# GENETIC PROGRAMMING
# FOR
# ADAPTIVE DIGITAL SIGNAL PROCESSING

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND ELECTRICAL ENGINEERING
OF GLASGOW UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Anna Isabel Esparcia Alcázar
May 1998

A Angel, Pepa y Alicia.

# Abstract

This thesis is devoted to presenting the application of the Genetic Programming (GP) paradigm to a class of Digital Signal Processing (DSP) problems. Its main contributions are

- a new methodology for representing Discrete-Time Dynamic Systems (DDS) as expression trees. The objective is the state space specification of DDSs: the behaviour of a system for a time instant $t \geq t_0$ is completely accounted for given the inputs to the system and also a set of quantities which specify the state of the system. This means that the proposed method must incorporate a form of memory that will handle this information.

  For this purpose a number of node types and associated data structures are defined. These will allow for the implementation of local and time recursion and also other specific functions, such as the sigmoid commonly encountered in neural networks. An example is given by representing a recurrent NN as an expression tree.

- a new approach to the channel equalisation problem. A survey of existing methods for channel equalisation reveals that the main shortcoming of these techniques is that they rely on the assumption of a particular structure or model for the system addressed. This implies that knowledge about the system is available; otherwise the solution obtained will have a poor performance because it was not well matched to the problem.

  This gives a main motivation for applying GP to channel equalisation, which is done in this work for the first time. Firstly, to provide a unified technique for a wide class of problems, including those which are poorly understood; and secondly, to find alternative solutions to those problems which have been successfully addressed by existing techniques.

  In particular, in the equalisation of nonlinear channels, which have been mainly addressed with Neural Networks and various adaptation algorithms, the proposed GP approach presents itself as an interesting alternative.

- a new way of handling numerical parameters in GP, **node gains**. A node gain is a numerical parameter associated to a node that multiplies its output value. This concept

was introduced by (Sharman and Esparcia-Alcázar 1993) and is fully developed here.

The motivation for a parameterised GP is addressed, together with an overview of how it has been addressed by other authors. The drawbacks of these methods are highlighted: there is no established way of determining the number of parameters to use and their placement; further, unused parameters can be unnecessarily adapted while, on the other hand, useful ones might be eliminated. The way in which node gains overcome these problems is explained. An extra advantage is the possibility of expressing complex systems in a compact way, which is labelled "compacting effect" of node gains.

The costs of node gains are also pointed out: increase in the degrees of freedom and increased complexity. This, in theory, results in an increase of computational expense, due to the handling of more complex nodes and to the fact that an extra multiplication is needed per node. These costs, however, are expected to be of, at most, the same order of magnitude as those of the alternatives.

Experimental analysis shows that random node gains may not be able to achieve all the potential benefits expected. It is conjectured that optimisation of the values is needed in order to attain the full benefits of node gains, which brings along the next contribution.

- a mathematical model is given for an adaptive GP. As concluded from the previous point, adaptation of the values of the node gains is needed in order to take full advantage of them. A Simulated Annealing (SA) algorithm is introduced as the adaptation algorithm. This is put in the context of an analogy: the adaptation of the gains by SA is equivalent to the learning process of an individual during its lifetime.

  This analogy gives way to the introduction of two learning schemes, labelled Lamarckian and Darwinian, which refer to the possibility of inheriting the learned gains.

  The Darwinian and Lamarckian learning schemes for GP are compared to the standard GP technique and also to GP with random node gains. Statistical analysis, done for both fixed and time-varying environments, shows the superiority of both learning methods over the non-learning ones, although it is not possible at this stage to determine which of the two provides a better performance.

- a number of interesting results in the channel equalisation problem. These are compared to those obtained by other techniques and it is concluded that the proposed method obtains better or similar performance when extreme values (maximum fitness or minimum error) are considered.

# Acknowledgements

First and foremost I would like to thank my supervisor, Dr. Ken Sharman, whose experience, eagerness and passion for science were always an inspiration to me. From his obsession with perfection I also hope to have benefited (it would only be fair, since it has tortured me so often) and I sincerely hope he will not consider his efforts wasted.

Thanks also to Dr. David Fogel, who showed the utmost patience and helped me a great deal with the statistical analysis, and to Professors Wilkinson and Murray-Smith, who gave me some useful hints.

To my colleagues and people from Glasgow University, who helped to alleviate the sorrows of student life, to name but a few: Euan, Gary, Graham, Tom (my favourite technician) and the lads from the Centre for Music Technology. Special mention to Iain and Henrik, who helped me with LaTeX , to Jesús, who provided many an interesting philosophical discussion and to the *popmobility* teachers in the Sports and Recreation Service, who helped me keep fit while working on this thesis.

Last but not least, thanks to my parents and my sister, who were an endless source of support, both moral and financial. I hope they'll be proud enough to shed a few tears over this thesis.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Overview

As developments in the communications field succeed one another at a fast pace, the engineer is faced with increasingly complex problems which often cannot be approached with existing ("classical") techniques.

Natural algorithms present themselves as an alternative where more conventional methods fail. These are techniques that aim at exploiting some analogy to natural processes, based on the premise that Nature always finds robust solutions to existing problems.

This thesis develops an application of two such techniques, Genetic Programming (GP) and Simulated Annealing (SA), to a class of Digital Signal Processing problems. This chapter is meant as a brief introduction to both methods.

Firstly, the abstract class of Evolutionary Computation techniques is presented in section 1.2. Further down in the hierarchy is the class of Genetic Algorithms (GAs), whose main features are described in 1.3. Section 1.4 focuses on Genetic Programming. A succinct review of its characteristics is followed by an enumeration of the reasons that make it worth studying. One of main characteristics of GP, the representation via expression trees, is then studied here in more detail. Section 1.5 presents the SA algorithm, its main features and possible variants.

Finally, section 1.6 gives the summary of the thesis.

## 1.2 Background on Evolutionary Computation

Evolutionary Computation techniques employ simulated evolution to solve a wide variety of problems. The guiding principle of these techniques is the Darwinian *survival of the fittest.* Darwinian evolution is intrinsically a robust search and optimisation mechanism. Biological species have solved problems that involve chaos, chance, temporality and nonlinear interactivity (Fogel 1994). These characteristics are shared by many engineering problems, for which there exist no heuristic solutions or they provide unsatisfactory results.

In the neo-Darwinian paradigm, evolution is driven by four basic processes: reproduction, mutation, competition and selection. These processes are mimicked by Evolutionary Computation techniques, which are usually divided into three main groups: genetic algorithms, evolutionary programming and evolution strategies. All three techniques employ a population of candidate solutions [1] and explore the search space by successive application of one or more variation operators.

While evolutionary programming and evolution strategies rely on mutation as the main variation operator, genetic algorithms are characterised by the strong emphasis placed on crossover. Crossover refers to the exchange of genetic information between two or more individuals (the *parents*) resulting in a number of new ones (the *children*).

Within the main class of genetic algorithms, a subclass that stands on its own right is Genetic Programming (GP). The next sections begin by summarising the main characteristics of GAs and then proceed with the specific features of GP.

## 1.3 Genetic Algorithms (GAs)

The introduction of GAs is generally attributed to Holland (1975), although other authors had previously proposed similar algorithms to simulate genetic systems[2].

The implementation of a GA is typically done as follows:

1. A *fitness function* is defined to rate the performance of any potential solution to the problem at hand.

2. An *initial population* of prospective solutions is generated at random. Candidate solutions are also referred to as *individuals* or *chromosomes*, with elements called *genes.*

3. A *new population* is generated by selecting individuals from the previous one, in a way that gives preference to individuals with higher fitness values.[3]

[1] And for this reason they lend themselves easily to parallelisation.

[2] See (Fogel 1994) for references

[3] Alternatively, the problem could also be expressed in terms of the minimisation of some cost function. Because traditionally GAs and GP are implemented as maximisation problems the same approach will be taken here

4. *Crossover* is implemented by selecting random pairs among the population and exchanging segments of data between them.

5. *Mutation* takes place by changing the value of a randomly selected gene with a given probability

6. Steps 3 through 5 are repeated until some termination criterion is met (either a solution is found or time has expired).

For a more in-depth study of GAs the reader is referred to (Goldberg 1989).

## 1.4 Genetic Programming

### 1.4.1 Characteristics

As said above, GP is a subclass of the standard GA (Koza 1992, Koza 1994). The main characteristics that give it a separate status are usually listed as

**objective** The goal of GP is to evolve a *program*, in contrast with the standard GA, whose aim is usually to evolve *data*. The term program is taken in a loose sense, and can mean such dissimilar items as a mathematical function or a game playing strategy.

**representation** the individuals evolved in GP are symbolic expression trees, or S-trees, written in polish notation. This is explained in detail below.

**operators** due to the use of a tree representation, the crossover operator in GP is syntax-constrained. This means that only crossovers that produce syntactically correct trees are allowed.

Although there have been other approaches to program induction using evolutionary techniques, there are a number of characteristics that make GP worth studying. (O'Reilly 1995) lists them as follows:

- GP is robust, in the sense that it has been shown to perform successfully on a wide range of problems, providing a single, unified approach.

- GP uses a variety of expressive primitives, or *node types* (see below).

- Because it works with primitives, the representation of GP is more flexible than others such as Learning Classifier Systems (LCS) or Neural Networks (NNs). LCSs use if-then rules and NNs weights and network connections; hence they require that the solution "fits to" these specialised structures.[4]

The next section describes the expression trees and their components.

[4]Incidentally, it will be shown later that structures such as NNs can be evolved by GP

### 1.4.2 Expression Tree and Node Vector.

**Definition 1.1** *A* **symbolic expression tree** *(or, for brevity, an S-tree or tree) is a variable length string of symbols written in polish (prefix) notation which is constructed according to certain grammar rules.*

**Definition 1.2** *A* **node vector**, $\vec{n}$

$$\vec{n} = \{n_0\ n_1\ \cdots\ n_{\ell-1}\} \tag{1.1}$$

*is an alternative representation for an expression tree, where:*

$\ell \in \aleph$ *is the length of the tree,*
$n_i : \Re^{\lambda(n_i)} \Longrightarrow \Re, \qquad n_i \in \mathcal{O}; \quad i = 0 \ \ldots\ \ell - 1$
$\lambda(n_i)$ *is the arity i.e. the number of arguments (or inputs) of* $n_i$
$\mathcal{O}$ *is a set of allowed operations*

The symbols $n_i$ that constitute the tree/vector are called *nodes.*

**Definition 1.3** *A* **node** $n$ *is a primitive function with one output and* $\lambda(n)$ *inputs.*

The set of operations $\mathcal{O}$ can be represented as the union of two subsets:

$$\mathcal{O} = \mathcal{O}_F \cup \mathcal{O}_T \tag{1.2}$$

which are usually referred to as *function set* and *terminal set*, respectively. The terminal set is characterised by

$$\lambda(n) \ = \ 0 \qquad\qquad \forall n \in \mathcal{O}_T \tag{1.3}$$

and the function set by

$$\lambda(n) \ > \ 0 \qquad\qquad \forall n \in \mathcal{O}_F \tag{1.4}$$

For $\vec{n}$ to be a syntactically correct tree, two necessary and sufficient conditions must be satisfied. These are described below.

**Completeness** The sum of the arities of all the nodes in the tree must be equal to the length of the tree minus one, i.e.

$$\sum_{i=0}^{\ell-1} \lambda(n_i) = \ell - 1 \tag{1.5}$$

The completeness condition guarantees that the tree is complete, that is, there are no missing arguments. Equation 1.5 can be easily deduced from the following facts:

1. the length of the tree must be *at least* equal to $1 + \lambda(n_0)$, where $\lambda(n_0)$ is the arity of the first node (or root)

2. any given node $n_i$ must have a number of succeeding nodes greater than or equal to its arity $\lambda(n_i)$.

**Definition 1.4** *A node $n_i$ is* **connected** *to another node $n_j$ if 1) $n_i$ is an argument to $n_j$ or 2) $n_i$ is an argument to a node that is connected to $n_j$*

**Definition 1.5** *A* **subtree** *of length $\ell$ and root $n_i$ is the set comprising $n_i$ and the $\ell - 1$ succeeding nodes connected to it.*

**Hierarchical connectivity** The sum of the arities of the nodes preceding a given node $n_i$, $0 < i < \ell - 1$, must be greater than $i$, i.e.

$$\sum_{j=0}^{i} \lambda(n_j) > i \qquad \forall i \quad 0 < i < \ell - 1 \tag{1.6}$$

For $i = \ell - 1$ the equality applies; this is the completeness condition.

The connectivity condition ensures that all subtrees in a vector are connected to the root node by way of preceding nodes, i.e. there are no isolated subtrees.

Examples of correct and incorrect trees are the following

| | | |
|---|---|---|
| {+ + a b * c d} | $\Longrightarrow$ | correct: $a + b + (c \cdot d)$ |
| {+ + a b * c} | $\Longrightarrow$ | incorrect: doesn't comply completeness condition |
| {+ a b * c } | $\Longrightarrow$ | incorrect: doesn't comply connectivity condition |

The vectorial representation lends itself easily to mathematical treatment. Nevertheless, in standard GP the preferred representation is the polish notation. A characteristic of this type of notation is that it renders parentheses unnecessary. Using them, however, increases the readability of the expressions and for this reason they are widely applied. In this way, the first tree in the previous example would be represented as:

```
(+   (+ a b)   (* c d))
```

Any function can be written as an expression tree or node vector, provided that appropriate nodes are defined. For example, the two-input function

$$f(x_1, x_2) = x_1 + k \cdot x_2 \tag{1.7}$$

can be represented as a vector as follows

$$\vec{n} \quad = \quad \{ +, \; x1, \; *, \; k, \; x2 \} \tag{1.8}$$

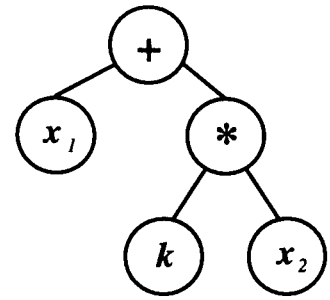


Figure 1.1: A simple expression tree representing equation 1.7.

or as an expression tree

```
(+ x1 (* k x2))
```

This tree is shown graphically in figure 1.1.

The output of the tree for a given pair of values of $x_1$ and $x_2$ would be the value of $f(x_1, x_2)$ for those particular values. If $x_1$ and/or $x_2$ are time–varying, the output of the tree is a time series.

**Definition 1.6** *To* **evaluate** *a tree in a particular environment (i.e. the problem at hand) is to obtain the output value of the function it represents.*

**Definition 1.7** *A* **branch** *of length $\ell$ and root $n_i$ is a subtree satisfying the Completeness and Hierarchical Connectivity conditions with respect to its root $n_i$, i.e.*

$$\sum_{j=i}^{\ell+i-1} \lambda(n_j) = \ell - 1 \tag{1.9}$$

$$\sum_{j=i}^{\ell+i-1} \lambda(n_j) > k \qquad \forall k: \quad i < k < \ell + i - 1 \tag{1.10}$$

In the example above, the subtree

```
(* k x2)
```

is a branch, as it can be seen as a tree in itself.

This definition is useful when specifying the mechanics of the crossover operator. Crossover is restricted by the fact that only syntactically correct trees can be generated. In practice, this means that only whole branches (and not just subtrees) can be exchanged; this is considered as the main feature of Genetic Programming.

## 1.5 Background on Simulated Annealing

Simulated Annealing (SA) is a stochastic searching strategy based on an analogy to the annealing process in statistical mechanics (i.e. the behaviour of systems with many degrees of freedom in thermal equilibrium at a finite temperature).

If a molten substance is cooled quickly (rapid quenching), it will solidify into a defective crystal. However, if the substance is slowly cooled, thermal diffusion will cause agitation of the particles, which will probabilistically fall into minimum energy configurations.

In simulated annealing, a system is "heated" up to its melting point and then the temperature is slowly reduced; the system being allowed to reach thermal quasi- equilibrium in each temperature step. The thermal noise is simulated by introducing random perturbations, the variance of which is dependent on the temperature.

First introduced by (Metropolis *et al.* 1953) it was not until the 80's that it became extensively applied to a number of optimisation problems (Kirkpatrick *et al.* 1983, Szu and Hartley 1987).

A Simulated Annealing algorithm is characterised by the following:

1. a perturbation-generating probability distribution
2. an acceptance/rejection probability distribution
3. a cooling schedule

For a full account of the SA algorithm, the reader is referred to appendix A.

## 1.6 Summary of the thesis.

Chapter 2 provides the key for the representation of Discrete–Time Dynamic Systems as expression trees, which constitutes the first main contribution of this thesis.

First of all the objective is enunciated as the state space specification of DDSs: the behaviour of a system for a time instant $t \geq t_0$ is completely accounted for given the inputs to the system and also a set of quantities which specify the state of the system. This means that the GP must incorporate a form of memory that will handle this information.

For this purpose a number of node types and associated data structures are defined. These will allow for the implementation of local and time recursion and also other specific functions, such as the sigmoid commonly encountered in neural networks. An example is given by representing a recurrent NN as an expression tree.

Chapter 3 introduces the channel equalisation problem, for which two examples will be given. A survey of existing methods for channel equalisation reveals that the main shortcoming of these techniques is that they rely on the assumption of a particular structure or model for the system addressed. This implies that knowledge about the system is available; otherwise the solution obtained will have a poor performance because it was not well matched to the problem.

This gives a main motivation for applying GP to channel equalisation, which is done in this work for the first time. Firstly, to provide a unified technique for a wide class of problems,

including those which are poorly understood; and secondly, to find alternative solutions to those problems which have been successfully addressed by existing techniques.

Chapter 4 presents the second main contribution of this thesis: a new way of handling numerical parameters in GP, *node gains*. A node gain is a numerical parameter associated to a node that multiplies its output value. This concept was introduced by (Sharman and Esparcia-Alcázar 1993) and is fully developed here.

The motivation for a parameterised GP is addressed, together with an overview of how it has been addressed by other authors. The drawbacks of these methods are highlighted: there is no established way of determining the number of parameters to use and their placement; further, unused parameters can be unnecessary adapted while, on the other hand, useful ones might be eliminated. The way in which node gains overcome these problems is explained. An extra advantage is the possibility of expressing complex systems in a compact way, which is labelled "compacting effect" of node gains.

The costs of node gains are also pointed out: increase in the degrees of freedom and increased complexity. This, in theory, results in an increase of computational expense, due to the handling of more complex nodes and to the fact that an extra multiplication is needed per node. These costs, however, are expected to be of, at most, the same order of magnitude as those of the alternatives.

Experimental analysis shows that random node gains may not be able to achieve all the potential benefits expected. It is conjectured that optimisation of the values is needed in order to attain the full benefits of node gains. This sets the scene for the following chapter.

In Chapter 5 a Simulated Annealing (SA) algorithm is introduced as a means of adapting the values of the node gains. This, the third main contribution of this thesis, is put in the context of an analogy: the adaptation of the gains by SA is equivalent to the learning process of an individual during its lifetime. This analogy gives way to the introduction of two learning schemes, labelled Lamarckian and Darwinian, which refer to the possibility of inheriting the learned gains. Both are compared to the standard GP technique and also to GP with random node gains. The comparison shows the superiority of both learning methods, although it is not possible at this stage to determine which of the two provides a better performance.

In Chapter 6, the channel equalisation problem is revisited and more results are provided using the full node gain GP+SA method. These results are compared to those obtained by other techniques and it is concluded that the proposed method obtains better or similar performance when extreme values (maximum fitness or minimum error) are considered.

Finally, Chapter 7 presents a summary of the conclusions and outlines areas of future research.

# Chapter 2

# Representing Discrete–Time Dynamic Systems as Expression Trees

## 2.1 Introduction

This chapter is concerned with providing a representation for Discrete–Time Dynamic Systems (DDS's) as symbolic expression trees, which are the structures typically employed by Genetic Programming. These S–trees, or individuals, are computer programs written in the language defined by the representation (Kinnear 1994). The fitness function then executes these programs to assign a performance measure, or fitness value, to each one of them.

The S–trees are composed of nodes whose interpretation will be specific of the problem. The aim of this chapter is to provide definitions of the node types that will allow for the representation of discrete–time systems.

The rest of the chapter is structured as follows. Section 2.2 provides a description of the systems that are to be represented as S–trees. Section 2.3 focuses on the components of the S–trees: what requirements they must satisfy and the various kinds of nodes that will be employed in the remainder of this work. Section 2.4 describes the data structures needed by the nodes defined in Section 2.3 and how they operate when the S–tree is executed. In Section 2.5 an example is presented. Section 2.6 gives some details of the software implementation and finally Section 2.7 concludes the chapter.

## 2.2 State Space Specification of Discrete–Time Dynamic Systems

Consider a single output single input discrete–time system with input $X_t$ and output $Y_t$, as shown in figure 2.1.


Figure 2.1: A single input single output system.

The state of the system at instant $t_0$ is defined as a set of quantities which, together with the input for all instants $t \geq t_0$ , uniquely determines the output for all instants $t \geq t_0$ (Priestley 1988).

The system can be described by the equations :

$$\mathbf{s}_{t+1} = A(\mathbf{s}_t) + B(\mathbf{x}_t) \tag{2.1}$$

$$\mathbf{y}_t = C(\mathbf{s}_t) + D(\mathbf{x}_t) \tag{2.2}$$

where $\mathbf{x}_t$ is the input vector, $\mathbf{y}_t$ is the output vector and $\mathbf{s}_t$ is the state vector. The functions $A(\cdot)$, $B(\cdot)$, $C(\cdot)$ and $D(\cdot)$ are matrices whose components are scalar–valued nonlinear functions.

The aim of this chapter is to express equations 2.1 and 2.2 in a way that can be handled by Genetic Programming, so that discrete–time dynamic systems can be evolved.

This is done by writing the function that provides the current system output, $\mathbf{y}_t$ , as an expression tree. The components of the state and input vectors are updated and saved using special data structures.

Although for simplicity in the remainder of this work only single input / single output systems are considered, the techniques introduced could be easily extended to multiple input multiple output systems.

The following sections discuss in detail the components of the expression trees and the necessary data structures.

## 2.3 Node types

### 2.3.1 Requirements

The nature of the nodes $n_i$ has to be adapted to the problem at hand. In the case of Discrete-Time Dynamic Systems (or, in particular, Digital Signal Processing algorithms) they must be chosen so that the representation system has the following main characteristics:

1. the ability to capture the dynamics of the system, which is of vital importance in DSP.

2. an efficient way to represent and adapt the parameters of the system.

The former is achieved by the combined use of special tree nodes and data structures that endow the system with memory and local and time recursion ability.

This section is concerned with the definition of such a class of nodes, the associated data structures will be dealt with in section 2.4. The second requirement is the subject of chapters 4 and 5.

### 2.3.2 Basic node types

To represent Discrete–Time Dynamic Systems the following sets of operations are of interest:

$$O_F = \{ +,\ -,\ *,\ /,\ *2,\ /2,\ +1,\ -1,\ \texttt{nlN},\ \texttt{psh},\ \texttt{Z},\ \texttt{fN},\ \texttt{avgN} \} \tag{2.3}$$

is the function set and

$$O_T = \{ \texttt{1},\ \texttt{cN},\ \texttt{xN},\ \texttt{yN},\ \texttt{stkN},\ \texttt{argN} \} \tag{2.4}$$

is the terminal set.

The basic types are those used to implement arithmetic operations. These are addition, `+`, subtraction, `-`, multiplication, `*`, and protected division, `/` [1]. Other basic types used throughout this work are increment, `+1`, decrement, `-1`, double, `*2`, and half, `/2`. The meaning of the other nodes is as follows.

### 2.3.3 Time recursion nodes

These node types allow access to previous values of inputs, outputs and other internal variables. The index $N$, which will appear in what follows, represents an integer in a user-defined range, $[0 \cdots N_{max}]$, where the value $N_{max}$ varies according to the type of node.

