>
<expr>	becomes	x	since	15 % 2 = 1	x + x * x

reached, the process continues by wrapping back to the start of the string. If a maximum number of wrapping events has occurred and the individual is still not mapped, it is assigned a minimal fitness value since it does not contribute to the overall wellbeing of the population.

Genetic Operators

Since GE uses a GA as a search engine, the genetic operators that are applied at the genotype level are very much the same as those used in a GA. GE simply requires a set of integers, a grammar and some way to interpret / evaluate the solutions that are produced by the mapping process. In a sense, GE is not concerned about where these numbers come from, as long as they can be used to produce a fitness improvement with successive generations (O'Sullivan, 2003).

2.3 Alternative Learning Mechanisms

The ability to construct models from examples and make predictions from unseen examples is not restricted to the Evolutionary methods described earlier in this chapter. This section introduces another learner-predictor method that is used later in this work; Artificial Neural Networks.

2.3.1 Artificial Neural Networks

Evolutionary algorithms are not alone in borrowing fundamental ideas from biology. An Artificial Neural Network (ANN) is a model that, at a high level, simulates a network of nerve cells in an animal central nervous system. The word "artificial" is important; these models are as far from exact replicas of biological neural networks as Evolutionary algorithms are from accurate models of human evolution. More importantly and of significant interest to this work, is the ability of ANNs to learn, model and predict from observational data. As will be seen later in Chapter 4, a particular instance of ANN, the *multi-layer perceptron* is used as a *black-box* learner / predictor. In order to introduce the concepts and terminology that will be used later, a brief introduction to this class of neural network is given here. Broader sources of information on this and other classes of ANN can be found elsewhere (Graupe, 1997; Fausett, 1994; Haykin, 1994).

An ANN can be described as a mathematical model defining an overall function $f(X) = Y$ where X corresponds so some input value(s) and Y the output. The fundamental building block of an ANN is the neuron, a simple computational element that produces a single output from a set of inputs as shown in Figure 2.5.

A neuron can receive multiple input values, each with an associated weight value. The neuron's output is calculated by combining the inputs and the weights and passing this net input value through an activation function as shown:

$$y = f(\sum x_i w_i)$$

A commonly used activation function is the S-shaped sigmoid function, which has the property of transforming any input into a value between zero and one.

$$f(x) = Sg(x) = \frac{1}{1 + e^{-x}}$$

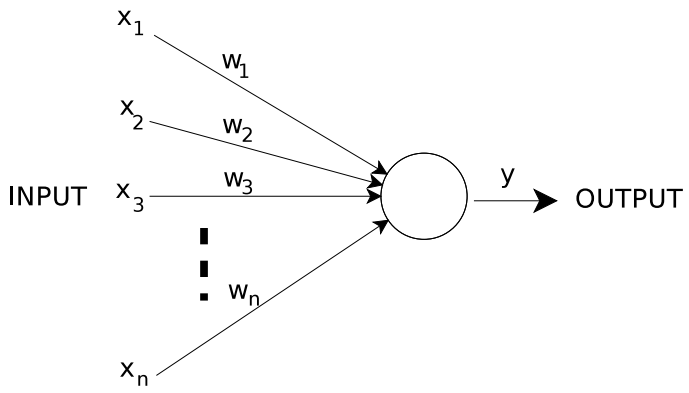


Figure 2.5: A neuron in an artificial neural network. A vector of input values, X is combined with a vector of weights W, to produce a single output value, y.

In a *multi-layer perceptron* network topology (see Figure 2.6), neurons are grouped into distinct layers. With the exception of the final layer the output of each layer is connected to input of neurons in the next layer. Inputs of the first layer are the inputs to the network.

Multi-layer perceptrons are commonly used in classification problems, whereby the approximation of a some function that maps an input vector X to one or more classes C_1, C_2, ..., C_n is performed. An iterative optimisation of the weights, supervised learning, occurs as training examples are presented to the function. The weights are then adjusted by an amount proportional to the error (distance from the network's guessed value and the actual target value). These adjustments are sent backwards into the preceding layers in a process called back-propagation. Repeating this process produces a function that maps the input vector to the target classes with minimum error and is then (ideally) able to make accurate classifications of unseen data.

2.4 Summary

This chapter has provided a general introduction Evolutionary Algorithms together with a set of examples which outline the operation of three sub-

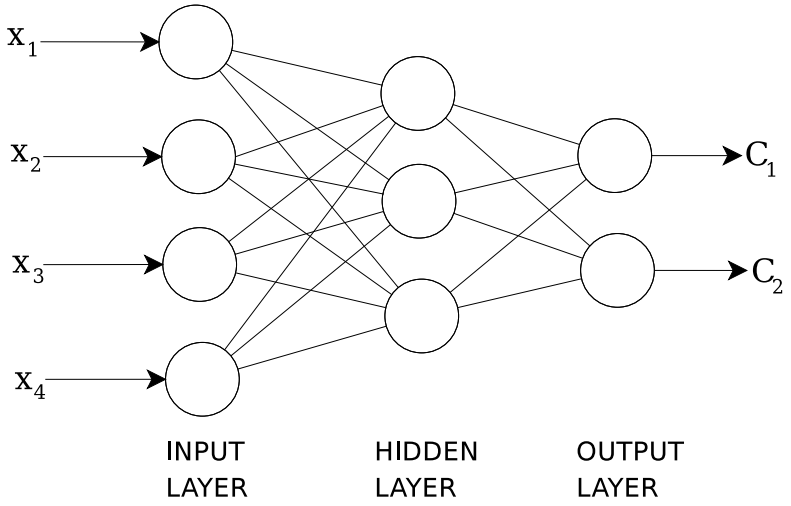


Figure 2.6: A multi-layer perceptron with one hidden layer used to classify a four-valued input vector X into one of two classes, C_1 or C_2.

disciplines, Genetic Algorithms, Genetic Programming and Grammatical Evolution. The sample problem chosen to illustrate the operation of these mechanisms was deliberate as it involves the search through a set of potential models for the best model which describes a set of observations.

Constructing models from observations is described as *supervised learning* and is a process also undertaken by alternate learning mechanisms such Artificial Neural Networks. Since this technique will be employed later in this work, a gentle introduction to its operation has also been provided.

As stated in Chapter 1, one of the central goals of the research presented in this thesis is to find a function that best models a set of observations of musical preferences expressed by a human. In later chapters we will see how the methods described in this chapter are employed to in an effort to accomplish this task.

Chapter 3

Background and Motivation

In the previous chapter, Evolutionary Algorithms were introduced and then described in the context context of a simple test problem. This chapter will describe how Evolutionary Algorithms have found application in the creative arts. It will also provide more details about the motivation for this work, and give a summary of some related research and other relevant background theory.

3.1 Genetic Music Composition

The marriage of Evolutionary Algorithms and music composition is by no means new, and many examples can be found in the literature (Burton & Vladimirova, 1999). The fundamental difference between the application of EAs to an engineering problem and a music composition problem lies in how the candidate solutions are evaluated. Typically, in the former case, an objective function exists that can help to guide the search for ideal solutions towards fitter states. This function can be formally defined and integrated into the evolutionary process. In the latter case, there does not exist a universally acceptable fitness function describing the "pleasantness" of a musical piece: fitness allocation is a subjective task and must be left to the user of the evolutionary system. Such systems are called Interactive Evolutionary

Algorithms (IEAs), and suffer from what has been referred as a "fitness bottleneck" (Biles, 1994): the evolutionary loop can only progress as fast as its slowest component, which, in cases like this, is a human.

3.1.1 GenJam

GenJam (Biles, 1994), short for Genetic Jammer, is an evolutionary system for the creation of jazz melodies. One of the aims of the system was to mimic the learning process of a novice musician while sitting in on "jam sessions", improvising fitting musical phrases and taking the feedback of it's peers as a judgement of the quality of what was played.

The system evolves two populations of measures (short musical sequences) and phrases (sequences of measures) guided by a user-allocated fitness score. GenJam runs in three modes: learning, breeding and demo. The first two modes are used as preparation for the third. In learning mode, phrases are evaluated for fitness by a human mentor. In this mode, only tournament selection is used, which allows the population to grow to a set of usable measures and phrases. Breeding mode then involves the use of genetic operators to combine the musical ideas formed in learning mode. Finally, demo mode is used for live performance of the system and simply employs tournament selection on bred individuals.

Regarding the process of fitness assignment, Biles makes the point that a fitness function clearly exists, although possibly not in an easily expressible algorithmic form. For this reason, he chooses himself as the fitness function since his brain contains the most qualified implementation of the *'I know what I like'* algorithm.

3.1.2 The GP-Music System

The GP-Music system (Johanson & Poli, 1998) allows the evolution of short musical phrases using interactive GP. Musical phrases take the form of short

melodic sequences without polyphony[1]. The sequences evolved by the system consist of notes and pauses (rests).

The system used was an extension of the *lil-gp* (Zongker & Punch, 1996) incorporating music-specific functions and terminals. Terminals used included notes, pseudo-chords (sequences of notes belonging to a specific chord played in sequence) and a 'rest' terminal for pauses. The function set included functions for concatenating sequences, adding pauses to sequences and transposing[2] sequences among others. It is claimed that this GP approach is more flexible than related GA approaches due to its ability to generate variable length sequences.

Fitness evaluation is carried out by a human evaluator through the use of a list interface displaying the current generation whereby individuals are assigned a fitness value ranging from 1-100. An interesting feature of the system is that the human-allocated fitness scores of sequences are "locked in" if they pass unchanged from one generation to the next. It is fair to assume that inconsistent scores for the same sequence could arise due to the continuously changing context of that sequence. Hence, locking in fitness scores forces consistency to be maintained.

To address the fitness bottleneck issue, the GP-Music system was extended using a neural network as a "automated fitness rater". The idea is that after a short run of user-rated sequences, a neural network trained on the user's data would then stand in as the fitness function for longer (yet faster) runs of the system. Unfortunately, it was found that while the auto-rater did pick up an ability to evolve "interesting, and pleasant" musical sequences, it could not do so in a consistent manner.

[1]No notes are sounded simultaneously.

[2]Moving each note in the sequence to a new key.

3.2 State of the Art

Burton and Vladimirova (Burton & Vladimirova, 1999) carried out an extensive survey of the state of the art in evolutionary music composition techniques, the main points of which follow.

A general term for automatic music generation is "algorithmic composition". Compositions constructed by computational means range from short musical phrases to themes to whole pieces, and many artificial techniques have been considered for this seemingly human-specific task. Examples include

- Artificial Neural Networks to generate musical fragments based on representative training data.
- Expert Systems that employ rule bases from music theory for music generation.
- Models of physical processes (Harley, 2004).
- Applications that deduce musical phrases from random events (for example, fractals, DNA and cosmic noise).

It can be argued that the above methods have some in-built limitations that can be overcome by the use of evolutionary techniques. One reason for this argument is the blindness of Evolutionary Algorithms: no explicit domain knowledge is needed for such systems to operate, unlike the methods listed. All that is needed by an EA is a quality metric to assess the worth of candidate solutions.

According to Burton and Vladimirova (Burton & Vladimirova, 1999), three important considerations should be taken on before applying an EA to a music composition task:

- **The search space:** For music composition problems, the potential search space of possible solutions is unthinkably vast (we will examine

this in more detail in section 5.2.1). In many cases it may be necessary to limit the duration, polyphony or key of the potential musical pieces.

- **The representation:** The format of the genetic material making up the evolving candidates can greatly affect the observed solutions. For example, perhaps a fixed-length, fixed-polyphony model is more appropriate than a variable-length, variable-polyphony model depending on the desired output. This is an important consideration for all EAs, not just interactive EAs.

- **The fitness function:** As with any EA, the choice of appropriate fitness function is critical. The fundamental difference between the application of EAs to engineering problems and a music composition problems lies in how the candidate solutions are evaluated. Typically, in the former case, an objective function exists that can help to guide the search for ideal solutions towards fitter states. This function can be formally defined and integrated into the evolutionary process. In the latter case, there is no such fitness function (due to the subjective biases of humans) although many music-theoretic formalisms exist to define structure (such as harmony). Typically, fitness allocation is a subjective task and is therefore left to the user of the evolutionary system. We will discuss this in more detail in the following sections.

In the work presented in this thesis, representation plays a crucial role since it is used as the input to the model discovery process employed by artificial methods. The choice of representation has an immediate bearing on the size of the search space of possible solutions and therefore influences the difficulty of the model discovery task.

Deterministic Fitness Functions

Deterministic fitness functions tend to use some mathematical relationship to assign fitness to musical pieces. This relationship comprises a function of the encoding of individuals representing the pieces, such as the degree of

similarity to a pre-existing, desirable pattern. This can be viewed as a form of improvisation on a theme, which is an important aspect of music creation that can bring together small musical ideas to form larger musical compositions. Rally (Ralley, 1995) combined deterministic fitness functions with user-allocated scores in his melodic development system. Previous work by Horner and Goldberg (Horner & Goldberg, 1991) used deterministic fitness functions for Thematic Bridging; this is the process by which one musical segment is transformed into another over a period of time. Metrics used in this system included a measure of similarity between a generated note pattern and a desired pattern and a measure of similarity of duration between two patterns.

A limitation of deterministic fitness functions is that they are very problem-specific and therefore difficult to generalise to other classes of automated music creation tasks. In the context of evolution this can be quite a serious constraint as it has the potential to "lock out" individuals that are musically valid (and may even contain some useful building blocks), but that do not conform to the rules specified by the fitness function.

Formalistic Fitness Functions

Since music can be described using mathematical concepts, it seems only natural to try to combine some of these concepts into a function for evaluating solution quality. The rules of harmony, for example can quite simply be encoded into a fitness function that is devoid of any subjectivity, which is a desirable characteristic to have if we are interested in processing a large amount of potential solutions within a suitable time-frame. McIntyre (McIntyre, 1994) devised a system for the evolution of Baroque harmony using a multi-tiered formalistic fitness function. Other researchers (Horowitz, 1994; Ralley, 1995; Thywissen, 1999) created similar systems in terms of fitness allocation: they employ a balance of user-defined measures of formalistic fitness functions. In this way, the importance of certain characteristics can be evolved to the user's taste yet evolution can still proceed at a steady pace.

Werner & Todd's (co-evolution) system cuts (Werner & Todd, 1997) out the user completely and considers two co-evolving populations: a (male) set of singers and a (female) set of evaluators, likened to songbirds. Note that this scheme used sexual selection as opposed to natural selection in that the individuals evolve to suit each other rather than behave optimally in some user-defined environment.

User-determined Fitness Functions

The most intuitive form of fitness function to use for this type of task is to use a human evaluator. Human subjects don't need to be able to judge musical pieces based on formal or music-theoretical grounds; rather, they just need to be able to allocate a score to a given musical piece. Biles' GenJam system (Biles, 1994) is probably one of the most famous examples of this type of system. Obviously, the "fitness bottleneck" (as Biles coined it) is the most significant drawback among systems using user-determined fitness functions since the speed of human evaluation is by far the slowest component in the overall process.

Neural Fitness Functions

In a similar manner to the Automated Fitness Rater described in section 3.1.2 other examples can be found in the literature which aim to to combat the bottleneck. The idea is to train Artificial Neural Networks to participate either in place of or as part of fitness functions. In work carried out by Gibson and Byrne (Gibson & Byrne, 1991), the authors used an ANN to classify 4-beat rhythms using examples from pre-classified training examples. Biles (Biles *et al.* , 1996) later attempted an admittedly less successful system using an ANN as a replacement for the user.

Conclusions

One of Burton and Vladimirova's main conclusions (which remains valid today) is that the fitness bottleneck involved with interactive, genetic music

composition systems creates the most significant drawback, mostly due to the sequential nature of fitness evaluation. The high degree of subjectivity involved with musical taste leads to difficulties with the realization of a general-purpose, please-all music creation system. Bypassing the bottleneck via neural networks appears to be only as effective as the training set used is representative.

A Note on Training Data

In many of the examples that have been cited so far, most training sets used come from what experts consider to be "good" and "bad" rather than what an average person would consider to be good, bad or otherwise. One of the contributions of the work presented in this research is a method for learning users musical tastes from their own input, rather than assuming that they reside in some training set of popular examples agreed upon by music domain experts.

3.2.1 Artificial Art Critics

Recent work by Machado *et al.* (Machado *et al.* , 2004) has laid the claim that artificial artists must first be able to perform simple aesthetic judgements (as this is a fundamental task among human artists) before they can be used to guide the evolutionary process. In earlier work (Machado *et al.* , 2003), a general framework is proposed for the development of Artificial Art Critics (AACs), the first stages of which involve author identification tasks. The authors use a "bottom-up" methodology to give their overall system a solid foundation based on a logical question: if AACs cannot learn to tell the differences in styles between different artists, how could they then be used for more sophisticated tasks? They therefore focus on this task for the musical and visual arts (2D images) domain.

The AACs are made up from two modules, a feature extractor and an evaluator. Depending on the domain, the feature extractor is used to reduce the raw input data (MIDI for music, pixels for images) to a set of useful

measures, which can then be used as input to the evaluator. The evaluator is made up from a multi-layer perceptron (MLP), which, when trained on a subset of input features, can learn to distinguish between authors (composers / artists) with a high success rate.

Feature Extraction

In work carried out by Manaris *et al.* (Manaris *et al.* , 2003), it was found that a particular mathematical / socio-economical distribution described by Zipf's Law (Zipf, 1949) describes patterns that can be found in several phenomena including music. By utilising these observations, the authors were able to produce a feature extractor. For example, in the case of the musical domain, a given musical piece can be analysed with a particular metric in mind (such as pitch), which is plotted on a log-log, frequency-rank format. If the slope of the plotted trendline produced is -1, then the pitch metric follows a Zipf distribution. The R^2 goodness-of-fit correlation of the trendline produced can then also be used as an indication of how close the metric is to an actual Zipf distribution. For the musical domain, the authors considered forty of these metrics (with each producing two real-values, slope and goodness-of-fit) and the number of notes in the musical piece being analysed, producing 81 features in total.

For the visual arts domain, the feature extractor used is based on the idea that image complexity is important when assessing images aesthetically. The authors use two types of complexity estimates for their feature extractor: jpeg and fractal compression. The complexity of a given image is calculated as the ratio between root mean square error resulting from its compression and the compression rate. In total, 198 metrics are generated in this manner by partitioning the image and splitting it into hue, saturation and lightness channels. For each channel, they also calculate the average, standard deviation, slope of the trendline of the Zipf distribution and the mean squared error.

Evaluation

The evaluator is the second module of the AAC, this takes the form of a multi-layer perceptron trained on the features extracted from a body of artists and subsequently tested to distinguish between them. The input layer of the network contains as many neurons as there are outputs from the feature extractor (normalised to values between -1 and 1). One hidden layer of hidden neurons was used, with 6 and 12 neurons producing the best results. The number of neurons in the output layer is equal to the number of authors considered in the test (except for Bach vs. the rest). The network was trained using standard backpropagation using a learning rate of 0.2 and the logistic function as the activation function.

Results

Results from the musical domain showed that the evaluator could very successfully distinguish between composers from the same and from different musical periods. The study used a total of 741 scores from the composers Bach, Chopin, Debussy, Purcell and Scarlatti. Interestingly, better results were obtained when using lower values for the number of training cycles. An attempt was also made to reduce the number of input neurons by assessing the contribution of each input metric. This is achieved by calculating the sum of the absolute values of the weights between each input neuron and the neurons of the hidden layer. Experiments were then re-run using the top 30 and 15 metrics as inputs. Results obtained indicated that 30 inputs were sufficient for the discrimination between five composers and a further reduction to 15 inputs still maintained a relatively high success rate.

The authors claim that analysis of the overall results from the musical domain reveals their AAC to be coherent with the results that could be expected from a human, although no empirical evidence is given to support this claim.

Results from the visual arts domain were comparable to those from the musical domain, even when the number of inputs was reduced. Interestingly,

the authors observed some increases in testing performance when certain input measures were removed. Overall, the experiments were successful, although (as admitted by the authors) the task represents the first step on a long road to the realisation of true artificial artists.

Subsequent work by the same team (Manaris *et al.* , 2005) involved the incorporation of musical features extracted using Zipf's law for as a basis for the construction of fitness functions that describe pleasant music. Their initial experiment indicates that the notions of pleasantness as expressed by humans can be modelled to some degree using the extracted features. The authors have also proposed the development of a fully automated music creation system which incorporates this functionality.

3.2.2 Machine Learning Subjective Preferences

Although not directly applied to the musical domain (however directly related), some interesting research has been carried out by Machwe and Parmee (Machwe & Parmee, 2006) on the use of machine learning techniques for the discovery of design preferences. In an Interactive Evolutionary system for exploring bridge designs, the authors examined the use of Case Based Reasoning (CBR) for automatically ranking (a subset of) the population of designs at each generation. To measure the effectiveness of the CBR system, the authors measured the amount of adjustments that the (human) user made to the machine-produced rankings. Results obtained from this study showed that the number of user-adjustments decreased over the course of the run, which is a good indication of how well the machine learning system is performing its task.

These results show lots of promise. However it is unclear from the study how many human users were used to produce the results. Also, if the goal is to produce a model that can effectively stand in for the human user, it would be instructive to attain a measure of the predictive power of the approach. This could be achieved by halting learning at a specific generation and then measuring the accuracy of the generated model on subsequent generations.

3.3 Summary

This chapter has given an introduction to the application of Evolutionary Algorithms in the musical domain. We have seen how the (mostly) necessary human evaluator the biggest obstacle to the otherwise speedy process of artificial evolution. In a recent paper by Jon McCormack (McCormack, 2005), this forms the basis for an open problem in evolutionary music and art, namely "The Problem of Aesthetic Selection". The problem is reprinted here as follows:

> To devise formalised fitness functions that are capable of measuring human aesthetic properties of phenotypes. These functions must be machine representable and practically computable.

This task of solving this problem remains the primary focus of the research described in this thesis.

Chapter 4

Proof of Concept

The previous chapters have introduced the concepts of Artificial Evolution and also described relevant research combining these ideas with those of algorithmic composition and other areas within the field of Computer Music research. The problem of *aesthetic selection* (as described at the end of Chapter 3) remains an important open question in the field of Evolutionary Music and Art.

4.1 Hypothesis and Goals

The goal of the research described in this chapter is to show that the answer to this question may also be found in Artificial Evolution. It is the intention to show that the task of modelling the subjective fitness functions of human users is achievable. Moreover, Genetic Programming is a suitable tool for this task.

Automatically discovering subjective human fitness functions in an Interactive Evolutionary context first requires a suitable problem of study. To best perform such a study, a system with the following components is necessary:

1. Candidate solutions suitable for subjective evaluation
2. A solution structure that can be accurately represented in an Evolutionary Algorithm

3. An interactive mechanism for human-allocated fitness assignment

4. A descriptive record of the decisions made by a human user

With these in place, the solution structures together with the choices made over the course of an Interactive Evolutionary run may be retained for subsequent analysis.

This chapter describes the implementation of two evolutionary systems for producing pleasing musical compositions. In both cases, the systems employ a Genetic Algorithm to produce the musical segment which undergoes evaluation by a human user. Genetic Programming is then used to analyse the preferences made by the human users, using data gathered during the course of the evolutionary runs of the system.

As seen in Chapter 3, the use of GAs for musical composition is by no means new (Biles, 1994; Johanson & Poli, 1998; Burton & Vladimirova, 1999). What is novel about the research described in this chapter is the further use of artificial evolution (GP) to produce *models* of human preferences used during the GA evolution.

4.2 Evolving Rhythms: The DrumGA

The problem of evolving pleasing drum rhythms was chosen due to the fact that simple rhythms are relatively easy to represent using a Genetic Algorithm, yet there can also exist a large amount of variety (and thus quality) in the candidate solutions produced.

The *DrumGA* system, a java-based Interactive Evolutionary drum rhythm generator was created to meet the requirements set out at the start of this section (a screen-shot is shown in Figure 4.1).

The DrumGA breeds a population of simple drum patterns towards fitter states through the use of a human-guided fitness function. An initial population of 12 drum patterns is created and presented to the user, who then listens to each pattern and allocates a score, particular to his or her taste.

Once all individuals have been rated, the user proceeds to the next generation and repeats the process with an evolved set of patterns. This process continues until the user is satisfied with the quality of the patterns created.