- *Input node*, `xN`, and *output node*, `yN`, are terminals ($\lambda = 0$) representing the input to and output from the system, respectively, both delayed by $N$ samples.

$$\texttt{xN} = x_{n-N} \tag{2.5}$$

$$\texttt{yN} = y_{n-N} \tag{2.6}$$

- *Delay node*, `Z`, is a function of arity 1 returning the value of its argument delayed by one time sample.

$$\texttt{Z}(arg_n) = arg_{n-1} \tag{2.7}$$

The implementation of these nodes requires the use of two kinds of data structures (registers and stacks) which will be described in section 2.4.

[1] The protected division returns the quotient between the first argument and the second when the latter is different from zero, and zero otherwise. Alternatively, in the second case a high value could be returned.

### 2.3.4 Local recursion nodes

With the `yN`, nodes defined in the previous section recursion from the output of the tree can be achieved by accessing the output value $N$ instants before. This section introduces nodes that allow for local recursion within the tree. These are called **psh** and **stkN**. The former is a function of arity 1; when evaluated it pushes the value returned by the branch below it onto a stack. The latter is a terminal (i.e. its arity is zero) and returns the value of the $N^{th}$ position in the stack.

Internal recursion is important for developing modular solutions to certain problems. For example, the biquadratic digital filter section in canonical form is described by the coupled equations, (Proakis and Manolakis 1992),

$$p_n = x_n + c_1 \cdot p_{n-1} + c_2 \cdot p_{n-2} \tag{2.8}$$

$$y_n = p_n + c_3 \cdot p_{n-1} + c_4 \cdot p_{n-2} \tag{2.9}$$

A possible tree coding for equations 2.8 and 2.9 using **psh** and **stkN** nodes is,

```
(+ (psh  (+ x0 (+ (* c1 stk0) (* c2 (Z stk0))))  )(+ (* c3 stk0) (*
c4 (Z stk0))))
```

In this expression tree, the sub-tree framed evaluates the term $p_n$ and pushes this value onto the stack memory ready for the next cycle. The `stk0` node therefore returns the value, $p_{n-1}$, which can be delayed by the `Z` node to get $p_{n-2}$.

It could be argued that an equivalent result can be achieved by using `Z`, `x` and `Y` nodes only. In DSP practice, however, this would imply a loss of significant digits in the obtained parameters which doesn't occur when internal recursion is used (Proakis and Manolakis 1992).

An alternative way of achieving internal recursion would be keeping track of the output of every node in the tree in each evaluation (thereby eliminating the need for **psh** nodes) and defining a node type, similar to **stk**, that would address this information in the next evaluation. This is left for further study.

### 2.3.5 Special nodes

The nodes defined above are sufficient for the representation of discrete–time systems. It is interesting, however, to define other node types to perform special functions. A few of these are introduced here.

- *Non-linear transfer function*, `nlN`, implements a sigmoid function,

$$g(x) = \frac{1 - \exp^{-\beta x}}{1 + \exp^{-\beta x}} = \tanh\left(\frac{\beta}{2}x\right) \tag{2.10}$$

where the amount of non-linearity $\beta$, is a linear function of the index $N$ as follows

$$\beta(N) = \beta_{lo} + \frac{N}{N_{max}}(\beta_{hi} - \beta_{lo}), \qquad \beta \in [\beta_{lo} \ \ldots \ \beta_{hi}] \tag{2.11}$$

The range $[\beta_{lo} \ldots \beta_{hi}]$, is partitioned into $N_{max}$ equally spaced subintervals that provide $N_{max+1}$ possible values for $\beta$; the parameter $N$ simply addresses each of these values, e.g. for `nl0` $\beta = \beta_{lo}$ and for `nlNmax` $\beta = \beta_{hi}$.

The numerical values used throughout this work are

$$\beta_{hi} = 10 \tag{2.12}$$

$$\beta_{lo} = 0.1 \tag{2.13}$$

$$N_{max} = 255 \tag{2.14}$$

so `N0` represents the curve $\tanh(0.05x)$ and `N255` represents $\tanh(5x)$; the latter is approximately equivalent to the sign function, while the former gives a very smooth transition between -1 and 1.

The sigmoid function implemented by the `nlN` node is of interest when it is an objective to evolve functions with the structure of a Neural Network.

- *Function node*, `fN`, executes the $N^{th}$ subroutine tree. These, also called automatically defined functions (Koza, 1994) are in every respect the same as any other function used by the main tree except that they can have a variable number of arguments. Subroutine trees are intrinsic to a particular main tree and are created and evolve together with it, not being accessible by any other trees. Thus, an expression tree would be properly defined as the set of a main tree and all its associated subroutine trees, if any.

  Function nodes are important in addressing the problem of scalability, (i.e. the increment in the size of the expression trees as the complexity of the system increases). Code reuse by means of function nodes provides a compact way of expressing repetitive tasks, so complex systems can be expressed as small trees.

- *Argument node*, `argN`, is the $N^{th}$ argument to a function node. These appear only in the definition of the function as terminals and are replaced by their corresponding values in the main tree.

- *Average node*, `avgN`, returns the average of its $N$ inputs.

- *Constant node*, `cN`, returns the $N_{th}$ entry of a constant table, whose values can be randomly initialised or preselected by the user.

Table 2.1: The Function and Terminal Node Set for Signal Processing Systems

| Symbol | Arity ($\lambda$) | Description | Symbol | Arity ($\lambda$) | Description |
|---|---|---|---|---|---|
| `+, -` | 2 | Addition, Subtraction | `xN` | 0 | System input data. $N$ indicates the delay†. (e.g. `x2` returns $x_{n-2}$) |
| `*, /` | 2 | Multiplication, Division If second argument is 0, then the node output is set to zero or a large maximum value | `yN` | 0 | Previous output from the expression tree. The index $N$ indicates the delay (e.g. `y2` returns $y_{n-2}$) |
| `+1,-1` | 1 | Increment, Decrement | `Z` | 1 | Unit sample time delay |
| `*2, /2` | 1 | Multiply, Divide by two | `psh` | 1 | Push the argument value onto the stack |
| `cN` | 0 | Constant value. $N$ is an index to a table of constants whose values may be predefined or chosen at random. | `stkN` | 0 | Retrieve the $N^{th}$ item from the stack. |
| `nlN` | 1 | Non-linear transfer function. $N$ indicates the amount of non-linearity. | `fN` | variable | Execute the $N^{th}$ function tree. |
| `avgN` ‡ | $N$ | The average of its $N$ arguments. | `argN` | 0 | The $N^{th}$ argument to a function tree. |

†The suffix N that appears in many of the node symbols is an integer in a user defined range.
‡The nodes `fN`, `argN` and `avgN` are not used in the present implementation.

## 2.4 Data structures

A number of structures are necessary to handle data and keep track of the state of the dynamic system. These are:

**registers** Fixed-size vectors that store current and previous values of inputs and outputs. These values are read by `xN` and `yN` nodes.

**local recursion reading and writing stacks** In order to maintain data coherency, two stacks are used whose size equals the maximum number of `psh` nodes allowed in a tree. In a given iteration of the evaluation process (e.g. for the $n^{th}$ input) one of the stacks is used for writing whenever a `psh` node is encountered; the other stack is used for reading whenever an `stk` is encountered. In the next evaluation (for the $n + 1^{th}$ input) the two stacks are swapped and what was written before is now read.

**Z reading and writing stacks** The Z stacks have as many positions as `Z` nodes appear in a particular tree (up to a certain maximum). During the evaluation of a tree, when a particular `Z` node is found the value at the associated position of the reading stack is returned as output of the node and its input is stored in the same position of the writing stack.

**constant table** A fixed vector that stores the values which will provide the output of `cN` nodes.

Since the trees are evaluated sequentially[2] the data structures are common for all trees.

At the beginning of the evaluation process of a particular tree the registers are set to zero and the stacks are empty. The constant table has been initialised with random (or prefixed) values.

An evaluation iteration begins by reading and storing the current system input onto the top position of the X (or input) register.

As explained, any `xN` or `yN` nodes will read their return values from the $N^{th}$ position of the relevant register; `stkN` nodes read theirs from the $N^{th}$ position of the reading stack and `psh` nodes write the value of their argument onto the current position of the writing stack, returning that same value.

`Z` nodes work with both Z stacks simultaneously: the input value is written onto the current position of the writing stack and the return value is read from the same position of the reading stack.

The evaluation iteration ends by storing the output value of the tree onto the top position of the Y register. Then the registers are shifted one position back[3] and the reading and writing

[2] At this stage no parallel processing is considered

[3] This is more efficiently implemented by means of a circular buffer; thus, rather than shifting the whole contents of the register, a single pointer is advanced

Table 2.2: Node types for DSP and their associated data structures

| node | arity | action | reads from | writes to |
|---|---|---|---|---|
| xN | 0 | retrieve $N^{th}$ value of the X register | X register | - |
| yN | 0 | retrieve $N^{th}$ value of the Y register | Y register | - |
| z | 1 | push value of argument to Z writing stack<br>retrieve same position of Z reading stack | Z reading stack | Z writing stack |
| psh | 1 | push value of argument to stack | - | writing stack |
| stkN | 0 | retrieve $N^{th}$ position of the stack | reading stack | - |
| cN | 0 | retrieve $N^{th}$ constant value | constant table | - |

stacks are swapped, leaving the system ready for the next iteration.

Due to the mechanics of the process all the values stored in the Z stacks will be used during evaluation. This is not the case for the local recursion stacks, as there might be more **psh** nodes than the highest N for **stk** nodes. The opposite is also possible: the value of N might be higher than one plus the number of **psh** nodes. In this case, N is clipped to one minus the number of **psh** nodes; if there are none, then all **stk** return zero.

This means that redundancy might arise, e.g. a tree could have **stk** nodes but no **psh** ones, or vice versa. This could be avoided if, as explained in section 2.3.4, the output of all nodes was stored. Further study would be required to show which way of proceeding is more efficient in terms of memory requirements and computation expense.

Figure 2.2 shows the state of the system during evaluation of a tree when processing a certain sample $i$. For the next sample, $i + 1$, the registers are shifted one position and the reading and writing stacks are swapped (so that what was written in iteration $i$ can be read in iteration $i + 1$).

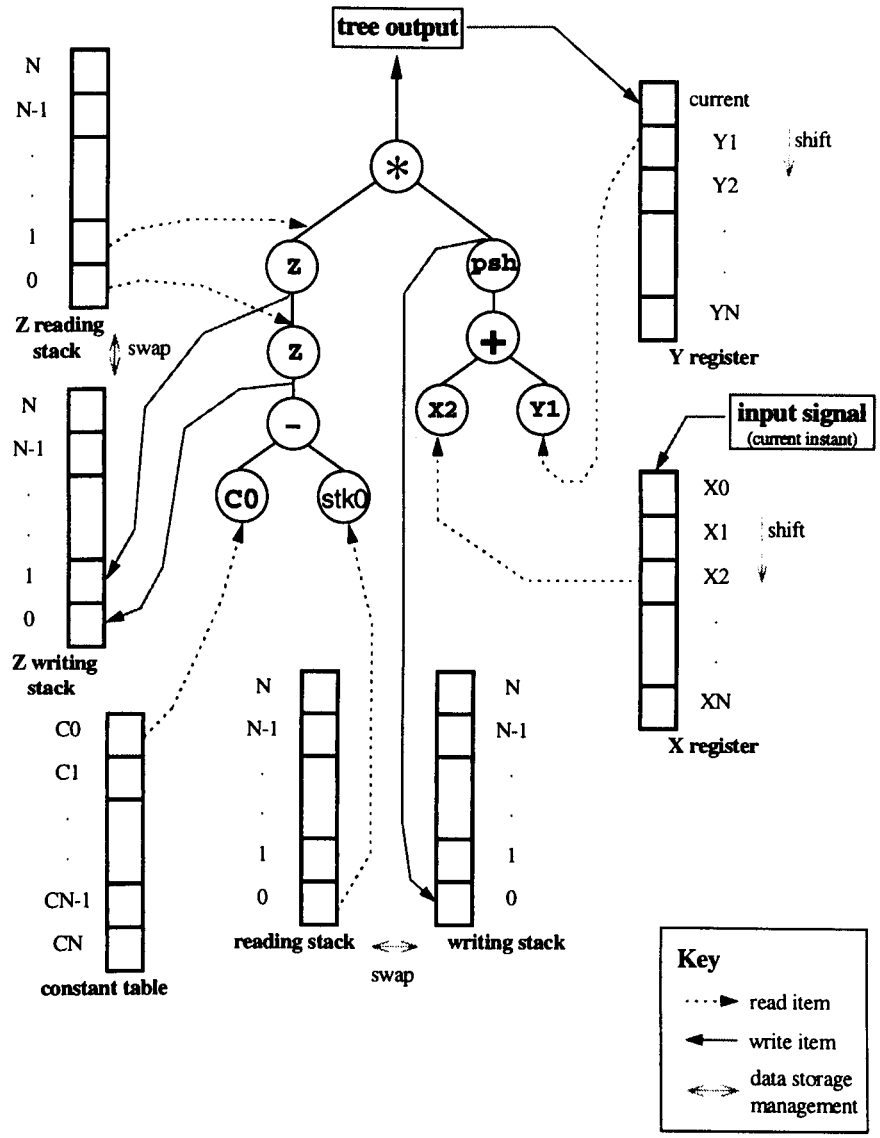


Figure 2.2: The tree above and its associated structures represent the equations

$$p_n = x_{n-2} + y_{n-1}$$
$$y_n = C0 \cdot p_{n-3} \cdot (x_{n-2} + y_{n-1})$$

The first step in obtaining the output at instant $i$ consists of reading the input signal at $i$ and storing it in the X register. Evaluation ends by storing the calculated output of the tree in the Y register, swapping the stacks and shifting the registers. Note that the combination of `psh` and `stk` nodes produces internal recursion. Also note how the output depends not only on the current input value but also on what was stored in the stacks and registers, which provide the state of the system. The only structure that remains unaltered through evaluation is the constant table. This is initialised with random or predetermined values at the beginning of the run

## 2.5 Example - A Recurrent Neural Network

It has been shown that a three-node recurrent neural network can be used as an efficient signal processor for equalisation of noisy non-linear communications channels, (Kechriotis *et al.* 1994). The system architecture of such a neural network is shown in Figure 2.3. Using the node definitions given above, one way of expressing this as a GP tree is to use function nodes as follows,

```
y  = ( + f1 ( * c0 (+f2 f3)))
f1 = ( psh ( nl1 avg4 ( x0 stk0 stk1 stk2 )))
f2 = ( psh ( nl2 avg4 ( x0 stk0 stk1 stk2 )))
f3 = ( psh ( nl3 avg4 ( x0 stk0 stk1 stk2 )))
```

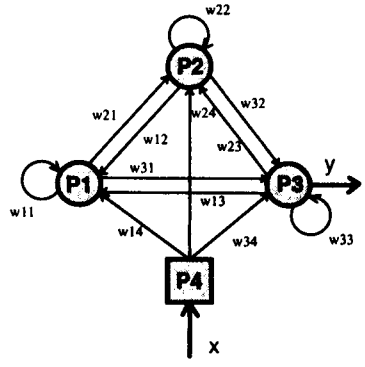


Figure 2.3: A Fully Recurrent Neural Network.
Each processing cell labelled P1 to P3 implements a sigmoidal transfer function on the average of the cell's input values. The connecting links have independent strengths labelled $w_{ij}$. The system output is taken from cell P3 and the input is applied to each cell simultaneously.

Here, each cell in the network is represented by its own function (`f1`, `f2` and `f3`) which are invoked by the main tree, `y`. The main tree executes the tree associated to each function, but only returns the value of the output node, which is represented by function `f1` (assuming `c0` = 0). The `psh` nodes in each function tree store the computed cell outputs on the stack, and these are accessed from the previous cycle using the `stkN` nodes.

Note that, for simplicity, the weights in the neural network have not been represented in the tree; the way to do this will be explained in chapter 4. Here it is sufficient to point out that *each occurrence* of a node (e.g. every `stk0` node that appears in the tree) has a different weight value, so that all the $\omega_{ij}$ would be expressed.

Thus, the basic set of GP node definitions shown above is able to code for recurrent neural network architectures. This leads to the possibility that neural networks and other such systems can be evolved by GP.

## 2.6 Software implementation

The software used in the experiments presented in this work is an extensively modified version of GPCPP4, a package developed by Adam Fraser of the University of Salford. GPCPP4 was the only public domain GP package in C++ existing at the time of starting this research. This was thought interesting for two reasons. First, due to the possibility of developing and working with an Object Oriented GP package. And second, because it would allow linking with software for Signal Processing that had already been developed in C++ by the author.

Converting GPCPP4 into **GP+SA**, the code required for the experiments described later in this thesis, involved major modifications which were undertaken by this author. Of these, some were necessary, others were meant to facilitate the understanding of what the program was doing and the rest were conceived in order to ease the realisation of experiments.

A summary of these modifications is described in the following sections.

### 2.6.1 Main additions

These mainly involved creating the necessary structures the DSP operations would use.

**Input, output and desired output signals** These are instances of previously created class Signal and are used during evaluation to calculate the fitness of each individual. The first two provide the values for xN and yN nodes; the first and last are equal for all individuals and the middle one is particular of each one.

**Registers and stacks** Two registers are needed for keeping track of previous values of the inputs and outputs, which are associated to `xN` and `yN` nodes. Two stacks handle `psh` and `stkN` nodes and two more handle `Z` nodes. The way these structures work is shown in figure 2.2.

**Node gains** As will be explained in chapter 4, each node may have a gain value associated to it. This is the basis of the learning approach using SA

**Indexing** Some of the nodes differ only by a parameter (e.g. `x1` and `x2`), so instead of redefining them for each value, the parameter is accessed by an index.

**Fitness type and return value of a node** These were defined as `unsigned int` in GPCPP4, but due to the nature of the problems tackled in this work and the fact that the implementation is done in a PC with a Pentium processor, their type can be changed to `double`.

### 2.6.2 Other improvements

**Initialisation** In GPCPP4 a few parameters values were read from an initialisation file, others were passed from the command line and the vast majority were part of the code

itself, which meant that when a minor change in a variable was made it was necessary to recompile the whole program.

This has been eased by the introduction of a new class, `EnvironmentManager`. An instance of this class is created for each run, which is in charge of reading all the necessary parameters from an initialisation file and passing them to wherever they are needed.

An example of an initialisation file is given in appendix E.

**Termination criteria** In GPCPP4 the termination criterion was simply reaching a given number of generations. This does not seem appropriate for two reasons. First of all, in *steady state* GP there is no proper definition of "generation" (it was artificially defined as a number of crossovers, including the ones that were aborted, equal to the population size). And second, the number of generations is just a form of time limit, which does not give any information on whether an optimal solution has been reached or not. Instead, five other criteria are introduced, as follows

- maximum fitness criterion, $t_{fit}$

$$t_{fit} = (max_i(fitness_i) \geq \texttt{MAX\_FITNESS}) \quad \text{where} \quad i \in [0, \texttt{PopulationSize} - 1] \tag{2.15}$$

- minimum Bit-Error-Rate criterion, $t_{ber}$

$$t_{ber} = (min_i(BER_i) \leq \texttt{MIN\_BER}) \text{where} \quad i \in [0, \texttt{PopulationSize} - 1] \tag{2.16}$$

- time limit criterion, $t_{tlim}$

$$t_{tlim} = (\text{elapsed\_time} \geq \texttt{TIME\_LIMIT}) \tag{2.17}$$

- maximum number of births criterion, $t_{Nbirths}$

$$t_{Nbirths} = (\texttt{births} \geq \texttt{BIRTH\_LIMIT}) \tag{2.18}$$

  where *births* is the number of individuals born

- user request criterion, $t_u$

$$t_u = (kbhit() \cap (getchar() == s)) \tag{2.19}$$

  i.e. $t_u = 1$ when the user presses s

The global termination criterion, $t_c$, is defined as the combination of one or several individual criteria.

$t_c$ is checked every time after a number of births equal to `ReportingInterval`. The program is stopped if any of the individual criteria is met. The values of the variables (`MAX_FITNESS`, `MIN_BER`, `TIME_LIMIT`, `BIRTH_LIMIT` and `ReportingInterval`) are read from the initialisation file (see Appendix E for an example).

**ReadTree** This is a read-and-evaluate routine used to calculate the fitness, output and/or bit-error-rate of any given tree after evolution. Its importance stems from two reasons: firstly, it allows to test solutions with a set of data different from that used during evolution; secondly, it serves as a test of the performance of the main program.

## 2.7 Conclusions

It has been shown here how the equations describing discrete–time systems can be represented as symbolic expression trees. Various kinds of node types and their associated data structures have been described to complete the representation system. The power of this system has been illustrated by an example, the coding of a Recurrent Neural Network as a list of S-trees.

So far, the first of the necessary requirements described in Section 2.3.1 has been fulfilled, namely capturing the dynamics of discrete–time systems. Further chapters will cover the second requirement: the representation and adaptation of the parameters of the systems.

# Chapter 3

# Channel Equalisation

## 3.1 Introduction

This chapter investigates the application of GP to channel equalisation. This is an application of great topical interest to the Signal Processing community. A review of the literature shows that currently a variety of algorithms are applied to this problem. These include neural networks (NNs), infinite impulse response (IIR) filters and finite impulse response (FIR) filters.

All these techniques provide tailored solutions to particular problems: a specific model structure (linear or nonlinear) is assumed and then the associated parameters are optimised by some adaptation algorithm. The success of each method at solving the problem at hand is then highly dependent not only on the characteristics of the algorithm, but also on whether or not the structure adopted is convenient for the problem at hand.

Under these circumstances it seems appealing to search for a general technique that can be applied to a wide class of problems. This implies being able to adapt the structures of the solutions and not only the parameters.

For this reason, the channel equalisation problem presents itself as an ideal test bed for the Genetic Programming method.

A number of applications will be presented here that have been addressed in the past with different equalisation methods. It will be shown in this work that all these applications will be tackled within a unified GP-based approach.

The chapter is structured as follows. Section 3.2 provides a background of current approaches to channel equalisation, pointing out the interest of applying GP to this problem. In Section 3.3 the implementation of GP in channel equalisation is explained. Section 3.4 concentrates on two specific problems to which the proposed method was successfully applied. Some conclusions are given in section 3.5 and finally section 3.6 summarises the chapter.

## 3.2 Background on Channel Equalisation

*Intersymbol interference* (ISI) is a phenomenon that arises when information is transmitted through bandwidth limited communication channels. If left uncompensated, ISI can be an important source of errors. All the techniques devoted to removing ISI at the receiver's end are called *channel equalisation techniques* (Proakis 1995, p. 636, Haykin 1996, p. 217).

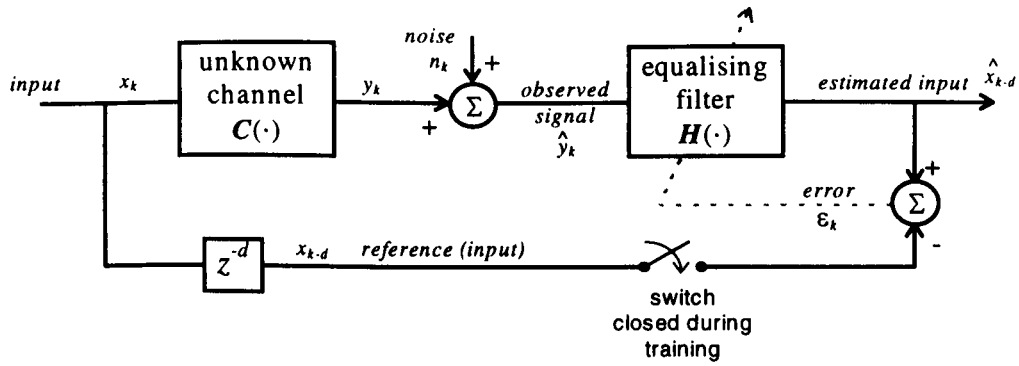


Figure 3.1: The Channel Equalisation Problem.
The unobservable input sequence, $x$, is distorted by the channel $C(\cdot)$ and corrupted by the additive noise, $n$. The objective of the equalising filter is to restore x from the noisy observations, $\hat{y}$. The lower part of the diagram indicates a signal path used during trained adaptation of the restoring filter.

Figure 3.1 shows the block diagram of a generic equalising system. Initially, the system undergoes a *training* process, which involves sending a known signal (the *training sequence*) that acts as a reference. The error is calculated as the difference between this signal and the actual output of the equaliser.

Once the training process is finished, the transmission of the data begins. At this stage some form of *test signal* can be transmitted, in order to measure the performance of the equaliser and the success of the training.

### 3.2.1 Linear equalisation

A common technique for removing ISI is labelled linear equalisation and consists of adapting the coefficients of a transversal, or FIR, filter (see Figure 3.2) until some cost function is minimised. For the problem to be easily addressed mathematically, the cost function must be a linear function of the filter's coefficients. A popular choice is the mean squared error (MSE).

Let the error signal, $\{\epsilon_k\}$, be the difference between some desired response $\{d_k\}$ and the

actual filter output, $\{y_n\}$.

$$\epsilon_n = y_n - d_n \tag{3.1}$$

The MSE is defined as

$$MSE = E(\epsilon_k^2) \tag{3.2}$$

where $E(\cdot)$ is the mathematical expectation.

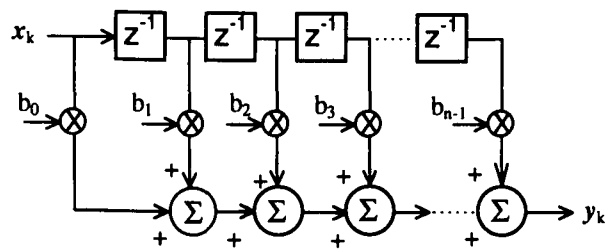


Figure 3.2: An $n$-tap FIR digital filter. The filter coefficients are represented by $b_i$ and $z^{-1}$ represents a unit time delay.

The problem can be formulated as follows. Given a channel whose transfer function is $C(z)$ the ideal equaliser must comply that its transfer function $H(z)$ is the inverse of $C(z)$, i.e.

$$H(z) = \frac{1}{C(z)} \tag{3.3}$$

This is referred to as *inverse filtering.*

If the channel impulse response is modelled as an Auto Regressive (AR) process, the effective transfer function of the channel is

$$C(z) = \frac{1}{b_0 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2} \cdots + b_n \cdot z^{-n}} \tag{3.4}$$

In this case the appropriate equaliser is an FIR filter as follows

$$H(z) = b_0 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2} \cdots + b_n \cdot z^{-n} \tag{3.5}$$

A widely used method for adapting the coefficients $\{b_i\}$ for this class of problems is the recursive least squares (RLS) algorithm (see appendix D for details).

When the channel is better modelled as a moving average (MA) or, more generally, as an auto regresive moving average (ARMA) system, an equaliser such as the one given in equation 3.5 may not be sufficient, even when the number of taps ($n$) is high. In such cases, an IIR equaliser may be employed, such as the one given by 3.6

$$H(z) = \frac{b_0 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2} \cdots b_n \cdot z^{-n}}{a_0 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2} \cdots a_m \cdot z^{-m}} \tag{3.6}$$

Despite providing reduced computational complexity, IIR equalisers have been traditionally less employed because the two fundamental approaches to the adaptation of the coefficients $a_i$ and $b_j$, known as equation-error and output-error methods, present major problems (Shynk 1989). The equation-error approach can lead to biased estimates of the coefficients. On the other hand, the output-error approach can converge to a local minimum of the error surface (which is not quadratic and may have multiple local minima), leading to an incorrect estimate of the coefficients. A trade-off must be found between the two.

Furthermore, adapting the coefficients of an IIR equaliser involves an additional problem: stability must be guaranteed by ensuring that all poles of $H(z)$ are inside the unit circle.

Recently other approaches have been taken to the adaptation of IIR equalisers, such as Evolutionary Programming (EP) (Chelapilla *et al.* 1997), Genetic Algorithms (GAs) (Etter *et al.* 1982, Ma and Cowan 1996) and Simulated Annealing (SA) (Nambiar and Mars 1992), showing the potential of these techniques.

Both instances of inverse filtering (with FIR and IIR equalisers) are dependent on two factors:

1. the channel $C(\cdot)$ is linear
2. the inverse of $C(z)$ is realisable. This implies that $C(z)$ has no zeros on or outside the unit circle; if it did, its inverse $H(z)$ would have poles on or outside the unit circle, and would therefore be oscillatory or unstable.

An additional problem appears when the channel has deep spectral nulls, i.e. zeros inside but close to the unit circle. Linear equalisers tend to compensate a deep null by placing a high gain at that frequency. This results in noise enhancement and a poor performance.

### 3.2.2 Nonlinear equalisation

In other situations the channel $C$ will have nonlinear distortion. Such channels are found, for example, in data transmission over digital satellite links, especially when the signal amplifiers operate in their high gain limits (Kechriotis *et al.* 1994). If the distortion is severe, linear equalisers perform poorly. Nonlinear equalisers must then be employed but, in general, the mathematical treatment of such models is complex.

An alternative has been found in neural networks. Neural networks, such as multilayer perceptrons (MLPs) and Radial Basis Function (RBF) networks have been applied to channel equalisation. This is done at the expense of turning the equalisation problem into a classification one: the transmitted data are assumed to be symbols belonging to some finite alphabet and the network, acting as a classifier, must determine which symbol was transmitted.

One important drawback of NNs lies in the determination of the structure: there exists no established procedure for determining the number of layers and nodes (Mulgrew 1996).

The second main problem of MLPs and RBF networks is their being feedforward structures. When nonlinearity is the main impairment, feedforward NNs perform well. This is the case of the examples reported in (Chen *et al.* 1990, Gibson *et al.* 1991, Theodoridis *et al.* 1995). It was shown how, for the high levels of noise involved in these examples, nonlinear classifiers were required.[1]

However, for higher values of the signal to noise ratio (SNR) (as should be expected in a telephone channel, for instance) the need for nonlinear compensation is balanced or overcome by the need for recurrence, or feedback.

To be able to cater for this, feedforward NNs require a large number of nodes, which increases their complexity. This hinders the study of their behaviour, as well as their hardware implementation, which prevents their use in real time applications (Nair and Moon 1997*a*, Nair and Moon 1997*b*, Nair and Moon 1995).

More recently, equalisation with recurrent neural networks (RNNs) has also been reported in the literature (Kechriotis *et al.* 1994, Parisi *et al.* 1997). Their less wide use is due to the complexity of the training algorithm, which may become unstable.

RNNs have the advantage of being more compact than their feedforward counterparts, but the issue of determining the structure remains.

### 3.2.3 A new approach

In view of all the problems involved in linear and nonlinear equalisation methods, it is desirable to find an equalisation technique that allows for adaptation of the structure, while catering at the same time for recurrence and nonlinearity.

Thus, taking into account the properties of Genetic Programming and the tree representation described in chapters 1 and 2, the scene is set for addressing the channel equalisation problem with GP.

## 3.3 Genetic Programming in Channel Equalisation

In the equalisation methods shown above, the structure of the equalising filter had to be selected by the user. Another constraint regarded the selection of a cost function to guide the adaptation process. This was chosen in order to ease mathematical tractability. It is the aim of this section to show how these constraints are relaxed when applying GP.

### 3.3.1 Fitness function

It was shown in chapter 2 how discrete time systems could be represented as expression trees to be evolved by GP. All that is necessary now in order to employ GP in channel equalisation

[1] As pointed out above, these constitute cases of the detection problem, rather than equalisation.

is to define a suitable fitness function. This will be a measure of the "goodness" of the candidate solutions.

In early implementations of GAs and GP the fitness was used directly as the probability of survival, hence it was defined in the interval [0,1]. This is no longer the case in the present implementation, but, for the sake of convenience, the same interval has been adopted. Thus, the fitness range is constant and independent of the problem tackled [2].

A simple way of achieving this is to define a cost function $J(\cdot)$ as some suitable error measure and then calculate the fitness, $f$, as follows

$$f = \frac{1}{1+J} \tag{3.7}$$

Hence, for $J \geq 0$, this results in $0 \leq f \leq 1$.

In linear equalisation the cost function had to be a linear function of the equaliser's coefficients, thus the wide use of the MSE. On the other hand, GP is not restricted by such constraints and any error measure can be used. These include

- the maximum absolute error,

$$\max |\epsilon_i|$$

  or MABS; this criterion is usually referred to as $H_\infty$ or minimax (Proakis 1995, p. 607),

- the average absolute error,

$$\frac{1}{N}\sum_{i=1}^{N} |\epsilon_i|$$

- and the average exponential error,

$$\frac{1}{N}\sum_{i=1}^{N} e^{-|\epsilon_i|^2}$$

An alternative definition of fitness stems from viewing the equalisation problem as a decision problem or a pattern classification task and is based on the bit-error rate (BER), which is defined as the fraction of misclassified symbols in the transmitted sequence[3]. This exploits the binary nature of the transmitted data, which is assumed to belong to the alphabet {-1, 1 }.

Since $BER \in [0,1]$ , $f$ may be defined as

$$f = 1 - BER \tag{3.8}$$

[2]The interest of this will be seen in further chapters.

[3]I.e. the number of incorrectly classified symbols divided by the total number of symbols in the transmitted sequence

which again results in $0 \leq f \leq 1$.

This may be contrasted with NN implementations, in which the MSE is generally employed during training but the performance tested using the BER.

Other characteristics of the output signal, such as the frequency response, can also be used to define a fitness function. Any of these criteria can be employed in isolation or combined with others. This is regarded as an important advantage of evolutionary techniques over classical methods.

## 3.4 Experiments

### 3.4.1 Objectives

Experiments were conducted using two channels with different characteristics. The aims of the experiments were as follows. Firstly, to show how GP can be applied to channel equalisation using the node types and data structures described in chapter 2. Secondly, to study the structure of the systems thus obtained and compare them with that of the theoretical equalising filters. To attain this aim, the impulse response and the response to a sinewave are studied. And finally, to compare the performance of the GP-evolved filters with that of a number of FIR filters, with different numbers of taps, adapted by the RLS algorithm.

### 3.4.2 Theoretical equalising filters

For the first experiment the channel model employed is

$$y_n = \frac{1}{2}\left(x_n + x_{n-1}\right) \tag{3.9}$$

Because the output of the channel is the average of the current and previous inputs, it will be referred to as *averaging channel.*

This channel has a zero on the unit circle. Hence, its inverse will have a pole on the unit circle, which means the ideal equaliser its an *oscillator*, whose difference equation is

$$y_n = 2 \cdot x_n - y_{n-1} \tag{3.10}$$

The second experiment involved a nonlinear channel, modelled by the following equations

$$y_n = c_n + 0.15 \cdot c_n^2 + 0.01 \cdot c_n^3 + n_n \tag{3.11}$$

$$c_n = 0.3482 \cdot x_n + 0.8704 \cdot x_{n-1} + 0.3482 \cdot x_{n-2} \tag{3.12}$$

Figure 3.3 gives the model for this channel.

Because the channel is nonlinear its inverse is not defined; hence there is no theoretical equaliser for this channel.

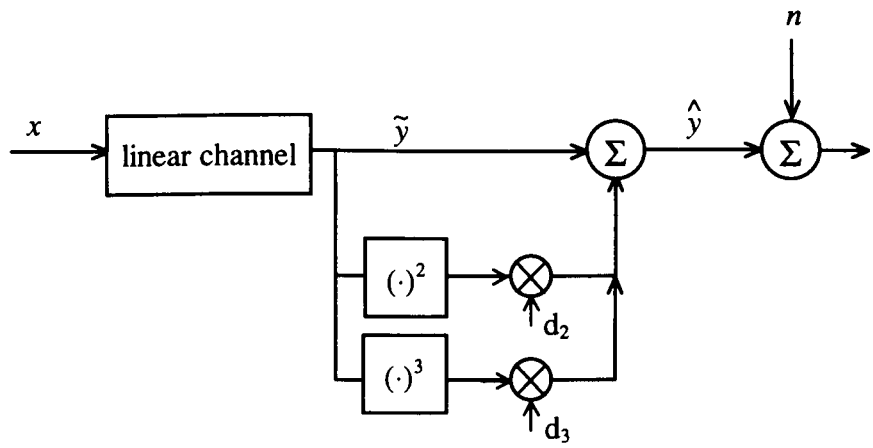


Figure 3.3: Model for the nonlinear channel used in the experiments

### 3.4.3 Averaging channel

#### No noise

A pseudo-random binary signal (PRBS) of 250 samples was used to train both a GP equaliser and eight FIR- RLS equalisers, whose order ranged from 3 to 10 taps.

The set up for the GP run is summarised in Table 3.1.

The GP-evolved equaliser was as follows:

```
( (NL232 ( + X1 ( + ( NL232 ( + STK1 X1 ) ) ( + ( % ( - ( + ( %
( - ( + X1 ( + X0 X0 ) ) STK3 ) 1 ) ( + ( + ( NL232 ( - ( + X1
X0 ) STK3 ) ) ( + X0 ( + ( + ( + X0 ( + ( % ( - ( - X1 ( + X0 X1
) ) X2 ) 1 ) ( - X1 ( * ( % X0 STK0 ) ( -1 ( +1 X0 ) ) ) ) ) ) (
+ ( % ( - ( + ( % ( - ( + X1 ( + ( + X0 X1 ) STK3 ) ) STK3 ) 1 )
( + ( + ( NL232 ( - ( + X1 X0 ) STK3 ) ) X0 ) X1 ) ) Y1 ) 1 ) (
+ ( + X0 X1 ) X0 ) ) ) X0 ) ) ) STK3 ) ) Y1 ) 1 ) X1 ) ) ) ) )
```

The fitness of this tree at the end of the evolution process (i.e. with the training signal) was 1.

The length of the tree is 91 and its depth is 21, but this can be reduced by editing and the tree greatly simplified. In particular, it must be noticed that, since there are no psh nodes, the value of `stk0`, `stk1` and `stk3` is zero. The simplified tree is shown in figure 3.4.

The GP equaliser was then tested with a PRBS of 1000 samples and the performance compared with that of several FIR-RLS equalisers with different number of taps. The results are summarised in Table 3.2. The GP-evolved equaliser beats the smaller FIR equalisers, both in terms of BER and fitness (which is equivalent to MSE). Although some of the longer

FIR equalisers also achieve a BER of zero, their fitnesses are consistently below 1.

We now proceed to study the behaviour of the GP equaliser in more detail. First we study the impulse response, which is plotted in figure 3.5.

Figure 3.5 shows that the GP equaliser is indeed an oscillator, as was the ideal inverse of equation 3.10. Furthermore, the two impulse responses (the theoretical and the obtained by GP) only differ in the first sample.

Next, the response to a sinewave of equation

$$s_n = \sin(0.1n)$$

is plotted in figure 3.6.

It can be observed that, as would be expected due to the presence of NL nodes, the obtained solution is nonlinear, since the response to a sinewave is a square wave rather than another sinewave. It can also be concluded that the GP equaliser behaves as a saturated amplifier.

### Signal to noise ratio SNR = 5 dB

The GP-evolved equaliser for this case was as follows

```
( ( NL74 ( + X0 ( + C2 X1 ) ) ) )
```

where C2 is 2. This tree is shown in figure 3.7.

This tree is much smaller than the one in the previous example. This is motivated by the phenomenon of noise enhancement (Gibson *et al.* 1991), which, in highly noisy environments, causes that equalisers of high order have a higher error than lower order ones. Hence, in this situation, a smaller, lower order equaliser would be preferable. The same can be observed for FIR-RLS equalisers.

The performance of the GP and RLS equalisers is given in table 3.3. It can be noticed that GP achieves similar performance than the RLS algorithm for this case.

Table 3.1: Set up for noiseless and SNR=5dB cases of the averaging channel ($C(z) = 0.5 + 0.5 \cdot z^{-1}$) equalisation experiment

| | |
|---|---|
| Function set | `+ - * % +1 -1 *2 /2 1 Z PSH NL0...NL255` |
| Terminal set § | `X0...X3 Y1...Y2 C0...C255 STK0...STK4` |
| $\beta$ limits for `NL` nodes | $\beta_{hi} = 10$ $\beta_{lo} = 0.1$ |
| Population size | 500 |
| Mutation probability | 0.01 |
| Size restrictions | at creation: maximum depth = 6<br>at crossover: none |
| Input signal (X) | output of the channel when fed with a Pseudo-Random Binary Signal (PRBS) |
| Reference signal | the same PRBS delayed by one sample |
| Fitness function | $\frac{1}{1+MSE}$ $0 \leq f \leq 1$ |
| Number of training samples | 250 |
| Termination criterion for each run | maximum fitness = 1<br>or 30 minutes of CPU time |
| Number of testing samples | 1000 |

§The 256 entries in the constant table are chosen uniformly within the interval [-1,1]

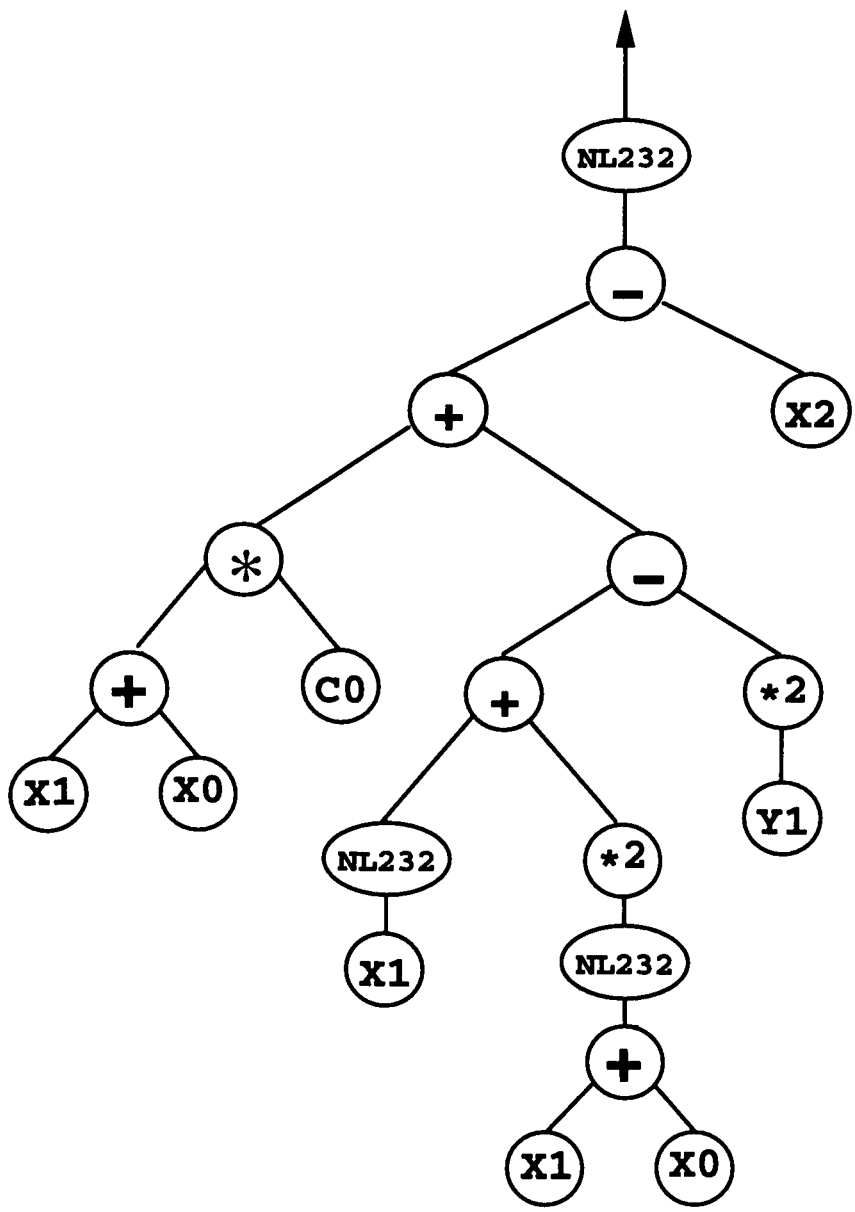


Figure 3.4: A GP equaliser for the averaging channel $C(z) = \frac{1}{2}(1 + z^{-1})$ (after editing). The value of the constant C0 is taken to be 8 and the node NL232 implements the function $\tanh(4.55x)$

Table 3.2: Performance summary of FIR-RLS and GP-evolved equalisers for averaging channel –noiseless case

| Equaliser | fitness | BER |
|---|---|---|
| FIR 3 taps | 0.800076 | 0.065 |
| FIR 4 taps | 0.832898 | 0.033 |
| FIR 5 taps | 0.855144 | 0.019 |
| FIR 6 taps | 0.873442 | 0.003 |
| FIR 7 taps | 0.888787 | 0 |
| FIR 8 taps | 0.900752 | 0 |
| FIR 9 taps | 0.909973 | 0 |
| FIR 10 taps | 0.918320 | 0 |
| GP-evolved | 1 | 0 |

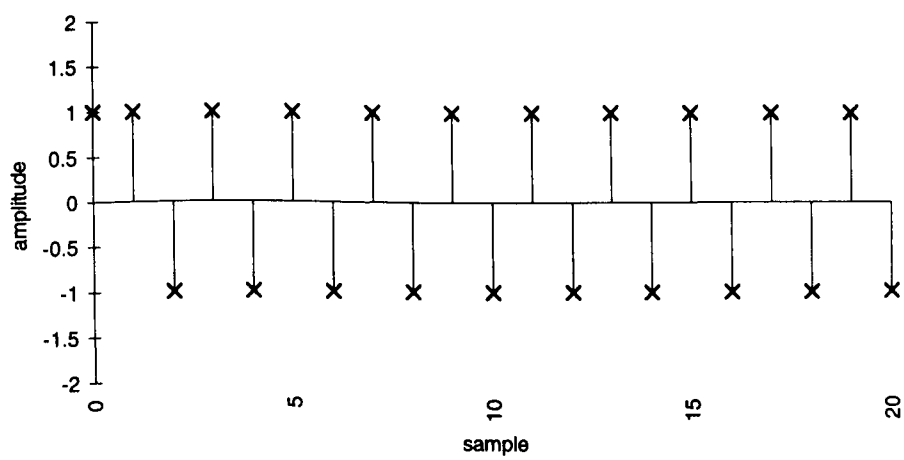


Figure 3.5: Impulse response of GP equaliser for averaging channel – Noiseless case

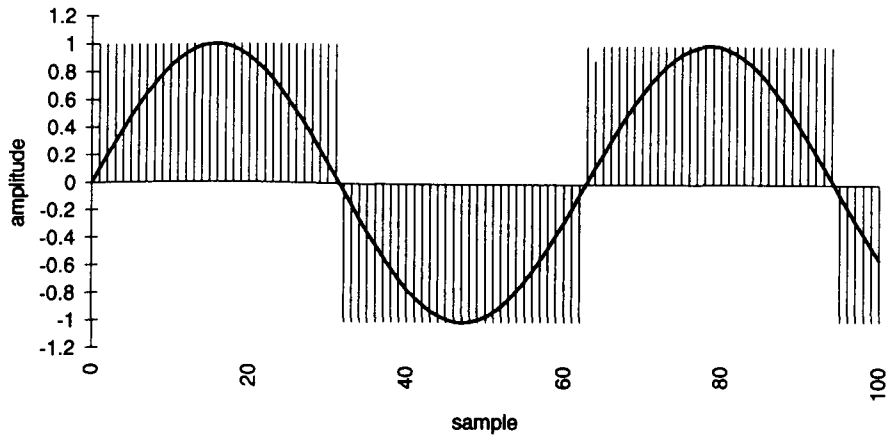


Figure 3.6: Output of GP equaliser when fed with a sinewave

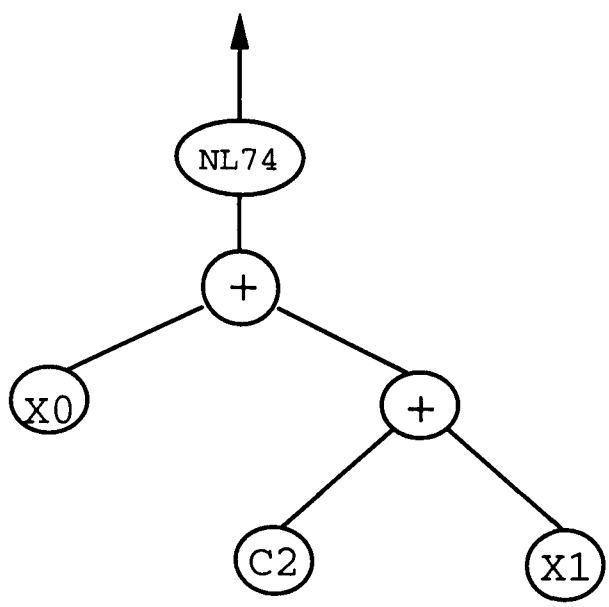


Figure 3.7: A GP equaliser for averaging channel with SNR = 5 dB. The value of C2 is 2 and the node NL74 implements the function $\tanh(1.49x)$.