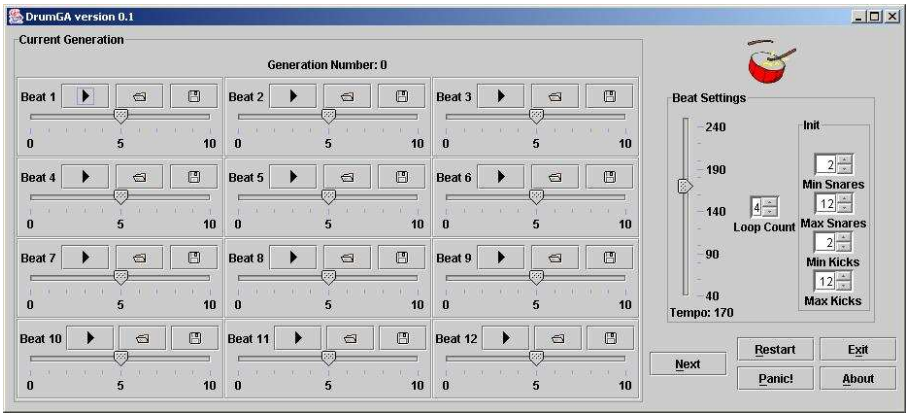


Figure 4.1: A screen-shot of the DrumGA system, which was developed for the user-guided evolution of simple drum rhythms. An evolving population of 12 candidates is displayed as a palette on the screen, each with its own playback and fitness allocation controls. A user listens to a candidate drum rhythm by clicking the play button and allocates a fitness score using the slider. Once all individuals have been evaluated, the user clicks the next button to proceed to the next generation. Initialisation settings are also shown on the right hand side of the screen.

4.2.1 Representation and Initialisation

The mechanism behind the DrumGA is a simple genetic algorithm (Goldberg, 1989) using fitness proportionate selection, 1-point crossover and bitwise mutation. Each drum pattern is represented as an 80 bit binary string, which is transformed into a sequence of MIDI events ((MMA), 1996) that can be sounded using a drum machine or software synthesizer[1]. An example drum pattern created from a bitstring is shown in Figure 4.2; each of the five 16-bit segments making up an individual are transformed into 16 "ticks" of a drum pattern for a specific instrument. Each set of 16 ticks can be thought of as

[1]The synthesizers used here were the deluxe soundbank provided with the Java Runtime Environment, and a Roland TD8 Sound Percussion Module.

a bar in musical terms; by default, four bars (iterations) of each pattern are played back to the user for evaluation.

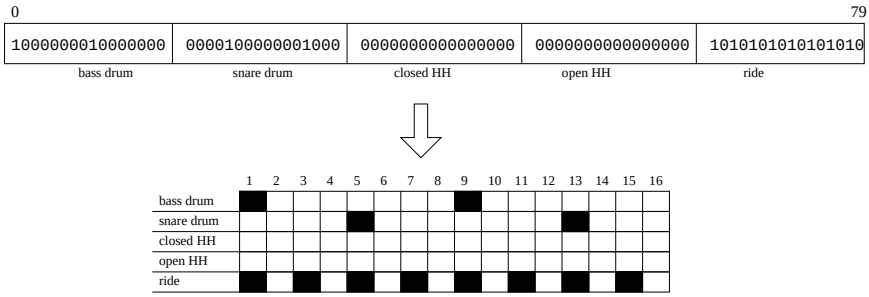


Figure 4.2: Individuals are broken into five 16-bit segments and transformed to midi events that can be sounded by the synthesizer. The lower part of the figure shows a "piano-roll" representation of a simple drum pattern. This representation shows how musical notes (or drum sounds, in this case) are sounded over a time period. Users of the DrumGA have the option to introduce their own changes to evolving patterns through a piano-roll interface.

The initialisation process imposes a certain degree of order on each member of the population. In the initial generation, each pattern has:

- either ride cymbals or closed hi-hats on every tick or every second tick
- open hi-hat cymbals only sounding before snare drum sounds
- all other events randomly created

With most drum patterns across a vast range of styles of popular music, the defining characteristics tend to centre on the placement of the snare, bass drum and (to a lesser extent) open hi-hat sounds, while the closed hi-hat and ride cymbals are usually used to give patterns a continuous feel. The purpose of the initialisation used here is to jump-start the evolutionary process so that all patterns start out with this continuous feel built in. It can also be viewed as a time saving measure; with all events created at random, it can take a significantly larger number of evaluations to get patterns to sound like actual drum beats. After initialisation, however, evolution is free

to take patterns in any direction, guided by the user, so if irregular patterns are preferred, they can easily come about.

Fatigued Distraction

With human users acting as fitness functions in an Interactive Evolutionary system, over time, the process becomes increasingly tiresome. This makes the task of judging individuals more and more difficult, which can then lead to inconsistent decisions on fitness. We term this state "fatigued distraction" and identify it as a potential barrier to the process of creating accurate models of human decisions. Against this effect and in the context of experimentation described later in this chapter, time is costly and the best possible use of it must be made. Jump-starting the initialisation process in the manner described above is therefore argued to be a worthwhile defence against this seemingly inevitable human condition.

4.2.2 Clone Replacement

Early experiments with the DrumGA settings indicated that the system was suffering from premature convergence, as the population quickly became saturated with a single pattern, or a set of almost identical patterns. In an effort to deter this from happening, and to give users more freedom to explore the space of possible patterns, the DrumGA was extended to incorporate a clone-replacement strategy. After selection, crossover and mutation, each pattern in the population is assigned a checksum value based on the positions and counts of the events it contains. Individuals with identical checksums are then tested for equality and clones removed, for example if a group of n identical individuals is found within the population, $n - 1$ of these are replaced with freshly created patterns (using the initialisation rules) before being presented to the user as part of the next generation.

4.2.3 Data Collection

As users proceed from one generation to the next, a record of the genetic makeup of each pattern and the score allocated to it is kept in a history file. Later we will see how the history data from users is used as input to a GP system for symbolic regression.

4.3 Experimental Setup

This section describes how data from user runs of the DrumGA was collected from human evaluators and subsequently analysed using GP.

4.3.1 Obtaining the data

Unlike many experiments involving Evolutionary Algorithms where a batch of runs can be set off on a computer and left unsupervised until they are completed with the results neatly tabulated, the experiments carried out here relied on the input of human evaluators. Since no well established dataset for this type of study was publicly available, it was necessary to gather a dataset for experimental purposes. Nineteen subjects[2] were each given one hour in which to carry out three tasks: complete a questionnaire, perform a Triangle (odd-one-out) test and finally, perform the DrumGA run. Note that in many studies involving human subjects and computers, typical figures for the number of participants are quite low and it is often acceptable to use four or five subjects for experimental purposes (von Mayrhauser & Vans, 1995).

The questionnaire asked for the following details from the subjects:

- age and gender
- if they would consider their hearing to be normal
- if they are musicians, if so, what level (novice, intermediate, expert)

[2]All subjects were graduate students from either a Music Technology or Interactive Media course, and paid for their time rather than the quantity and/or quality of data they produced.

Table 4.1: Summary of the Triangle test results and amount of DrumGA history data gathered from 19 subjects.

Subject	**Triangle Test #evals, score**	**DrumGA #gens**
1	6, 1.0	9
2	7, 0.86	7
3	7, 1.0	11
4	9, 1.0	13
5	10, 0.78	12
6	7, 1.0	13
7	7, 0.71	6
8	7, 0.71	9
9	6, 0.67	12
10	7, 0.71	12
11	8, 0.75	8
12	7, 0.57	12
13	7, 0.57	10
14	7, 0.57	12
15	0, ?	16
16	0, ?	12
17	8, 0.5	16
18	3, 0.67	19
19	6, 0.5	10

- level of knowledge of music theory (on a scale of 1–5)
- favourite radio station
- preferred style of music

The purpose of the Triangle[3] test was to establish each subject's ability to tell the difference between similar patterns. For each trial of the test, the subject listened to three drum patterns, two identical and one slightly different and was then asked to choose the odd-one-out. The odd-one-out in each trial of this test differs to the other two by one bit in each 16-bit segment. All subjects were given the same set of patterns for this test, although the location of the odd-one-out was different for each user to prevent subjects from collaborating on the answers. Subjects were given approximately 10 minutes to carry out a maximum of ten trials.

With the remaining time in the session, subjects were then presented with the DrumGA and asked to perform one run of as many generations as was comfortably possible. A summary of the data collected from the above experiments is given in Table 4.1. It is worth noting that working with human subjects almost always means that something unforeseen will happen. For example, it was hoped that all subjects would carry out the Triangle test, and evaluate fifteen generations worth of drum patterns at the very least, but, unfortunately, two subjects were either unable or unwilling to do the test, and very few managed to get past fifteen generations.

4.3.2 Analysing the data

The results from the Triangle test were used to split the subjects into four groups as follows:

- Those that performed 6 or more evaluations and achieved a score greater than 0.75 (Group 1)

[3]Similarly to that carried out in (Truong, 2002)

- Those that performed 6 or more evaluations and scored between 0.55 and 0.75 (Group 2)
- Those whose results from the test were unknown (Group 3)
- Others, who either scored less than 0.55 or performed less than 6 evaluations (Group 4)

The following experiments will only deal with members of groups 1 and 2 as these are the subjects that it is felt are most likely to give consistent results. Furthermore, we know the least about groups 3 and 4, making it difficult to draw conclusions about how GP fares at learning their functions.

Analysis of the history data gathered from each subject focused on the last four generations; of these four, the first three generations were used for training with the final generation used for testing. Tests were performed on both raw and filtered history data.

The raw data was created by taking each boolean value from the 80 bit segment representing a pattern and converting each bit into a numerical value (0.0 or 1.0). For training, 36 cases of 80 variables and 1 target were used. Testing data (the final generation) consisted of 12 cases.

Domain-specific filters were designed and applied to the history files to reduce the number of input variables to 7 using:

- Average distance between closed hi-hat events (x_1).
- Average distance between ride cymbal events (x_2)
- Number of bass drum events (x_3)
- Distance between first and last bass drum events (x_4)
- Number of snare drum events (x_5)
- Distance between first and last snare drum events (x_6)

- Proportion of "correct" open hi-hat events[4] (x_7)

Variables x_1 and x_2 are used to give an indication of the level of continuity within a pattern, while variables x_3 through x_6 focus the bass and snare drum events, which we regard as the defining characteristics of the patterns. The number of such events coupled with the distance between first and last events helps to give an idea of the level of clutter within each pattern. The last variable gives a measure of correctness of the open hi-hat cymbal events; in general these events immediately precede snare drum events across a variety of music styles.

As with the raw data, filtered training data consisted of 36 cases with 12 cases used for testing.

Symbolic Regression

Due to the subjective nature of human fitness functions, we can make no assumptions about their functional form. Symbolic Regression with GP seems an appropriate data-mining technique for the task, as it allows a flexible interaction between the input variables.

The experiments carried out used a fast GP system for symbolic regression that uses interval arithmetic and linear scaling (Keijzer, 2003). The system employs a steady state algorithm, tournament selection, subtree crossover and node and branch mutation. Replacement is carried out via an inverse tournament. For each dataset, 30 runs were carried out using varying numbers of generations from 10 to 200. Training performance is calculated using Mean Squared Error:

$$MSE(t,p) = \frac{1}{N}\sum_{i}^{N}(t-p)^2 \tag{4.1}$$

Where t and p are vectors of actual targets and predicted values respectively. A summary of the experimental settings is given in Table 4.2.

[4]A correct open hi-hat event is judged (by this function) to be one which occurs one tick before a snare drum event, as is commonly found in most drum patterns.

Table 4.2: Experimental settings used for Symbolic Regression analysis of the data gathered during the DrumGA experiments.

Generations	10 – 200
Crossover rate	1.0
Mutation rate	1.0 (either a single node or branch)
Tournament size	3
Function set 1	$\{+, -, *, /\}$
Function set 2	$\{+, -, *, /, x^2, \sqrt{x}, exp(x), ln(x), sin(x), cos(x)\}$
Terminal set 1	Raw inputs and ERC, $\{x_1, .. , x_{80}, \Re\}$
Terminal set 2	Filtered inputs and ERC, $\{x_1, .. , x_7, \Re\}$
Raw fitness	MSE(targets, predictions)

Testing Performance

After each run on the training data the best of run is applied to the testing data. The actual fitness score, t is classified, as either bad ($0 \leq t < 3$), okay ($3 \leq t < 7$) or good ($7 \leq t \leq 10$) as is the predicted score, p. The performance measure, S, is then calculated as the proportion of successful classifications.

For each data set, two variations of parametric settings were applied. The first used a restricted function set consisting of $\{+, -, *, /\}$ only, and the second used a larger set of functions, $\{+, -, *, /, x^2, \sqrt{x}, exp(x), ln(x), sin(x), cos(x)\}$. Crossover was applied at a rate 100% and children created always underwent either node or branch mutation.

Random Choice

As a baseline for establishing if the GP-generated models are doing more than simply guessing randomly, the results that follow compare the GP scores with those that would be expected by a simple *Random Choice* method. Based on the ranges of values defining bad, okay and good, the probability that a random guess method matches the actual target value as chosen by the user is approximately 0.339. This value is used in a Null Hypothesis Significance Test (with alpha level 0.05) to indicate if the performance of a GP-generated

model is significantly better than a wild guess.

4.4 Results

Results from the symbolic regression experiments on the raw data are shown in Tables 4.3 and 4.4, while Tables 4.5 and 4.6 show the corresponding results from experiments on the filtered data.

Overall, we can see that GP does appear to have the capability to learn the fitness functions, although there is room for improvement in certain cases. All tables show that the GP results are significant except in three out of the fifty six cases shown, which is a very positive result.

Statistical tests comparing the use of raw and filtered data reveals no significant difference between the two sets of results. A further comparison between the experiments using a restricted and larger function set also show no significant difference.

4.4.1 Conclusions

The results show that GP appears capable of performing meaningful symbolic regression on real and extremely noisy data. This is pleasing, if somewhat surprising; however the results also show cases where the GP-induced models perform quite poorly. This is especially disappointing in cases where the history data is taken from a subject with a high score in the odd-one-out test.

Nevertheless, the GP scores do compare favourably with those of the Random Choice predictor, suggesting that GP is doing more than simply taking a blind guess at the DrumGA user's choice.

What can be said for certain about the results above is that there is definitely room for more experimentation; we have cases where good models of human preferences are generated by GP, which is an excellent basis for further research. If it were the case that all models produced very poor results, it would be safe to close off this avenue of investigation. Given that

Table 4.3: Results from symbolic regression on the raw data with function set {+,-,*,/}. The mean test performance and standard error (calculated from 30 runs) achieved on each subject's history file sampled at 10, 20, 50, 100, 150 and 200 generations is shown. The table shows the best score found over all runs and also reports whether the best GP result achieved is significantly superior to that which would be expected from a method taking random guesses.

	Generation												Best	Significant	P-value
	10		20		50		100		150		200				
	Mean	Std Err	Mean	Std Err	Mean	Std Err	Mean	Std Err	Mean	Std Err	Mean	Std Err	(30 runs)		
01	0.6917	0.0175	0.7139	0.0190	0.6722	0.0203	0.6528	0.0212	0.6139	0.0198	0.6139	0.0209	0.9167	YES	2.47E-018
02	0.7278	0.0149	0.6861	0.0186	0.5944	0.0298	0.5444	0.0225	0.5306	0.0238	0.5194	0.0225	0.8333	YES	1.12E-021
03	0.4556	0.0125	0.4389	0.0207	0.4250	0.0197	0.3833	0.0194	0.3389	0.0215	0.3556	0.0174	0.5833	YES	2.96E-010
04	0.5389	0.0182	0.5361	0.0142	0.5389	0.0158	0.5528	0.0185	0.5528	0.0181	0.5556	0.0171	0.7500	YES	2.45E-013
05	0.5417	0.0168	0.4944	0.0183	0.4250	0.0205	0.4167	0.0230	0.4000	0.0227	0.4000	0.0205	0.7500	YES	8.36E-013
06	0.6972	0.0123	0.7250	0.0161	0.8028	0.0176	0.7583	0.0162	0.7750	0.0127	0.7778	0.0135	1.0000	YES	8.79E-022
07	0.6250	0.0164	0.6472	0.0173	0.6139	0.0206	0.6139	0.0209	0.6333	0.0242	0.6278	0.0210	0.8333	YES	3.51E-017
08	0.4306	0.0288	0.4417	0.0250	0.4389	0.0219	0.4694	0.0213	0.4833	0.0231	0.4583	0.0236	0.7500	YES	8.03E-007
09	0.6417	0.0150	0.6222	0.0191	0.6500	0.0213	0.6667	0.0165	0.6472	0.0137	0.6306	0.0199	0.9167	YES	1.91E-018
10	0.7361	0.0223	0.7222	0.0185	0.6778	0.0229	0.6306	0.0214	0.6167	0.0242	0.6194	0.0232	0.9167	YES	3.62E-017
11	0.9139	0.0028	0.9000	0.0062	0.9028	0.0058	0.8778	0.0087	0.8694	0.0086	0.8806	0.0077	0.9167	YES	1.61E-047
12	0.5083	0.0241	0.5194	0.0221	0.5222	0.0207	0.5111	0.0222	0.5111	0.0222	0.5056	0.0233	0.7500	YES	1.01E-009
13	0.3306	0.0217	0.2944	0.0191	0.2750	0.0196	0.2972	0.0255	0.2944	0.0235	0.2722	0.0211	0.6667	NO	7.00E-001
14	0.4000	0.0185	0.3861	0.0251	0.3194	0.0208	0.2722	0.0236	0.2639	0.0253	0.2389	0.0249	0.5833	YES	2.57E-003

Table 4.4: Results from symbolic regression on the raw data with function set $\{+, -, *, /, x^2, \sqrt{x}, exp(x), ln(x), sin(x), cos(x)\}$. The mean test performance and standard error (calculated from 30 runs) achieved on each subject's history file sampled at 10, 20, 50, 100, 150 and 200 generations is shown. The table shows the best score found over all runs and also reports whether the best GP result achieved is significantly superior to that which would be expected from a method taking random guesses.

	Generation												Best	Significant	P-value
	10		20		50		100		150		200				
	Mean	Std Err	Mean	Std Err	Mean	Std Err	Mean	Std Err	Mean	Std Err	Mean	Std Err	(30 runs)		
01	0.7306	0.0148	0.8028	0.0141	0.7611	0.0125	0.6972	0.0176	0.6750	0.0234	0.6889	0.0164	0.9167	YES	1.70E-024
02	0.7611	0.0111	0.7361	0.0183	0.6222	0.0207	0.5389	0.0199	0.5139	0.0196	0.5000	0.0155	0.8333	YES	2.75E-026
03	0.4194	0.0129	0.3722	0.0163	0.3222	0.0225	0.3028	0.0248	0.2806	0.0176	0.2917	0.0225	0.5833	YES	8.77E-007
04	0.4806	0.0124	0.4889	0.0104	0.5528	0.0123	0.5750	0.0176	0.5972	0.0145	0.5972	0.0150	0.7500	YES	9.06E-017
05	0.5472	0.0124	0.4972	0.0224	0.4444	0.0209	0.4111	0.0222	0.4000	0.0201	0.3833	0.0228	0.7500	YES	1.90E-016
06	0.6806	0.0070	0.7306	0.0190	0.7417	0.0129	0.7306	0.0111	0.7278	0.0126	0.7111	0.0153	1.0000	YES	6.50E-024
07	0.6194	0.0202	0.6861	0.0182	0.6861	0.0173	0.6750	0.0197	0.6667	0.0196	0.6611	0.0191	0.8333	YES	5.68E-018
08	0.3083	0.0179	0.3250	0.0205	0.4167	0.0233	0.4111	0.0249	0.4139	0.0209	0.4056	0.0207	0.5833	YES	2.35E-003
09	0.6194	0.0137	0.6028	0.0186	0.6333	0.0217	0.6278	0.0182	0.6528	0.0175	0.6472	0.0190	0.9167	YES	2.89E-017
10	0.7556	0.0215	0.6806	0.0183	0.5583	0.0175	0.5806	0.0194	0.6000	0.0217	0.5972	0.0200	1.0000	YES	3.84E-018
11	0.9167	0.0000	0.9139	0.0028	0.8861	0.0085	0.8833	0.0076	0.8694	0.0077	0.8750	0.0077	0.9167	YES	0.00E+000
12	0.5556	0.0330	0.5222	0.0195	0.5333	0.0130	0.5056	0.0211	0.4889	0.0229	0.4833	0.0269	0.8333	YES	3.50E-007
13	0.2889	0.0153	0.2667	0.0152	0.2694	0.0148	0.2917	0.0153	0.3028	0.0198	0.2806	0.0202	0.5000	NO	7.72E-002
14	0.4056	0.0104	0.3694	0.0182	0.3417	0.0224	0.2778	0.0166	0.2639	0.0183	0.2806	0.0206	0.5833	YES	5.07E-007

Table 4.5: Results from symbolic regression on the filtered data with function set {+,-,*,/}. The mean test performance and standard error (calculated from 30 runs) achieved on each subject's history file sampled at 10, 20, 50, 100, 150 and 200 generations is shown. The table shows the best score found over all runs and also reports whether the best GP result achieved is significantly superior to that which would be expected from a method takine random guesses.

	Generation												Best	Significant	P-value
	10		20		50		100		150		200				
	Mean	Std Err	Mean	Std Err	Mean	Std Err	Mean	Std Err	Mean	Std Err	Mean	Std Err	(30 runs)		
01	0.7722	0.0068	0.7944	0.0087	0.7750	0.0081	0.7500	0.0160	0.7556	0.0159	0.7361	0.0155	0.9167	YES	2.81E-030
02	0.8139	0.0086	0.8222	0.0104	0.7694	0.0177	0.7222	0.0244	0.6806	0.0183	0.6611	0.0226	0.9167	YES	8.05E-029
03	0.3972	0.0124	0.3750	0.0159	0.3694	0.0137	0.3528	0.0124	0.3361	0.0167	0.3556	0.0149	0.5833	YES	6.12E-005
04	0.6472	0.0103	0.6250	0.0125	0.6306	0.0142	0.6556	0.0186	0.6472	0.0177	0.6444	0.0154	0.8333	YES	1.29E-016
05	0.5833	0.0000	0.5806	0.0028	0.5333	0.0153	0.4750	0.0200	0.4750	0.0188	0.4500	0.0181	0.6667	YES	0.00E+000
06	0.6722	0.0068	0.7361	0.0133	0.6861	0.0086	0.6583	0.0073	0.6528	0.0099	0.6611	0.0126	0.8333	YES	2.52E-023
07	0.6361	0.0094	0.6083	0.0114	0.6167	0.0124	0.6222	0.0168	0.6139	0.0176	0.5889	0.0195	0.8333	YES	4.43E-024
08	0.3500	0.0193	0.2444	0.0169	0.2139	0.0111	0.2083	0.0096	0.2139	0.0103	0.2194	0.0109	0.5833	NO	5.74E-001
09	0.6083	0.0107	0.6111	0.0140	0.5639	0.0148	0.5333	0.0186	0.5194	0.0131	0.5111	0.0153	0.7500	YES	3.78E-018
10	0.7778	0.0193	0.7667	0.0180	0.7500	0.0155	0.6861	0.0182	0.6361	0.0189	0.6139	0.0172	0.9167	YES	4.96E-020
11	0.8972	0.0095	0.8472	0.0155	0.7500	0.0000	0.7750	0.0091	0.7833	0.0103	0.7972	0.0111	0.9167	YES	1.12E-031
12	0.4806	0.0210	0.4583	0.0118	0.3861	0.0147	0.3806	0.0163	0.3583	0.0165	0.3778	0.0182	0.6667	YES	2.18E-007
13	0.3083	0.0081	0.3194	0.0058	0.3250	0.0061	0.3056	0.0115	0.3111	0.0174	0.3222	0.0125	0.5000	YES	2.98E-002
14	0.3889	0.0176	0.4333	0.0146	0.4361	0.0158	0.4583	0.0178	0.4778	0.0195	0.4917	0.0162	0.7500	YES	2.35E-010

Table 4.6: Results from symbolic regression on the filtered data with function set $\{+, -, *, /, x^2, \sqrt{x}, exp(x), ln(x), sin(x), cos(x)\}$. The mean test performance and standard error (calculated from 30 runs) achieved on each subject's history file sampled at 10, 20, 50, 100, 150 and 200 generations is shown. The table shows the best score found over all runs and also reports whether the GP best result achieved is significantly superior to that which would be expected from a method taking random guesses.