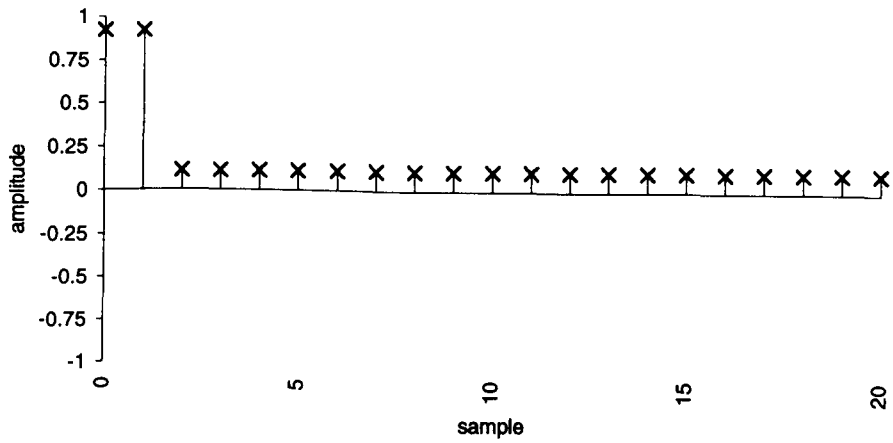


Figure 3.8: Impulse response of GP equaliser for averaging channel – SNR = 5 dB

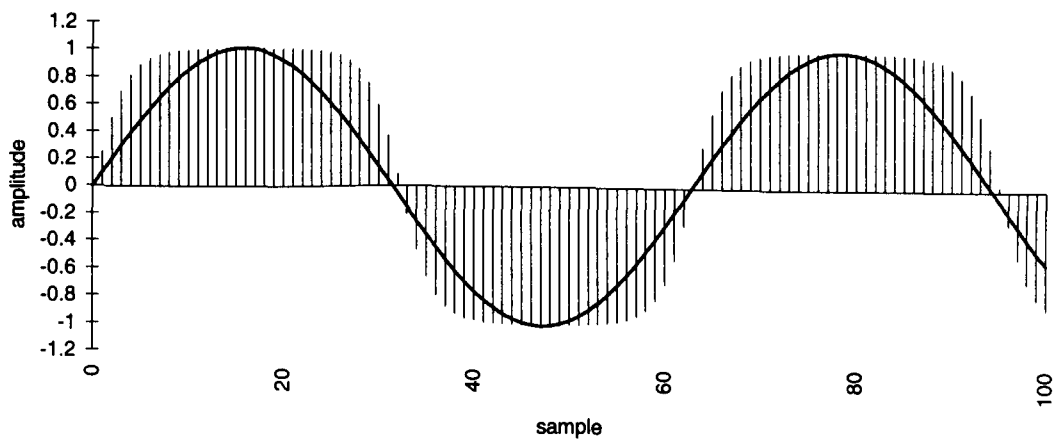


Figure 3.9: Response to a sinewave of GP equaliser for averaging channel – SNR = 5 dB

Table 3.3: Performance summary of GP-evolved and FIR-RLS equalisers for averaging channel (SNR= 5dB). The test signal was a PRBS of 1000 samples

| Equaliser | fitness | BER |
|---|---|---|
| FIR 3 taps | 0.659632 | 0.158 |
| FIR 4 taps | 0.65997 | 0.157 |
| FIR 5 taps | 0.657134 | 0.169 |
| FIR 6 taps | 0.65505 | 0.179 |
| FIR 7 taps | 0.653528 | 0.181 |
| FIR 8 taps | 0.653275 | 0.18 |
| FIR 9 taps | 0.65321 | 0.185 |
| FIR 10 taps | 0.654391 | 0.178 |
| GP-evolved | 0.648368 | 0.179 |

### 3.4.4 Nonlinear channel

For this channel the definition of fitness employed was based on the BER rather than the MSE, as was the case for the previous example. Hence,

$$f = 1 - BER \tag{3.13}$$

The linear part of the channel, given by equation 3.12, is a mixed phase channel. Such channels are usually equalised introducing a delay in the reference (Chen *et al.* 1990). This, however, was not done here in order to show the potential of the GP method.

The set up for the GP run is summarised in Table 3.4.

The equaliser obtained was

```
( ( - ( - ( + ( - X0 C11 ) ( - X0 ( Z X0 ) ) ) ( / ( NL123 ( Z
C17 ) ) X0 ) ) 1 ) )
```

where C11 = -0.577828 and C17 = 1.40172.

This is shown in figure 3.10

During evolution the fitness of this tree was 1, i.e. the BER was 0. The tree was then tested with a further 1000 samples, for which the fitness based on the MSE and the BER were measured, giving values of 0.213564. and 0, respectively.

Table 3.4: Set up for nonlinear channel equalisation experiment

| | |
|---|---|
| Function set | + - * % +1 -1 *2 /2 1 Z<br>PSH NL0...NL255 |
| Terminal set § | X0...X3 Y1...Y2<br>C0...C255 STK0...STK4 |
| $\beta$ limits for NL nodes | $\beta_{hi} = 10$ $\beta_{lo} = 0.1$ |
| Population size | 500 |
| Mutation probability | 0.01 |
| Size restrictions | at creation: maximum depth = 6<br>at crossover: none |
| Input signal (X) | output of the channel when fed with a Pseudo-Random Binary Signal (PRBS) |
| Reference signal | the same PRBS |
| Fitness function | $f = 1 - BER$ $0 \leq f \leq 1$ |
| Number of training samples | 250 |
| Termination criterion for each run | maximum fitness = 1<br>or 30 minutes of CPU time |
| Number of testing samples | 1000 |

§The 256 entries in the constant table are chosen uniformly within the interval [-1,1]

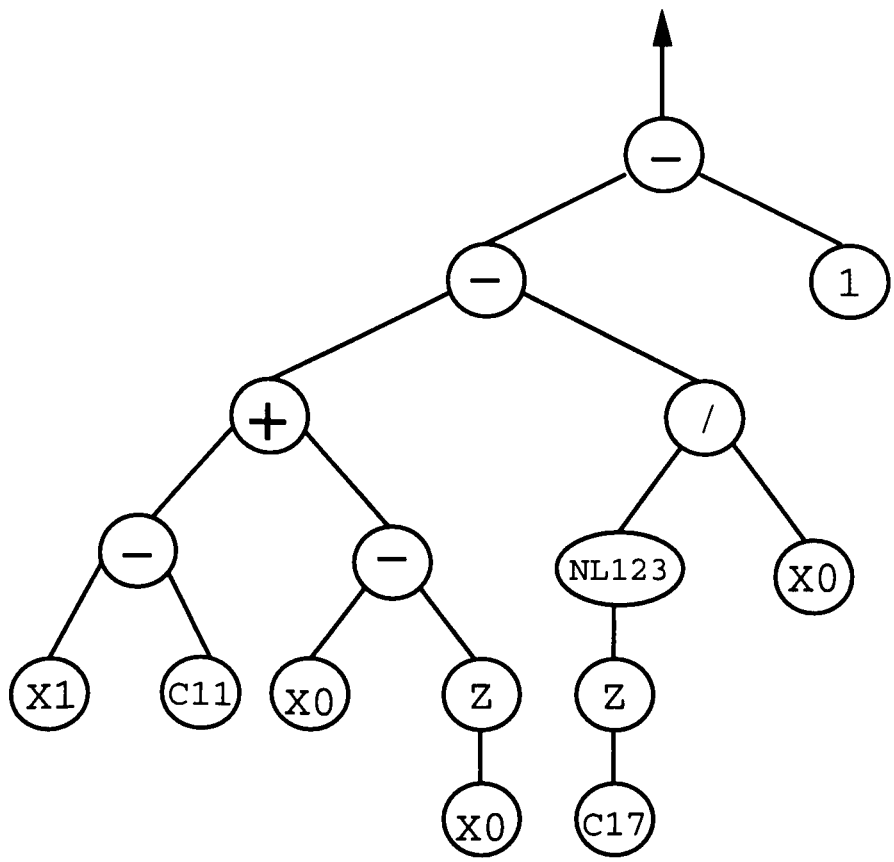


Figure 3.10: A GP equaliser for the nonlinear channel given by equations 3.11 and 3.12 (no noise). The values of the constants are C11 = -0.577828 and C17 = 1.40172. The node NL123 implements the function $\tanh(2.44x)$

## 3.5 Conclusions

It has been shown here how GP can be applied to channel equalisation using the node types defined in chapter 2.

GP serves to overcome problems of classical equalisation techniques by allowing the structure of the solution to evolve rather than being preselected by the user. In this way, nonlinear and recursive structures can be obtained.

Stability monitoring is intrinsic to the evolution process: unstable candidate solutions will yield a high value of the error and therefore a low fitness and will eventually be culled from the population.

However, in the examples presented here the evolved solutions tended to be complex or did not yield a good performance. It would be desirable to decrease complexity in some

cases (in order to increase readability and facilitate possible implementation) and improve the performance in others.

This points will be addressed in later chapters.

## 3.6 Summary

Channel equalisation is a commonly encountered problem in digital communications and a number of techniques to address it have been extensively reported in the literature. These techniques tend to perform well under certain assumptions but they all share a common failing: the structure of the solution must be determined in advance. Also, the cost function employed for adaptation of the structure's parameters must be selected in order to ease mathematical treatment.

GP has been presented here as a way to avoid these problems. Because the structure of potential solutions is allowed to evolve, nonlinearity and recurrence can arise without being specifically introduced by the designer. Also, the selection of a fitness, or alternatively, a cost function, is not restricted by the constraints of other methods: any user-defined function that associates a candidate solution to a numerical value can be employed.

The application of GP to channel equalisation has been illustrated with two examples, which show the great potential of the method.

Issues of interest are reducing the complexity of the solutions and improving their performance. These will be dealt with in further chapters.

# Chapter 4

# Node gains

## 4.1 Introduction

This chapter introduces node gains (Sharman and Esparcia-Alcázar 1993) as a means of representing the values of numeric parameters in Genetic Programming.

Firstly the motivations for a parameterised GP are explained. This is followed by a review of previous approaches to parameter estimation in GP, whose drawbacks are subsequently analysed. These shortcomings justify the exploration of alternatives and it is in this context that node gains are introduced.

The concept of node gains will be explained and it will be shown how they waive the shortcomings of other techniques, as well as attaining other benefits.

The costs of node gains will also be analysed and it will be shown how these can be outweighed by the advantages.

Experimental analysis will show that although not all the potential benefits are achieved at this preliminary stage, developments in forthcoming chapters will guarantee that this is the case.

## 4.2 Parameter estimation in Genetic Programming

### 4.2.1 Motivation for a parameterised GP

Let us assume a simple GP system, which employs the following function set:

$$\mathcal{O}_F = \{ +, -, *, / \} \tag{4.1}$$

and the terminal set

$$\mathcal{O}_T = \{ X \} \tag{4.2}$$

where $X$ is a variable (e.g. an input to the system).

Even though no constant numerical values are explicit in this representation, they can be implicitly generated, as shown in the following examples.

Consider the tree in figure 4.1. This, assuming $X \neq 0$, codes for the number 1. Correspondingly, the tree in figure 4.2 codes for the number zero.

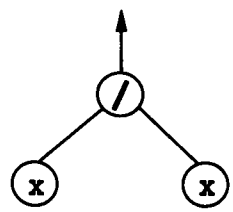


Figure 4.1: A tree representing the number one

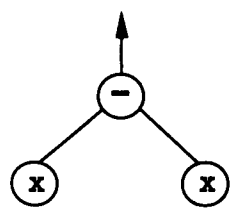


Figure 4.2: A tree representing the number zero

Any rational number can be expressed by combinations of such trees. For instance, the number 0.5 could be expressed as shown by figure 4.3.

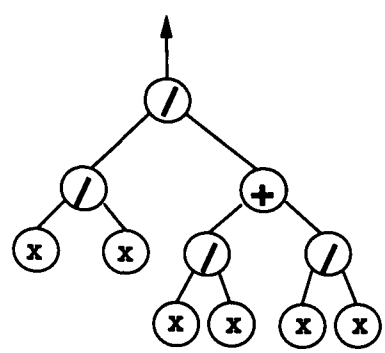


Figure 4.3: A tree representing the number 0.5

If more complicated numbers, say 0.5137, are required, the size of the tree becomes larger. This leads to an increase in both computation and evolution time. Clearly this is not a very efficient way of proceeding - a different way of handling numerical parameters would be desirable.

## 4.3 Background

The first attempt at introducing numerical values in GP was done by Koza (1992), who defined an ephemeral random constant, $\Re$, as a terminal node which is assigned a random value every time it appears in the first generation. These values remain constant for the rest of the run.

A problem with this approach is that the number and values of the available parameters depend on what happens in the first generation. Also, constants may be eliminated from (selected out of) the population but they cannot be created.

Although these two problems may be addressed by introducing a point mutation operator or some form of adaptation of the values, Koza did not apply any of these. This was somewhat justified by the simple nature of the problems he addressed.

Successive approaches to the numerical parameter problem can be classified into two main categories.

In the first one numerical parameters are included in the terminal set, i.e. the return value of specific terminal nodes is a number.

In the second one, they are expressed as a constitutive part of specific functions or terminals, i.e. the definition of certain nodes involves numerical parameters.

Combinations of both can also be found. In all cases, some sort of adaptation can be performed, by means of classical or evolutionary techniques.

An example of category I are the ephemeral random constants described above: the value returned by the $\Re$ node is the numerical parameter itself.

The nonlinear node `nlN`, described in chapter 2, can be used to illustrate category II. As explained, this node implements the function

$$\frac{1-\exp^{-\beta x}}{1+\exp^{-\beta x}} = \tanh\left(\frac{\beta}{2}x\right) \tag{4.3}$$

where $\beta$ is a linear function of the index $N$,

$$\beta(N) = \beta_{lo} + \frac{N}{N_{max}}(\beta_{hi} - \beta_{lo}), \qquad \beta \in [\beta_{lo} \ \ldots \ \beta_{hi}] \tag{4.4}$$

Thus, the parameter $\beta$ (or, correspondingly, the index N) is a constitutive part of (or implicit in) the function `nlN` i.e. the parameter is used to calculate the return value of the node but it is not the return value itself.

The first category includes the works of Angeline (1996), Chellapilla (1997) and Chellapilla *et al.* (1997) in Evolutionary Programming, and Montana and Czerwinski (1996) and Howard and D'Angelo (1995) in GP.

Angeline (1996), Chellapilla (1997) and Chellapilla *et al.* (1997) employ a numerical terminal similar to Koza's ephemeral random constant with the difference that its value can be modified by a mutation operator. The operator selects a single real valued numerical

terminal in a given tree and adds to it Gaussian noise with a particular variance (typically equal to 0.1).

Montana and Czerwinski (1996) and Howard and D'Angelo (1995) define a number of parameters associated to each individual tree. The values of these parameters are encoded in an appropriate way and evolved by means of a Genetic Algorithm. Each tree has its own set of values for the parameters, although these may or may not be present in the tree structure.

The tree in figure 4.4 illustrates this idea.

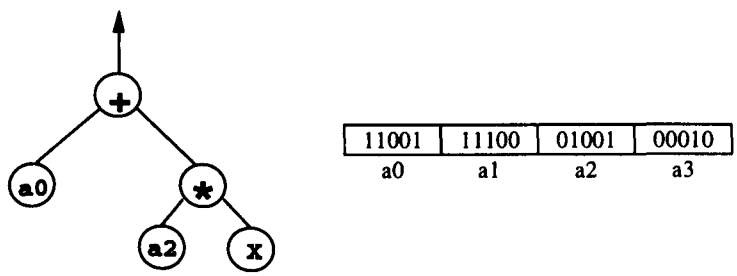


Figure 4.4: In this system, an individual is represented by the pair of a tree and a binary string. The terminal set in this system includes the nodes a0 .. a3. These nodes can be present in any tree but the actual values returned by them are encoded in the binary string and hence they are specific to each tree. In this particular tree only a0 and a2 are present, which means the resources devoted to store and adapt the values of a1 and a3 are wasted (Howard and D'Angelo 1995).

In the second category can be cited Andre (1994), Nguyen and Huang (1994), Gray *et al.* (1997), Iba *et al.* (1994) and Marenbach *et al.* (1996). Because the parameters are part of certain nodes and these are defined for particular problems, the methodologies become very problem-specific.

For instance, both Gray *et al.* and Marenbach *et al.* address system identification and control problems and for this purpose implement nodes that represent first and second order systems, time delays etc. An example is given in figure 4.5.

A different approach is that of Andre (1994) in the field of pattern recognition. It employs a class of functions, `ifdfn`, whose associated parameters, `dfn`, are not numerical values but two dimensional arrays of pixels ("hit-miss" matrices).

Nguyen and Huang (1994) apply GP to evolve 3-D aeroplane models. In their implementation a number of dummy functions serve the purpose of holding certain parameters. For instance, the function `SketchBody` holds three parameters (`midbodyLength`, `tailLength` and `bodyDiameter`) which are relevant to the task of designing the body of the plane.

Iba *et al.*'s implementation for system identification involves nodes whose transfer functions are polynomials of the node's two input arguments, as seen in figure 4.6.

A less clear example of category II is provided by the work of Koza *et al.* (1997*b*). They apply GP to evolve electronic circuits and for this purpose use a number of component-creating functions. These functions have one or more arguments, one of which is numerical

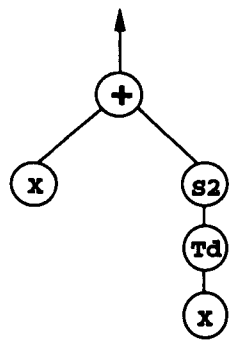


Figure 4.5: Model for a dynamic system. The function $S2$ represents a second order system of transfer function $\frac{K \cdot \omega_n^2}{s^2 + 2\omega_n \xi + \omega_n^2}$; the parameters associated to it are $K$, $\omega_n$ and $\xi$. $Td$ represents a time delay of transfer function $e^{-sT}$; the associated parameter is $T$ (Marenbach *et al.* 1996).

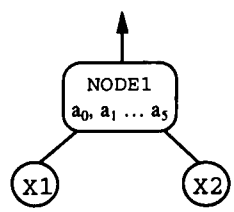


Figure 4.6: A polynomial GP node. The output of NODE1 is given by the equation $y_1 = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_1 x_2 + a_4 x_1^2 + a_5 x_2^2$ Every inner node carries a set of six associated parameters, $a_0 \cdots a_5$ (Iba *et al.* 1994).

and determines the value of the created component (see figure 4.7).

The argument $V1$ is an arithmetic performing subtree, composed only of arithmetic functions (addition and subtraction) and random constants. In this sense, this implementation seems to correspond to category I (where parameters are numerical terminals) but it has been included here because only certain specific functions make use of the arithmetic-performing subtrees; it can therefore be considered that the numerical parameter is part of the function.

An approach which can't be classified into any of the two categories is the one taken by McKay *et al.* (1996) and Hiden *et al.* (1997). They define a model structure, some parts of which are evolved by GP; the rest are parameters that are determined by other means.

### 4.3.1 Problems with existing prameter representation methods

Many of these approaches involve a (predefined) fixed number of parameters per tree (or per function). This is appropriate in some cases, where knowledge about the problem is available

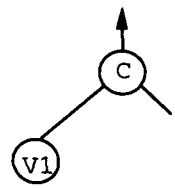


Figure 4.7: The C function creates a capacitor whose value is determined by mapping the numerical argument $V1$ onto an appropriate range of values (Koza *et al.* 1997*b*).

and the nature of the relevant parameters is understood a priori. This is the case of Gray *et al.* (1997), Marenbach *et al.* (1996), Nguyen and Huang (1994) and Koza *et al.* (1997*b*). For instance, in Nguyen and Huang's problem it seems only logical that, when designing a aeroplane, an important parameter should be its length.

In other cases, however, the determination of what parameters are relevant and should be included in the prospective solution is done in a rather arbitrary way. In this group fall Howard and D'Angelo (1995) and Andre (1994); for instance the latter employs six hit-miss matrices, but there doesn't seem to be a justification for the election of this particular number.

It seems desirable to find an alternative mechanism that **avoids or rationalises the selection of the parameters**. This was the first goal of the work described in this chapter.

Further problems are posed by the use of numerical terminals. In the case of constant terminals, as explained above, everything depends on whether the user's choice is appropriate or not, or, for randomly selected terminals, what the situation is in the first generation.

Numerical terminals face the problem of being placed in a position where no possible mutation can obtain any improvement in the performance. An otherwise useful terminal risks being selected out of the population due to this misplacement and the highly extended technique of assigning less probability to terminal crossover points, also termed leaf crossover (Koza 1992, pp. 114-116), seems designed to make matters worse (Angeline 1996).

As an example, let us suppose a model for the system represented in figure 4.8 is desired.

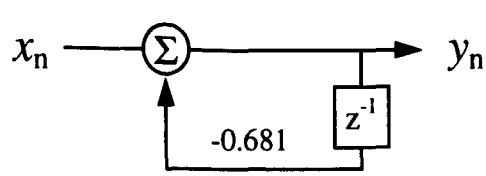


Figure 4.8: A first order system.

The system difference equation is given by 4.5.

$$y_n = x_n - 0.681 \cdot y_{n-1} \tag{4.5}$$

This system can be represented as a tree, using the nodes described in chapter 2. This is shown in figure 4.9.

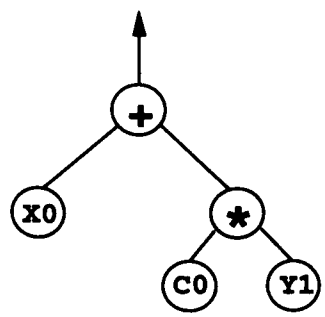


Figure 4.9: A tree representing a first order system ($x_n + C0 \cdot y_n$), with a numerical terminal.

The system equation for this tree is given by equation 4.6.

$$y_n = x_n + C0 \cdot y_{n-1} \tag{4.6}$$

so the tree will be a model for the system in Figure 4.9 as long as the value of $C0$ is approximately -0.681. Consider, however, what the situation would be if the numerical terminal $C0$ happens to be in the "wrong" place, as shown in figure 4.10.