	Generation												Best	Significant	P-value
	10		20		50		100		150		200				
	Mean	Std Err	Mean	Std Err	Mean	Std Err	Mean	Std Err	Mean	Std Err	Mean	Std Err	(30 runs)		
01	0.7722	0.0154	0.7806	0.0102	0.7389	0.0137	0.7056	0.0143	0.6778	0.0195	0.6889	0.0219	0.9167	YES	6.18E-028
02	0.7139	0.0131	0.6500	0.0213	0.5778	0.0219	0.5667	0.0234	0.5250	0.0208	0.4972	0.0181	0.8333	YES	7.58E-023
03	0.4222	0.0132	0.4472	0.0129	0.4278	0.0242	0.4194	0.0176	0.4250	0.0185	0.4056	0.0203	0.6667	YES	3.21E-009
04	0.5722	0.0203	0.5861	0.0206	0.5778	0.0164	0.5611	0.0207	0.5611	0.0164	0.5528	0.0162	0.7500	YES	8.74E-013
05	0.5694	0.0070	0.5139	0.0192	0.4417	0.0184	0.4167	0.0204	0.3806	0.0242	0.3833	0.0225	0.6667	YES	1.71E-024
06	0.6667	0.0000	0.6639	0.0028	0.6694	0.0063	0.6861	0.0077	0.7028	0.0131	0.7194	0.0152	0.9167	YES	3.51E-021
07	0.5361	0.0158	0.4444	0.0140	0.4861	0.0160	0.4917	0.0220	0.4778	0.0178	0.4806	0.0199	0.8333	YES	3.63E-013
08	0.4222	0.0164	0.3417	0.0166	0.2667	0.0157	0.2556	0.0169	0.2722	0.0169	0.2778	0.0166	0.5833	YES	2.14E-005
09	0.5917	0.0073	0.5667	0.0062	0.5417	0.0104	0.5917	0.0135	0.5806	0.0109	0.5722	0.0153	0.7500	YES	9.10E-018
10	0.7250	0.0227	0.7417	0.0224	0.6722	0.0219	0.5806	0.0363	0.5556	0.0318	0.5083	0.0349	0.9167	YES	2.80E-017
11	0.9000	0.0084	0.9111	0.0039	0.8889	0.0115	0.8639	0.0135	0.8556	0.0138	0.8444	0.0143	0.9167	YES	2.56E-043
12	0.5472	0.0153	0.5194	0.0131	0.5500	0.0202	0.4944	0.0229	0.5056	0.0195	0.5028	0.0176	0.7500	YES	2.48E-011
13	0.3417	0.0146	0.3833	0.0194	0.3972	0.0163	0.3917	0.0208	0.3972	0.0168	0.3750	0.0182	0.5833	YES	1.67E-003
14	0.3833	0.0202	0.3972	0.0168	0.3750	0.0148	0.3833	0.0206	0.3778	0.0195	0.3694	0.0199	0.6667	YES	1.67E-003

there is some promise shown, it would be a worthwhile pursuit to try to improve on the findings thus far.

4.5 Musical Evolution: the MelodyGA

Experiments on the DrumGA showed that some merit may exist in using GP for producing models of the musical preferences of human users. Feedback from users who took part in the experiment mentioned that fatigue was experienced by many participants as the experiment progressed, making it difficult to accurately score evolving populations of drum patterns which were starting to sound more and more alike as time went on. In an effort to make the evolving patterns more interesting, and to ultimately lead to more consistent user-evaluations, the DrumGA was updated to incorporate an extra layer consisting of a simple melody.

Similar to the DrumGA, the MelodyGA is an Interactive Genetic Algorithm for producing pleasing two-track melodies consisting of piano and drums. The system starts off by presenting the user with a population of twelve short melodies, each of which must be listened to and interactively evaluated.

Once all members of the population have been scored, the user proceeds to the next generation with an evolved set of melodies. The evolutionary operators selection, crossover and mutation are applied to the population between one generation and the next. The selection operator is used to take the fittest members of the population and place them into a breeding pool. Crossover is then applied to sets of two parents from the breeding pool in order to exchange genetic material between them and pass it on to their offspring. Mutation is then used to introduce small genetic changes to the offspring created via crossover. The process is terminated after a number of generations when the user is satisfied with the quality of one or more of the musical pieces evolved.

In the earlier DrumGA experiments, the subjects were asked to use an

evaluation scale of 0 – 10 when rating individuals. At each generation, the initial value was pre-set to the neutral value of 5 prior to evaluation. The rating scale was reduced to 0 – 7 for the MelodyGA experiments, removing a neutral value and persuading users to make a positive or negative decision about the fitness of each individual. The effect of this slight bias is likely to be exploited by the underlying fitness-proportionate selection process of the Genetic Algorithm. The intent here is to make it easier for the MelodyGA users to notice the consequences of their decisions.

Each musical piece in the evolving population comprises two instruments, piano and drums. The drum set used consists of five instruments: bass drum, snare drum, closed hi-hat, open hi-hat and ride cymbal. The piano instrument used is limited insofar as it can only play one of 48 notes per "tick" ranging from C1 to B4. Pseudo-polyphony can occur within the piano instrument due to overlapping notes (of three note lengths permitted), but no two notes are allowed to be sounded from the same starting tick [5]. An example individual is depicted in Figure 4.3. To aid evaluation, each decoded individual is played back to the user in a loop of four iterations.

As users running the MelodyGA progress from one generation to the next, a record of the genetic makeup of each piece and the fitness score allocated to it is kept in a history file. Later, we will see how this data can be used as input to a Genetic Programming system which attempts to learn the user's subjective fitness functions.

4.6 Experiments

At a European conference on Genetic Programming[6], participants were invited to take part in a song contest where competing entries were evolved by the MelodyGA. Apart from being a fun event in itself, the contest was

[5]This property is enforced throughout the run. If crossover produces an individual which has the option to sound two different notes, the note is chosen randomly from the parents.

[6]EuroGP 2004

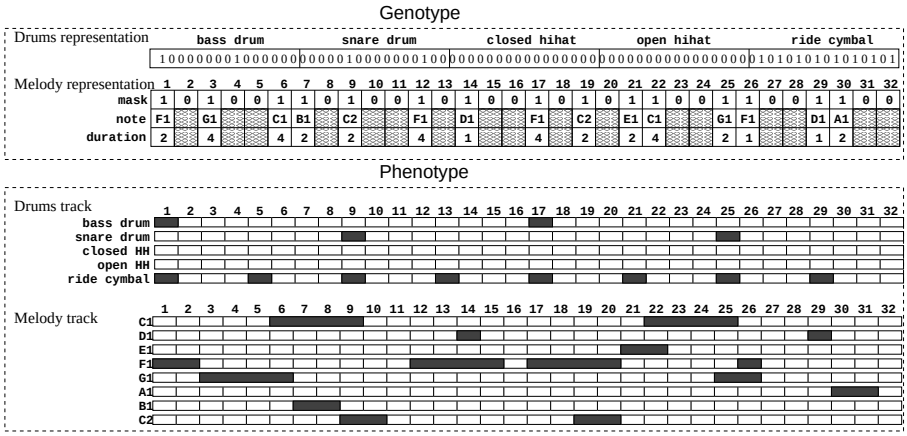


Figure 4.3: Representation of individuals in the MelodyGA. The top half (genotype) shows the genetic makeup of each musical piece as evolved by the system. The bottom half (phenotype) gives a "piano-roll" style representation of the genotype. The drums are represented as an 80-bit binary string where five 16-bit segments are used to represent each drum instrument. This is then scaled up to a 32-tick drum track. The melody instrument is represented by three 32-value arrays: a 32 bit mask which dictates whether a note should be sounded or not, followed by two integer values specifying the pitch and duration of each note (independent of the mask value). For simplicity, note and duration values in the piano representation under a 0 value in the mask are shaded out.

actually an experiment in disguise, the aim of which was to gather data for the research described in this chapter.

4.6.1 Song Contest

The contest consisted of three stages; each participant took part in a Triangle test, followed by approximately one hour of musical evolution. Finally, once all songs were evolved, contestants then voted on each other's musical creations.

The Triangle test followed the same structure as described earlier with the DrumGA experiments in Section 4.2 except that the sound samples used were MelodyGA individuals, containing both drums and melody.

Table 4.11 (left hand side) summarises the results obtained from this part

of the experiment; the second column shows the triangle test score attained and the third column shows the number of generations of musical pieces evolved by each participant. The number of generations evolved plays an important role in the stage that follows; the higher this number, the more data we have at our disposal for analysis with GP.

4.6.2 Symbolic Regression Analysis

In order to apply GP for symbolic regression to the data collected, each subject's history data file is split into two parts, one for training and one for testing. In the training phase, GP uses the examples supplied to build a model that fits the data with minimum error. In the testing phase, the model is then supplied with a set of previously unseen examples and asked to output a set of predictions.
Each subject's history file is analysed in three ways by supplying GP with:

1. **All Data** – this data set contains all of the genotype data, which consisted of 80 variables for the drums part and 96 variables for the piano, giving a total of 176 variables and 1 target;

2. **Melody Only** – this data set contains only the piano part of the genotype, 96 variables and 1 target;

3. **Rhythmic Events** – this set contains the drums part of the genotype and the piano's bit-mask, 112 inputs and 1 target.

The amount of training examples depends on the amount of generations evolved by each subject, if N generations were evolved then $N-1$ of these are used for training; testing is then carried out on the last generation. Training performance is calculated using Mean Squared Error.

To measure testing performance, the actual fitness score, t is classified as either bad ($0 \leq t \leq 2.33$), okay ($2.33 < t \leq 4.66$) or good ($4.66 < t \leq 7$) as is the predicted score, p. The performance measure, S, is then calculated as the percentage of successful classifications.

Using a variation of GP settings and multiple views of the data, a set of experiments were conducted consisting of a total of 14400 runs. The experimental settings used are summarised in Table 4.7.

Table 4.7: GP run settings for the experiments carried out on the data gathered from the MelodyGA participants. Four sets of 50 runs were carried out for each participant, where each set used a different function set. Three views of each participants' data was also considered, giving a total of 14400 runs.

Population Size	1000
Generations	200
Crossover rate	0.7
Mutation rate	0.02
Tournament size	3
Function set 1	{+, -, *, /}
Function set 2	{+, -, *, /, x^2, $\sqrt{x}$}
Function set 3	{+, -, *, /, x^2, $\sqrt{x}$, $\exp(x)$, $\ln(x)$}
Function set 4	{+, -, *, /, x^2, $\sqrt{x}$, $\exp(x)$, $\ln(x)$, $\sin(x)$, $\cos(x)$}
Terminal set 1	{x_1, .. , x_{176}, $\Re$ }
Terminal set 2	{x_1, .. , x_{96}, $\Re$ }
Terminal set 3	{x_1, .. , x_{112}, $\Re$ }
Raw fitness	MSE(targets, predictions)
Standardised fitness	Same as raw fitness for this problem (minimising)
Testing Performance	Proportion of correctly classified predictions

4.7 Results

The results from all runs on the 24 subjects are summarised in Table 4.8, Table 4.9 and Table 4.10. This information is summarised further in Table 4.11 together with extra information including the Triangle test scores and the amount of data available for each subject.

In most cases, GP has been able to construct good models of the functions, with the maximum testing performance score reaching 70% or more in 10 out 24 cases. In many other cases, a trend towards better testing performance is seen as the run progresses despite not reaching the (arbitrarily chosen)

Table 4.8: Summary of the best GP results over all 24 subjects obtained using all input data over the four function sets tested. In the case of each function set, the table shows the mean best GP performance and standard error(50 runs). Also shown is the result of a null hypothesis significance test, showing whether the GP results obtained are significantly better than those that would be expected by a method taking random guesses.

	Function Set 1				Function Set 2				Function Set 3				Function Set 4			
	Mean	Std Err	Signif.	P-Value	Mean	Std Err	Signif.	P-Value	Mean	Std Err	Signif.	P-Value	Mean	Std Err	Signif.	P-Value
01	0.8300	0.0033	YES	2.54E-066	0.8250	0.0060	YES	8.13E-054	0.8300	0.0058	YES	8.13E-054	0.8150	0.0055	YES	3.91E-055
02	0.6850	0.0096	YES	1.26E-036	0.6800	0.0113	YES	4.20E-033	0.6867	0.0091	YES	4.20E-033	0.6600	0.0143	YES	3.28E-027
03	0.7300	0.0127	YES	1.97E-033	0.7100	0.0115	YES	1.92E-034	0.7183	0.0132	YES	1.92E-034	0.6983	0.0141	YES	1.02E-029
04	0.6683	0.0017	YES	1.85E-072	0.6600	0.0052	YES	1.16E-047	0.6667	0.0000	YES	1.16E-047	0.6650	0.0017	YES	3.06E-072
05	0.4700	0.0114	YES	5.37E-015	0.4817	0.0117	YES	7.24E-016	0.4767	0.0147	YES	7.24E-016	0.4733	0.0138	YES	1.45E-012
06	0.4833	0.0130	YES	2.01E-014	0.5083	0.0137	YES	3.51E-016	0.5017	0.0136	YES	3.51E-016	0.5217	0.0168	YES	2.92E-014
07	1.0000	0.0000	YES	0.00E+000	0.9967	0.0023	YES	3.60E-080	0.9983	0.0017	YES	3.60E-080	0.9933	0.0052	YES	7.11E-063
08	0.5200	0.0162	YES	1.26E-014	0.5083	0.0179	YES	2.81E-012	0.4967	0.0190	YES	2.81E-012	0.5000	0.0192	YES	1.17E-010
09	0.4467	0.0146	YES	5.87E-009	0.4117	0.0193	YES	9.23E-004	0.4133	0.0137	YES	9.23E-004	0.3983	0.0189	YES	5.86E-003
10	0.4250	0.0060	YES	2.55E-018	0.4233	0.0062	YES	3.21E-017	0.4283	0.0063	YES	3.21E-017	0.4367	0.0105	YES	1.08E-011
11	0.4317	0.0094	YES	1.96E-012	0.4500	0.0130	YES	1.14E-010	0.4517	0.0117	YES	1.14E-010	0.4483	0.0111	YES	1.57E-012
12	0.6650	0.0017	YES	3.06E-072	0.6600	0.0047	YES	4.26E-050	0.6667	0.0000	YES	4.26E-050	0.6650	0.0017	YES	3.06E-072
13	0.9083	0.0055	YES	4.86E-059	0.9067	0.0045	YES	7.37E-063	0.9050	0.0048	YES	7.37E-063	0.8967	0.0094	YES	3.91E-047
14	0.8350	0.0050	YES	9.50E-058	0.8417	0.0049	YES	1.29E-058	0.8417	0.0055	YES	1.29E-058	0.8517	0.0064	YES	2.49E-053
15	0.2917	0.0165	YES	2.79E-003	0.2700	0.0181	YES	1.64E-004	0.2600	0.0185	YES	1.64E-004	0.2983	0.0169	YES	9.61E-003
16	0.7183	0.0101	YES	1.40E-037	0.7033	0.0072	YES	1.13E-043	0.7033	0.0093	YES	1.13E-043	0.6800	0.0099	YES	1.13E-035
17	0.9167	0.0000	YES	0.00E+000	0.9167	0.0000	YES	0.00E+000	0.9167	0.0034	YES	0.00E+000	0.9167	0.0000	YES	0.00E+000
18	0.4000	0.0117	YES	1.42E-005	0.4117	0.0151	YES	4.37E-005	0.3967	0.0120	YES	4.37E-005	0.3900	0.0113	YES	1.55E-004
19	0.4183	0.0113	YES	2.71E-008	0.4183	0.0017	YES	2.11E-041	0.4167	0.0067	YES	2.11E-041	0.4233	0.0047	YES	3.00E-022
20	0.6850	0.0096	YES	1.26E-036	0.6800	0.0113	YES	4.20E-033	0.6867	0.0091	YES	4.20E-033	0.6600	0.0143	YES	3.28E-027
21	0.6967	0.0091	YES	2.48E-038	0.7033	0.0076	YES	1.36E-042	0.6967	0.0091	YES	1.36E-042	0.6950	0.0074	YES	1.11E-042
22	0.4467	0.0142	YES	2.94E-009	0.4583	0.0131	YES	1.39E-011	0.4533	0.0170	YES	1.39E-011	0.4533	0.0133	YES	8.08E-011
23	0.6383	0.0127	YES	5.07E-028	0.6133	0.0173	YES	1.26E-020	0.6033	0.0162	YES	1.26E-020	0.5867	0.0149	YES	1.78E-021
24	0.5033	0.0186	YES	2.47E-011	0.5250	0.0240	YES	8.88E-010	0.5183	0.0217	YES	8.88E-010	0.5133	0.0209	YES	1.35E-010

Table 4.9: Summary of the best GP results over all 24 subjects obtained using input data restricted to melodic events only over the four function sets tested. In the case of each function set, the table shows the mean best GP performance and standard error(50 runs). Also shown is the result of a null hypothesis significance test, showing whether the GP results obtained are significanty better than those that would be expected by a method taking random guesses.

	Function Set 1				Function Set 2				Function Set 3				Function Set 4			
	Mean	Std Err	Signif.	P-Value	Mean	Std Err	Signif.	P-Value	Mean	Std Err	Signif.	P-Value	Mean	Std Err	Signif.	P-Value
01	0.8150	0.0031	YES	2.54E-066	0.8125	0.0050	YES	8.13E-054	0.8200	0.0050	YES	1.55E-054	0.8100	0.0040	YES	3.91E-055
02	0.6683	0.0120	YES	4.44E-031	0.6133	0.0121	YES	2.57E-027	0.6150	0.0134	YES	2.21E-025	0.6283	0.0120	YES	1.39E-028
03	0.7350	0.0152	YES	3.84E-030	0.7083	0.0124	YES	9.68E-033	0.6867	0.0187	YES	1.33E-023	0.7100	0.0137	YES	7.70E-031
04	0.6633	0.0067	YES	1.12E-042	0.6617	0.0037	YES	4.00E-055	0.6667	0.0000	YES	0.00E+000	0.6650	0.0017	YES	3.06E-072
05	0.4767	0.0079	YES	5.66E-022	0.4850	0.0081	YES	1.46E-022	0.4867	0.0069	YES	6.26E-026	0.4850	0.0074	YES	2.63E-024
06	0.4833	0.0130	YES	2.01E-014	0.5083	0.0137	YES	3.51E-016	0.5017	0.0136	YES	1.04E-015	0.5217	0.0168	YES	2.92E-014
07	0.9983	0.0017	YES	2.24E-087	1.0000	0.0000	YES	0.00E+000	1.0000	0.0000	YES	0.00E+000	0.9983	0.0017	YES	2.24E-087
08	0.6417	0.0153	YES	1.01E-024	0.6467	0.0146	YES	6.28E-026	0.6200	0.0167	YES	1.07E-021	0.6283	0.0147	YES	1.40E-024
09	0.4033	0.0291	NO	4.60E-002	0.4017	0.0103	YES	8.75E-007	0.4067	0.0094	YES	2.06E-008	0.3717	0.0107	NO	1.20E-002
10	0.4233	0.0058	YES	1.52E-018	0.4200	0.0086	YES	8.43E-012	0.4383	0.0075	YES	4.29E-017	0.4350	0.0087	YES	3.73E-014
11	0.4567	0.0093	YES	2.15E-016	0.4383	0.0116	YES	1.16E-010	0.4383	0.0095	YES	2.08E-013	0.4600	0.0093	YES	7.26E-017
12	0.6667	0.0000	YES	0.00E+000	0.6667	0.0000	YES	0.00E+000	0.6683	0.0038	YES	3.23E-055	0.6700	0.0041	YES	1.69E-053
13	0.9083	0.0036	YES	5.00E-068	0.9100	0.0040	YES	1.28E-065	0.9083	0.0049	YES	2.80E-061	0.9150	0.0017	YES	1.76E-084
14	0.8317	0.0017	YES	3.98E-081	0.8333	0.0000	YES	0.00E+000	0.8333	0.0000	YES	0.00E+000	0.8333	0.0000	YES	0.00E+000
15	0.2967	0.0158	YES	4.53E-003	0.2983	0.0143	YES	2.59E-003	0.2967	0.0182	NO	1.25E-002	0.3033	0.0190	NO	3.86E-002
16	0.7467	0.0075	YES	3.55E-045	0.7533	0.0041	YES	2.61E-058	0.7417	0.0099	YES	3.36E-039	0.7233	0.0084	YES	1.30E-041
17	0.9167	0.0000	YES	0.00E+000	0.9167	0.0000	YES	0.00E+000	0.9167	0.0041	YES	2.77E-065	0.9167	0.0000	YES	0.00E+000
18	0.3817	0.0117	YES	2.12E-003	0.4000	0.0114	YES	9.96E-006	0.3900	0.0090	YES	5.27E-006	0.3967	0.0094	YES	8.52E-007
19	0.4250	0.0080	YES	1.13E-013	0.4183	0.0017	YES	2.11E-041	0.4200	0.0033	YES	8.35E-028	0.4250	0.0055	YES	8.05E-020
20	0.6683	0.0120	YES	4.44E-031	0.6133	0.0121	YES	2.57E-027	0.6150	0.0134	YES	2.21E-025	0.6283	0.0120	YES	1.39E-028
21	0.6817	0.0078	YES	9.03E-041	0.7000	0.0079	YES	1.43E-041	0.6950	0.0056	YES	2.56E-048	0.6917	0.0060	YES	5.54E-047
22	0.4250	0.0043	YES	3.53E-024	0.4150	0.0029	YES	3.67E-029	0.4200	0.0106	YES	3.64E-009	0.4167	0.0041	YES	6.57E-023
23	0.6183	0.0156	YES	8.62E-023	0.6017	0.0153	YES	4.91E-022	0.6100	0.0171	YES	1.25E-020	0.5883	0.0169	YES	2.57E-019
24	0.5433	0.0243	YES	9.13E-011	0.4967	0.0219	YES	7.44E-009	0.5317	0.0274	YES	1.15E-008	0.4817	0.0263	YES	3.40E-006

Table 4.10: Summary of the best GP results over all 24 subjects obtained using input data restricted to rhythmic events only over the four function sets tested. In the case of each function set, the table shows the mean best GP performance and standard error(50 runs). Also shown is the result of a null hypothesis significance test, showing whether the GP results obtained are significanty better than those that would be expected by a method taking random guesses.