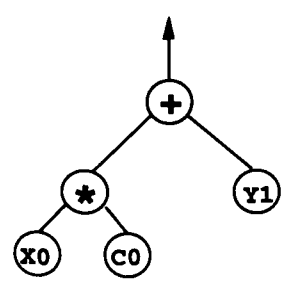


Figure 4.10: Another tree representing a similar system ($C0 \cdot x_n + y_n$).

The system equation for this tree is:

$$y_n = C0 \cdot x_n + y_{n-1} \tag{4.7}$$

Even if the value of C0 is -0.681, this tree does a very bad job at approximating the system in figure 4.8, so its fitness will be low. This could be waived by exchanging the values of `x0` and `y1` using some sort of permutation operator. Alternatively, the terminal `C0` could still be of use if placed in another tree by leaf crossover.

These procedures are discouraged in standard GP (Koza 1992) and the last one in particular is purposely avoided in many implementations. In such a situation, it is unlikely that the subtree (* x0 C0) will be disrupted and, because it confers no advantage to the individual carrying it, it is to be expected that it will eventually disappear from the population, which results in the loss of a useful constant.

This situation highlights two problems: **parameter misplacement** and **possibility of loss of parameters.**

### 4.3.2 Adaptation

Most techniques reviewed above employ some form of adaptation of the parameters: Gaussian mutation (Angeline 1996, Chelapilla 1997, Chelapilla *et al.* 1997), genetic algorithms (Montana and Czerwinski 1996, Howard and D'Angelo 1995, Andre 1994, Nguyen and Huang 1994), genetic programming (Koza *et al.* 1997*b*), simulated annealing (Gray *et al.* 1997) or classical regression techniques (Iba *et al.* 1994, Marenbach *et al.* 1996, McKay *et al.* 1996, Hiden *et al.* 1997).

However, adaptation seems rather pointless in the case of the "misplaced parameter" explained in the previous section.

In other cases (Howard and D'Angelo 1995, Andre 1994) it is possible to spend computational effort adapting parameters that are not used by the structure (as exemplified by figure 4.4), to no avail. This points at the fourth objective to attain: avoiding the unnecessary adaptation of unused parameters.

### 4.3.3 Summary of objectives

The problems thus exposed seem justification enough to investigate possible alternative methods of handling numerical parameters. The questions to be answered (i.e. the objectives to be attained) by such a method can be listed as follows:

1. how to determine the number of parameters to use
2. how to prevent the parameters from being selected out of the population
3. how to avoid the unnecessary adaptation of unused parameters
4. how to rationalise the placement of parameters

These questions are fully answered by the introduction of node gains.

## 4.4 Node gains

### 4.4.1 Concept

This is an approach that diverges from the two categories explained above and for this reason it has not been included in the previous review. Node gains were introduced by Sharman and Esparcia–Alcázar (1993) and subsequently developed in Sharman, Esparcia–Alcázar and Li (1995) and Esparcia–Alcázar and Sharman (1996, 1997*a*,*b*) and Esparcia–Alcázar (1997).

The concept of node gains was inspired by work in the Neural Networks field. In a Neural Network, numerical parameters in the form of weights are associated to the links between neurones, so that the data are modified as they pass from one neurone to another through the net.

In GP it is equivalent and representation-wise more convenient to define node gains as follows:

**Definition 4.1** *A* **node gain** *is a numerical parameter that multiplies the output value of a node.*

Let us consider the link between the output of a node labelled P and the input to a parent (or upper) node labelled Q. The link has a strength of $\alpha_{pq}$ and the relationship between the value at the output of node P, x, and the input to node Q, y, is

$$y = \alpha_{pq} \cdot x \tag{4.8}$$

This is shown graphically in figure 4.11.

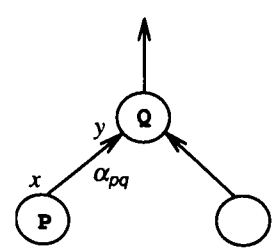


Figure 4.11: Graphic representation of a node gain.

For the purposes of this work the gains will be real numbers; this is no limitation, however, as complex, integer or binary gains can be implemented in the same way.

**Definition 4.2** *A* **gain vector**, $\vec{g}$, *is a vector of node gains associated to a node vector of the same length,* $\vec{n}$, *so that the* $i^{th}$ *component of* $\vec{g}$, $g_i$, *is the gain of the* $i^{th}$ *component of* $\vec{n}$, $n_i$.

Thus, a GP individual $I$ consists of the pair of vectors

$$I = \langle \vec{n}, \vec{g} \rangle \tag{4.9}$$

where

$$\vec{n} = \{ n_0\ n_1\ \cdots\ n_{\ell-1} \} \tag{4.10}$$

$$\vec{g} = \{ g_0\ g_1\ \cdots\ g_{\ell-1} \} \qquad g_i \in \Re \qquad i = 0 \ldots \ell - 1 \tag{4.11}$$

### 4.4.2 Benefits of node gains

By introducing node gains, all the questions posed in the previous section have been answered, namely:

- no decision must be taken regarding the number of parameters: there are as many gains as nodes.
- the parameters cannot disappear from the population, since they are attached to the nodes.
- all parameters have an influence on the output; there are no unused parameters.
- because all the nodes have a gain value, the placement of the parameters is not an issue.

Let us go back to the first order system of figure 4.8. A tree to represent such a system, considering node gains is shown in figure 4.12.

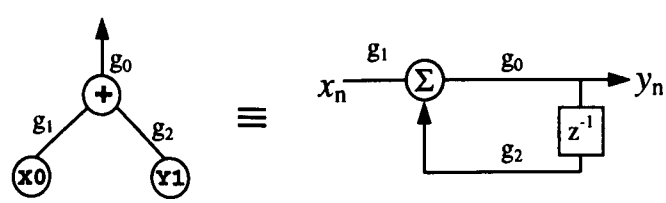


Figure 4.12: A tree representing a first order system, using node gains.

The system equation is now:

$$y_n = g_0 \cdot (g_1 \cdot x_n + g_2 \cdot y_{n-1}) \tag{4.12}$$

Note that, using an adequate adaptation algorithm, it will always be possible to find a set of gains (i.e. a gain vector, $\vec{g}$ ) that accurately models the system.

For example, the pair

$$\vec{n} = \{ +,\ \mathtt{x0},\ \mathtt{y1} \} \tag{4.13}$$

$$\vec{g} = \{ 0.5,\ 2.0,\ -1.362 \} \tag{4.14}$$

or, in compact polish notation:

```
(  [0.5] +  [2.0] x0  [-1.362] y1  )
```

represents a model for the system.

This is also equivalent to the system represented by the tree in figure 4.9, making $C0 = -0.681$. There is a striking difference, however: the tree in figure 4.9

$$\vec{n} = \{ +, \texttt{x0}, *, \texttt{y1}, \texttt{C0} \} \tag{4.15}$$

requires two extra nodes.

Besides, if other parameters were of interest, such as the ones present in equation 4.12, further constants and multiplication nodes should be introduced.

This leads us to an important characteristic of the node gain approach: by using node gains the same system can be represented by shorter, more compact trees. This "compacting effect" can affect the performance of the method in two ways:

- shorter subtrees are less susceptible of disruption by crossover
- handling shorter trees is less computer intensive.

### 4.4.3 Implementation issues

- **Initialisation**

  When creating a random tree for the initial population the node gains can also be initialised at random[1] or set to a predefined value (typically 1).

- **Genetic operators for node gains**

  Two approaches can be taken to crossover when working with node gains. The first one involves considering the gains as part of the structure; since a gain is attached to a specific node, it "moves" with it as a result of crossover.

  The second approach regards the gains in a more independent manner; thus, when a tree is created by crossover, the gains are reinitialised at random. This distinction will become of interest later, when an adaptation algorithm is introduced.

  Mutation can also be implemented for node gains. A gain can be selected for mutation in the same way as a node. The mutation itself can be done by either replacing the current value by a randomly generated one or by adding a random quantity to it, in the same way as done by Angeline (1996), Chellapilla (1997) and Chellapilla *et al.* (1997).

- **Extensions**

  It has been considered so far that every node has a gain value. This does not necessarily have to be so: only certain nodes or types of nodes may have gains (in the same way as in neural networks) and these would be marked by a "gain flag". The remaining gains would be set to unity.

[1] The gain values $g_i$ could, for instance, be obtained from normal ($g_i \sim N(1, \sigma^2)$) or uniform ($g_i \sim U(-1, 1)$) distributions

### 4.4.4 Cost of node gains

The achievements related in the previous sections have come at three costs:

- increased complexity of the nodes: a node must carry the value of the gain $g_i$ as well as the information on its type.
- increase in the computational cost, due to the additional multiplication operation per node
- increase in the degrees of freedom, equal to the number of nodes. This highlights the need for an adaptation algorithm to obtain adequate values for the gains.

It must be pointed out that these should not be considered in an absolute manner, but they must be compared separately to the different costs generated by the other possible approaches to the numerical values problem. It is an objective of this work to investigate whether the costs are overcome by the advantages brought by the use node gains.

With regard to the first point, the increase in complexity of the node representation is considered to be comparable (i.e. of the same order of magnitude) to the complexity introduced by the alternatives. More efficient ways of handling numerical parameters imply that either more complex structures are used or more knowledge about the problem at hand is available.

As per the second point, it will be shown that, despite the increased dimensionality, searching in the gain space using a continuous adaptation algorithm can, for certain problems, be easier and more efficient than searching in discrete spaces, as exemplified by crossover.

The reason is that for continuous spaces the fitness landscape is smoother than for discrete ones, i.e. small (big) changes in the parameters produce small (big) changes in the fitness, on average.

This applies as well to other continuous parameter adaptation schemes as the ones used by Angeline (1996), Chellapilla (1997), Chellapilla *et al.* (1997), Gray *et al.* (1997), Marenbach *et al.* (1996) and Iba *et al.* (1994) and partly by McKay *et al.* (1996) and HIden *et al.* (1997).

## 4.5 Experimental analysis

### 4.5.1 Objective

The purpose of this section is to make an experimental comparison between standard GP and the proposed node gain GP, in which all nodes have a gain value.

The objectives of this comparison are threefold:

- expose the shortcomings of standard GP, that make it desirable to find a different way of handling numerical parameters

- show the benefits of node gains, both immediate and potential, in terms of the fitness of the obtained solutions.
- determine whether the size of the solutions obtained by the node gain method is bigger or smaller than the standard GP ones.

### 4.5.2 Method

To attain these objectives a symbolic regression experiment is devised. This consists in approximating a function $y(x)$ given a number of points $(x,\ y)$, where $x$ is uniformly sampled from the interval $[x_{lo},\ x_{hi}]$. The solutions obtained are then tested with an extra set of pairs $(x, y)$, sampled from the same interval.

The experiment is done in two parts; for the first one the function to be approximated is as follows:

$$y = 2x^2 + 3x + 1 \tag{4.16}$$

The second part aims at approximating the function [2]

$$y = 2.718 \cdot x^2 + 3.1416 \cdot x + 1.3579 \tag{4.17}$$

In both cases the set up of the experiment is the same, as summarised in table 4.1.

### 4.5.3 Results

- **Fitness**

  When performing the first part of the experiment using standard GP, the solutions obtained in all runs but one achieved a fitness of 1 in the test (the one that didn't, had a fitness value close to zero). On the other hand, when the same was repeated for part 2 of the experiment, one tenth of the runs got solutions with fitness values close to zero and the maximum fitness of all was 0.997. Thus, just changing the numerical parameters in the problem causes a significant degradation in the performance of the obtained solutions.

  Part 2 of the experiment was also repeated using the node gain GP method. In this case, one third of the runs (ten in each set of thirty) obtained solutions with fitnesses close to zero, but at least one run in each set of thirty obtained a solution with a fitness value greater than 0.999, i.e. greater than the maximum obtained by standard GP.

  A $t$-test with a significance of 0.05 was performed for each set of thirty runs, with the conclusion that the mean of the fitnesses obtained by standard GP is higher than that of the fitnesses obtained by node gain GP.

[2] This is similar to an example given by Koza (1992). However, due to an erratum in the transcription of that work it is impossible to establish any comparisons.

Therefore it can be concluded that, on the average, the solution fitness in standard GP is higher than in node gain GP; however, the latter can achieve higher maximum values. This justifies further investigation into this technique.

- **Size**

  To allow for a size comparison between the two methods, the runs were conducted without any size restrictions (as opposed to the common practice in GP).

  The shortest solution of all, whose length was equal to 5 nodes, was obtained by node gain GP. It is interesting to note that the fitness of this solution was 0.829, i.e. much higher than the 2.61E-06 scored by the shortest standard GP solution, whose length was 7.

  A $t$-test is used for part 2 of the experiment to compare standard GP with node gain GP as per the length of the obtained solutions. The result indicates that the average of the solution size in standard GP is smaller than its counterpart in node gain GP, with a significance of 0.05. Thus, the "compacting effect" described in section 4.4.2 is not present on average.

Table 4.1: Set up for symbolic regression experiments

| | |
|---|---|
| Function set | `+ - * % +1 -1 *2 /2 1` |
| Terminal set † | `X0 C0 ... C255` |
| Nodes with gain | all |
| Population size | 500 |
| Mutation probability | 0 |
| Size restrictions | at creation: maximum depth = 4<br>at crossover: none |
| Fitness function ‡ | $\frac{1}{1+MSE}$ $0 \leq f \leq 1$ |
| Number of training samples | 10 pairs $(x, y)$ where $x = U(-1, 1)$ |
| Number of runs § | Two sets of 30 |
| Termination criterion for each run | 5000 births |
| Number of testing samples | 100000 pairs $(x, y)$ where $x = U(-1, 1)$ |

†The 256 entries in the constant table are chosen uniformly within the interval [-1,1]
‡The MSE or mean squared error is the average of the squared differences between the expected and the obtained values of y.
§The number of runs was chosen equal to 30 so that the Central Limit Theorem allows for the application of the $t$-test, even though the involved distributions may not be normal (Devore 1995, p. 235).

## 4.6 Conclusions

Experimental analysis shows that there is a significant degradation in the performance of standard GP when problems involving rational parameters (as opposed to integers) are tackled. This is considered to be justification enough for the investigation of new approaches to handling numerical parameters.

The introduction of node gains does not necessarily bring an advantage over standard GP, and, on the average, it proves to be disadvantageous. However, the results show the potential benefits that can be attained, exemplified by a higher maximum fitness and a smaller minimum size.

Although the use of node gains has many potential benefits, these are difficult to attain if the values of the gains are random: adaptation is needed if we want to take full advantage of the node gains.

## 4.7 Summary

The problem of handling numerical parameters in GP has been identified and various different approaches to it have been reviewed.

A number of problems of several or all of these approaches have been listed: identifying the parameters to evolve, ensuring that these parameters are not lost during the search, ensuring that there is no adaptation of unused parameters and rationalising the way parameters are placed.

An alternative approach is presented in the form of node gains. A node gain (similar to link weights in neural networks) is a numerical parameter associated to a node that multiplies the output value of the node.

Introducing node gains solves the problems listed above and also produces another effect: a given system can be described using smaller trees. This reduction in size has been named "compacting effect".

The costs involved are an increase in the complexity of the node representation and also in the degrees of freedom of the system. It will be proven that these disadvantages are far outweighed by the benefits.

Experimental results show that the potential advantages can be lost if the node gain values are left at random. This highlights the need for an adaptation algorithm.

An issue is thus left open: what the adaptation (or learning) algorithm should be. This will be discussed in the following chapter.

# Chapter 5

# Adaptive GP

## 5.1 Introduction

This chapter is devoted to the issue of adapting the node gains via an optimisation algorithm. For this purpose, an analogy between such an algorithm and the learning processes in nature is established.

Let us begin by defining what is going to be understood by learning in the remainder of this work.

**Definition 5.1** *In node gain GP,* ***learning*** *is the adaptation of the gain vector by means of an optimisation algorithm. In this work the algorithm employed for this purpose is Simulated Annealing.*

The chapter is structured as follows. In section 5.2 the motivation for adapting the gains, as concluded in Chapter 4, is summarised.

Section 5.3 focuses on the relationship between learning and adaptation. The first part of the section concentrates on the effects of learning in natural systems and how these effects can be exploited in artificial ones. The second half is devoted to the simulation of learning in artificial systems. A brief summary of previous models and their main shortcomings is followed by an account of how these shortcomings have been addressed by the proposed method, the limitations of which are also stated.

Section 5.5 extends the natural analogy of Darwinian evolution to the hypothetical domain of Lamarckian inheritance. The way to implement such a scheme is explained.

In Section 5.6 the Simulated Annealing algorithm is presented as an adaptive learning method and its implementation in node gain GP is discussed. Section 5.7 presents an experimental comparison between the Lamarckian and Darwinian learning methods and two other forms of GP: standard GP and GP with random node gains. Several measures of performance are presented and statistical analysis performed on them.

Finally, some conclusions are given in section 5.8 and section 5.9 summarises the chapter.

## 5.2 Motivation

In Chapter 4 node gains were introduced and it was shown how they give GP the *potential* of getting higher fitnesses and smaller solution sizes. However, statistical tests showed that this potential was not exploited on average: the mean of the solutions fitnesses was smaller and that of the sizes bigger when using node gains, as opposed to setting all gains to 1. It was concluded that leaving the node gains at random brought no real advantage while making the system more complex.

Thus, it was established that, in order to get full advantage of the potential benefits of node gains, an adaptation algorithm was needed to adjust the gain values.

Because GP is based on an analogy to the evolutionary processes of a population of individuals, it is interesting to exploit the analogy further and equate the adaptation of the gain values to the learning of single individuals. The rest of the chapter is devoted to establishing this analogy and explaining the adaptation algorithm used.

## 5.3 Natural and artificial learning

### 5.3.1 Analogies

Atmar (1994) defines natural learning as the selective retention of behaviours that have been accumulated through stochastic trial and error. It is a process inherent to and indistinguishable from evolution itself.

Following this definition, three forms of learning can be observed in natural evolution:

**Phylogenetic learning,** where the learned knowledge is stored in the species genetic code; the time scale of this process is the lifetime of a species.

**Sociogenetic learning,** where the knowledge is stored in the form of social culture; the time scale is the lifetime of a group.

**Ontogenetic learning,** where the knowledge is stored in an individuals memory; the time scale is the lifetime of the individual.

The first type of learning, phylogenetic learning, is implicit in the Genetic Programming algorithm. Attempts have also been made at modelling the second type of learning by providing GP with a notion of culture (Spector and Luke 1996, and others).

This chapter will be concerned with the simulation of the third type, ontogenetic learning, which from now on will be referred to simply as learning. The key idea is to provide each individual with the opportunity to adapt to its environment and modify its behaviour accordingly.

Individual learning affects the whole population in various ways. For instance it has been postulated (Anderson 1995) that one effect is the decrease in the reproduction rate,

as time devoted to learning cannot be spent in reproduction[1]. More importantly, learning can also influence evolution. The process by which learning assists the integration of genetic components of behaviours into the gene pool has been termed *Baldwin effect* (Hinton and Nowlan 1987, Belew 1990, French and Messinger 1994, Anderson 1995, Parisi and Nolfi 1996).

The effect of learning in fixed environments is a speed up in the search for the optimum, because instead of a phylogenetic (and possibly blind) search for an optimal genotype, the optimal phenotype can be found via an ontogenetic ("directed") search.

However, the main advantage of learning comes in variable environments.

From the individual's point of view, learning allows it to adapt to the new environment and therefore increases its chances to survive under the modified conditions.

From the population's point of view, it is attributed to learning the maintenance of genotypic diversity (Anderson 1995). This means there will be a higher probability that at least some of the genotypes will have a high fitness under the new environment, than in the case where only few genotypes were present before the change took place.

### 5.3.2 Simulating learning

Many attempts have been made at artificially modelling the natural learning processes, both to gain insight into the learning process itself (Hinton and Nowlan 1987, Belew 1990, French and Messinger 1994, Anderson 1995, Parisi and Nolfi 1996) or to exploit its potential benefits when applied to engineering problems (Boers *et al.* 1995, Gruau and Whitley 1993, Lucas 1996). Most of these models involve significant simplifications of the complexity of natural learning. In particular, criticism to (Hinton and Nowlan 1987)'s classical simulation, learning the weights of a neural network, is based on three points (Parisi and Nolfi 1996):

1. learning consists of random changes, each learning trial being independent of the result of previous ones (there isn't a "learning path").

2. there is no environment and therefore no phenotype, only genotype. Hinton & Nowlan's search for a particular neural network is in reality a search for a particular bit string.

3. the evolutionary task and the learning task coincide: good performance on the learning task implies high fitness.

Various authors have addressed these points, mainly the first two, in different ways. In the model presented here, the first one is overcome by the use of Simulated Annealing as learning algorithm, as will be shown below. The solution to the second criticism is implicit in the aim of the method, which is to find a solution to an engineering problem – this constitutes the environment. For this same reason, the third point is of less relevance here than in the

[1] While I sit here writing this thesis, other people are breeding

case of simulations that aim at studying the learning process itself, and therefore will be overlooked.

Two additional points apply to the proposed method. First, the individual learns at no cost to itself. The cost to the user is the increase in computational time required to produce a given number of individuals. But, from an engineering point of view, this is irrelevant if less "births" are needed to reach a solution.

Second, the amount of effort an individual devotes to learning is controlled externally: it is neither congenital to the individual nor controlled by it. This issue can be addressed by modifying the implementation such that the amount of learning effort is regulated by a "learning gene".

Another interesting development would be to run the simulation in parallel and allow the individuals to learn or reproduce "at will". These concerns are left for future study.

## 5.4 Basic details of Adaptive GP

Let $\vec{n} \in \aleph$ be a syntactically correct vector of nodes (according to the rules given in section 1.4.2) and let $\vec{g} \in \mathcal{G}$ be a vector of gains. Define the function $\ell(\vec{v})$ to be the number of elements in a vector $\vec{v}$. An individual genotype $\vec{h} \in \mathcal{H}$ can be represented as the pair of vectors

$$\vec{h} = \langle \vec{n}, \vec{g} \rangle \quad : \quad \ell(\vec{n}) = \ell(\vec{g}) \tag{5.1}$$

We now proceed to define a GP phenotype. Let $e \in \mathcal{E}$ be the environment in which a GP individual operates. In the context of Digital Signal Processing, the environment is the input data sequence and the training output (if required).

The *fitness function*, $f \in \mathcal{F}$, is

$$f \quad : \mathcal{H} \times \mathcal{E} \longrightarrow \quad (0 \ldots 1) \tag{5.2}$$

Let $\omega$ be a learning algorithm:

$$\omega : \mathcal{H} \times \mathcal{E} \times \mathcal{F} \longrightarrow \mathcal{G} \tag{5.3}$$

The above learning algorithm produces a new set of gain values for a given individual and fitness function in a particular environment. Details of a Simulated Annealing algorithm will be given in appendix A.

An individual phenotype, $p \in \mathcal{P}$, is represented as the 4-tuple:

$$\vec{p} = \left\langle \vec{h},\ f,\ \omega,\ r \right\rangle \tag{5.4}$$

Hence, before learning we have

$$f = (\vec{h}, \vec{e}) = f\left(\langle \vec{n}, \vec{g} \rangle\ \ \vec{e}\right) \tag{5.5}$$

and after learning

$$f(\vec{h}', \vec{e}_2) \tag{5.6}$$

where

$$\vec{h}' = \left\langle \vec{n}, \omega(\vec{h}, \vec{e}_1) \right\rangle \tag{5.7}$$

where $\vec{e}_1$ is the environment used for learning and $\vec{e}_2$ is the environment where it will operate afterwards.

Finally, we define a *reproduction function* as

$$r \quad : \quad \mathcal{H} \times \mathcal{H} \longrightarrow \mathcal{H} \tag{5.8}$$

or, alternatively, as

$$r \quad : \quad \aleph \times \aleph \longrightarrow \aleph \tag{5.9}$$

depending on whether or not the learned gain values are inherited during crossover (details given in section 5.5).

## 5.5 Darwinism vs. Lamarckism

In standard GP evolution takes place following Darwinian rules. Darwinian evolution is a two-step process, taking place as follows, (Gould 1980)

1. **random** genetic variations take place, caused by recombination and mutation only
2. selection (natural or artificial) favours the survival of the fittest, or rather, the **culling of the less fit** (Atmar 1994), among these variants.

The former implies that individual learning does not affect the genetic material and therefore cannot be inherited.

On the other hand, another classical theory of evolution, Lamarckism, is essentially a theory of *directed variation*. In the face of an environmental change, an organism would react by incorporating preferentially favourable genetic information, which would then be transmitted to offspring. The latter, also known as "inheritance of acquired characters", has taken over the meaning of the word Lamarckism, and it is by this definition that we will use it.

Although Lamarckian evolution (in any of its meanings) has not been observed in biological history[2], it can be said that the evolution of human culture (or learning in higher

[2]Although genetic changes can be due to exposure to radiation or chemical agents, these changes are random, not directed. (Gould, 1980)

mammals) is Lamarckian in character: knowledge is transmitted from one generation to another.

Lamarckian evolution is implemented in the GP/SA system by allowing the annealed node gains to be inherited by the offspring. In Darwinian evolution, the gains are set to 1 or initialised with random values before the annealing takes place and are not inherited.

Let $T_1$ and $T_2$ be two syntactically correct trees selected for crossover, with lengths $\ell$ and $m$ respectively, and whose expressions are:

$$T_1 \equiv \begin{cases} \vec{n}_1 = \{ & n_{1,0} & n_{1,1} & \cdots & n_{1,(\ell-2)} & n_{1,(\ell-1)}\} \\ \vec{g}_1 = \{ & g_{1,0} & g_{1,1} & \cdots & g_{1,(\ell-2)} & g_{1,(\ell-1)}\} \end{cases} \tag{5.10}$$

$$T_2 \equiv \begin{cases} \vec{n}_2 = \{ & n_{2,0} & n_{2,1} & \cdots & n_{2,(m-2)} & n_{2,(m-1)}\} \\ \vec{g}_2 = \{ & g_{2,0} & g_{2,1} & \cdots & g_{2,(m-2)} & g_{2,(m-1)}\} \end{cases} \tag{5.11}$$

Let us assume that $T_1$ is acting as the "mother" and T2 as the "father"; this means $T_1$ provides the root of the tree and $T_2$ the branch to swap. Further assume that the crossover points are $i$, $0 \leq i < \ell$, for $T_1$ and $j$, $0 \leq j < m$, for $T_2$.

Let the subtrees starting at $i$ and $j$ comprehend all the nodes up to $p$ and $q$ in their respective trees.

The result of the crossover, $T_{1x2}$ , is a tree of length $\ell - p + q$, whose expression is:

$$T_{1x2} \equiv \begin{cases} \vec{n}_{1x2} = \{ & n_{1,0} & \cdots & n_{1,(i-1)} & n_{2,j} & \cdots & n_{2,(j+q-1)} & n_{1,(i+p)} & \cdots & n_{1,(\ell-1)}\} \\ \vec{g}_{1x2} = \{ & g_{1,0} & \cdots & g_{1,(i-1} & g_{2,j} & \cdots & g_{2,(j+q-1)} & g_{1,(i+p)} & \cdots & g_{1,(\ell-1)}\} \end{cases} \tag{5.12}$$

The gain vector $\vec{g_{1x2}}$ has components inherited from $T_1$ and $T_2$. In an alternative scheme, the components of $\vec{g_{1x2}}$ would be set to 1 or initialised with random values. These two ways of proceeding are labelled here *Lamarckian and Darwinian learning schemes.*

Using the notation given in section 5.4, the Darwinian learning scheme implies defining the reproduction function $r$ as follows

$$r = r(\vec{n}) \tag{5.13}$$

i.e. $r$ is only a function of the node vector $\vec{n}$. On the other hand, in Lamarckian evolution

$$r = r(\vec{n},\ \omega(\vec{h}, \vec{e})) \tag{5.14}$$

$r$ is a function of the genotype $\vec{h}$ modified by the learning function $\omega$ in the environment $\vec{e}$

## 5.6 Learning by Simulated Annealing

The reasons why Simulated Annealing was chosen as a learning algorithm are twofold. On the computational front, due to the simplicity of implementation. On the philosophical front, due to the similarity to the learning process in nature, in several ways.

First, as an individual undergoes annealing, the probability of a big gain jump decreases. This is in accordance with what is observed in nature: the amount that an individual can learn decreases in time.

Second, big decreases in fitness are mainly restricted to the early stages of the learning process. Pursuing the natural analogy, when the individual is young it can still accept a decrease in "status" but that is less likely as it grows older.

Finally, the starting point of each iteration depends on the outcome of the previous one: the process is cumulative.

For details on the implementation of the SA algorithm see Appendix A.

## 5.7 Experimental analysis

### 5.7.1 Four methods

This section provides results obtained by four different methods. The first one is the standard GP algorithm, in which the gains are fixed and equal to 1. This is referred to as **NGNL** (no gains - no learning). The other three methods employ node gains, whose values are initialised at random in the first population.

In the second method studied the gain values remain fixed (save in the case of a random mutation) and thus are inherited by the offspring. This is labelled **RGNL** (random gains - no learning).

The remaining two methods employ learning of the gains. In the **Darwinian learning** scheme the gains are also initialised at random for every individual that is born and then subjected to annealing. In **Lamarckian learning** the annealed gains of the parents are inherited by the offspring, which then undergo their own annealing process.

The four methods will be applied to three channel equalisation problems, as explained in Chapter 3. The reader is also referred to (Sharman, Esparcia-Alcázar and Li 1995) and (Esparcia-Alcázar and Sharman 1996, Esparcia-Alcázar and Sharman 1997*b*) .

The GP settings for these experiments are shown in Table 5.1, while Table 5.2 gives the SA settings.

### 5.7.2 Fixed environments

#### Overview

Strictly speaking a fixed environment would be one in which the population has reached equilibrium, i.e. the average fitness remains constant. When tackling DSP or other engineering problems, however, the behaviour in the equilibrium is not usually observed. The concern in these cases is finding an optimal solution; hence, the process is stopped once it is considered that no further improvement can be achieved.

Table 5.1: Settings for four method comparison

| | |
|---|---|
| Function set | `+ - * % +1 -1 *2 /2` |
| Terminal set | `1 X0 ... X3 Y1 Y2 C0 ... C6`<br>`C7...C255` (NGNL only)† |
| Constant table | 0.1 0.5 2 10 -1 3 0.7 |
| Population size | 500 |
| Mutation probabilities | of a node: 0.01<br>of a gain: 0<br>0.01 (RGNL only) |
| Size restrictions | at creation: maximum depth = 4<br>at crossover: maximum depth = 6 |
| Fitness function ‡ | $f = \frac{1}{1+MSE}$ $0 \leq f \leq 1$ |
| Number of training samples ($\vec{e}_1$)§ | 70 |
| Signal to Noise (SNR) ratio | 30 dB |
| Number of runs | LC1: 50 runs<br>NLC: 24 runs<br>LC1 $\longrightarrow$ LC2: 51 runs |
| Termination criterion for each run | node evaluations |
| Number of testing samples ($\vec{e}_2$) § | 10100 |

†Entries 7 to 255 in the constant table for NGNL are chosen uniformly within the interval [-1,1]

‡The MSE or mean squared error is the average of the squared differences between the expected and the obtained values of the output.

§The first 20 and 100 samples are rejected for fitness calculation as transient

Table 5.2: Annealing settings for Darwinian and Lamarckian methods.

| | |
|---|---|
| Learning Probability | 1 |
| Perturbation Distribution | Cauchy |
| Perturbation Scheme | Elastic |
| Cooling schedule | Exponential |
| Starting Temperature ($T_0$) | 2 |
| Temperature Decay ($\alpha$) | 0.99 |
| Learning Iterations (`maxIterations`) | 400 |
| Trials per temperature (`maxTrials`) | 1 |
| Maximum Invariant Fitness Trials | 20 |

In the particular case of the experiments related here we are interested in both the solutions and the evolutionary process itself. A fixed environment will then be defined as a GP run of the channel equalisation problem in which the unknown channel does not change for the duration of the run. The run proceeds for up to a given number of node evaluations (as explained below) regardless of whether or not a suitable solution has been found or equilibrium has been reached.

Two cases are studied: a linear channel (LC1) and a nonlinear one (NLC). The difference equations for these channels are

$$x_n = y_n + 0.9 \cdot y_{n-1} \tag{5.15}$$

for LC1 and

$$p_n = y_n + 0.6 \cdot y_{n-1} + 0.5 \cdot p_{n-1} \tag{5.16}$$

$$x_n = p_n + 0.15 \cdot p_{n-2} \tag{5.17}$$

for NLC, where $y_n$ and $x_n$ are the inputs to the channel and equaliser respectively at instant $n$ and $n_n$ is the additive white Gaussian noise.

### Study of the performance

The study of the performance can be divided into two aspects. The first one is related to how well the solutions obtained perform with unseen data, i.e. the generalisation ability.

Table 5.3: Comparison of results (50 runs for LC1)

| | fitness (in test) | BER (in test) | fitness (after evolution) | discrepancy d |
|---|---|---|---|---|
| Darwinian | 0.9765 | 0 | 0.9825 | 0.0018 |
| Lamarckian | 0.9679 | 0.00806 | 0.9704 | 0.0002 |
| RGNL | 0.9731 | 0.00006 | 0.9890 | 0.0037 |
| NGNL | 0.8231 | 0.03045 | 0.8933 | 0.0487 |

Table 5.4: Comparison of results (24 runs for NLC)

| | fitness (in test) | BER (in test) | fitness (after evolution) | discrepancy d |
|---|---|---|---|---|
| Darwinian | 0.9620 | 0.00564 | 0.9816 | 0.0013 |
| Lamarckian | 0.9261 | 0.01071 | 0.9620 | 0.0052 |
| RGNL | 0.7418 | 0.02920 | 0.9655 | 0.2090 |
| NGNL | 0.8049 | 0.10371 | 0.9303 | 0.0336 |

It is a matter of discussion among the GP community whether or not the existence of local learning schemes decreases the generalisation ability (and therefore the quality) of the potential solutions. Some argue that learning will cause overfitting of the training data and therefore, when the data is changed, the performance will be poor because the solution was biased towards the training data.

In order to test whether or not this is the case with the Darwinian and Lamarckian learning schemes, each experiment was run a number of times using 70 samples (of which the first 20 were rejected for the fitness calculation as transient). The termination criterion for evolution was a number of node evaluations equal or exceeding a given limit (1e8 for LC1, 3e8 for NLC).

The solutions were tested with a further 10100 samples (of which the first 100 are rejected) to obtain a fitness value and a bit-error-rate. These two values themselves provide a measure of "how good" the solutions are.

The fitness in the test was also compared with the one obtained during evolution and the discrepancy measured as follows:

$$d = \frac{1}{N}\sum_{N}\left(\text{fitness}_{\text{test}} - \text{fitness}_{\text{after evaluation}}\right)^2 \tag{5.18}$$

where $N$ is the number of runs.

This gives an idea of how well the fitness during evolution can predict the subsequent behaviour of the solutions obtained by each method, therefore being a measure of reliability.

Tables 5.3 and 5.4 give the averages of these values.

Table 5.5: Success rates for LC1 and NLC

| | LC1 (50 runs) | NLC (24 runs) |
|---|---|---|
| Darwinian | 1 | 0.92 |
| Lamarckian | 0.96 | 0.83 |
| RGNL | 1 | 0.92 |
| NGNL | 0.78 | 0.76 |

These results show that, on average, Darwinian learning outperforms the other methods, the differences being more noticeable in the case of NLC, which is a more difficult problem. On the other hand, the discrepancy in LC1 is lower for Lamarckian learning, whereas in NLC the lowest value corresponds to Darwinian learning; both results show that learning methods generalise better than non learning ones.

The second aspect in measuring the performance is the success rate. This is given by the ratio of successful runs over the total number of runs. We define a successful run as one in which the fitness of the best solution measured during the evolution is greater than an arbitrary value, here chosen as 0.9. The values are shown in Table 5.5.

Darwinian learning maintains the advantage because it has both high success rates and low discrepancies. RGNL has higher success rates than Lamarckian learning, but on the other hand, as shown in the previous tables, it also has a greater discrepancy, which makes its solutions less reliable.

### 5.7.3 Variable environments

#### Overview

A variable environment is one in which the unknown channel is modified during the run.

The implementation of this is as follows. Evolution proceeds as explained in the previous section for LC1, up to $10^8$ node evaluations approximately. Then a new set of data is generated for the modified channel LC2. LC2 has the same structure as LC1 but one of its coefficients is slightly different in absolute value and has opposite sign. The difference equation for LC2 is

$$y_n = x_n - 0.99 \cdot x_{n-1} \tag{5.19}$$

The population is then re-evaluated and re-annealed and evolution continues for approximately up to $3 \cdot 10^8$ node evaluations.

Table 5.6: Comparison of results in a variable environment. Averages of 51 runs for LC1 $\longrightarrow$ LC2

| | fitness (in test) | BER (in test) | fitness (after evolution) | discrepancy d |
|---|---|---|---|---|
| Darwinian | 0.9078 | 0.002047 | 0.9592 | 0.0030 |
| Lamarckian | 0.8930 | 0.001986 | 0.9606 | 0.0208 |
| RGNL | 0.8296 | 0.038525 | 0.9146 | 0.0313 |
| NGNL | 0.6062 | 0.195249 | 0.7719 | 0.1242 |

Table 5.7: Success rates (51 runs for LC1 $\longrightarrow$ LC2)

| | |
|---|---|
| Darwinian | 0.9608 |
| Lamarckian | 0.9804 |
| RGNL | 0.8431 |
| NGNL | 0.5882 |

**Study of the performance**

The solutions are tested as explained before with 10100 samples generated for the channel LC2. A comparison of the solution performances is given in table 5.6.

Darwinian learning maintains the advantage, both in fitness and reliability, clearly over RGNL and NGNL and slightly over Lamarckian learning.

After studying the success rates, given in Table 5.7, it turns out that the Darwinian learning scheme has a slightly lower probability of success than Lamarckian learning. Nevertheless, the performance of the solutions obtained by this method is higher due to the low discrepancy.

### 5.7.4 Analysis

The results obtained in previous sections are analysed statistically by means of a **Kruskal-Wallis test**. This is a multiple comparison non parametric test (Conover 1980), which means the $k$ populations compared are not assumed to have normal distributions. The details of the test are given in Appendix C. Briefly, the hypotheses tested are as follows:

$H_0$ : All of the $k$ populations compared are identical

$H_a$ : At least one of the populations tends to yield larger observations than at least one of the other populations (i.e. the $k$ populations do not have identical means)

For a confidence level of 0.1 the result of the test for all three cases studied was rejection of $H_0$. A procedure was then employed to determine which pairs of populations tended to differ. The results, for the same confidence level, are summarised in Table 5.8.

Table 5.8: Multiple comparisons with the Kruskal-Wallis test

| Methods | | LC1 | NLC | LC1⟶LC2 |
|---|---|---|---|---|
| Darwinian | Lamarckian | same | same | same |
| Darwinian | RGNL | same | different | different |
| Darwinian | NGNL | different | different | different |
| Lamarckian | RGNL | same | different | different |
| Lamarckian | NGNL | different | different | different |
| RGNL | NGNL | different | different | different |

The conclusion is that, for this confidence level, the Darwinian and Lamarckian learning schemes cannot be distinguished. Also, in the case of LC1, RGNL cannot be distinguished from either the Darwinian or Lamarckian learning schemes.

### 5.7.5 Influence of learning on evolution.

To trace the influence of learning on evolution a new variable is introduced: the fitness at birth or $fab$. The statistics of the $fab$ will be compared to those of the fitness after learning, or $fit$. With no learning, $fit = fab$. We are interested in the variations in the distributions of $fit$ and $fab$ as the run proceeds.

For this study 24 runs are performed for Darwinian, and Lamarckian learning and RGNL with the same set up shown in tables 5.1 and 5.2.

In this case the emphasis is not placed on the solution but on the evolutionary process itself and therefore the experiments run for a longer time. In practise the termination criterion for evolution is a number of node evaluations greater than or equal to $3 \cdot 10^8$.

The distributions for LC1 are shown in Figures 5.1 and 5.2. The distributions for NLC showed similar results and are not displayed to avoid repetition.

It can be seen that the $fit$ distribution moves to the right as the run proceeds. This displacement is slowest in Darwinian learning and fastest in RGNL. In more difficult problems we have observed that the fit distribution in RGNL doesn't reach the higher values of the scale. Instead, it gets "stuck" at suboptimal values.

The $fab$ distribution shows an increasing peak at zero in Darwinian learning. This means that individuals that can learn are selectively preferred to those that are naturally fit. The opposite occurs in Lamarckian learning: the $fab$ distributions are displaced towards the right, as would be expected.

In both cases the displacement of the $fab$ distributions is much slower than that of the $fit$ for RGNL (remember that for RGNL $fit = fab$). This shows how the presence of learning slows down the evolution of the genotype.

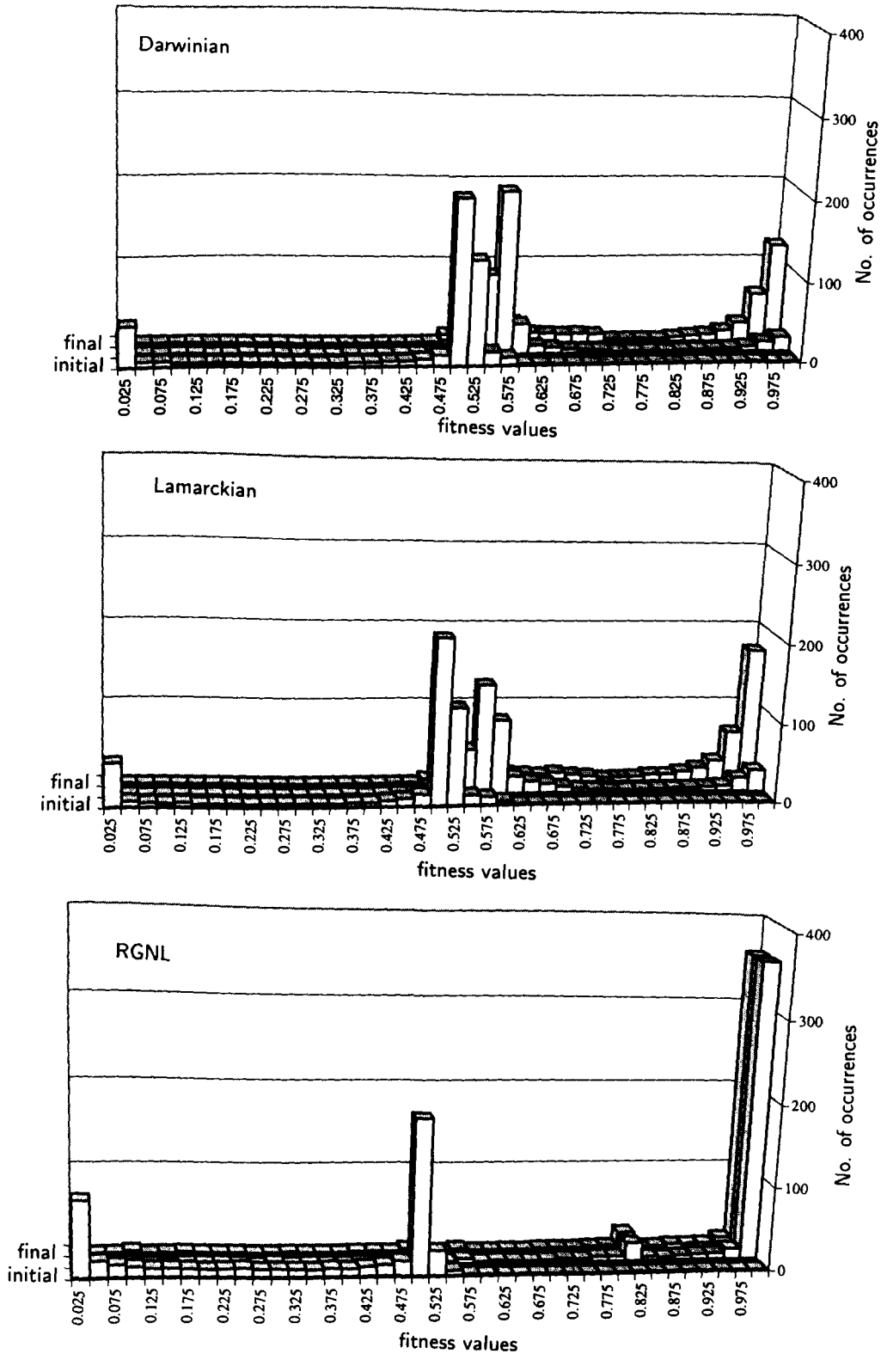


Figure 5.1: Evolution of the $fit$ for the Darwinian (top), Lamarckian (middle) and RGNL (bottom) methods. Averaged histograms for initial, middle and final stages of the run.

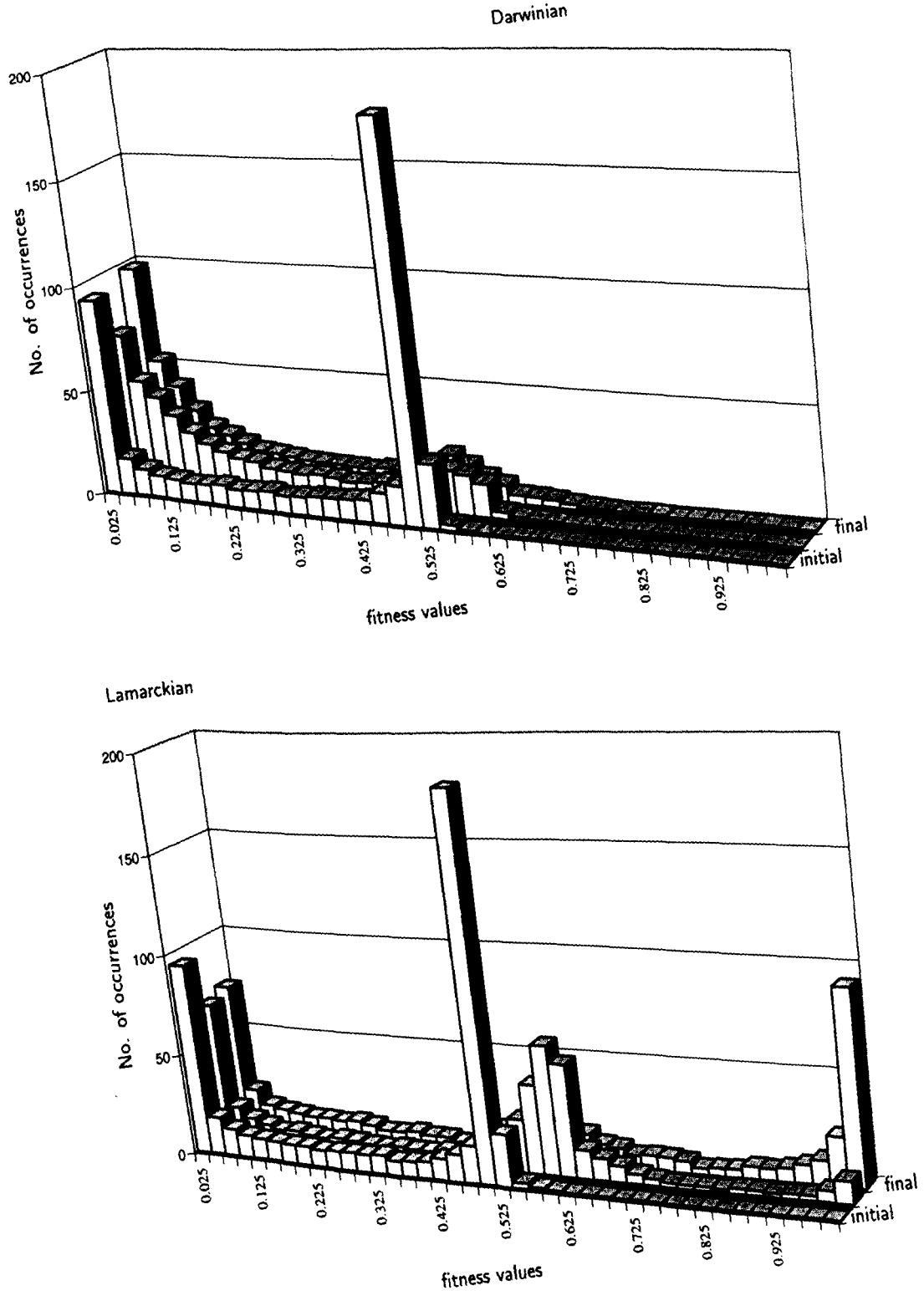


Figure 5.2: Evolution of the $fab$ for Darwinian (top) and Lamarckian (bottom) learning. Average histograms for initial, middle and final stages of the run.