	Function Set 1				Function Set 2				Function Set 3				Function Set 4			
	Mean	Std Err	Signif.	P-Value	Mean	Std Err	Signif.	P-Value	Mean	Std Err	Signif.	P-Value	Mean	Std Err	Signif.	P-Value
01	0.8300	0.0033	YES	2.54E-066	0.8100	0.0045	YES	8.13E-054	0.8250	0.0050	YES	1.55E-054	0.8200	0.0055	YES	3.91E-055
02	0.7100	0.0096	YES	4.12E-038	0.6950	0.0123	YES	3.14E-032	0.6933	0.0108	YES	8.40E-035	0.7033	0.0090	YES	3.80E-039
03	0.7050	0.0099	YES	3.12E-037	0.7117	0.0083	YES	3.71E-041	0.7067	0.0107	YES	1.15E-035	0.6967	0.0103	YES	7.18E-036
04	0.6667	0.0000	YES	0.00E+000	0.6633	0.0023	YES	5.48E-065	0.6667	0.0000	YES	0.00E+000	0.6667	0.0000	YES	0.00E+000
05	0.4883	0.0137	YES	3.10E-014	0.4850	0.0113	YES	8.51E-017	0.4783	0.0130	YES	6.29E-014	0.4700	0.0095	YES	5.93E-018
06	0.5450	0.0162	YES	9.04E-017	0.5433	0.0157	YES	3.54E-017	0.5317	0.0173	YES	1.24E-014	0.5233	0.0112	YES	3.64E-021
07	1.0000	0.0000	YES	0.00E+000	0.9900	0.0070	YES	1.26E-056	0.9983	0.0017	YES	2.24E-087	0.9983	0.0017	YES	2.24E-087
08	0.5167	0.0147	YES	6.63E-016	0.5250	0.0157	YES	1.26E-015	0.5067	0.0159	YES	9.78E-014	0.5150	0.0128	YES	4.83E-018
09	0.3967	0.0094	YES	8.52E-007	0.4117	0.0044	YES	1.83E-020	0.4167	0.0000	YES	0.00E+000	0.4000	0.0086	YES	3.28E-008
10	0.4467	0.0085	YES	2.48E-016	0.4517	0.0096	YES	2.99E-015	0.4583	0.0124	YES	2.79E-012	0.4517	0.0086	YES	6.83E-017
11	0.4583	0.0147	YES	4.13E-010	0.4767	0.0145	YES	3.27E-012	0.4450	0.0155	YES	3.75E-008	0.4500	0.0158	YES	1.75E-008
12	0.6650	0.0017	YES	3.06E-072	0.6583	0.0049	YES	6.43E-049	0.6633	0.0033	YES	2.02E-057	0.6667	0.0000	YES	0.00E+000
13	0.8900	0.0125	YES	5.89E-041	0.9100	0.0047	YES	2.04E-062	0.9017	0.0081	YES	2.49E-050	0.9067	0.0057	YES	3.22E-058
14	0.8600	0.0065	YES	1.85E-053	0.8583	0.0099	YES	1.51E-044	0.8500	0.0086	YES	3.61E-047	0.8633	0.0103	YES	7.13E-044
15	0.2367	0.0220	YES	1.22E-005	0.1967	0.0176	YES	5.64E-011	0.2467	0.0245	YES	2.41E-004	0.2617	0.0243	YES	1.43E-003
16	0.6750	0.0090	YES	2.17E-037	0.6600	0.0078	YES	2.97E-039	0.6517	0.0113	YES	3.46E-031	0.6633	0.0092	YES	3.72E-036
17	0.9167	0.0024	YES	5.89E-077	0.9167	0.0000	YES	0.00E+000	0.9183	0.0017	YES	1.33E-084	0.9167	0.0000	YES	0.00E+000
18	0.3883	0.0132	YES	1.41E-003	0.3950	0.0114	YES	4.10E-005	0.4033	0.0144	YES	1.32E-004	0.4033	0.0069	YES	1.94E-011
19	0.4200	0.0112	YES	1.20E-008	0.4167	0.0000	YES	0.00E+000	0.4250	0.0112	YES	2.88E-009	0.4183	0.0044	YES	6.15E-022
20	0.7100	0.0096	YES	4.12E-038	0.6950	0.0123	YES	3.14E-032	0.6933	0.0108	YES	8.40E-035	0.7033	0.0090	YES	3.80E-039
21	0.7150	0.0133	YES	8.85E-032	0.7033	0.0124	YES	1.70E-032	0.7250	0.0122	YES	5.06E-034	0.7000	0.0119	YES	3.83E-033
22	0.4783	0.0148	YES	4.58E-012	0.4917	0.0141	YES	4.37E-014	0.4983	0.0102	YES	4.62E-020	0.4883	0.0137	YES	3.10E-014
23	0.5167	0.0161	YES	1.98E-014	0.5150	0.0159	YES	1.68E-014	0.4933	0.0180	YES	5.94E-011	0.4883	0.0168	YES	2.45E-011
24	0.5167	0.0172	YES	1.59E-013	0.5367	0.0191	YES	1.37E-013	0.5450	0.0216	YES	1.96E-012	0.5483	0.0186	YES	7.91E-015

70% threshold. Additionally, almost all of the mean best performance scores attained by GP are statistically significant (as shown in Tables 4.8, 4.9 and 4.10)

An example of where GP has done particularly well is shown in Figure 4.4. In this case we can see better performance results when we include the variables that incorporate piano events; when this information is removed there is a decline in performance, possibly indicating that this subject paid more attention to the piano melody than the drum rhythm when assigning fitness scores to the evolving musical pieces.

Figure 4.5 shows an example of where GP has failed to reach a testing performance score of 70%, although traces of learning are evident when GP is focused on the drum events and the piano bit-mask, perhaps indicating that this subject was more rhythm-oriented.

4.8 Discussion

Although GP has shown some promise in modelling the musical preferences of some of the human subjects who took part in this study, for the most part the results were quite poor. It was hoped that by adding extra melody into the mix, the users would have more interesting musical candidates to evolve and would therefore produce more consistent datasets for analysis.

Unfortunately, if anything, the extra inputs presented to the symbolic regression system made the problem far too difficult in many cases. To check if this shortcoming was limited to GP, the experiments were repeated using an artificial neural network (ANN) as a learner-predictor.

The ANN used was a multi-layer perceptron with one hidden layer consisting of eight neurons and an output layer of three neurons representing bad, okay and good as with the GP experiments. For each subject, thirty runs of the ANN were performed with the data split up into training / test sets in the same way as with GP. In each run, the network was trained using 10000 iterations of back-propagation and then tested on the (unseen) test

Table 4.11: Summary of results from all 24 subjects. This table first reports the Triangle test score achieved by each subject together with the number of generations of the MelodyGA performed. A summary of the best GP results is then given showing whether improvement in GP test performance score is observed over time (GP Learning), and whether GP test performance scores greater than 70%. The Coefficient of Determination value, R^2 is also reported, computed from the Triangle test scores and best GP scores, and also from the MelodyGA generations and the best GP scores.

	Triangle Test score	**MelodyGA Gens**	**GP Learning?**	**GP Best**	**GP $\geq$ 70%**
01	0.60	13	Yes	0.830	Yes
02	0.40	5	Yes	0.710	Yes
03	0.50	8	Yes	0.735	Yes
04	0.90	19	No	0.668	No
05	0.50	9	No	0.488	No
06	0.71	7	Yes	0.545	No
07	0.50	21	Yes	1.000	Yes
08	0.40	16	Yes	0.647	No
09	0.60	5	Yes	0.447	No
10	0.90	13	No	0.458	No
11	0.80	15	Yes	0.477	No
12	0.40	13	Yes	0.670	No
13	0.70	7	No	0.915	Yes
14	0.80	20	Yes	0.860	Yes
15	0.60	26	Yes	0.303	No
16	0.80	8	Yes	0.753	Yes
17	0.00	25	Yes	0.918	Yes
18	0.20	6	No	0.412	No
19	0.70	6	No	0.425	No
20	0.50	30	Yes	0.710	Yes
21	0.80	9	Yes	0.725	Yes
22	0.40	12	Yes	0.498	No
23	0.30	6	Yes	0.638	No
24	0.30	9	Yes	0.545	No
	Coefficient of Determination	Triangle Test and GP Best, R^2			0.0060
		MelodyGA Gens and GP Best, R^2			0.0552

data. Test performance is reported as the proportion of successful classifications.

Results from these experiments are summarised in Table 4.12. It can be seen from this table that very similar results are obtained to those from the GP experiments in the case of subjects 1, 13, 14 and 16 (for set 1 and 2). This is an interesting finding as it further suggests the existence of patterns within the training sets that lead to the construction of good predictive models[7].

4.9 Conclusions

It can be seen from the results from both the DrumGA and MelodyGA experiments that it has been possible to produce good models of subjective fitness functions using GP. Of course, the process has not been an overwhelming success although the existence of good results is very pleasing.

The significance tests show that the GP models created are doing more than taking wild guesses. The less successful results can not be ignored, however. The following possible explanations for these cases have become apparent:

- the amount of user data collected is not large enough
- there are too many input variables in the input set and not enough fitness cases to learn from
- conflicting examples may exist in the datasets, due to uneven user behaviour
- the GP-based Symbolic Regression system appears to either

    1. converge on a constant, or

[7]It is worth noting that the ANN experiments undertaken here did not undergo the same degree of fine-tuning as those performed using GP, therefore it is not unreasonable to assume that the ANN results could be improved upon to highlight this finding more

Table 4.12: Results summary from experiments using a neural network (ANN) on the song contest participants' data sets. Similarly to the GP experiments, the data sets were split into three; set 1 contains all data, set 2 has the drum information removed and set 3 only contains rhythm information. Cases where a test performance score greater than 70% have been achieved are highlighted in bold. Note that a reminder of the best GP result found is also given in the last column of the table.

	ANN Results			**GP: Best score**
	Set 1	Set 2	Set 3	
01	**0.7609**	**0.7618**	0.6958	0.830
02	0.6629	0.6581	0.4363	0.710
03	0.5563	0.5556	0.5278	0.735
04	0.4659	0.4673	0.4340	0.688
05	0.3118	0.3132	0.3351	0.488
06	0.3468	0.3468	0.4005	0.545
07	0.3875	0.3875	0.3875	1.000
08	0.5081	0.5111	0.5085	0.647
09	0.3021	0.2972	0.3549	0.447
10	0.4123	0.4148	0.3674	0.458
11	0.3722	0.3752	0.3683	0.477
12	0.4146	0.4148	0.3595	0.670
13	**0.7917**	**0.7823**	**0.7641**	0.915
14	**0.7808**	**0.7795**	0.6814	0.860
15	0.2493	0.2500	0.2568	0.303
16	**0.7020**	**0.7008**	0.5603	0.753
17	0.5897	0.5898	0.5818	0.918
18	0.2856	0.2839	0.3878	0.412
19	0.3222	0.3217	0.3028	0.425
20	0.4844	0.4885	0.3892	0.710
21	0.6674	0.6670	0.5861	0.725
22	0.2967	0.2934	0.3869	0.498
23	0.3394	0.3422	0.3789	0.638
24	0.6212	0.6340	0.5764	0.545

2. use a very small amount of input variables in the evolved solutions, resulting in a poor prediction.

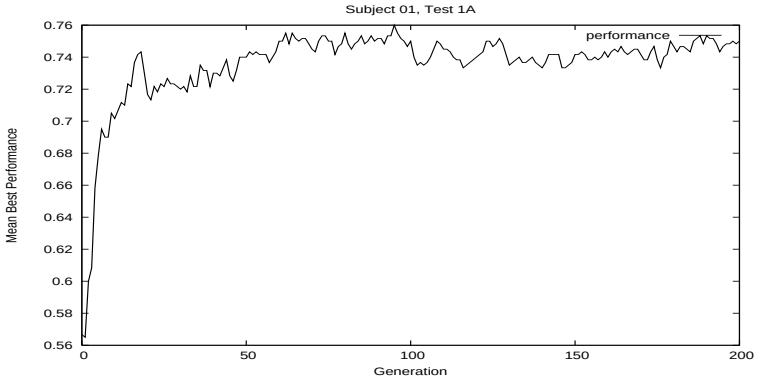
- the problem is too difficult; the fitness landscape is a needle-in-a-haystack problem, with a moving needle. It might be too ambitious to expect *any* learning method to be able to tackle a problem of this magnitude.

- the search-space is far too large to expect any system to arrive at a usable solution.

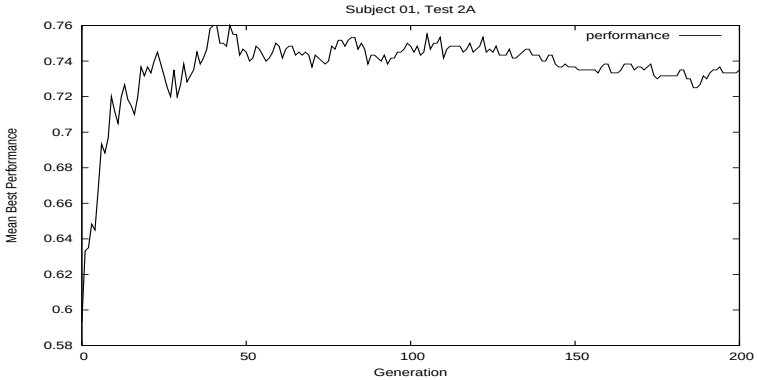
Additionally, the academic background of the subjects that took part in the two studies may have had some influence on the results. For the first set of experiments, all the subjects were pursuing postgraduate courses in either Interactive Media or Computer Music, both of which would involve some degree of musical knowledge. In contrast, most of the subjects who took part in the second experiment would have had a very good understanding of the evolutionary aspect of the application[8]. It is interesting to note that when all of the subjects from the second experiment voted on the best songs produced by the group, both the winning song and the song that was voted second best were actually produced by two of the few subjects who *did not* have an academic background in Evolutionary Computation.

The issues highlighted above will be addressed in more detail in Chapter 5.

[8]So much so that some even tried to break it.



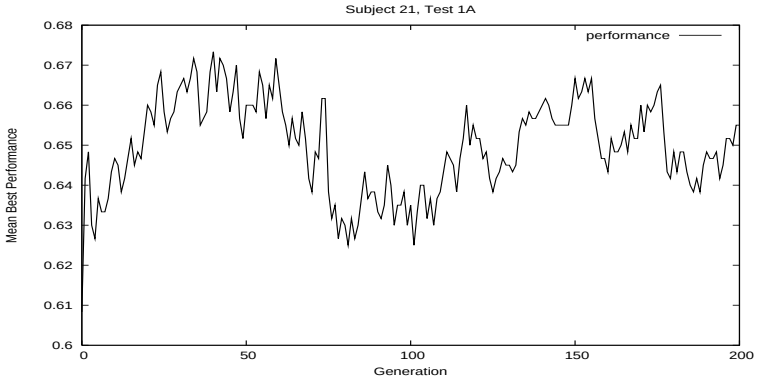
(a) Subject 1, Experiment 1; GP using all 176 inputs



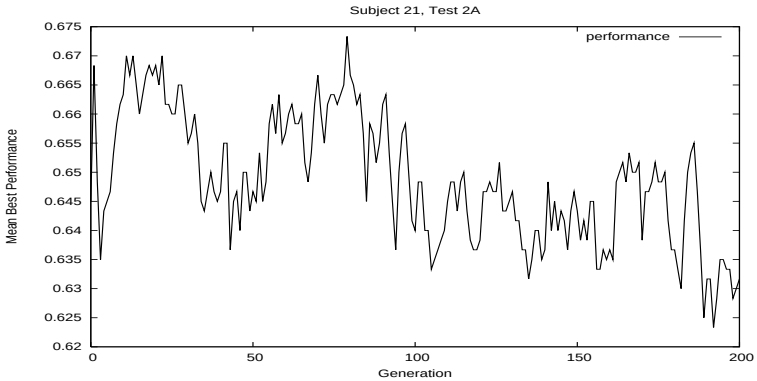
(b) Subject 1, Experiment 2; GP using 96 inputs (drums removed)

(c) Subject 1, Experiment 3; GP using 112 inputs (drums and piano bit-mask)

Figure 4.4: Testing performance results achieved on 3 views of the data collected from subject 1; In the first plot, GP is given access to all of the genotype variables comprising a MelodyGA music piece. In the middle plot, only information about the piano events is given. In the last plot, GP is given the drum events and information about whether a piano note is played or not (regardless of its pitch and duration). Each plot shows the mean performance of the best individual, across 50 runs.



(a) Subject 21, Experiment 1; GP using all 176 inputs



(b) Subject 21, Experiment 2; GP using 96 inputs (drums removed)

(c) Subject 21, Experiment 3; GP using 112 inputs (drums and piano bit-mask)

Figure 4.5: Testing performance results achieved on 3 views of the data collected from subject 21; In the first plot, GP is given access to all of the genotype variables comprising a MelodyGA music piece. In the middle plot, only information about the piano events is given. In the last plot, GP is given the drum events and information about whether a piano note is played or not (regardless of its pitch and duration). Each plot shows the mean performance of the best individual, across 50 runs.

Chapter 5

New Directions

Experience from the results obtained so far has shown that in certain situations, it is possible to use GP to produce a model from data that can make good predictions about future events. This is supported by corresponding results in cases where GP is substituted with a Neural Network. From this it is apparent that with the right conditions in place, learning from the data is possible. The research presented in this chapter is founded on the following hypothesis:

> *With more consistent user behaviour, modelling user preferences and predicting future decisions becomes an achievable task.*

The goals are therefore summarised as follows:

- To re-examine the DrumGA and MelodyGA systems to identify areas that are leading to noisy, inconsistent behaviour.
- To show the results that can be expected when noise and inconsistent human behaviour is not an issue.
- To perform a new set of experiments based on lessons learned from past experience.
- To identify the conditions under which successful artificial models can be created.

Re-examination of previous systems leads to the "bottom-up" construction of a new system which also aims to build models of human musical preferences with high predictive power. It will be shown here how Grammatical Evolution (Ryan *et al.* , 1998) can be used to effectively construct models from artificial datasets which mimic real-world user-generated history data. These models will ultimately substitute for the subjective fitness functions that human users employ during Interactive Evolution of pleasing musical melodies.

5.1 Reducing Noise, Improving Consistency

Recall that in Section 4.2, we examined the use of Genetic Programming for modelling the preferences of human users of an evolutionary drummer, the DrumGA. This system allowed drum sequences to be evolved towards fitter states through human guidance, while keeping records of the scores allocated to each individual for subsequent analysis with GP. The GP performed symbolic regression using the raw genetic data combined with the user-allocated score as inputs, with the aim of producing a function of the form:

$$f(sequence) = score$$

To address this, we decided to make the candidate solutions more interesting by including a melody track into the mixture (Section 4.5). A second set of experiments was then initiated using the MelodyGA. The data collected was fed through the symbolic regression system (as before with the DrumGA experiments) however, despite having this extra melody information, the results showed few signs of improvement in general, although some successful results came about using data from certain participants. It is likely in some cases, the extra inputs presented to the symbolic regression system made the problem too difficult to expect solutions with consistently good predictive power over all.

5.2 System Simplification

Two areas of concern are identified here as potential contributors to inconsistent, noisy data. The first is the size of the search space of musical candidates. If this is too large, the human user taking the role of the fitness function is less likely to reach a good optimum before "fatigued distraction" is experienced. Be decreasing the size of the search space, there is a better chance of coverage by the user, which (it is assumed) is less of a distraction. The more focused the user, the more consistent (and less noisy) the generated history data is expected to be.

The second is the make-up of the user interface that receives the fitness allocation from the human subject. Presenting too many individuals for parallel evaluation is a likely contributor to fatigue. Reducing the information overload should make things less taxing on the subject which should also reduce noise and make past experience easier to create good predictive models from.

5.2.1 Search Space Reduction

With the DrumGA, each individual is represented by 80 boolean values, therefore 2^{80} possible solutions, which is quite a large number compared to the amount of data gathered from human subjects in the experiments; the examples that we gathered only represented a tiny fraction of the potential cases – and this is without adding melody to the mix. When we do add information about the melody, we are adding 32 new boolean values and 64 (constrained) integer values (32 for the note pitch numbers and 32 for the note durations). We can calculate the number of potential solutions as follows:

$$2^x \times l^{n+d+1}$$

where x represents the number of bits making up the drum sequence, l represents the number of notes in the melody, n is the maximum number of possible note values and d is the number of note durations.

With x=80, l=32, n=12 and d=3 (these values have been used in all MelodyGA experiments to date), the number of possible solutions is approximately 2.53×10^{30}; if it takes an average user of the MelodyGA system one hour to rate 240 individuals according to the format above, then it would take approximately 1.2×10^{24} *years* to evaluate them all. This is clearly infeasible and suggests that it would be a good idea to try and reduce the size of the search space. This is achieved by making the following modifications to the system:

- The drum sequence that an individual uses is taken from a pool of W possibilities
- The melody that an individual uses is taken from a pool of X possibilities
- The tempo is fixed at one of a set of Y values
- The instrument that plays the melody is taken from a set of Z possibilities.

If we set W=32, X=32, Y=32 and Z=32 we have a total search-space size of 1048576 (182.04 days for an average human evaluator to process at the same rate as in the previous example), which seems to be a much more realistic figure to expect a black-box system to be able to build models from.

5.2.2 Tournament-style Evolution

In previous incarnations of the DrumGA and MelodyGA software, users were presented with a palette of candidate solutions, representing each generation, all of which needed to be scored in parallel before proceeding to the next generation. A point taken from the feedback of the subjects who took part in the evaluations was that it can be quite difficult for a user to efficiently and consistently score several songs (12, in our experiments) in parallel. This point is addressed by altering the system so that the user processes two

candidates at a time (see screen-shot in Figure 5.1). It is worth noting here that a similar practice is used by opticians when testing patients for new lenses [1] – reducing the problem to a set of binary decisions is less taxing on the brain, therefore more likely to arrive at the intended solution.

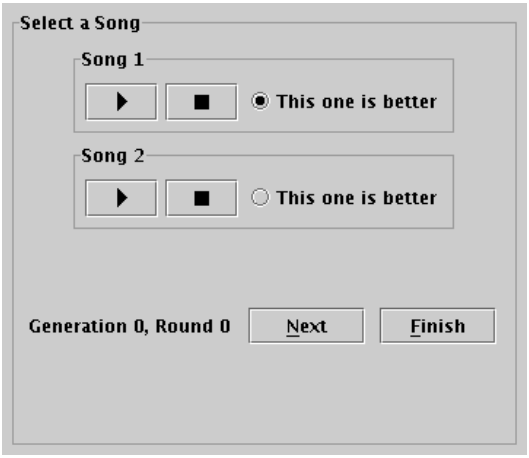


Figure 5.1: Tournament-style GUI for the MelodyGA. Due to the simplified nature of the selection process, the interface itself may be made much easier for the user to operate.

We will therefore use a tournament system for all future experiments, based on the following conjecture:

> A user is better able to say that song x_1 is better/worse than song x_2 as opposed to ranking n songs ($n > 2$) in order of preference, according to an arbitrary scale.

If this is true then it would follow that a tournament-style system would produce a more consistent dataset for subsequent analysis than the generational-style system. Note that we could also use a league system, however the idea is to produce data that is as consistent as possible. It may be the case that expecting the user to make more comparisons (as would be needed by a league) will lead to the same user-fatigue that was exhibited in the generational system, leading to inconsistent data.

[1] This type of test is also referred to as a discrimination test or ABX test.

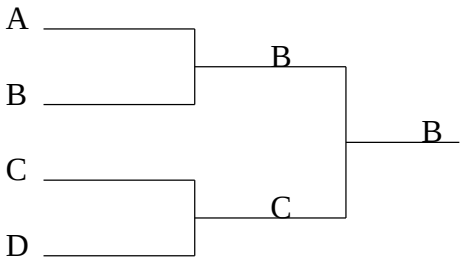


Figure 5.2: A simple tournament situation involving four competing melodies. Rather than allocate a score to a set of competing candidates, the human subject is instead asked to choose a single favourite from two possible choices. The process is repeated until an overall favourite emerges.

For the data analysis, the learning method cannot simply act as a regression system, making future predictions from historical examples because the history data would have a different form. Instead, we alter it to more closely follow the tournament-style path that is taken by the user. Figure 5.2 depicts an example of four songs competing in a user-guided tournament. In this setting, an accurate artificial model would contain a choice function, C_m, such that

$$C_m(x_1, x_2) = C_u(x_1, x_2)$$

where C_u is the choice made by the human user. Simply put, the aim is to discover a function that mimics the user's behaviour consistently throughout the tournament, making the same decisions as those made by the user. The inputs to the metric are the genetic representations of the songs. The metric may be scored by using a comparison with the decisions allocated by the user for each round of the tournament. To use the example shown in Figure 5.2 we could have the scenario given in Table 5.1.

5.3 Choosing the *choice()* function

Now that we know what the choice function should do, the next task is to decide how such a function should be designed. We know that the inputs

Table 5.1: An example comparison of the choices made by a human subject C_u and those made by an artificial model C_m. By matching up actual choices and modelled choices, a fitness function for model accuracy can be easily formulated.

Round	1	1	2
C_u	B	C	B
C_m	B	D	B
Score	1	0	1

to the function are two songs and each song is presented in its raw, genetic representation. For example, lets assume that the genetic makeup of two individuals is as follows:

```
X) [x0, x1, x2, x3]
Y) [y0, y1, y2, y3]
```

Of course, this is a simplified case since an individual would typically be composed of more than 4 components, however the idea remains the same for any number of components (not just with boolean values for Xi and Yi). We can define a grammar for the *choose()* function as follows:

```
choose  ::= f( <inputs> )
inputs  ::= <vars> | <inputs> <op> <vars>
vars    ::= x0 | x1 | x2 | x3 | y1 | y2 | y3 | y4
op      ::= and | or
```

Another issue that needs to be addressed is the format of the historical data from user-runs. Common sense dictates that it is better to have too much data rather than too little since we can trim data but not invent it. At the least, it will be necessary to know the following:

- the complete genetic makeup of each song
- the outcome when one song competes with another (chosen by the user)

For example, if we use the tournament outcome depicted in Figure 5.2, the output of the runs (and input to the black box learning system) might be:

```
A,B,1
C,D,0
B,C,0
```

Where the first and second columns refer to the competitors and the third column refers to the winner of the tournament. Note that in order to produce more data (and to help prevent any bias), the black box learning system could also be reinforced with the same results in the opposite order:

```
B,A,0
D,C,1
C,B,1
```

The goal of the black box learner will be to produce a prediction for each set of competitors. The set of predictions may then be compared against the actual user choices in order to compute the fitness of the predictive model. As hinted at by the use of a grammar above, we will use Grammatical Evolution (O'Neill & Ryan, 2003) as the black box learner in the experiments that follow.