### 5.7.6 Further comments: Extending the analogy

A number of limitations apply to our simulations, in particular:

- all individuals have the same probability of learning (equal to 1)
- the maximum number of annealing iterations is fixed
- in the experiment with varying environment, the unknown channel is modified only once.

It would be of great interest to relax these constraints, as well as, as already mentioned, letting the individuals "decide" on their own learning and parallelise the simulation. Further work will aim at achieving these objectives, which would be easily incorporated in the implementation.

## 5.8 Conclusions

In complex engineering problems where node gains become necessary, an adaptation algorithm is required to optimise the values of these gains. By drawing an analogy between such an adaptation algorithm and the learning process in nature, one is able to envisage the way in which individual adaptation affects the whole system.

A step beyond the natural analogy is the introduction of Lamarckian inheritance, which is easily implemented in the node gain GP system.

The learning algorithm is implemented by means of Simulated Annealing, due to its similarities to natural learning. It also offers a number of possibilities in the selection of parameters and different schedules used, in contrast with the GP algorithm which is somewhat more rigid.

Introducing SA in the node gain GP system provides a faster way of finding solutions to engineering problems and a more robust one in the case of variable environments.

Applying a learning algorithm might not be advisable in all cases. Two general decision rules as per when to employ learning can be given :

- when the performance of a system is sensitive to parameter variations; for instance, when a wrong selection of parameters can make a system unstable
- when environmental changes are expected; for instance in equalisation in mobile systems, where the unknown channel is constantly varying.

Two possible learning schemes for GP have been addressed here, namely Darwinian and Lamarckian. It has been shown that, in the examples presented, using node gains provides better results than those obtained by standard GP. Furthermore, the use of learning improved the performance in the two more complex problems addressed.

A measure of the generalisation ability has been introduced which shows that overfitting is not a problem in any of the learning schemes presented; on the contrary, both generalise better than the non learning methods. This might be due to the fact that the solutions obtained with any of the learning schemes tend to be smaller than those obtained without learning. Further analysis is required to prove this point.

Although statistical tests do not allow us to reach a conclusion as to which performs better, the Darwinian scheme looks intuitively more indicated for a variety of problems, as the more robust of the two.

## 5.9 Summary

The optimisation of node gains has been transformed by a useful analogy into the learning of individuals in a given population. The analogy is exploited to investigate what the effects of such an optimisation algorithm will be on the GP system. Potential benefits have been discussed, which point out to the situations when it is most convenient to use a learning algorithm. Also, the limitations of the analogy have been discussed.

The implementation of learning via simulated annealing has been justified with the various properties that make SA an interesting learning algorithm.

Experimental analysis has aimed at comparing learning and non learning methods via three equalisation problems. This has been done using various performance measures and statistical analysis. On the whole, the superiority of the learning techniques has been shown.

# Chapter 6

# Further Results in Channel Equalisation

## 6.1 Introduction

This chapter revisits the channel equalisation problem presented in Chapter 3. The objective is to provide further results in this problem using GP enhanced with node gains and Simulated Annealing and compare them with those provided by the existing bibliography.

The chapter is structured as follows. Section 6.2 gives an overview of three channel equalisation problems, pointing out the interest of applying GP to them. Sections 6.3, 6.4 and 6.5 concentrate on three specific problems that have been addressed by other authors and to which the proposed method was successfully applied. Section 6.6 compares the results of presented in the previous sections to those given by the literature. Conclusions are given in section 6.7 and finally section 6.8 summarises the chapter.

## 6.2 Overview

The examples discussed in this chapter are cases for which it has been shown (Chen *et al.* 1990, Gibson *et al.* 1991, Gibson *et al.* 1989, Theodoridis *et al.* 1995) that nonlinear equalisation techniques can provide better results than linear ones. The unknown channel to equalise will be:

- a linear channel with high levels of noise.
- a linear partial response channel.
- a nonlinear channel.

The results yielded by the proposed method will be compared to those obtained by training a 20-tap ($19^{th}$ order) Finite Impulse Response (FIR) equaliser with the Recursive Least Squares (RLS) algorithm, as done in (Kechriotis *et al.* 1994).

Table 6.1: Annealing settings for all channel equalisation experiments

| | |
|---|---|
| Perturbation Type | Cauchy |
| Perturbation Scheme | Elastic |
| Starting Temperature ($T_0$) | 1.5 |
| Cooling Schedule | Inverse linear |
| Scale perturbation by | 0.2 |
| Annealing Iterations (`maxIterations`) | 100 |
| Trials per temperature (`maxTrials`) | 5 |
| Maximum Invariant Fitness Trials | 20 |

The set up for the experiments is as follows. A training signal of 250 samples is used to train both a FIR-RLS and a GP+SA equalisers. For the latter, the first 50 samples are rejected as transient in the calculation of the fitness during the evolution process.

After adaptation, a further 100100 samples of a signal of the same noise realisation are processed by both filters and the bit-error rate calculated (rejecting the first 100 samples in both cases).

The parameters of the Simulated Annealing algorithm, which are given in Table 6.1, are common to all three examples.[1]

## 6.3 Linear channel with high noise

Let us consider the linear minimum phase channel described by the transfer function

$$H(z) = 1 + 0.7z^{-1} \tag{6.1}$$

In the low noise situation it is possible to find an equaliser for this channel to any specified accuracy by employing a finite impulse response (FIR) filter of sufficient length (or order). When the noise is high, however (i.e. the signal to noise ratio, SNR, is lower than 10 dB) the phenomenon of noise enhancement appears, which means that any increase in order results

[1]This can be done because, due to the way it was defined in chapter 3, the fitness always lies in the interval [0,1]. If the fitness had different limits for each problem, the annealing settings (in particular, the starting temperature, $T_0$) should be modified accordingly

Table 6.2: Values of the bit-error rate obtained by equalisers for the channel $H(z) = 1 + 0.7z^{-1}$ over 30 runs.

| | average BER | | minimum BER | |
|---|---|---|---|---|
| SNR(dB) | FIR-RLS | GP+SA | FIR-RLS | GP+SA |
| 2.5 | 0.126724 | 0.129697 | 0.11284 | 0.10519 |
| 5 | 0.071622 | 0.082892 | 0.06121 | 0.05491 |
| 7.5 | 0.033807 | 0.034803 | 0.02731 | 0.01299 |
| 10 | 0.009246 | 0.014868 | 0.00653 | 0.0012 |

in a decrease in efficiency of the equaliser (Gibson *et al.* 1991). It is therefore interesting to try and find low order equalisers which employ some form of nonlinearity.

Results for different values of the SNR are given in Table 6.2 and shown graphically in Figure 6.1. Average and minimum values of the BER for 30 runs (per point) are presented, showing that the GP+SA method can obtain lower minimum values than the FIR-RLS, especially for values of the SNR of 7.5 and 10 dB.

For values of the SNR below these the noise is too high for the equalisation method to make a significant difference. For values above, both methods give consistently BERs of zero, so the comparison loses meaning. Nevertheless, it is interesting to note that in these cases several GP+SA equalisers yielded a fitness of 1 (i.e. MSE = 0), while this was never achieved by any of the FIR-RLS equalisers.

Table 6.3: Set up for linear channel (H(z) =1+0.7z-1 ) equalisation experiment

| | |
|---|---|
| Function set | + - * % +1 -1 *2 /2 1 Z<br>PSH NL0...NL255 |
| Terminal set§ | X0...X3 Y1 Y2 C0...C255 STK0...STK4 |
| $\beta$ limits for NL nodes | $\beta_{hi} = 10$ $\beta_{lo} = 0.1$ |
| Nodes with Gain | X Y STK C NL |
| Population size | 500 |
| Mutation probability | 0.01 |
| Size restrictions | at creation: maximum depth = 4<br>at crossover: maximum length = 25 |
| Fitness function† | $\frac{1}{1+MSE}$ $0 \leq f \leq 1$ |
| Input signal (X) | output of the channel when fed with a Pseudo-Random Binary Signal (PRBS) |
| Reference signal | the same PRBS delayed by one sample |
| Number of training samples | 250 samples;<br>the first 50 are not considered for fitness calculation. |
| Number of runs | 30 |
| Termination criterion for each run | 30 minutes of CPU time |
| Number of test samples | 100100 samples;<br>the first 100 are not considered for fitness calculation. |

§The 256 entries in the constant table are chosen uniformly within the interval [-1,1]
†The MSE or mean squared error is the average of the squared differences between the values of the output (Y) and those of the reference signal.

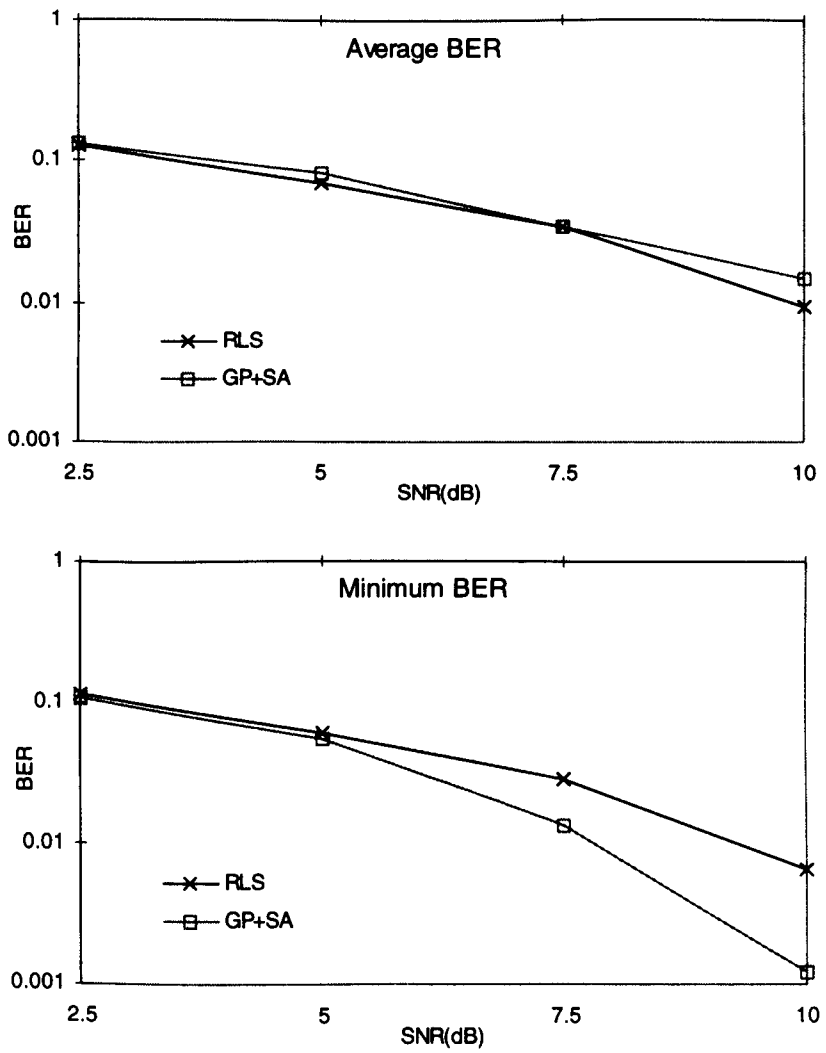


Figure 6.1: Average and minimum values of the BER obtained by GP+SA and RLS equalisers for the channel $H(z) = 1 + 0.7z^{-1}$ over 30 runs (per point).

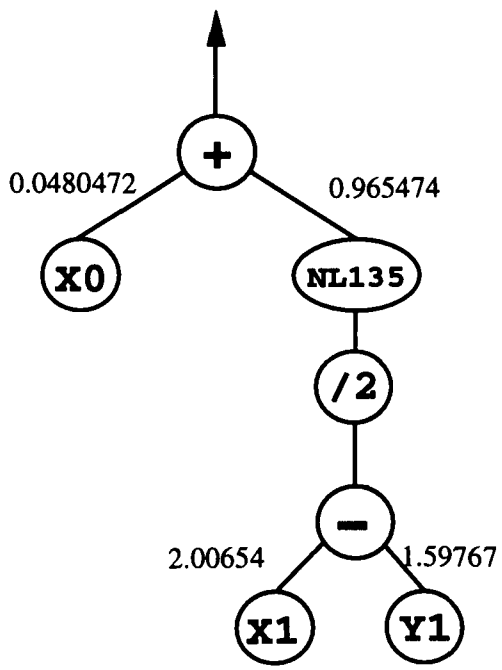


Figure 6.2: An equalising filter for the channel $H(z) = 1 + 0.7z^{-1}$. The node NL135 represents the function $\tanh(2.67x)$. Note that, although the channel is linear, the equaliser is nonlinear. In the presence of high levels of noise (in this case, the SNR was 7.5 dB), nonlinear equalisers can perform better than linear ones (Chen *et al.* 1990, Gibson *et al.* 1991, Gibson *et al.* 1989, Theodoridis *et al.* 1995).

## 6.4 Partial response channel

The transfer function of partial response channels has zeros on the unit circle. Such channels are frequently encountered in magnetic recording (Kechriotis *et al.* 1994) and since the inverse of the channel is undefined, there exists no linear filter that would sufficiently equalise them. Therefore nonlinear methods have to be used to reconstruct the originally transmitted signal.

Following (Kechriotis *et al.* 1994) the channel employed in the experiments had the transfer function

$$H(z) = 1 - 2 \cdot z^{-1} + z^{-2} \tag{6.2}$$

This channel has a double zero on the unit circle.

The performance of the proposed method was compared to that of the RLS algorithm, first for the noiseless case and then for different realisations of the signal to noise ratio. Thirty runs were performed in all cases and the values of the bit-error rate obtained are given in Table 6.4.

Table 6.4: Values of the bit-error rate obtained by equalisers for the channel $H(z) = 1 - 2 \cdot z^{-1} + z^{-2}$ over 30 runs.

| SNR(dB) | average BER | | minimum BER | |
|---|---|---|---|---|
| | RLS | GP+SA | RLS | GP+SA |
| 10 | 0.07537 | 0.10266 | 0.06197 | 0.0014 |
| 12.5 | 0.06866 | 0.09047 | 0.05944 | 0 |
| 15 | 0.05895 | 0.08109 | 0.04295 | 0 |
| 17.5 | 0.05135 | 0.11199 | 0.04356 | 0 |
| 20 | 0.04471 | 0.07155 | 0.03426 | 0 |
| 22.5 | 0.03498 | 0.06836 | 0.02914 | 0 |
| $\infty$ | 0.00623 | 0.07644 | 0.00515 | 0 |

A $t$-test was used to compare the two methods in the absence of noise showing that the average BER of the solutions was lower for the RLS algorithm. However, none of the RLS solutions obtained a BER of zero, while, on the other hand, this was achieved by a number of the GP+SA solutions.

One of the solutions obtained by the proposed method is shown in Figure 6.3.

In view of these results, the question that arises is as follows: If GP+SA can outperform the RLS algorithm by such a margin, why isn't the average BER of the GP+SA solutions much lower? The reason is that in many cases the algorithm converges towards a solution that is suboptimal, despite having a relatively high fitness. In the present case, the deceptive solution had the form shown in Figure 6.5.

This solution arose for various values of $n$ and the gains of $g_0$ and $g_1$. One possible cause for this is that the algorithm did not run for long enough to reach the optimum. A compromise

Table 6.5: Set up for partial response channel ($H(z) = 1 - 2 \cdot z^{-1} + z^{-2}$ ) equalisation experiment

| | |
|---|---|
| Function set | `+ - * % +1 -1 *2 /2 1 Z PSH NL0...NL255` |
| Terminal set | `X0...X3 Y1...Y2 C0...C255 STK0...STK4` |
| $\beta$ limits for `NL` nodes | $\beta_{hi} = 10 \qquad \beta_{lo} = 0.1$ |
| Nodes with Gain | `X Y STK C NL` |
| Population size | 500 |
| Mutation probability | 0.01 |
| Size restrictions | at creation: maximum depth = 4<br>at crossover: maximum length = 25 |
| Fitness function | $\frac{1}{1+MSE} \qquad 0 \leq f \leq 1$ |
| Input signal (X) | output of the channel when fed with a Pseudo-Random Binary Signal (PRBS) |
| Reference signal | the same PRBS delayed by one sample |
| Number of training samples | 550 samples (noiseless); 250 (with noise)<br>the first 50 are not considered for fitness calculation. |
| Number of runs | 30 |
| Termination criterion for each run | 30 min. of CPU time (with noise);<br>60 min. (noiseless) |
| Number of test samples | 100100 samples;<br>the first 100 are not considered for fitness calculation. |

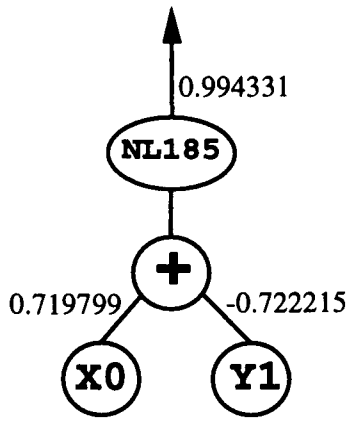


Figure 6.3: An equalising filter for the channel $H(z) = 1-2z^{-1}+z^{-2}$. The node NL185 represents the function $\tanh(3.64x)$. This was obtained with a training signal of SNR = 12.5 dB and when tested with signals of other SNR realisations (10, 12.5, 15, 17.5 and 20 dB), yielded a BER of zero in every case but one (for SNR = 10dB the BER was 0.00159). The other solutions were similar in structure to this one.

must then be found between a bigger number of shorter runs and a smaller number of longer, and possibly more successful ones.

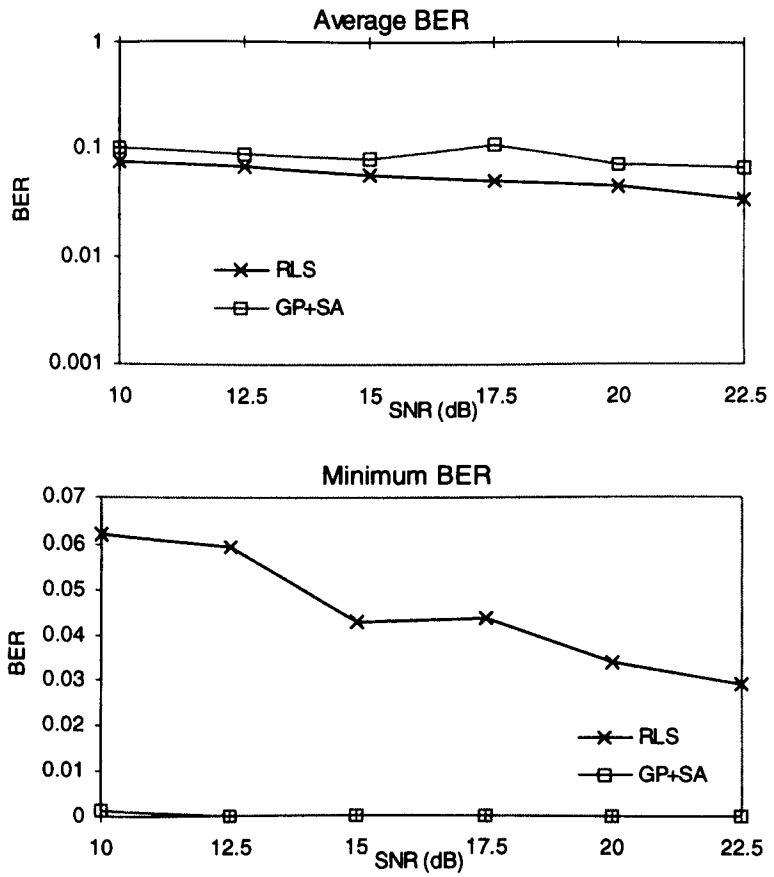


Figure 6.4: Average and minimum BER for different SNR realisations obtained by GP+SA and RLS equalisers over 30 runs (per point). The channel employed was $H(z) = 1 - 2z^{-1} + z^{-2}$. Note that in this case the minimum BER is displayed using a linear scale and not in the commonly employed logarithmic scale, because most of the values for GP+SA are zero.


Figure 6.5: A suboptimal equaliser for the channel $H(z) = 1 - 2z^{-1} + z^{-2}$

## 6.5 Non linear channel

Nonlinear channels are the obvious cases to tackle with nonlinear equalisation techniques. Such channels are frequently encountered in data transmission over digital satellite links, especially when the signal amplifiers operate in their high-gain limits (Kechriotis *et al.* 1994).

The channel used in the experiments (Kechriotis *et al.* 1994, Chen *et al.* 1990) follows the model shown in Chapter 3 (Figure 3.3) and is given by equations 6.3 and 6.4.

$$\tilde{y}_n = 0.3482 \cdot x_n + 0.8704 \cdot x_{n-1} + 0.3482 \cdot x_{n-2} \quad (6.3)$$

$$\hat{y}_n = \tilde{y}_n + 0.2 \cdot \tilde{y}_n^2 \quad (6.4)$$

The channel output, $\hat{y}$ , is further corrupted by the addition of white Gaussian noise.

The procedure for training and testing for GP+SA and the RLS algorithm is as explained for the linear channel and the settings for this problem are the same as those given in Table 6.3.

The values of the bit-error-rate obtained are given in Table 6.6 and displayed graphically in Figure 6.7. From it we can conclude that GP equalisers outperform linear equalisers trained by the RLS algorithm, both on the average and the minimum values.

An example of a solution obtained by the proposed method is given in Figure 6.6. This tree was obtained with a training signal whose SNR was 15 dB. Other solutions obtained for various SNR realisations had a similar structure, with an NL node at the top and a subtree with a + (or -) node at the root and Y1 and X0 nodes as branches.