5.4 Experiments

Although it would seem tempting at this point to proceed directly to a set of experiments on live data gathered from human subjects, past experience suggests that the time would be better spent validating this new approach on some controllable artificial problems to investigate if the foundations are secure. If it turns out to be the case that the system cannot produce meaningful results on consistent, artificial data then we cannot expect it to build models of more complex human-generated data (which is costly to gather). With this in mind we have set up our datasets to mimic the following scenarios:

1. The user consistently picks a song that has a specific drumbeat, ignoring all other attributes. This mimics a user who is only interested in the percussion part and is able to filter out melody, instrument and tempo information.

2. The user always picks a song that has a particular drumbeat and melody, which work well together. This mimics a situation where a user is only interested in a particular percussion and melody combination, with no interest in anything else.

3. The user likes a particular drums and melody combination when it is played at a particular tempo, but not at other tempos.

4. The user likes a drums and melody combination played with a particular instrument and tempo.

5. The user is only interested in the percussion sequence but has a set of preferred beats (1, 2, 3, 4) over others. Songs with drumbeat 1 are chosen over all other songs. Songs with drumbeat 2 are chosen over all other songs except those with drumbeat 1. Songs with drumbeat 3 are chosen over all songs except those with drumbeats 1 and 2. Finally, songs with drumbeat 4 are chosen over all other songs except those with drumbeats 1, 2 and 3.

For each problem we generated 100 training cases and 100 test cases and performed 30 runs of Grammatical Evolution using the BNF grammar shown in Figure 5.3. The GE algorithm settings are given in Table 5.2. Note that the `x` in the grammar refers to the array of 8 values which correspond to a competition between two songs. The `musFunc` productions can also be explained by the following example; if `x` contains the values {1,2,3,4,5,6,7,8} then a call to `song1UsesDrumbeat(x, 1)` would return true since the array contains the value of 1 in the position corresponding to the drums preset for the first song. Similarly, a call to `song2UsesDrumbeat(x, 5)` would also return true.

In all problems, the datasets assume the availability of 32 preset drum sequences, melodies, instruments and tempo values. Random noise was also introduced into each dataset whereby a random set of presets is generated for two competitors along with a randomly chosen outcome; each dataset contains 10% noise. The introduction of random noise into the datasets is done so that the problems are not made too easy for the system. Adding noise in this way is also done to introduce very slight inconsistencies into the training set, as could reasonably be expected from real-world data.

```
<choose>    ::= define f(x) { return <body>; }

<body>      ::= <musFunc> | <body> <binary_op> <musFunc>  |
                <unary_op> <body>

<musFunc>   ::= song1UsesDrumBeat(x, <val> ) |
                song1UsesMelody(x, <val> ) |
                song1UsesInstrument(x, <val> ) |
                song1UsesTempo(x, <val> ) |
                song2UsesDrumBeat(x, <val>) |
                song2UsesMelody(x, <val>) |
                song2UsesInstrument(x, <val>) |
                song2UsesTempo(x, <val>)

<val>       ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
                9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
                17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
                25 | 26 | 27 | 28 | 29 | 30 | 31

<binary_op> ::= and | or

<unary_op>  ::= not
```

Figure 5.3: Actual grammar used by the GE experiments on the artificial datasets.

5.4.1 Assessing Performance

As was the case with previous experiments from the DrumGA and MelodyGA, a baseline for assessing the performance of GE here is a Null Hypothesis Significance test that establishes whether a statistically significant difference exists between the scores attained by GE and those that would be expected by a method taking a random guess. An implementation of the *choice()*

Table 5.2: Experimental settings for the GE runs.

Population size	200
Number of generations	100
Crossover rate	0.9
Mutation rate	0.01
Replacement strategy	Steady-state
Other Settings	Sensible initialisation (Ryan & Azad, 2003)
Fitness function	The number of cases where the guesses match the targets (maximising)

function that returns a random, unbiased, result (from two possible values) would have an expected mean performance score of 50%.

5.5 Results and Discussion

This section gives a brief summary of the results from each problem, followed by an overall discussion of the outcome as a whole.

5.5.1 Problem 1

In problem 1, both training and test datasets show candidates with a drum preset value of 1 getting chosen as the winner of the tournament. Each dataset contains examples of the form:

1 15 2 15 25 24 9 25 0

The first set of 4 values refers to the first song and the second 4 to the second song. The final value refers (0 or 1) determines which song is chosen by the human user. The first song uses the drumbeat preset 1, melody preset 15, instrument preset 2 and tempo preset 15. In this fitness case,

the left song was chosen (hence the value of 0 in the last position) since it contains drumbeat 1. To avoid confusion, there are no examples in the datasets where the two songs competing use the same preset. Results from 30 runs are shown in Figure 5.4 and a selection of the best-of-run individuals is given below:

```
define f(x) { return song2UsesDrumBeat(x, 1 ); }

define f(x) { return song2UsesDrumBeat(x, 1 ) or
                     song1UsesInstrument(x, 17 ); }

define f(x) { return not (song1UsesDrumBeat(x, 1 ) or
                          song1UsesTempo(x, 19 )); }

define f(x) { return song2UsesDrumBeat(x, 1 ) or
                     song1UsesDrumBeat(x, 25 ) or
                     song1UsesDrumBeat(x, 27 ); }
```

In fact, every solution produced contains some mention of "UsesDrumBeat(x, 1)", making it clear that GE has picked up on the importance of the use of drumbeat 1. The testing performance scores reported in Figure 5.4 also support this finding.

5.5.2 Problem 2

In problem 2, both training and test datasets show candidates with a drum preset value of 1 and a melody preset of 2 getting chosen as the winner of the tournament. For this problem, each dataset contains examples of the form:

11 15 2 15 **1 2** 9 25 1

Here we can see that the right song was chosen (value of 1 at the end) since it contains drumbeat 1 and melody 2. Results from the second artificial problem are shown in Figure 5.5 and a selection of the best-of-run individuals is given below:

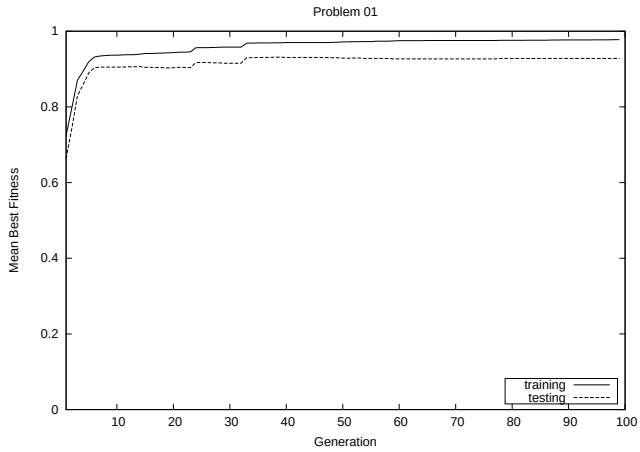


Figure 5.4: Training and Testing performance scores on the first artificial problem. Performance scores are the mean best fitness from 30 runs, showing training and testing performance at each generation. Training was performed on 100 fitness cases and testing was carried out on 100 unseen cases. The scores have been normalised to values between 0 and 1.

```
define f(x) { return not (not (song2UsesDrumBeat(x, 1 ))
                           or song1UsesMelody(x, 2 )); }

define f(x) { return not (song1UsesMelody(x, 2 ))
                     and song2UsesMelody(x, 2 )
                      or song1UsesMelody(x, 6 ); }

define f(x) { return not (not (song2UsesMelody(x, 2 ))
                           or song1UsesMelody(x, 2 )); }

define f(x) { return not (song2UsesDrumBeat(x, 4 )
                       or song1UsesMelody(x, 2 ))
                      and song2UsesDrumBeat(x, 1 )
                       or song1UsesInstrument(x, 28 ); }
```

5.5.3 Problem 3

Results from the third artificial problem are shown in Figure 5.6. A selection of the best-of-run individuals from the experiments is given below:

```
define f(x) { return not (song1UsesMelody(x, 2 )
```

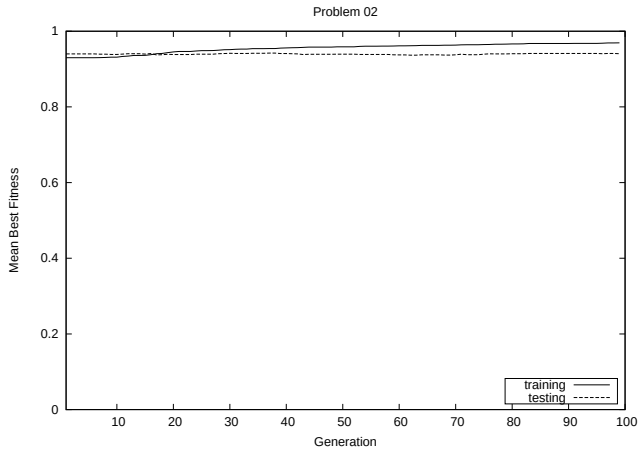


Figure 5.5: Training and Testing performance scores on the second artificial problem. Performance scores are the mean best fitness from 30 runs, showing training and testing performance at each generation. Training was performed on 100 fitness cases and testing was carried out on 100 unseen cases. The scores have been normalised to values between 0 and 1.

```
                    and song1UsesDrumBeat(x, 1 )); }

define f(x) { return song2UsesMelody(x, 31 ) or
                    song2UsesDrumBeat(x, 1 ) and
                    song2UsesTempo(x, 3 ); }

define f(x) { return not (song1UsesTempo(x, 3 )
                    and song1UsesDrumBeat(x, 1 )
                     or song1UsesMelody(x, 26 )); }

define f(x) { return not (not (song2UsesDrumBeat(x, 1 ))
                           or song2UsesMelody(x, 16 )); }
```

In this problem all of the fitness cases take the form:

1 2 3 15 25 24 9 25 0

In the generated solutions, the focus in most cases is on the input combinations that resulted in a successful choice by the artificial user. The notions of using melody 2, drumbeat 1 and tempo 3 show up in a majority of solutions.

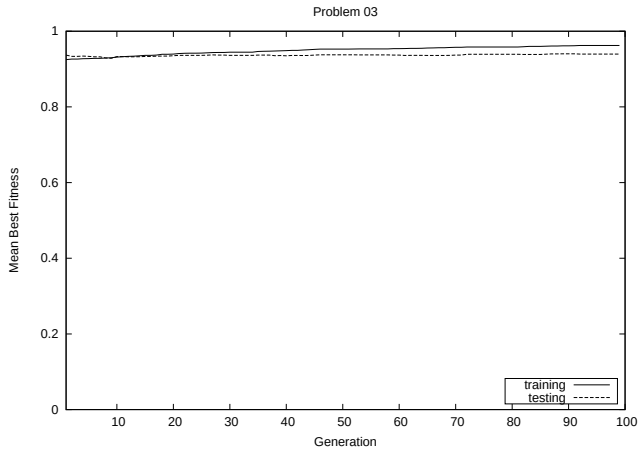


Figure 5.6: Training and Testing performance scores on the third artificial problem. Performance scores are the mean best fitness from 30 runs, showing training and testing performance at each generation. Training was performed on 100 fitness cases and testing was carried out on 100 unseen cases. The scores have been normalised to values between 0 and 1.

5.5.4 Problem 4

Results from the fourth artificial problem are shown in Figure 5.7. A selection of the best-of-run individuals from the experiments on this problem is given below:

```
define f(x) { return song2UsesInstrument(x, 4 ) and
                     song2UsesMelody(x, 2 ) or
                     song1UsesMelody(x, 23 ); }

define f(x) { return song2UsesMelody(x, 2 ) and
                     song2UsesInstrument(x, 4 ); }

define f(x) { return song2UsesMelody(x, 2 ) and
                     song2UsesDrumBeat(x, 1 ) or
                     song1UsesDrumBeat(x, 28 ) or
                     song2UsesTempo(x, 2 ); }

define f(x) { return song2UsesMelody(x, 2 ) or
                     song2UsesTempo(x, 2 ) or
                     song2UsesInstrument(x, 13 ) and
                     song2UsesDrumBeat(x, 13 ); }
```

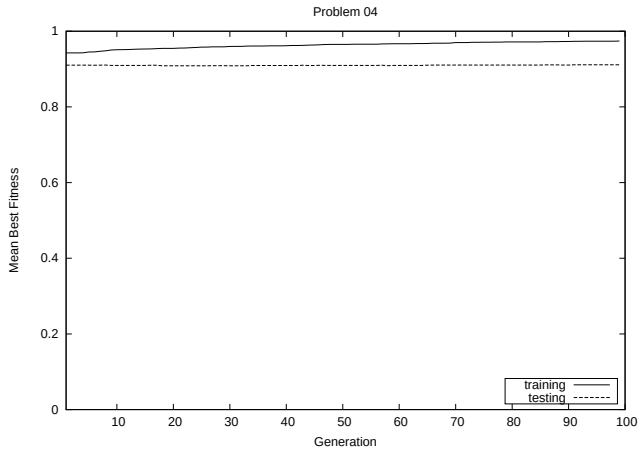


Figure 5.7: Training and Testing performance scores on the fourth artificial problem. Performance scores are the mean best fitness from 30 runs, showing training and testing performance at each generation. Training was performed on 100 fitness cases and testing was carried out on 100 unseen cases. The scores have been normalised to values between 0 and 1.

5.5.5 Problem 5

Results from the fifth artificial problem are shown in Figure 5.8. A selection of the best-of-run individuals from the experiments on this problem is given below:

```
define f(x) { return song2UsesDrumBeat(x, 3 ) and
                     song1UsesDrumBeat(x, 4 ) or
                     song2UsesDrumBeat(x, 2 ) or
                     song2UsesDrumBeat(x, 1 ); }

define f(x) { return song2UsesDrumBeat(x, 1 ) or
                     song1UsesDrumBeat(x, 4 ) or
                     song1UsesTempo(x, 4 ) or
                     song1UsesTempo(x, 21 ) or
                     song1UsesMelody(x, 4 ); }

define f(x) { return song1UsesMelody(x, 1 ) or
                     song2UsesTempo(x, 27 ) or
                     song1UsesDrumBeat(x, 4 ) or
                     song2UsesDrumBeat(x, 1 ) or
                     song1UsesInstrument(x, 3 ); }
```

What is interesting here is that in most of the generated best-of-run

solutions, the musical functions that make up the solutions focus primarily on the "UsesDrumBeat" notion. This is particularly apparent in the first of the best-of-run solution shown above. Furthermore, the solutions that do focus on the "UsesDrumBeat" notion have few inputs other than those in the preferred set ({1, 2, 3, 4}). However, this problem also seems to be one which GE has more difficulty solving, in terms of the generalisation performance.

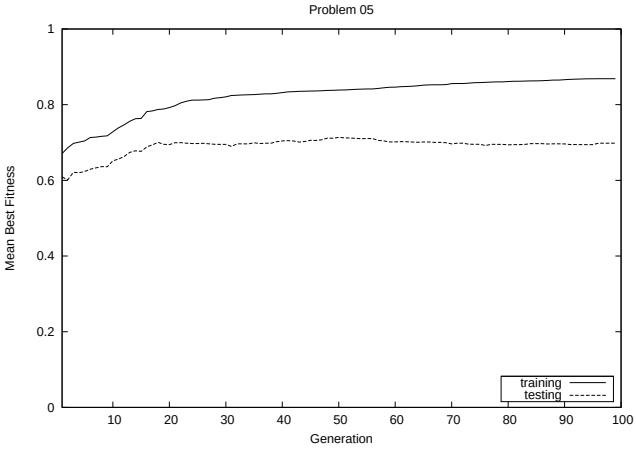


Figure 5.8: Training and Testing performance scores on the fifth artificial problem. Performance scores are the mean best fitness from 30 runs, showing training and testing performance at each generation. Training was performed on 100 fitness cases and testing was carried out on 100 unseen cases. The scores have been normalised to values between 0 and 1.

5.5.6 Summary

It should be clear from the testing performance scores achieved across all problems that GE is solving them with a high degree of success. The worst performance (approximately 70%) was on problem 5, which is arguably the most realistic problem of the set since it mimics a situation where multiple levels of preference ordering are at work in the mind of the user. Despite the fact that there is room to improve on this outcome, it is felt that this would be a very desirable result to achieve on real-world, non-contrived user data.

Results from the application of statistical hypothesis tests as shown Table 5.3 reveal that the GE models created are significantly superior to those that would be exected from an unbiased coin toss.

Table 5.3: Null Hypothesis Statistical test results using the end-of-run GE test performance score, using a t-test with alpha-level 0.01. If the resulting *p-value* from the test is less than 0.01, a statistically significant outcome is reported.

Problem	GE	Significant	P-Value
1	0.9313	Yes	2.96E-047
2	0.9423	Yes	8.92E-037
3	0.9403	Yes	6.72E-048
4	0.9113	Yes	1.83E-043
5	0.7137	Yes	1.24E-018

In relation to solution structure, in some cases it has been clear that the GE-evolved functions are focused on the input combinations which correspond to a "correct" choice, however it is not immediately clear in others. Of course, the question then emerges: should we be concerned about such solution structures as long as they are achieving good predictive performance on unseen data? Clearly, our goal has been to construct an accurate model of user behaviour; while it may be interesting to examine the inner-workings of a particular successful model, it is not essential that this can be performed – provided that the predictive performance of the model is sufficiently high.

5.6 An Initial Experiment with Real Data

In the previous section it was shown how a GE-based learning system can be used to form meaningful models of (contrived) fitness functions using a set of artificial problems. This is an encouraging result and represents a significant milestone on the journey towards the modelling of fitness functions from real-world data. It is clear from this result that when consistent data is part of the input, GE can be used to build good models.

Table 5.4: Instruments and scales used to generate preset melodies. Note that while the scale of the preset melody cannot change after it has been generated, the system does allow the instrument and drumbeat to be changed during evolution. The size of the search space using this setup is 16384.

Instruments available		Melody Scale	Drumbeat Style
Piano	Harpsichord	Chromatic scale	Rock (1)
Xylophone	Hammond organ	E Major	Rock (2)
Steel string guitar	Overdriven guitar	F Harmonic Minor	Rock (3)
Acoustic bass	Slap bass	G Melodic Minor	Funk (1)
Violin	Cello	Blues Scale in A	Funk (2)
Harp	Synth strings	B Diminished	Pop
Choir aahs	Saxophone	C Persian	Reggae
Oboe	Flute	C Major	Samba

An obvious next step would be to see if the system can still produce meaningful models when we introduce a human into the equation. In accordance with the "bottom-up" nature of the previous experiments and due to the relatively high cost of gathering data from a large cohort of human users, this initial experiment will focus on data gathered from a single subject before proceeding to a larger set of participants.

5.6.1 Presets Used

In the experiment that follows, eight pre-programmed drumbeats were used including the styles rock, funk, pop, reggae and samba. Sixteen instruments and eight musical scales were used to generate a total of 128 melodies. All settings used are shown in Table 5.4. The total number of presets that can come about using this configuration is 16384 (this is the size of the search space).

5.6.2 MelodyGA Settings

From a user point of view, the MelodyGA system is nothing more than a binary decision: two candidate songs are presented and a decision must be made as to which song is better. As mentioned earlier in Section 5.2, this is done to ease user-fatigue and aid user-concentration by making things as

Table 5.5: Summary of the MelodyGA settings used in the first "bottom-up" experiment on human user data.

Population Size	40
Initial Score	0
Update Amount	10
Replacement Interval	10 evaluations
Cull Cutoff	-11
Tournament Size	3
Crossover Rate	0.7
Mutation Rate	0.01

simple as possible. The job of the GA, then is to make the best use of this information to drive the evolution forward. The way that information is fed back to the GA for this experiment is as follows. If a song is selected as a winner, it gets an increased score; if it loses, its score is decreased. After a predetermined number of comparisons (the replacement interval), the population is sorted by score and those songs with the lowest scores are removed. Tournament selection, one-point crossover and point mutation then takes place to replace the individuals that have been removed. This setup allows for a larger, more diverse population since it is not absolutely necessary for every individual to be evaluated in order to stay in the population. All that is needed for an individual to survive is a score that is greater than a *cull cutoff* value, which will always be the case unless a user-comparison takes place in which the individual loses. A summary of the GA settings used is given in Table 5.5.

5.6.3 Learning from the Data

Over a one hour period, a human subject[2] performed 219 comparisons, which were then used to produce a dataset containing 438 fitness cases. This data was then split into a training and test set containing 292 and 148 fitness cases respectively. Thirty runs of GE were then performed using the settings

[2]The author of this dissertation.

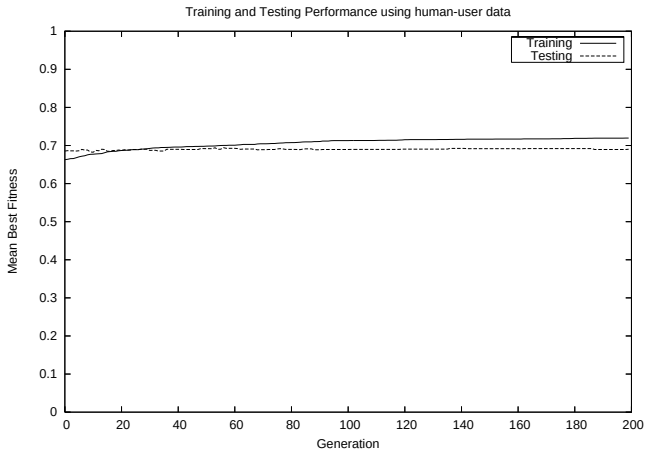


Figure 5.9: Training and Testing performance using MelodyGA data gathered from a human subject. Performance scores are the mean best fitness from 30 runs, showing training and testing performance at each generation. Training was performed on 292 fitness cases and testing was carried out on 148 unseen cases.

described in Table 5.7 with the grammar described in Table 5.6 used to specify solutions.

Plots of mean best performance on the training and testing data are given in Figure 5.9. As can be seen, the mean best training and testing performance stays above 65% for the duration of the run with test performance coming in just below 70% at the end of the run. If we zoom in on the test performance score (Figure 5.10) we can see that the best test score obtained was 69.4% at generation 56.

As used previously, a desirable result for GE to achieve would be statistically significantly better than that which would be expected by from an unbiased coin toss. A Null Hypothesis Significance test (alpha level 0.01) yields a P-Value of 4.833E-28, indicating that a significantly superior performance is achieved by GE. This is very impressive result given that it has been obtained from non-contrived data. Recall that in the most complex of the artificial problems examined in Section 5.5.5 a comparable test performance score was obtained. The fact that the learning method has maintained this

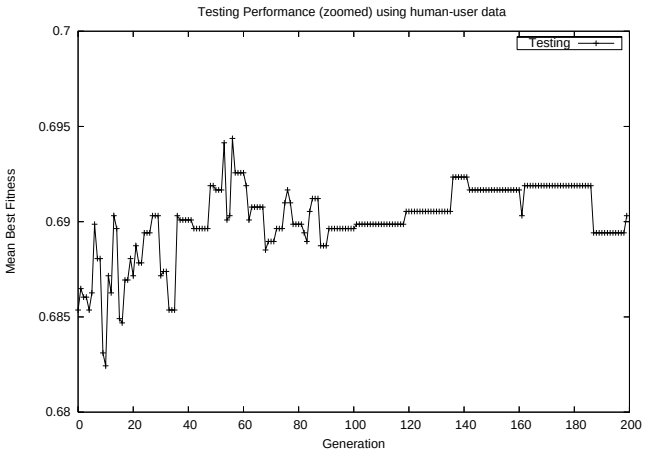


Figure 5.10: Detailed plot of the testing performance score from Figure 5.9. The best score obtained was 69.4% at generation 56.

level is very encouraging and represents another significant milestone.

5.7 Music to the Masses

So far in this chapter we have de-constructed and rebuilt a system for discovering human user's fitness functions from the bottom up. Experiments on artificial problems using the system have yielded encouraging results and a subsequent speculative experiment using non-contrived user data has indicated that the system is ready to be re-tested against a wider audience.

In the previous data-gathering experiments, the human subjects that took part did so in an environment which was specifically set up for data collection (in both previous cases, this was a teaching lab in a university). This constraint was removed for the experiments reported in this section. Although a lot of the data collected was still taken from students in computer labs, other interested parties could also take part in the experiment from anywhere else with an internet connection and a Java-enabled web browser. The overall architecture of the system is depicted in Figure 5.11.

There are no constraints on how many times a user runs the application

Table 5.6: Grammar used in the initial experiment on a single human subject.

```
<body>     ::= <musFunc> | <body> <binaryop> <musFunc>
             | <unaryop> ( <body> )
<musFunc>  ::= song1UsesDrumBeat(x, <val8> ) | song1UsesMelody(x, <val128> )
             | song1UsesScale(x, <val8> ) | song1UsesInstrument(x, <val16> )
             | song2UsesDrumBeat(x, <val8> ) | song2UsesMelody(x, <val128> )
             | song2UsesScale(x, <val8> ) | song2UsesInstrument(x, <val16> )
<binaryop> ::= and | or
<unaryop>  ::= not
<val8>     ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
<val16>    ::= <val8> | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15
<val128>   ::= <val16> | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24
             | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35
             | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46
             | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57
             | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68
             | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79
             | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 91
             | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101
             | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110
             | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119
             | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127
```

nor on the amount of time that it takes for a user to complete a given run. The Melody Evolver user interface only permits the user to progress to the next set of songs to be evaluated once the play button for each song has been pressed. The system keeps a record of the total run time, the total number of comparisons and the time taken for each comparison. History files for analysis by GE can then be downloaded from the system using appropriate selection criteria to filter out records below certain unusable levels.

To collect the data, two classes of first-year computing students from two Irish universities were given access to the system during their lab sessions. A total of 177 users took part[3] and of these, 15 produced enough data to be considered for this study according to the selection criteria shown below. By these criteria, the user must have:

1. Performed at least 50 comparisons; and

2. Spent at least 20 minutes doing so.

[3]This means that this number of distinct users visited the application URL, downloaded the applet and clicked the next button at least once.

Table 5.7: Experimental settings used by GE on the first "bottom-up" experiment on human user data.

Population size	500
Number of generations	200
Crossover rate	0.9
Mutation rate	0.01
Replacement strategy	Steady-state
Fitness function	The proportion of cases where the guesses match the targets (maximising)

A list of runs that meet these criteria is given in Table 5.8.

5.7.1 Results from Online Experiments

For each subject that was included in this study, the run data generated by the subject was split into a training and testing set using the first 85% of the data for training and the latter 15% for testing. Multiple runs of GE were then performed on each subject's data using the settings in Table 5.10. The GE grammar was adapted from that used in the previous speculative experiment on human user data (from Section 5.6) to include the use of conditional statements as shown in Table 5.9.

Results from the online experiments are given in Figure 5.12 and Figure 5.13 and summarised in Table 5.11. From the plots and the table it can be seen that the GE models achieved a performance score of at least 60% in 8 out of 15 cases.

A Null Hypothesis Significance test is carried out at each generation by taking the best solution discovered on the training data and calculating its corresponding test score from the unseen data. This test score is compared with that which would be expected from an unbiased coin toss. It turns out that the GE models produce results that are statistically significant (using

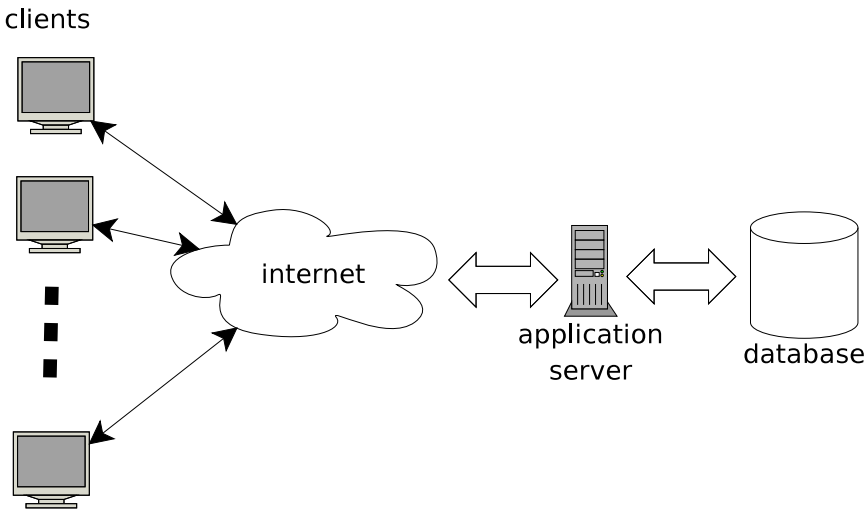


Figure 5.11: System architecture. Melody Evolver clients (Java applets) download their configuration settings from a central server. Users then take part in the experiment by repeatedly choosing their favourite songs. When finished, results are posted back to the server and persisted to a database. Results files from selected individuals may then be downloaded by the (GE) learning system for the purposes of building models of their fitness functions.

a t-test with alpha-level 0.01) in all but two cases. Table 5.11 shows the minimum p-value together with the generation at which it is calculated for those cases where a significant difference is observed.

Learning Curves

Inspection of the training lines shown in Figures 5.12 and 5.13 indicates a slow rate of learning in the training phase – the lack of steepness of the training curves in the initial generations suggest that the randomly created GE individuals from generation zero are solving the bulk of the problems, with only minor enhancements occurring over the course of the evolutionary run. Re-running the experiment above with a smaller population size does not produce drastically different training curves, as can be seen from Figures 5.14 and 5.15.

Initial examination of the plots suggests little difference between the re-

Table 5.8: User-runs with a minimum number of 50 comparisons which also took 20 minutes or more to complete.

Subject ID	Total Comparisons	Run Time (mins)
265	54	23.04
267	72	23.88
317	97	29.52
318	72	30.30
333	92	22.75
345	67	20.12
352	151	22.55
354	129	22.92
357	71	27.33
359	55	28.07
360	123	24.41
366	73	31.67
371	142	24.41
372	133	32.31
373	104	33.51

sults obtained from the use of a large population (5000) and a small one (50). A re-run of the significance tests shows an increase in the number of cases where no difference is observed to four. Hence, the increased population size offers better results, even if the best solutions appear in early generations.

5.8 Discussion

Producing accurate models of human user behaviour in the musical setting as described in this chapter has been a difficult task. Although it is clear that the use of GE has significantly outperformed a random choice method, a higher test performance score on each subject would have been a more desirable outcome.

The results that have been achieved emphasise the difficulty of the problem that is learning from (and predicting) human behaviour when making subjective, musical judgements. Despite introducing measures to simplify the problem by reducing the search space in the musical GA and altering the

Table 5.9: Grammar used in the online experiments. This grammar adds the availability of conditional statements to the evolving models.

```
<body>          ::= <if_else> | <if_elseif>
<if_else>       ::= if <conditional> then return <true_false>
                    else return <true_false> end
<if_elseif>     ::= if <conditional> then return <true_false>
                    <elseifs>
                    else return <true_false> end
<elseifs>       ::= elseif <conditional> then return <true_false>
                  | <elseifs> elseif <conditional> then return <true_false>
<conditional>   ::= <musFunc> | <conditional> <binaryop> <conditional>
                  | <unaryop> ( <conditional> )
<musFunc>       ::= song1UsesDrumBeat(x, <val8> ) | song1UsesMelody(x, <val128> )
                  | song1UsesScale(x, <val8> ) | song1UsesInstrument(x, <val16> )
                  | song2UsesDrumBeat(x, <val8> ) | song2UsesMelody(x, <val128> )
                  | song2UsesScale(x, <val8> ) | song2UsesInstrument(x, <val16> )
<binaryop>      ::= and | or
<unaryop>       ::= not
<true_false>    ::= true | false
<val8>          ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
<val16>         ::= <val8> | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15
<val128>        ::= <val16> | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24
                  | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35
                  | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46
                  | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57
                  | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68
                  | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79
                  | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 91
                  | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101
                  | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110
                  | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119
                  | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127
```

Table 5.10: Experimental settings for the GE runs using data gathered from the online Melody Evolver experiments.

Number of runs (per subject)	30
Population size	5000
Number of generations	50
Crossover rate	0.99
Mutation rate	0.1
Replacement strategy	Steady-state
Fitness function	The number of cases where the guesses match the targets (maximising)

user interface to transform fitness allocation into a binary decision, it is still unlikely that all unpredictable aspects of the behaviour of human users have been removed from the system. It may be the case that the more successful artificial models produced by GE correspond to cases where the human user has behaved in a certain way when making decisions about the quality of the musical pieces undergoing evolution. To investigate this idea further, we can use some of the (non-subjective) information about each user's run to discover if a relationship exists between the way in which the system is used and the corresponding performance of the GE-evolved models on the user's run data.

5.8.1 Predicting the Predictability

In addition to the raw run data accumulated by each subject, the online experiments also recorded other information about a given user run, such as the total number of song comparisons (evaluations), the run time and the average time taken to compare a set of competing songs. This information is used to produce a set of tuples of the form:

Table 5.11: Summary of the testing performance results achieved by GE from the online experiments. With a desirable result set at 60%, GE meets this level in 8 out of 15 cases. In all but two cases, the GE evolved models performance scores are significant based on a t-test with alpha level 0.01. In these cases, the minimum P-value together its associated generation number is reported.

Subject	Best Test Score	$\geq 60\%$	Significant	Minimum P-value	Generation
265	0.622	Yes	Yes	3.06E-009	9
267	0.545	No	Yes	1.33E-003	38
317	0.738	Yes	Yes	3.33E-016	46
318	0.553	No	Yes	1.18E-003	5
333	0.610	Yes	Yes	6.46E-012	11
345	0.585	No	Yes	0	19
352	0.671	Yes	Yes	0	5
354	0.486	No	No	-	-
357	0.611	Yes	Yes	3.16E-007	38
359	0.579	No	Yes	9.14E-008	45
360	0.643	Yes	Yes	0	3
366	0.595	No	Yes	2.82E-007	26
371	0.603	Yes	Yes	4.95E-013	19
372	0.507	No	No	-	-
373	0.664	Yes	Yes	0	1

$$[x_1, x_2, x_3, y]$$

where x_1 is the total number of comparisons performed by a human subject, x_2 is the run time (in seconds), x_3 is the average comparison time and y is the testing performance score achieved by the GE-induced model of the subject's choice metric. To investigate if a relationship exists between the inputs and the target values, GE is used here again with the following grammar:

```
<body>           ::= (function() return <expr> end) ()
<expr>           ::= <expr> <bin_op> <expr> | <unary_op>(<expr>) | <summary_metric>
<summary_metric> ::= total_comps(x) | run_time(x) | avg_comp_time(x)
<unary_op>       ::= math.log | math.sin | math.cos
<bin_op>         ::= + | - | * | /
```

The system is trained using eight of the fifteen fitness cases (chosen at random) and the models produced are tested against the remaining seven cases. All experimental settings are given in Table 5.12. To calculate test performance, the best models discovered during training are applied to the test data. A test hit occurs if the guessed value falls within 10% of the target value. Overall test performance is then calculated by measuring the proportion of test hits achieved.

Table 5.12: Experimental settings for the GE runs used to investigate a relationship between the way in which a user run was carried out and the resulting test performance score obtained from the artificial models of the user's choices.

Number of runs	30
Population size	500
Number of generations	100
Crossover rate	0.9
Mutation rate	0.01
Replacement strategy	Steady-state
Fitness function	Maximise the hits; a hit occurs when the guessed value and the target value differ by at most 0.01.
Test Performance	The percentage of cases where the guessed value and the target value differ by at most 0.1.

Results

After performing thirty runs of the GE configuration described above, a test performance score of 90.95% was obtained. This means that in over 90% of cases, the GE-induced model comes within 10% of the target value. A selection of best of run individuals is given below:

```
function() return math.sin(math.sin(math.sin(
  math.cos(math.sin(avg_eval_time(x))))))
end

function() return math.sin(math.sin(math.sin(
  math.cos(math.sin(math.sin(
  math.log(math.cos(math.sin(
  eval_time(x) * math.cos(math.sin(
  math.log(math.log(eval_time(x)))))))) - avg_eval_time(x)))))))
end

function() return math.sin(math.cos(math.cos(
  math.sin(math.sin(total_evals(x)) -
     math.sin(math.cos(total_evals(x) +
  math.cos(avg_eval_time(x)) - avg_eval_time(x)
  * eval_time(x) + math.sin(eval_time(x) +
  eval_time(x) * math.sin(total_evals(x) /
  total_evals(x)) * total_evals(x)))) + total_evals(x)))))
end
```

These results show that a relationship does exist between the way in which a human user performed a run and the performance obtained by a GE system producing artificial models of that user's behaviour.

We have seen from the results reported in Section 5.7.1 that the artificial models produced are not perfect. By deriving this relationship, we can start to find the boundaries of where this work can be useful.

This is very significant; such information helps in identifying the conditions under which artificial models of human fitness functions *can be created*.

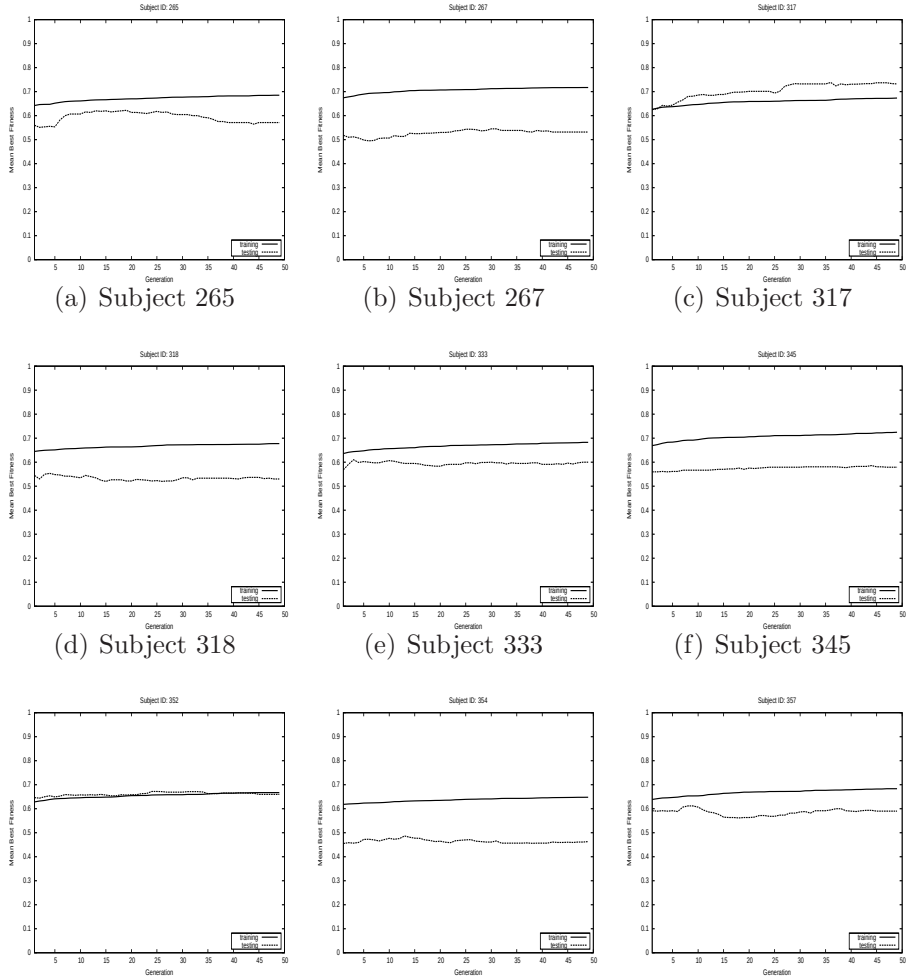
5.9 Conclusions

At the beginning of this chapter it was hypothesised that consistent user behaviour is the key to the derivation of models of human choices that have good predictive power. This has been initially demonstrated by the substitution of real-world data with controllable, artificial datasets. Given usable inputs, Grammatical Evolution produces models which perform very well when presented with previously unseen examples.

Moving out of the artificial, controllable domain and onwards to experimentation on human subjects, it was hoped that changes to the size of the search space and to the make-up of the user interface would help to filter out some of the noise that is to be expected from experimentation with human subjects. An initial experiment on a single human subject showed the measures to be working well, with a good predictive performance of almost 70% achieved. A subsequent set of experiments on a much larger group of participants has shown a wider range of results.

It is difficult to determine exactly how effective these simplification measures have been at *ensuring* consistent behaviour from all subjects tested. Based on the results achieved (as summarized in Table 5.11) it would be fair to assume that there are other potential factors at work that cause a given subject to produce consistent data.

This idea is investigated further by measuring some of the external conditions as experienced by subjects taking part in the experiments (as described in Section 5.8.1) and attempting to discover a relationship between these conditions and to the production of a good GE model. A better understanding of this is very likely to aid data noise reduction and ultimately to better artificial models.

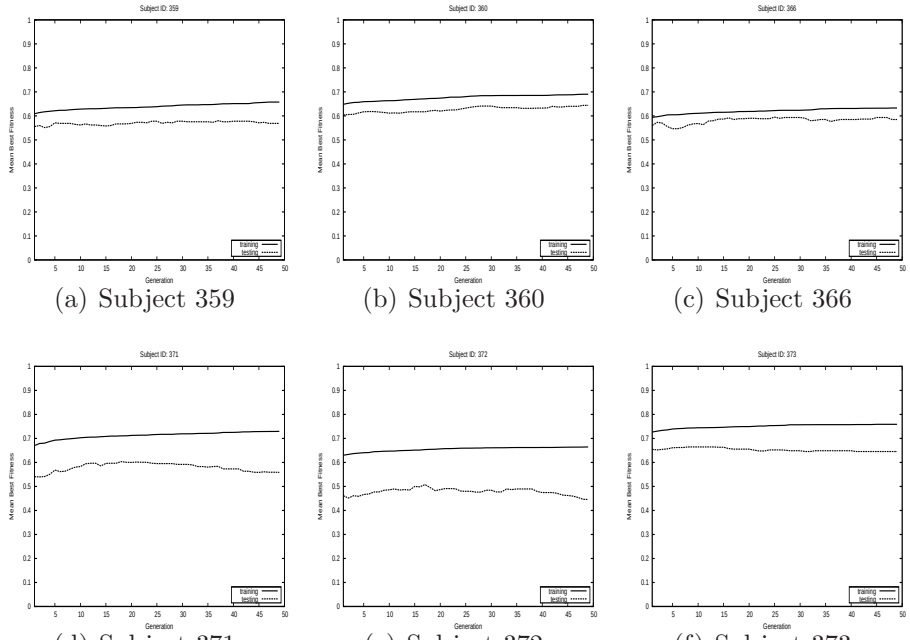


(a) Subject 265 (b) Subject 267 (c) Subject 317

(d) Subject 318 (e) Subject 333 (f) Subject 345

(g) Subject 352 (h) Subject 354 (i) Subject 357

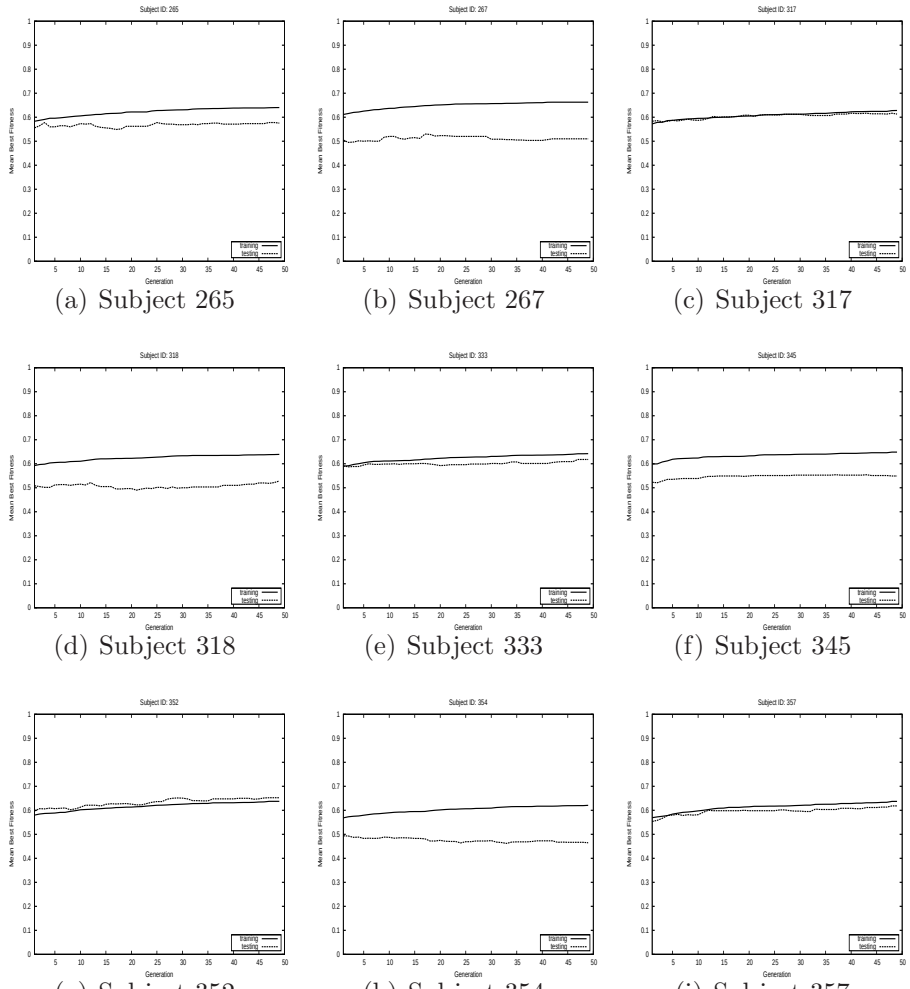
Figure 5.12: GE results from the first set of subjects that took part in the online experiments. Each plot shows the mean training and testing performance scores gathered from 30 runs.



(a) Subject 359 (b) Subject 360 (c) Subject 366

(d) Subject 371 (e) Subject 372 (f) Subject 373

Figure 5.13: GE results from the second set of subjects that took part in the online experiments. Each plot shows the mean training and testing performance scores gathered from 30 runs.

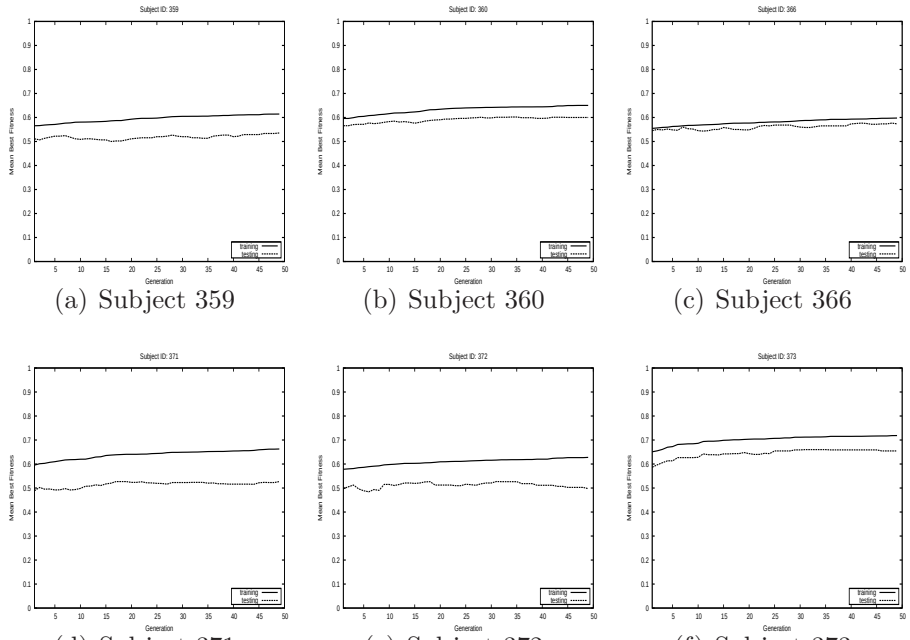


(a) Subject 265 (b) Subject 267 (c) Subject 317

(d) Subject 318 (e) Subject 333 (f) Subject 345

(g) Subject 352 (h) Subject 354 (i) Subject 357

Figure 5.14: GE results (using population size of 50) from the first set of subjects that took part in the online experiments. Each plot shows the mean training and testing performance scores gathered from 30 runs. Note that the training lines still appear quite flat despite the massive reduction in the population size.



(a) Subject 359 (b) Subject 360 (c) Subject 366

(d) Subject 371 (e) Subject 372 (f) Subject 373

Figure 5.15: GE results (using population size of 50) from the second set of subjects that took part in the online experiments. Each plot shows the mean training and testing performance scores gathered from 30 runs. Note that the training lines still appear quite flat despite the massive reduction in the population size.

Chapter 6

Improving Generalisation in Genetic Programming

In Chapter 4, Genetic Programming was applied to the task of of musical fitness function prediction and it was seen that certain success stories exist, which is an excellent result for GP in the context of a difficult real-world problem.

For the less successful cases, it would be unfair to blame GP. In fact, any case of poor performance is much more likely to be a result of the small, noisy, inconsistent dataset that was provided to it. This claim is supported by the corresponding predictions which were made by other "black-box" learning methods on the same data.

In this chapter, we initially detour from modelling subjective fitness functions and instead focus on the generalisation of GP. We examine this on some well-studied test problems and also critically examine the performance of some well known GP improvements from a generalisation perspective. From this, the need for GP practitioners to provide more accurate reports on the generalisation performance of their systems on problems studied is highlighted. Based on the results achieved, it is shown that improvements in training performance thanks to GP-enhancements represent only half of the battle.

The goal of the work presented in this chapter is to try to improve the generalisation of GP, particularly in the context of recent advances to the field. It is argued here that the blind application of improvement techniques is not always a good idea, rather, there is a balance that can be struck which has desirable consequences for making predictions about unseen events.

These aims are summarized as follows:

- To investigate and improve GP's generalisation abilities
- To examine recent enhancements to the field in order to show how getting the balance right is important

In particular, the combination of two recent GP-improvement techniques, *Linear Scaling* (Keijzer, 2003) and *No Same Mates* (Gustafson *et al.* , 2005) is recommended for achieving better generalisation on forecasting and prediction problems.

Following an initial demonstration on artificial problems, this positive effect is also demonstrated by revisiting the *DrumGA* experiments from Chapter 4.

6.1 Background and Motivation

Improvements to Genetic Programming are discovered and reported at Evolutionary Computation conferences and in articles every year. A typical effort involves the discovery of a new technique followed by its comparison with standard Koza-style GP (often on the same problem). These comparisons tend to show that due to a statistically significant improvement between the proposed method and standard GP on *training* data, the new method is therefore declared to be successful.

Unfortunately, what many such studies fail to address is the performance of the proposed GP-improvements on *unseen* data [1]. A central message

[1]Even the seminal, award winning papers by Keijzer (Keijzer, 2003) and Gustafson *et*

of this chapter is that approaches such as these must take generalisation performance into account before any declaration of success is made.

It should be stressed that the motivation for much of the work described here is not founded in an attempt to discount the valuable contributions that researchers have made to the GP field. On the contrary: this work aims to increase the robustness of such contributions, making them stronger competitors against other methods within the field of Artificial Intelligence and beyond.

6.2 On Problem Difficulty

In assessing the generalisation ability of GP, it would be prudent to examine it in a simple form and strip away some limiting factors. For example, if we want to assess limitations, there is no point in trying to assess these on a difficult problem, since this will only add to the limitations. Instead, it would be useful to choose a problem that has been studied well and acts as a useful benchmark for assessing GP.

To gain an idea of the difficulty of a GP problem Koza suggests the measurement of the number of individuals that must be processed in order to satisfy the problem's *success predicate* with a chosen probability. A strict success predicate can be defined as the discovery of a 100% correct solution to the problem. A slightly less strict version of the predicate deems a GP-generated program to be successful if it achieves N "hits" (where N is the number of fitness cases and a hit occurs when a predicted outcome and a target outcome fall within a pre-specified error window).

If the amount of processing that must take place to achieve a successful solution to a problem is correlated with problem difficulty, then it follows that the lower the value of this measurement (or the larger the error window is), the easier the problem is for GP to solve. Two obvious factors that

al (Gustafson *et al.* , 2005) are examples. The two examples cited have made significant contributions to the GP field – the fact that these particular studies do not directly address generalisation performance should not deter from this.

influence the amount of processing are the population size M and number of generations G. Koza describes the process of measuring the amount of processing required by estimating the probability $\gamma(M, i)$ that a particular run with population size M first yields a successful individual at generation i.

The *cumulative probability of success*, $P(M, i)$, then refers to the probability of achieving a successful result for all generations between 0 and i.[2] Koza defines

$$R(M, i, z) = \lceil \frac{1 - z}{log(1 - P(M, i))} \rceil$$

as the number of runs required to have a probability $z = 99\%$ of discovering a solution before generation i. Calculating the total number of individuals that must be processed to achieve a successful result is then a matter of straightforward multiplication by the population size and the number of generations (including the initial generation):

$$I(M, i, z) = m(i + 1)R(M, i, z)$$

As noted (and even criticised[3]) elsewhere (Luke & Panait, 2002), the above metrics are often used in GP literature to compare new GP-variants. For a more comprehensive study on problem difficulty in GP, the reader is referred elsewhere (Vanneschi *et al.* , 2005).

[2]It is calculated experimentally as follows:

$$P(M, i) = \frac{N_s(i)}{T}$$

where T is the total number of runs and $N_s(i)$ is the number of successful runs at generation i.

[3]The authors (Luke & Panait, 2002) note a difficulty with the use of ideal solution count measures GP research due to an observed lack of correlation with best-fitness-of-run measures – it was noted that in cases where there were very few successful runs, the results appear distorted suggesting a better overall performance than what was actually achieved.

6.2.1 A Simple Problem

If there is one symbolic regression problem that receives the most attention in GP research, it is probably that of (re-) discovering the quartic polynomial. The function is described as follows:

$$f(x) = x^4 + x^3 + x^2 + x \tag{6.1}$$

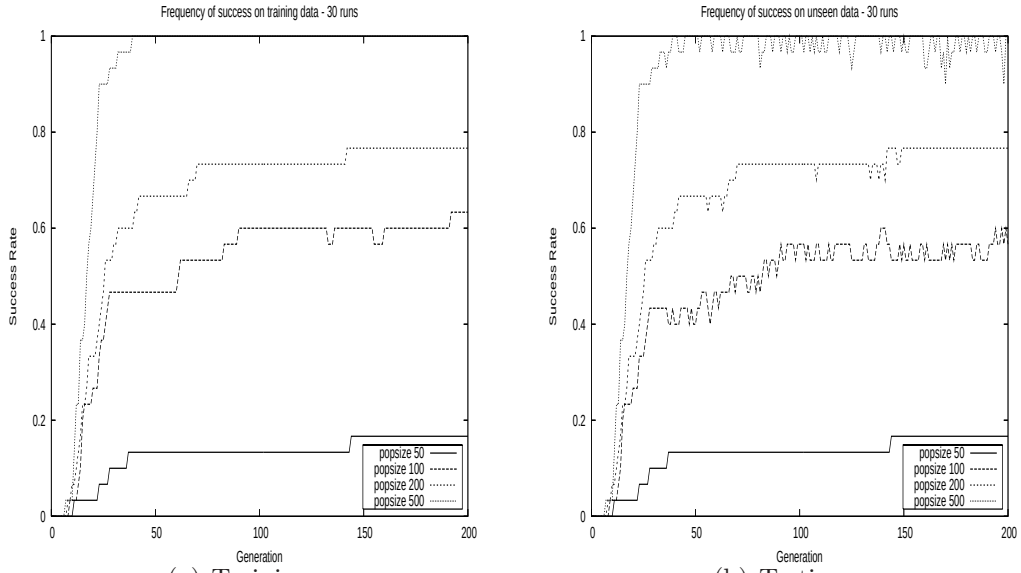
Due to the popularity of this problem in GP research, it is used here for the purposes of demonstration.

Making things *easy*

Now that we have a simple problem to study and a metric that can be used to approximate the difficulty of the problem, let us now turn our attention to the task of finding a configuration of the problem settings that achieve a high success rate. This task is achieved experimentally as follows. Using a Koza-style GP system (Costelloe, 2008), we supply a set of configurations to the system, run each one and then choose the configuration that turns out to be the easiest for GP.

An *easy* configuration is (arbitrarily) defined here as one that produces an average success rate of at least 70% by the end of the run. Let us choose configuration value for population size; using the run parameters shown in Table 6.1, four sets of 30 independent runs are carried out using population size 50, 100, 200 and 500.

By examining the success rate plots in Figure 6.1 (left), we can immediately identify two "easy" configurations from the use of populations sizes 200 and 500 since the success rate passes the pre-defined target value of 70% in both cases. Also shown in Figure 6.1 (right) is the success rate as calculated from the number of hits on the testing dataset. It is encouraging to note that the performance on unseen data holds up almost as well as the training performance, which is a very desirable characteristic for any black box learner/predictor method.



(a) Training (b) Testing

Figure 6.1: Success rate plots from four sets of runs on the quartic polynomial problem using a training dataset of 21 points in the range [-1:0.1:1] (left) and a test dataset of 11 points in the range [1:0.1:1] (right). Each set of runs uses a different population size.

This characteristic is so desirable that it should form the basis of a much more useful comparison between $GP + improvement$ and standard GP for forecasting problems. Put another way, if the training / testing plots exhibit very obvious differences then it would be wise to question the validity of the technique in a generalisation context.

6.3 A Selected Improvement: Linear Scaling

A number of improvements to GP for symbolic regression have been published in recent years (Fernandez & Evett, 1998; Topchy & Punch, 2001; Keijzer, 2003; Keijzer, 2004). Among these is Keijzer's application of linear scaling (Keijzer, 2003) to a suite of test problems and the subsequent proof (Keijzer, 2004) of the superiority of the scaled error measure over standard error measures (such as MSE) on symbolic regression problems. The method

Table 6.1: Run settings used to find a population size to use which produces a success rate of at least 70%.

Generations	200
Crossover rate	0.95
Mutation rate	0.02
Tournament size	5
Function set	$\{+, -, *, \%, sqr()\}$
Terminal set	$\{x\}$
Raw fitness	MSE(targets, predictions)
Standardised fitness	$\frac{1}{1+MSE}$
Hits criterion	Number of points where the GP program comes within 0.01 of the desired value

works by finding two values, a and b such that

$$a + bg(x) = t + \epsilon$$

It can be proved that the error (ϵ) is reduced via the application of linear scaling for $a \neq 0$ and $b \neq 1$. The process works as follows. If x is a set of independent variables, g is a GP-induced function that produces a set of predictions y and t is a set of target values derived from an unknown function f such that $f(x) = t$, a linear regression on the target values can be performed:

$$b = \frac{\sum_{i=0}^{n}(t - \bar{t})(y - \bar{y})}{\sum_{i=0}^{n}(y - \bar{y})^2} \tag{6.2}$$

$$a = \bar{t} - b\bar{y} \tag{6.3}$$

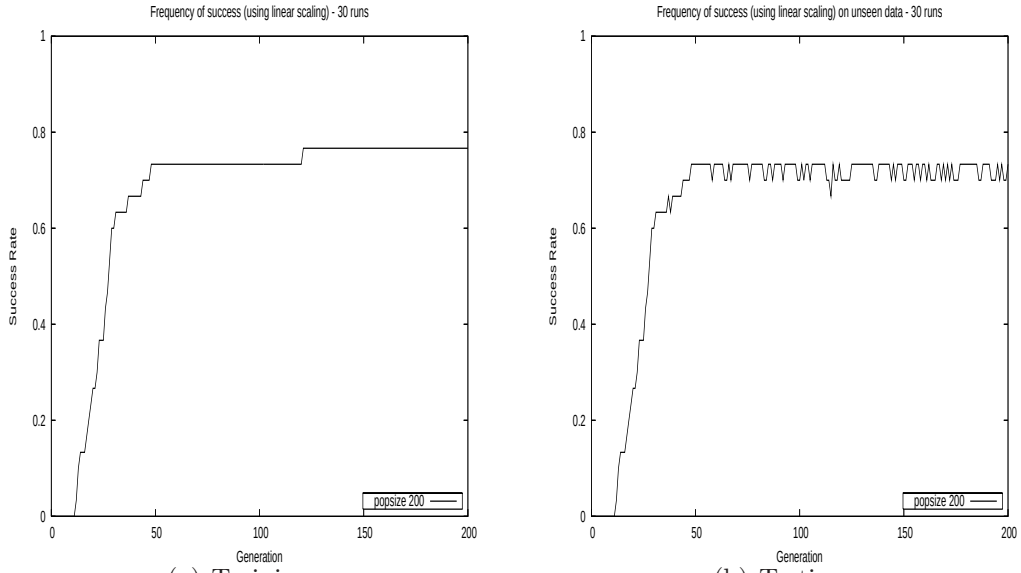
The error measure is then scaled as:

$$MSE(t, a + by) = \frac{1}{N}\sum_{i=0}^{N}((a + by) - t))^2 \tag{6.4}$$

Due to the relatively low computational cost of calculating a and b, the

implementation of this technique has been recommended to GP researchers and some successful results can be found in the literature (Raja *et al.* , 2007; Majeed & Ryan, 2006). The technique is not without its apparent limitations, however. In one particular example of its application on a real-world problem (Valigiani *et al.* , 2004), the findings suggest that the application of linear scaling leads to over-fitting, resulting in poor forecasting abilities[4].

6.3.1 Does Linear Scaling Over-fit?



(a) Training (b) Testing

Figure 6.2: Success rate plots on the quartic polynomial problem (population size 200) with linear scaling, using a training dataset of 21 points in the range [-1:0.1:1] (left) and a test dataset of 11 points in the range [1:0.1:1] (right).

Using a population size of 200 as chosen from section 6.2.1, let us examine the success rate plots when linear scaling is applied to quartic polynomial problem (with all other settings as per Table 6.1). The results from thirty

[4]It should be pointed out here, however, that the study (Valigiani *et al.* , 2004) led to this conclusion based on 20 independent runs. It has been noted elsewhere (Luke & Panait, 2002) that at least 30 runs should be carried out before any statistically sound conclusions may be made regarding performance comparisons.

independent runs are shown in Figure 6.2. What is clear from this test is that the application of linear scaling to this so-called *easy* problem configuration does not show any evidence of over-fitting. This can be seen in the right plot in the figure: the success rate does not drop off when applied to unseen data, which shows that the GP-derived models can forecast with a comparable amount of accuracy to that observed on the training set. Again, this sort of result is desirable; any experiences with GP-improvement techniques that do *not* follow this pattern should be called into question.

6.3.2 Comparing Performance

The use of sound statistics is of the utmost importance when comparing approaches such as these. As noted by Luke (Luke & Panait, 2002), many examples can be found in GP literature where conclusions regarding successful techniques are founded on questionable statistical practices.

A widely used test to determine whether there is a statistically significant difference between two sets of observations is the paired t-test. This test can be used to compare two methods for example, GP and *improved* GP with all else being equal (same random seeds, run settings and so on). The test is performed as follows; if X and Y are two sets of observations, let

$$\hat{X}_i = (X_i - \bar{X})$$

$$\hat{Y}_i = (Y_i - \bar{Y})$$

$$t = (\hat{X} - \hat{Y})\sqrt{\frac{n(n-1)}{\sum_{i=0}^{n}(\hat{X}_i - \hat{Y}_i)^2}}$$

This statistic has $n-1$ degrees of freedom and this information is used to estimate the P-value, that is the estimated probability of rejecting the *null hypothesis*. To bring this into context, we could let X be the set of best-of-

run values obtained from standard GP applied to a problem and let Y be the set of best-of-run values obtained using *improved* GP. The null hypothesis in this case would be that there is no difference between the two sets of observed values X and Y. The question that the study is asking is: *is there a significant difference in performance between the two methods?* This forms the *alternative hypothesis*, which states that there *is* a significant difference in performance between the methods. To claim statistical significance, the aim is to reject the null hypothesis.

Although the choice of threshold P-value varies across statistical studies, values of 0.05 and 0.01 are typical. In the former case, a P-value obtained that is less than 0.05 can be interpreted (in English) as: "a less than one in twenty chance of being wrong" (about rejecting the null hypothesis).

The only difficulty with the method described above is that the sets of observations X and Y are obtained from a single point in time: the end of the run. It would be statistically much stronger to state that no point over the duration of the run is there evidence that differences in performance are due to chance.

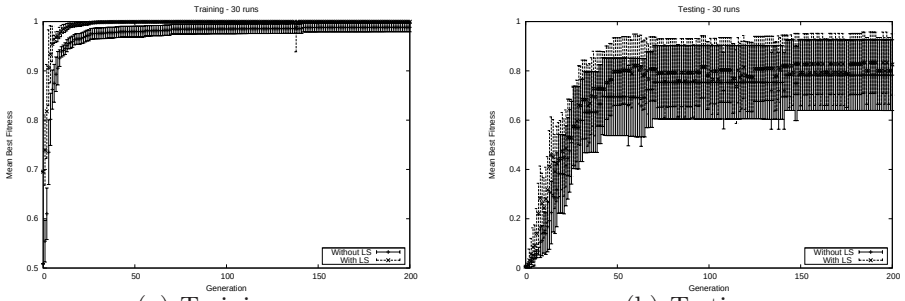
We can use confidence intervals to convey this information – this is statistically equivalent (Wasserman, 2004) to performing a paired t-test about each set of runs per generation. Using a sample of n (i.e the number of runs) observations of X, the 95% confidence limits are defined as:

$$\bar{X} \pm 1.96 \frac{\sigma}{\sqrt{n}}$$

where $\bar{X}$ and σ refer to the mean and standard deviation of the observations. What this measure tells us is that based on the sample provided, we can be 95% certain that the (statistical) population lies inside these limits. To apply this in a GP-context, this information can easily be incorporated into best-of-run plots (using error-bars, where the bars represent the confidence intervals) showing the differences between two competing methods. If there is *no overlap* between the error-bar plots then it is safe to assume that a

statistically significant difference in performance is observed.

Armed with this technique, we can now assess the performance of linear scaling. A comparison is shown in Figure 6.3. The figure shows the error bar plots obtained from both training and testing. We can clearly see by inspection (overlapping error-bars) that no significant difference in performance is observed for this particular problem.



(a) Training (b) Testing

Figure 6.3: Mean best fitness plots on the quartic polynomial problem (population size 200). Each plot compares the use of standard GP with and without linear scaling, using a training dataset of 21 points in the range [-1:0.1:1] (left) and a test dataset of 11 points in the range [1:0.1:1] (right). The error-bars shown represent 95% confidence intervals. Note that standardised fitness is used so all y-axis values fall between 0 and 1.

This simple demonstration does not provide enough information to conclude that GP with linear scaling does not outperform standard GP. This will be investigated by experimentation in the following section.

6.3.3 Widening the Scope

Using a set of test problems (most of which can be found in (Keijzer, 2003)) described in Table 6.2, let us now examine the performance differences between standard GP and standard GP with linear scaling on a selection of four problems. It has already been shown and also proven that training error is guaranteed to be reduced when linear scaling is applied, so the differences in training performance will not be shown here. Instead, we will examine

the generalisation performance on unseen data. Unless otherwise stated, the run settings used for 30 runs of GP both with and without linear scaling are given in Table 6.3.

Table 6.2: Test Problems under investigation

#	**Description**	**Training set**	**Test set**
1	$f(x) = x^4 + x^3 + x^2 + x$	401 cases: [0:0.01:4]	81 cases: [1:0.025:3]
2	$f(x) = 0.3xsin(2\pi x)$	161 cases: [0:0.025:4]	55 cases: [0:0.073:4]
3	$f(x, y) = x^y$	400 random cases: $x, y \in$ [0:1]	100 random cases: $x, y \in$ [0:1]
4	$f(x, y) = 6sin(x)cos(y)$	400 random cases: $x, y \in$ [0:1]	100 random cases: $x, y \in$ [0:1]

Table 6.3: Run settings for the 4 test problems.

Population Size	200
Generations	200
Crossover rate	0.7
Mutation rate	0.02
Tournament size	3
Function set	{+, -, *, %, sqr(), sin(), cos(), ln() }
Terminal set	$\{x\}$
Raw fitness	MSE(targets, predictions)
Standardised fitness	$\frac{1}{1+MSE}$
Hits criterion	Number of points where the GP program comes within 0.01 of the desired value

Results from the four test problems are summarised in Table 6.4. In terms of statistical significance, the results are quite disappointing; it is found that in none of four trials examined did the application of linear scaling contribute to a significant performance improvement, which is quite dramatic considering the huge performance gains that would be expected if we were to examine training performance alone.

Table 6.4: Summary of results obtained when comparing the generalisation performance of standard GP versus standard GP with Linear Scaling over four test problems. Note that a "No" value means that the improvement to GP has not resulted in performance significantly *better* than standard GP. An example confidence interval illustrating whether or not overlap occurs is also given.

Problem	**Significant**	**95% confidence interval**	
		GP	GP with LS
1	No	0.8680 ± 0.1040	0.9587 ± 0.0649
2	No	0.9317 ± 0.0128	0.9505 ± 0.0146
3	No	0.9911 ± 0.0015	0.9957 ± 0.0042
4	No	0.9309 ± 0.0197	0.9644 ± 0.0303

6.3.4 Discussion

The results obtained from the four test problems would not support any assertion that the use of linear scaling should improve generalisation performance. Variations of the experiments above were also re-run by incorporating the idea of *potency* whereby the level of application of the scaled error measure was both increased and decreased over the duration of the run. In neither case was it possible to improve upon the results from Table 6.4.

6.4 Powers Combined: No Same Mates

As indicated previously, the use of Linear Scaling with GP has been shown to offer little performance gains over standard GP when applied to unseen data. Given that the method has been proven to reduce the error on training data (Keijzer, 2004) it is unfortunate that the same gains are not evident when it comes to generalisation. However, the results from the experiments above should not necessarily lead us to the conclusion that GP with linear scaling cannot generalise. It could be the case that good generalisation is possible, however not necessarily with the application of Linear Scaling *alone*. It would appear that LS can cause the selection of individuals which match the

training data too well – the population quickly becomes saturated with these individuals which has a negative effect on the resulting generalisation. In this scenario, having some means to counter-act the selection pressure would be desirable.

Gustafson *et al* (Gustafson *et al.* , 2005) reported a simple GP improvement technique that forced the genetic operators to always use parents with different fitness values. The method was shown to provide a statistically significant performance improvement in the training phase when compared with standard GP. At the core of the work carried out by Gustafson *et al* was the fact that the probability of no change in solution quality increases with the similarity of solutions. By forcing the mating of dissimilar individuals, a significant improvement in solution quality was observed.

In the experiments that follow, we combine this *no same mates* idea with the application of Linear Scaling to see if it results in better generalisation. Using the same experimental settings as given in Table 6.3, standard GP is compared to GP with Linear Scaling using No Same Mates (NSM). Results from the four test problems are summarised in Table 6.5.

It can be seen from the table that the incorporation of NSM with Linear Scaling results in a significant improvement over standard GP in three out of four cases. Recall that when standard GP was compared with GP and Linear Scaling, a significant improvement was not observed in any of the four cases. To test that this improvement was due to the *combination* of the two techniques (LS and NSM) and not just NSM alone the experiment was re-run without Linear Scaling. In this case, no statistically significant improvement in generalisation performance was observed in any case. This is good news, both for Linear Scaling and for the No Same Mates technique. Although the combination of the two techniques has not been an overwhelming success, a step in the right direction is clearly observed. In the case of the second problem, no single configuration of settings was found to significantly outperform (or under-perform) another. Furthermore, this problem presented the greatest difficulty for all GP variants tested, resulting in the lowest number

of successful individuals discovered.

Table 6.5: Summary of results obtained when comparing the generalisation performance of standard GP versus standard GP with Linear Scaling and the No Same Mates technique over four test problems. Note that a "Yes" value means that the improvement to GP resulted in performance significantly *better* than standard GP. An example confidence interval illustrating whether or not overlap occurs is also given.

Problem	**Significant**	**95% confidence interval**	
		GP	GP with LS and NSM
1	Yes	0.8680 ± 0.1040	0.9907 ± 0.0047
2	No	0.9317 ± 0.0128	0.9525 ± 0.0112
3	Yes	0.9911 ± 0.0015	0.9975 ± 0.0010
4	Yes	0.9309 ± 0.0197	0.9809 ± 0.0043

6.5 DrumGA Revisited

The results obtained on the artificial problems described so far would appear to advocate the combination of Linear Scaling and the No Same Mates technique. To test the applicability of this idea to a real-world problem, let us return to the problem of predicting fitness values for drum-patterns, using data gathered from the DrumGA experiment as described in Chapter 4.

The raw data sets from the fourteen subjects were split into training and test sets as before. To guard against over-use of (or a reliance upon) the protected division operator, values of 0 (corresponding to rests or off-beats in the drum pattern) were converted to -1. The experiments performed used the *gpsr* system (Costelloe, 2008) in the following four configurations:

- GP (standard)
- GP with LS
- GP with NSM

- GP with LS and NSM

All four configurations used the same random seeds for thirty runs each. Settings for the runs are shown in Table 6.6.

To obtain a high-level view of the results, the highest test performance scores found by each GP-variant on each subject was extracted as shown in Table 6.7 (best values shown in bold type). In some cases, the highest value is shared by more than one GP-variant. Each of the four GP-variants are then rated by recording the number of "wins" as the number of times that a variant produced the highest test performance value. The number of "out-right wins" is also recorded as the number of times that a variant got the highest value, over all others.

Table 6.6: Run settings used in a comparison of GP with three other GP variants with Linear Scaling, No Same Mates and both.

Population Size	200
Generations	100
Crossover rate	0.7
Mutation rate	0.02
Tournament size	3
Function set	{+, -, *, %, sqr(), sin(), cos(), ln() }
Terminal set	$\{x\}$
Raw fitness	MSE(targets, predictions)
Test Performance	Proportion of correct classifications

As can be seen from the table, the GP+LS+NSM variant performs best since it records the most wins and out-right wins. This finding supports previous results which suggest that the combination of Linear scaling and the No Same Mates technique leads to better test performance.

6.6 Summary

This chapter has noted the relative absence of generalisation performance analysis in Genetic Programming research in recent years. This deficit stands

Table 6.7: Summary of a comparison of GP and variations of improvement techniques on real-world data taken from participants of the DrumGA experiments. The table reports the highest test performance score found by each method (averaged over 30 runs). Values in bold are the best of the four configurations tested (sometimes shared by more than one GP-variant). At the bottom of the table, the number of **wins** is reported as the number of times that a variant got the highest value. The number of **out-right wins** is the number of times that a variant got the highest value over all other methods.

Subject	**GP**	**GP+LS**	**GP+NSM**	**GP+LS+NSM**
01	**0.9167**	**0.9167**	**0.9167**	**0.9167**
02	**0.8111**	0.7944	0.8083	0.7944
03	0.6750	**0.6917**	0.6889	**0.6917**
04	0.3583	0.4056	0.3639	**0.4194**
05	0.4583	0.5083	0.5028	**0.5139**
06	**0.8333**	**0.8333**	0.8306	**0.8333**
07	**0.5861**	0.5833	0.5833	0.5833
08	**0.7472**	0.7306	**0.7472**	0.7306
09	0.4722	0.5639	0.4611	**0.5861**
10	**0.4944**	0.4917	0.4889	0.4917
11	0.5361	0.5389	**0.5528**	0.5333
12	0.5861	**0.6389**	0.5861	0.6111
13	0.6194	0.6750	0.6194	**0.7083**
14	0.3417	**0.5417**	0.3583	0.5278
Wins	6	5	3	7
Out-right wins	3	2	1	4

in stark contrast to the amount of reported successful GP-improvements stemming from their application to problems consisting solely of training data. The main motive for highlighting this apparent shortcoming has not been to de-value the contributions of GP researchers. Rather, we have aimed to strengthen the foundations by pointing to areas that appear to be lacking with the hope that future research may benefit.

In Section 6.2, a frequently used metric (Cumulative Frequency of Success) was used to display performance on unseen data for a simple GP problem. We suggest that the success rate plots resulting from unseen data should

not differ drastically from those achieved on training data if the method under investigation is to be *relevant* in terms of generalisation. Later, in Section 6.3, we examined the unseen data success rate on a popular GP improvement technique – Linear Scaling. An initial experiment on the quartic polynomial showed the success rate on unseen data to hold up well compared to the training phase.

This hopeful start lead to a more detailed investigation of differences in performance using commonly used measures of statistical significance in the context of generalisation. The finding (based on a set of test problems) has been quite dramatic: Linear Scaling does not always generalise well – that is to say, not significantly better than standard GP. This is an unfortunate outcome for a method that has been demonstrated to perform so well on training data. Some valid questions then emerge. Does this mean that Linear Scaling is *bad*? If so, should the practice of using it be discouraged? Not necessarily. It is equally plausible that the technique is simply *too good*. It was hinted in Section 6.4 that it causes too much pressure on the selection of individuals that match the training data as closely as possible, with poor generalisation as a consequence.

By combining the application of Linear Scaling with another simple improvement to GP that forces recombination between parents with different fitness values, we have found cases where better generalisation is possible. This second improvement technique appears to be a steadying counterbalance for the more aggressive characteristics displayed by Linear Scaling. While their symbiotic relationship has not resulted in perfect generalisation scores on all of the test problems studied, we are certainly experiencing a positive outcome. It is quite fortunate and somewhat pleasing that such a result has grown from the combination of methods which were initially criticised in Section 6.1.

What is even more pleasing is that this finding stands up when applied to the DrumGA human fitness function modelling problem. When we move from the controlled, restricted environments of artificial test problems to a

noisy, real-world problem we see that the best performance values found were achieved using the same combination of GP-improvement techniques.

Chapter 7

Conclusions

The core of this thesis has been about the ability of Evolutionary techniques to learn and predict from humans as they make subjective decisions. The musical domain within the creative arts was chosen as an appropriate area of study due to the fact that a single piece of music may be experienced in infinitely many different ways by human listeners. By recording the subjective choices made by humans as they listen to simple rhythms and melodies, it has been shown here that Evolutionary techniques such as Genetic Programming *can* be used to construct models of subjective fitness functions that have good predictive power.

In the case of the DrumGA experiments described in Chapter 4[1] and those of the later MelodyGA, cases were found where good predictive power is achievable for some subjects. Running the same data through another black-box learning mechanism (a Neural Network) showed corresponding successes. This has shown that is is possible to use the system in a way that lends itself to the production of good artificial models.

On the pessimistic side, both DrumGA and MelodyGA experiments also showed cases where the models constructed were poor predictors. As such it may have been too optimistic to assume that good models could be built for a majority of the human subjects tested.

[1] Also published in (Costelloe & Ryan, 2004).

Even so, the successful cases provided hope. In Chapter 5, the MelodyGA system was demolished and reconstructed on a firmer foundation, taking lessons learned from human subjects into account. Through the construction of a set of test problems, the new system was demonstrated to capably learn from noisy, historical data[2]. When compared to the performance that would be expected by a random predictor, the models generated using Grammatical Evolution were demonstrated to be significantly superior in terms of predictive power.

Although based on artificial data sets (with noise added to make things more challenging), the fact remains that GE can learn and accurately predict in these simple scenarios. Any failures that GE has can therefore be blamed on data that is too noisy, too inconsistent or too sparse.

With this more solid foundation in place, human subjects were again asked to take part in a data collection experiment (this time over the internet in somewhat less controlled situations than in other cases). A repeat application of GE has a model builder / predictor on the data gathered showed mixed results. It is encouraging to see that the GE performance is statistically significant in almost all cases. We also observed a test performance score of 60% or more in over half of the cases tested.

The positive side of this result is that by *predicting the predictability*, it has been shown how the conditions in which a human user takes part in the data-gathering experiment have an influence on the overall performance of the models derived from the data. In this way, we not only see that good models of human fitness functions *can* be created, but we can also start to understand *how*.

7.1 Answering Core Questions

This brings us back to the core questions addressed by this research, restated here as follows. In the context of fitness function modelling:

[2]Also published in (Costelloe & Ryan, 2007).

1. Can a system be constructed that displays the ability to learn subjective notions from humans?

2. How well do artificially created models compare with future (unseen) choices?

3. Can the causes of any inconsistent user behaviour be identified and restricted?

The answer to the first question is that yes, Evolutionary methods (such as GP and GE) can learn subjective notions from humans. We have seen successful cases therefore the ability is there.

Answering the second question is more difficult. For some subjects, under some conditions, the artificial models compare very well with their human counterparts. Sadly, these successful cases appear to be in the minority.

The lack in consistency of user behaviour is the most likely contributor to the less successful outcomes. This claim is supported by the substitution of a human user with a controllable, artificial agent on a set of musical test problems - when consistent historical data is used as an input, the Evolutionary learner can produce accurate models from the data.

The size of the search space and the style of the user interface were identified as two possible sources that lead to inconsistent behaviour leading to the recommendation of a tournament-style user interface and a restricted search space in order to create less noisy data. It is evident that the introduction of these changes alone is not enough to provide a thorough clean-up of the data across all subjects studied.

However all is not lost; it has also been shown here that a relationship exists between the manner in which a human subject takes part in the experiment and the resulting predictive performance of the artificial model. This helps us to identify conditions where good prediction is possible.

This research has made considerable advances towards answering the "Problem of Aesthetic Selection" (McCormack, 2005). Producing a formalisation of the subjective functions at work in the mind of the human decision

maker is advantageous in terms of alleviating the fitness bottleneck that comes about in Interactive Evolution. Overcoming the bottleneck in any way is clearly a highly valuable contribution.

Outside of the musical domain, the question was asked if recent advances in the theory have lead to practical improvements on real-world problems. As has been seen, prediction and generalisation has played a central role in the investigation into the modelling of subjective fitness functions. It is fitting then that the penultimate chapter focuses on generalisation in Genetic Programming. GP was used almost exclusively in the initial experiments. In an effort to improve on testing performance scores a selection of GP-improvement techniques were examined for symbolic regression problems (not restricted to modelling subjective functions).

It was startling to discover that recent GP improvement techniques did not appear to consider performance on unseen data. This stands in stark contrast to what is considered standard practice in other statistical modelling research fields. It was fortunate, to discover that a combination of two recent advances in the GP field leads to significant improvements in generalisation performance on a set of symbolic regression problems. Even more pleasing was the discovery that the same combination of techniques produces better performance when re-applied to a real-world problem in the musical domain.

7.2 Future Research Directions

Some possible avenues of future work now follow; for fitness function model building, it would be interesting to investigate if user behaviour can be made more consistent by examining more of the non-subjective characteristics gathered and applying lessons learned from them during early generations of the run. If it can be predicted when an artificial model is likely to perform poorly when building a model, then we can feed this back to the user to try to make better predictions possible.

It is true that when applied to a wide audience of subjects, there was

a mixture of results from the experiments described earlier in this thesis. However, let us briefly consider the type of user who is likely to benefit from this research. An active user of an Interactive Evolutionary system is likely to be doing so with the goal of discovering new, previously unknown ideas of artistic / creative merit for a particular problem. It is this goal-driven behaviour that sets such a user apart from the general class of user examined in previous experimentation. This same behaviour has a better chance of producing less noisy history data, making it easier to build models of the subjective fitness functions. If it is possible to identify this class of user(s), then it would be interesting to see how accurate the artificial models created are on a cohort of members of this class.

The serial evaluation of evolving musical candidates is a likely contributor to the *fatigued distraction* phenomenon described earlier in this thesis. Recent research has shown that serial evaluation of musical samples is not necessarily a prerequisite in order to select based on preferences (Fernström & McNamara, 2005). An interesting avenue of future work would be to investigate how a user interface permitting parallel evaluation of musical pieces leads to subsequent modelling by artificial means.

The central application of this research has been in the musical domain but it is not constrained to it. Evaluation of artistic pieces in the form of images may also be done in parallel (by showing more than one image on the screen at the same time). An investigation into the ability to model fitness functions in the visual arts would therefore be an interesting area of inquiry.

The field of Genetic Programming offers many opportunities for future research; improvements to the field are introduced every year however not all approaches consider generalisation performance. An examination of what makes a given improvement technique a good predictor constitutes a significant improvement to the field of research as a whole.

References

Back, Thomas, Fogel, David B., & Michalewicz, Zbigniew. 1997. *Handbook of Evolutionary Computation*. IOP Publishing Ltd.

Biles, J. A., Anderson, P. G., & Loggi, L. W. 1996. Neural Network Fitness Functions for a Musical IGA. *Pages B39–B44 of: Proceedings of the International ICSC Symposium on Intelligent Industrial Automation (IIA '96) and Soft Computing (SOCO '96).*

Biles, John A. 1994. GenJam: A genetic algorithm for generating jazz solos. *In: In ICMC Proceedings 1994. The Computer Music Association.*

Burton, A. R., & Vladimirova, T. 1999. Generation of Musical Sequences with Genetic Techniques. *Computer Music Journal*, **23**(4).

Costelloe, Dan. 2008. `gpsr`*: Genetic Programming for Symbolic Regression.* `http://gpsr.sourceforge.net`.

Costelloe, Dan, & Ryan, Conor. 2004. Genetic Programming for Subjective Fitness Function Identification. *Pages 259–268 of:* Keijzer, Maarten, O'Reilly, Una-May, Lucas, Simon M., Costa, Ernesto, & Soule, Terence (eds), *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*. LNCS, vol. 3003. Coimbra, Portugal: Springer-Verlag.

Costelloe, Dan, & Ryan, Conor. 2007. Towards models of user preferences in interactive musical evolution. *Pages 2254–2254 of:* Thierens, Dirk, Beyer, Hans-Georg, Bongard, Josh, Branke, Jurgen, Clark, John Andrew, Cliff, Dave, Congdon, Clare Bates, Deb, Kalyanmoy, Doerr, Ben-

jamin, Kovacs, Tim, Kumar, Sanjeev, Miller, Julian F., Moore, Jason, Neumann, Frank, Pelikan, Martin, Poli, Riccardo, Sastry, Kumara, Stanley, Kenneth Owen, Stutzle, Thomas, Watson, Richard A, & Wegener, Ingo (eds), *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, vol. 2. London: ACM Press.

Cramer, Nichael Lynn. 1985 (24-26 July). A representation for the Adaptive Generation of Simple Sequential Programs. *Pages 183–187 of:* Grefenstette, John J. (ed), *Proceedings of an International Conference on Genetic Algorithms and the Applications*.

Fausett, Laurene (ed). 1994. *Fundamentals of neural networks: architectures, algorithms, and applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Fernandez, Thomas, & Evett, Matthew. 1998. Numeric Mutation as an Improvement to Symbolic Regression in Genetic Programming. *Pages 251–260 of:* Porto, V. William, Saravanan, N., Waagen, D., & Eiben, A. E. (eds), *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming*. LNCS, vol. 1447. Mission Valley Marriott, San Diego, California, USA: Springer-Verlag.

Fernström, Mikael, & McNamara, Caolan. 2005. After Direct Manipulation – Direct Sonification. *ACM Trans. Appl. Percept.*, **2**(4), 495–499.

Gibson, P. M., & Byrne, J. A. 1991. NEUROGEN: musical composition using genetic algorithms and cooperating neural networks. *Pages 309–313 of: Second International Conference on Artificial Neural Networks*. New York: IEEE.

Goldberg, David E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, Massachusetts: Addison-Wesley Publishing Company.

Graupe, Daniel. 1997. *Principles of Artificial Neural Networks*. River Edge, NJ, USA: World Scientific Publishing Co., Inc.

Gustafson, Steven, Burke, Edmund K., & Krasnogor, Natalio. 2005. On Improving Genetic Programming for Symbolic Regression. *Pages 912–919 of:* Corne, David, Michalewicz, Zbigniew, Dorigo, Marco, Eiben, Gusz, Fogel, David, Fonseca, Carlos, Greenwood, Garrison, Chen, Tan Kay, Raidl, Guenther, Zalzala, Ali, Lucas, Simon, Paechter, Ben, Willies, Jennifier, Guervos, Juan J. Merelo, Eberbach, Eugene, McKay, Bob, Channon, Alastair, Tiwari, Ashutosh, Volkert, L. Gwenn, Ashlock, Dan, & Schoenauer, Marc (eds), *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, vol. 1. Edinburgh, UK: IEEE Press.

Harley, James. 2004. *Xenajis: His Life in Music*. London, UK: Taylor and Francis Books.

Haykin, Simon. 1994. *Neural Networks: A Comprehensive Foundation*. New York: Macmillan.

Holland, John H. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press.

Horner, A., & Goldberg, D. E. 1991. Genetic Algorithms and Computer-Assisted Music Composition. *Pages 437–441 of: Proc. of the Fourth International Conference on Genetic Algorithms*.

Horowitz, D. 1994. Generating rhythms with genetic algorithms. *Pages 142–143 of: Proceedings of the 1994 International Computer Music Conference*. San Francisco: ICMA.

Johanson, Brad, & Poli, Riccardo. 1998. GP-Music: An Interactive Genetic Programming System for Music Generation with Automated Fitness Raters. *Pages 181–186 of:* Koza, John R., Banzhaf, Wolfgang, Chellapilla, Kumar, Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max H., Goldberg, David E., Iba, Hitoshi, & Riolo, Rick (eds), *Ge-*

netic Programming 1998: Proceedings of the Third Annual Conference. University of Wisconsin, Madison, Wisconsin, USA: Morgan Kaufmann.

Keijzer, Maarten. 2003. Improving Symbolic Regression with Interval Arithmetic and Linear Scaling. *Pages 70–82 of:* Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., & Costa, E. (eds), *Proceedings of the Sixth European Conference on Genetic Programming (EuroGP-2003)*. LNCS, vol. 2610. Essex, UK: Springer Verlag.

Keijzer, Maarten. 2004. Scaled Symbolic Regression. *Genetic Programming and Evolvable Machines*, **5**(3), 259–269.

Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Luke, Sean, & Panait, Liviu. 2002. Is The Perfect The Enemy Of The Good? *Pages 820–828 of: GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Machado, P., Romero, J., Manaris, B., Santos, A., & Cardoso, A. 2003. Power to the Critics – A Framework for the Development of Artificial Critics. *Pages 55–64 of: Proceedings of 3rd Workshop on Creative Systems, 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*.

Machado, P., Romero, R., Santos, M. L., Cardoso, A., & Manaris, B. 2004. Adaptive Critics for Evolutionary Artists. *In: EvoMUSART Workshop, Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*. LNCS. Coimbra, Portugal: Springer-Verlag.

Machwe, Azahar T., & Parmee, Ian C. 2006 (May). Introducing Machine Learning within an Interactive Evolutionary Design Environment. *In: Proceedings of Design 2006, 9th International Design Conference*.

Majeed, Hammad, & Ryan, Conor. 2006. A re-examination of a real world blood flow modeling problem using context-aware crossover. *Chap. 14,*

pages – of: Riolo, Rick L., Soule, Terence, & Worzel, Bill (eds), *Genetic Programming Theory and Practice IV*. Genetic and Evolutionary Computation, vol. 5. Ann Arbor: Springer.

Manaris, Bill, Vaughan, Dallas, Wagner, Christopher, Romero, Juan, & Davis, Robert B. 2003. Evolutionary Music and the Zipf-Mandelbrot Law: Developing Fitness Functions for Pleasant Music. *Pages 522–534 of: Proceedings of the EvoWorkshops 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM*. LNCS, vol. 2611. Essex, UK: Springer Verlag.

Manaris, Bill, Machado, Penousal, McCauley, Clayton, Romero, Juan, & Krehbiel, Dwight. 2005. Developing Fitness Functions for Pleasant Music: Zipf's Law and Interactive Evolution Systems. *Pages 488–497 of:* Rothlauf, Franz, Branke, Juergen, Cagnoni, Stefano, Corne, David W., Drechsler, Rolf, Jin, Yaochu, Machado, Penousal, Marchiori, Elena, Romero, Juan, Smith, George D., & Squillero, Giovanni (eds), *Applications of Evolutionary Computing, EvoWorkshops2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*. LNCS, vol. 3449. Lausanne, Switzerland: Springer Verlag.

McCormack, Jon. 2005. Open Problems in Evolutionary Music and Art. *Pages 418–427 of:* Rothlauf, Franz, Branke, Juergen, Cagnoni, Stefano, Corne, David W., Drechsler, Rolf, Jin, Yaochu, Machado, Penousal, Marchiori, Elena, Romero, Juan, Smith, George D., & Squillero, Giovanni (eds), *Applications of Evolutionary Computing, EvoWorkshops2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*. LNCS, vol. 3449. Lausanne, Switzerland: Springer Verlag.

McIntyre, A. 1994. Bach in a box: The evolution of four part baroque harmony using the genetic algorithm. *Pages 852–857 of: Proc. IEEE Conference on Evolutionary Computation.*

(MMA), MIDI Manufacturers Association. 1996. *The Complete MIDI 1.0 Detailed Specification, v.96.1.*

O'Neill, Michael, & Ryan, Conor. 2003. *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*. Genetic programming, vol. 4. Kluwer Academic Publishers.

O'Sullivan, John. 2003. *An Investigation into Grammatical Evolution Search Strategies*. M.Phil. thesis, Deparment of Computer Science and Information Systems, University of Limerick, Limerick, Ireland.

Raja, Adil, Azad, R. Muhammad Atif, Flanagan, Colin, & Ryan, Conor. 2007. Real-Time, Non-Intrusive Evaluation of VoIP. *Pages 217–228 of:* Ebner, Marc, O'Neill, Michael, Ekárt, Anikó, Vanneschi, Leonardo, & Esparcia-Alcázar, Anna Isabel (eds), *Proceedings of the 10th European Conference on Genetic Programming*. Lecture Notes in Computer Science, vol. 4445. Valencia, Spain: Springer.

Ralley, D. 1995. Genetic algorithms as a tool for melodic development. *Pages 501–502 of: Proceedings of the 1995 International Computer Music Conference*. San Francisco: ICMA.

Ryan, Conor, & Azad, R. Muhammad Atif. 2003. Sensible Initialisation in Grammatical Evolution. *Pages 142–145 of:* Barry, Alwyn M. (ed), *GECCO 2003: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*. Chigaco: AAAI.

Ryan, Conor, Collins, J. J., & O'Neill, Michael. 1998. Grammatical Evolution: Evolving Programs for an Arbitrary Language. *Pages 83–95 of:* Banzhaf, W., Poli, R., Schoenauer, M., & Fogarty, T. C. (eds), *First European Workshop on Genetic Programming 1998*. Berlin: Springer.

Thywissen, Kurt. 1999. GeNotator: an environment for exploring the application of evolutionary techniques in computer-assisted composition. *Org. Sound*, **4**(2), 127–133.

Topchy, Alexander, & Punch, William F. 2001. Faster Genetic Programming based on Local Gradient Search of Numeric Leaf Values. *Pages 155–162 of:* Spector, Lee, Goodman, Erik D., Wu, Annie, Langdon, W. B., Voigt, Hans-Michael, Gen, Mitsuo, Sen, Sandip, Dorigo, Marco, Pezeshk, Shahram, Garzon, Max H., & Burke, Edmund (eds), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. San Francisco, California, USA: Morgan Kaufmann.

Truong, Brian. 2002. *Trancendence: An Artifical Life Approach to the Synthesis of Music*. M.Phil. thesis, School of Cognitive and Computing Sciences, University of Sussex, Brighton, UK.

Valigiani, Gregory, Fonlupt, Cyril, & Collet, Pierre. 2004. Analysis of GP Improvement Techniques over the Real-World Inverse Problem of Ocean Color. *Pages 174–186 of:* Keijzer, Maarten, O'Reilly, Una-May, Lucas, Simon M., Costa, Ernesto, & Soule, Terence (eds), *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*. LNCS, vol. 3003. Coimbra, Portugal: Springer-Verlag.

Vanneschi, Leonardo, Tomassini, Marco, Collard, Philippe, & Clergue, Manuel. 2005. A Survey of Problem Difficulty in Genetic Programming. *Pages 66–77 of:* Bandini, Stefania, & Manzoni, Sara (eds), *AI*IA 2005: Advances in Artificial Intelligence, 9th Congress of the Italian Association for Artificial Intelligence, Proceedings*. Lecture Notes in Computer Science, vol. 3673. Milan, Italy: Springer.

von Mayrhauser, A., & Vans, A. M. 1995. Program Understanding: Models and Experiments. *Pages 1–38 of:* Zelkowitz, Marvin (ed), *Advances in Computers*, vol. 40. Academic Press.

Wasserman, Larry. 2004. *All of Statistics: A Concise Course in Statistical Inference (Springer Texts in Statistics)*. Springer.

Werner, G.M., & Todd, P.M. 1997. Too many love songs: Sexual selection

and the evolution of communication. *Pages 434–443 of:* Husbands, P., & Harvey, I. (eds), *ECAL97*. Cambridge, MA: MIT Press.

Zipf, George K. 1949. *Human Behavior and the Principle of Least Effort*. Addison-Wesley (Reading MA).

Zongker, Douglas, & Punch, Bill. 1996. *lilgp 1.01 User's Manual*.