Table 6.6: Values of the bit-error rate obtained by equalisers for the nonlinear channel given by equations 6.3 and 6.4.

| SNR(dB) | average BER | | minimum BER | |
|---|---|---|---|---|
| | RLS | GP+SA | RLS | GP+SA |
| 5 | 0.153082 | 0.131444 | 0.13183 | 0.12431 |
| 7.5 | 0.111524 | 0.093784 | 0.09914 | 0.08232 |
| 10 | 0.084465 | 0.064212 | 0.06722 | 0.04311 |
| 12.5 | 0.055214 | 0.044546 | 0.04454 | 0.01466 |
| 15 | 0.039135 | 0.026311 | 0.02155 | 0.00349 |
| 17.5 | 0.028946 | 0.016541 | 0.01095 | 0.00025 |

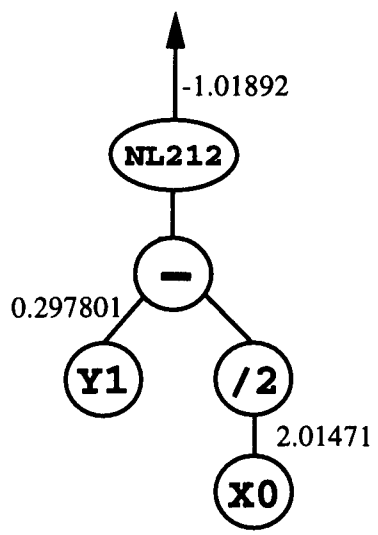


Figure 6.6: An equalising filter for the channel $H(z) = 0.3482 + 0.8704z^{-1} + 0.3482z^{-2}$ with nonlinear gain $d_2 = 0.2$. The node NL212 represents the function $\tanh(4.16x)$.

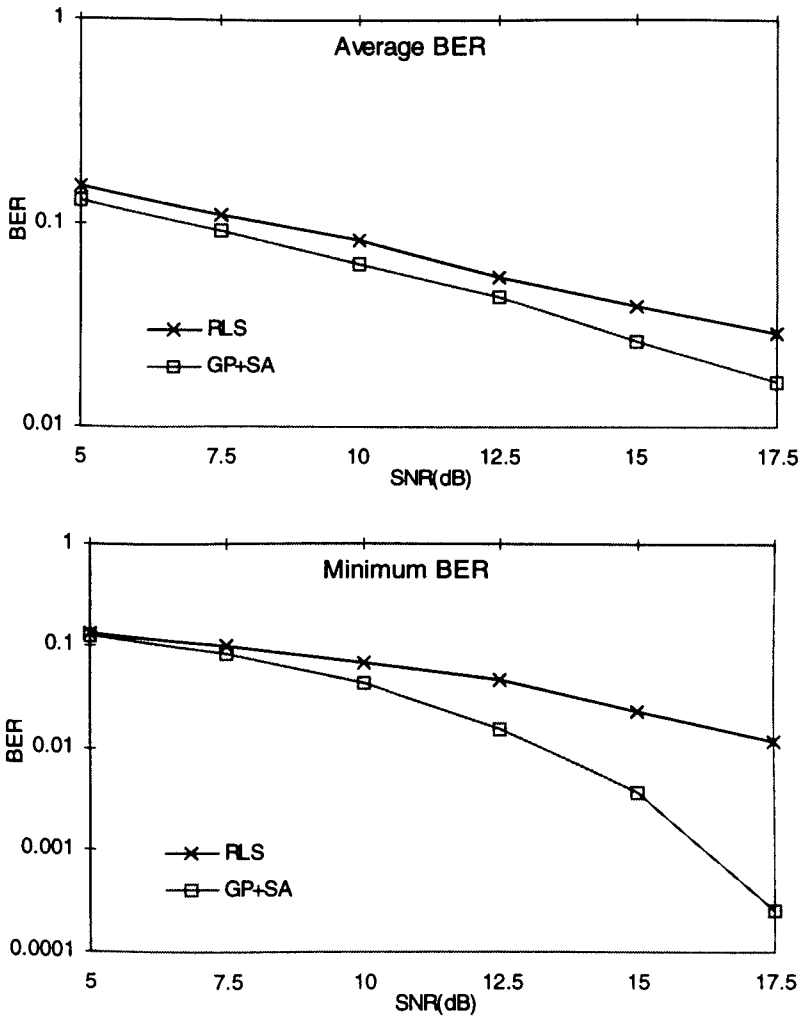


Figure 6.7: Average and minimum values of the BER for the channel $H(z) = 0.3482 + 0.8704z^{-1} + 0.3482z^{-2}$ with nonlinear gain $d_2 = 0.2$. Results obtained by GP+SA and RLS equalisers over 30 runs (per point).

## 6.6 Comparison

A comparison with the results obtained by (Kechriotis *et al.* 1994) and (Parisi *et al.* 1997) is attempted here. Such a task is not easy, due to the difficulty in obtaining reliable measures from published results. Furthermore, the methods employed by (Kechriotis *et al.* 1994) and (Parisi *et al.* 1997) are based on a different philosophy than that of the GP+SA method presented here.

The minimum BER results achieved by the proposed method were consistently lower that the average values obtained by (Kechriotis *et al.* 1994) and minimum values of (Parisi *et al.* 1997) for the linear and partial response channels, and similar for the nonlinear one.

However, the average results only outperformed those of (Kechriotis *et al.* 1994) for the high noise cases (SNR $\leq$ 7.5 dB) of the linear channel.

This is consistent with our expectation. Further adjustments in the method are needed to achieve optimal performance in these problems.

## 6.7 Conclusions

The node gain GP+SA method has proven successful in a number of difficult channel equalisation problems. However, it must be pointed out that superior performance of the proposed method over other techniques such as the RLS algorithm is not necessarily observable on the average (except perhaps in the case of the nonlinear channel of section 6.5) but rather on the extremes i.e. maximum fitness or minimum error across runs. GP+SA runs can be deceptive, but they can also lead to interesting solutions that would not be found by other techniques.

As a consequence and in order to guarantee success, a number of runs must be performed. This is not the case of the RLS algorithm, as shown by the small variance of the solution's fitness across runs.

The improvement in the performance gained by the use of GP+SA sufficiently justifies its use in channel equalisation instead of a less computationally expensive technique such as the RLS algorithm, as long as the specific situation allows for a number of runs to be performed and for the user to select the best solution of all of them.

As a summary of characteristics of the proposed method can be cited, first of all, that the structure of the adaptive filter is evolved, which allows for exploration of the configuration space.

Second, the possibility of using various criteria (even simultaneously) for measuring the fitness, not necessarily based on the mean squared error as is the case with the RLS algorithm. These could involve other measures of the error (such as the maximum absolute error or $H\infty$, the bit-error rate etc.) or other criteria such as the Fourier transform, correlation coefficients etc. This is important when non Gaussian signals are involved or when the mean squared error is not a meaningful source of information.

Third, the performance is relatively independent on the number of samples used in the training stage. GP+SA can produce good results even with small sample sizes, which is never the case of RLS and other algorithms.

All these good features could not come without drawbacks. The first one is the computational expense of the proposed method. In a field such as communications, this is a major inconvenient. However, other applications of signal processing do not have such time constraints, so GP+SA could be successfully applied in those cases. In any case, it must be pointed out that other techniques that are widely employed nowadays were regarded as impractically time-consuming in their early days.

And second, success is not guaranteed. Despite being a powerful technique there always exists the possibility of deception, however small. This can be minimised by selecting carefully the fitness function and the parameters of GP evolution (such as maximum tree size, sample size etc.). Furthermore, if , as has been said, it is possible to perform a number of runs, the risk of deception is minimal and the most convenient solution can be chosen.

## 6.8 Summary

This chapter has been devoted to the problem of channel equalisation. The shortcomings of existing techniques have been exposed and the main requirements to demand of a new method enumerated.

A description of the problem has been followed by an account of particular cases of equalisation, exemplified by three channels, both linear and nonlinear. Results have been obtained both with the proposed method and a classical technique, the RLS algorithm, showing the great potential of the former, especially in the case of nonlinear channels.

Disadvantages of the proposed method, such as the computational effort involved, have also been pointed out and it has been concluded that, upon the whole, the method deserves further investigation.

# Chapter 7

# Conclusions and further work

## 7.1 Conclusions

This thesis has been devoted to attaining the following objectives:

- presenting a new methodology for representing Discrete Dynamic Systems (DDS) as expression trees
- introducing a new way to represent numerical parameters in expression trees, in the form of node gains
- implementing a Simulated Annealing algorithm that performs local learning by adapting the values of the node gains

In chapter 2 it was shown how the equations describing discrete time systems can be represented as symbolic expression trees, or S-trees. Various kinds of node types and their associated data structures were described to complete the representation system.

The power of this system was illustrated by an example, the coding of a Recurrent Neural Network as a list of S-trees. This covered a first main requirement: capturing the dynamics of discrete-time systems.

Chapter 3 showed how GP can be applied to channel equalisation using the node types defined in chapter 2. GP serves to overcome problems of classical equalisation techniques by allowing the structure of the solution to evolve rather than being preselected by the user. In this way, nonlinear and recursive structures can be obtained.

Stability monitoring is intrinsic to the evolution process: unstable candidate solutions will yield a high value of the error and therefore a low fitness and will eventually be culled from the population.

However, in the examples presented in this chapter the evolved solutions tended to be complex or did not yield a good performance. This points out to the convenience of achieving

a decrease in complexity in some cases (in order to increase readability and facilitate possible implementation) and an improvement in the performance in others.

Chapters 4 and 5 covered a second main requirement: the representation and adaptation of the parameters of the systems.

Experimental analysis done in chapter 4 showed that there is a significant degradation in the performance of standard GP when applied to problems involving rational parameters, rather than integers. This justified the investigation of new approaches to handling numerical parameters.

Even though the introduction of node gains did not necessarily bring an advantage over standard GP, the results showed the potential benefits that can be attained, exemplified by a higher maximum fitness and a smaller minimum size. These benefits are difficult to attain if the values of the gains are random, hence the need for adaptation to take full advantage of node gains.

Such an algorithm was introduced in chapter 5. Here an analogy was drawn between an adaptation algorithm for the node gains and the learning process in nature. In this way one is able to envisage how individual adaptation affects the whole system.

A step beyond the natural analogy was the introduction of Lamarckian inheritance, whose implementation in the node gain GP system was presented here.

The learning algorithm was implemented by means of Simulated Annealing, chosen due to its similarities to natural learning. It also offers a number of possibilities in the selection of parameters and different schedules used, in contrast with the GP algorithm which is somewhat more rigid.

Introducing SA in the node gain GP system provided a faster way of finding solutions to engineering problems and a more robust one in the case of variable environments.

Two general decision rules as to when to apply a learning algorithm were given :

- when the performance of a system is sensitive to parameter variations; for instance, when a wrong selection of parameters can make a system unstable
- when environmental changes are expected; for instance in equalisation in mobile systems, where the unknown channel is constantly varying.

Two possible learning schemes for GP were addressed here, namely Darwinian and Lamarckian. It was shown that, in the examples presented, using node gains provided better results than those obtained by standard GP. Furthermore, the use of learning improved the performance in the two more complex problems addressed.

A measure of the generalisation ability was introduced which showed that overfitting is not a problem in any of the learning schemes presented; on the contrary, both generalised better than the non learning methods. This might be due to the fact that the solutions

obtained with any of the learning schemes tended to be smaller than those obtained without learning. Further analysis is required to prove this point.

Although statistical tests did not allow us to reach a conclusion as to which performed better, the Darwinian scheme looked intuitively more indicated for a variety of problems, as the more robust of the two.

In chapter 6 node gain GP+ SA method was successfully applied to a number of difficult channel equalisation problems. However, it must be pointed out that superior performance of the proposed method over other techniques such as the RLS algorithm is not necessarily observable on the average but rather on the extremes i.e. maximum fitness or minimum error across runs. GP+SA runs can be deceptive, but they can also lead to interesting solutions that would not be found by other techniques.

The improvement in the performance gained by the use of GP+SA sufficiently justifies its use in channel equalisation instead of a less computationally expensive technique such as the RLS algorithm, as long as the specific situation allows for a number of runs to be performed and for the user to select the best solution of all of them.

As a summary of characteristics of the proposed method can be cited, first of all, that the structure of the adaptive filter is evolved, which allows for exploration of the configuration space.

Second, the possibility of using various criteria (even simultaneously) for measuring the fitness, not necessarily based on the mean squared error as is the case with the RLS algorithm. These could involve other measures of the error (such as the maximum absolute error or $H\infty$, the bit-error rate etc.) or other criteria such as the Fourier transform, correlation coefficients etc. This is important when non Gaussian signals are involved or when the mean squared error is not a meaningful source of information.

Third, the performance is relatively independent on the number of samples used in the training stage. GP+SA can produce good results even with small sample sizes, which is never the case of RLS and other algorithms.

All these good features also involve drawbacks. The first one is the computational expense, which, in a field such as communications, is a major inconvenient. However, other applications of signal processing do not have such time constraints, so GP+SA could be successfully applied in those cases. It must also be pointed out that other techniques that are widely employed nowadays, were regarded as impractically time-consuming in their early days.

The second drawback is that success is not guaranteed. Despite being a powerful technique there always exists the possibility of deception, however small. This can be minimised by selecting carefully the fitness function and the parameters of GP evolution (such as maximum tree size, sample size etc.). Furthermore, if, as has been said, it is possible to perform a number of runs, the risk of deception is minimal and the most convenient solution can be chosen.

## 7.2 Further work

This work has left open a number of interesting threads for future research. Some of them are summarised below.

On the theoretical front, further work remains to be done on the adaptation (or *learning*) process. On the one hand, it would be interesting to determine the differences between Darwinian and Lamarckian evolution. This might help to give an insight as to why Lamarckian evolution is not observed in nature.

Also, a number of changes could be easily introduced in the implementation to make the adaptation process more similar to learning in Nature.

As per the implementation, another interesting point would be to introduce an umber of mutation operators, as done by (Chelapilla 1997), and hence produce a hybrid Genetic-Evolutionary Programming engine. This would help understand the relative importance of crossover and mutation operators in a tree-based evolutionary system.

On the applications front, the proposed method has still to be tried on a number of Digital Signal Processing problems, only a few of which have been attempted by this author (Esparcia-Alcázar 1997, Esparcia-Alcázar and Sharman 1996). These would include noise cancellation, interference removal, blind equalisation and others.

Also, applications in other areas, such as Control, could benefit by the introduction of the proposed method.

# Appendix A

# Implementation of the SA algorithm

In section 1.5 the main characteristics of a Simulated Annealing algorithm were listed as follows:

- a perturbation–generating probability distribution.
- an acceptance/rejection probability distribution.
- a cooling schedule.

These are described here in more detail. First, the main SA algorithm is outlined is section A.1. Section A.2 proceeds giving an account of the two most commonly employed probability distributions for the generation of the perturbations, the normal and the Cauchy-lorentzian, and those used for acceptance/rejection (Boltzmann and Fermi–Dirac) In section A.3 three different cooling schedules are described and their dependence on the choice of perturbation-generating probability distribution.

Finally, section A.4 introduces a new feature which is specific to the adaptation of gain vectors, the perturbation scheme. Two such schemes, labelled *independent* and *elastic*, are described here.

## A.1 Main algorithm

The SA algorithm for expression trees works as follows:

1. Initialise parameters ($T_0$ , `maxTrials`, `trials`, `maxIterations`, `iterations`, see below)
2. Perturb $\vec{g}(i)$ to get $\vec{g}(i)$
3. Evaluate the fitness, $f(i)$ using the perturbed gain vector $\vec{g}(i)$

4. If $(f(i) \geq f(i)$ then accept the perturbation: $\vec{g}(i+1) = \vec{g}(i)$ and continue
   else accept the perturbation with probability $p_\alpha$ and continue

5. If `trials` > `maxTrials` then `trials` = 0
   else `trials`++ and go back to step 2

6. Reduce the temperature $T$ according to annealing schedule

7. If `iterations` < `maxIterations` then `iterations`++ and go back to step 2.

The algorithm is summarised in figure A.1.

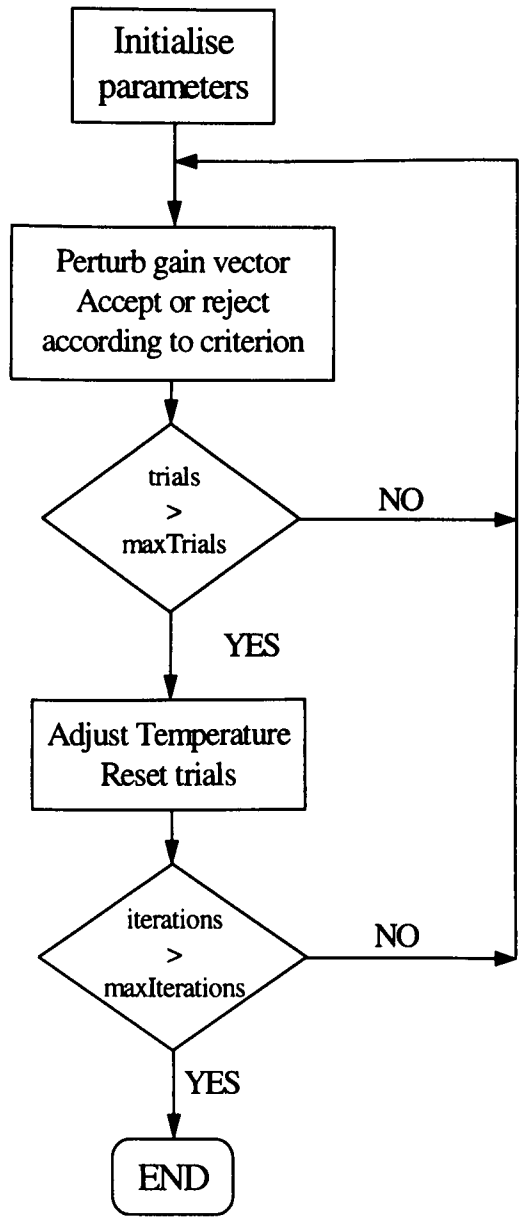


Figure A.1: The Simulated Annealing algorithm

## A.2 Perturbation–generating and acceptance/rejection probability distributions

With regard to the generation of the perturbation it is usual to distinguish between *classical* SA (CSA), which employs a normal distribution, and *fast* SA (FSA), employing a Cauchy-lorentzian distribution. The latter allows for big "jumps" in the perturbations, which might be more convenient for escaping local optima.

The acceptance/rejection probability can be calculated using the Boltzmann distribution

$$\exp^{\frac{-\Delta E}{kT}}$$

as in (Kirkpatrick *et al.* 1983), or the Fermi-Dirac distribution

$$\frac{1}{\exp^{\frac{\Delta E}{kT}+1}}$$

as in (Szu and Hartley 1987), where $\Delta E$ is the increase in energy involved in the transition and $k$ is Boltzmann's constant. For the purposes of this work $\Delta E$ will be substituted with $-\Delta f$ , (i.e. the decrease in fitness) and $k$ will be taken equal to 1.

Because the fitness always lies in the interval $[0, 1]$ then $-\Delta f \in [-1,\ 1]$, but only the sub interval $[0,\ 1]$ is considered, due to the fact that the probability is only invoked when the transition involves a decrease in fitness (i.e. $\Delta f\ <\ 0$).

In the region $-\Delta f \gg T$ (e.g. when $-\Delta f$ is close to 1 and $T$ is close to 0) the Fermi-Dirac distribution becomes identical with the Boltzmann one (Dekker, 1958, pp. 213–215). Elsewhere the former lies below the latter, which means the Fermi-Dirac distribution provides a more conservative annealing schedule than the Boltzmann distribution.

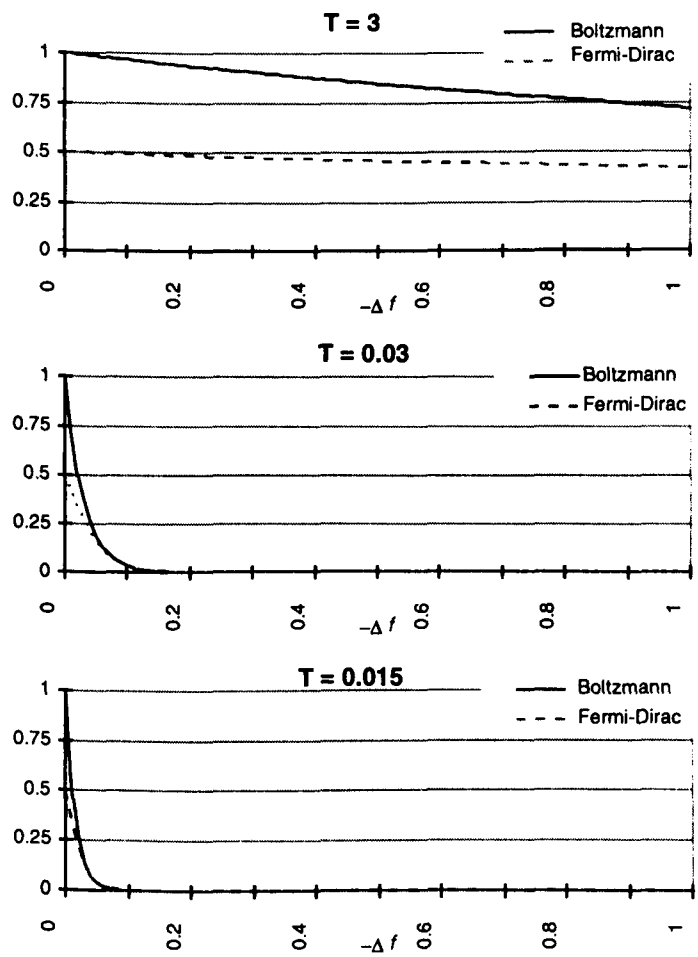


Figure A.2: Acceptance probabilities using the Boltzmann and Fermi-Dirac distributions at three different temperatures: 3, 0.03 and 0.015. These temperatures correspond to iterations 0, 100 and 200 for a cooling schedule inversely linear in time ($T = \frac{T_0}{n}$) with starting temperature $T_0 = 3$.
The Fermi-Dirac graph is always below the Boltzmann one, which means decreases in fitness are less likely to be accepted, thus resulting in a more *greedy* algorithm.

## A.3 Cooling schedules

The choice of distribution places a lower bound on the cooling schedule: the temperature $T$ should change no faster than $\frac{T_0}{\ln(n+1)}$ for CSA and $\frac{T_0}{n+1}$ for FSA, where $T_0$ is the starting temperature and $n$ the number of iterations. The first schedule is called *inverse logarithmic in time* and the second *inverse linear in time*. It is obvious that the latter provides a faster reduction in temperature, hence the name *fast* SA. However, a faster decrease in temperature also means that more effort should be spent to restore equilibrium. This is measured by the number of accepted trials per temperature step or, more simply, by the total number of trials per temperature step.

Another popular cooling schedule consists of setting the number of trials to 1 and updating the temperature as

$$T_n = \alpha T_{n-1} = \alpha^n T_0 \qquad n = 1, 2 \ldots \tag{A.1}$$

where $\alpha$ (the decay) is typically between 0.9 and 0.99 (van Laarhoven 1988). This will be referred to as an *exponential* cooling schedule.

The termination criterion for the algorithm can be expressed in terms of a given number of iterations or a maximum number of invariant trials. In this work both criteria will be used.

Thus, three basic parameters must be set: the starting temperature, $T_0$, the maximum number of trials per temperature, `maxTrials`, and the maximum number of iterations (or temperature steps), `maxIterations`. If the exponential cooling schedule is used, a further parameter to set is the decay, $\alpha$.

## A.4 Perturbation schemes

Two schemes for generating perturbations are devised, which will be referred to as "independent" and "elastic".

**Independent perturbations:** The components of the gain vector are random variables whose distribution is dependent only on the temperature. The amount of perturbation received by the gain of a particular node is therefore independent of the perturbations received by the gains of other nodes inside the same tree.

The algorithm for generating independent perturbations is as simple as:

1. Generate random values for $p_i$ (e.g. from $N(0, \alpha T)$, $\alpha \in \Re$)
2. The new gain vector is $\vec{g} = \vec{g} + \vec{p}$

**Elastic perturbations:** For these, the vectorial nature of the gains in a given tree is taken into account. The global perturbation affecting a tree is a vector whose modulus is a constant that depends only on the temperature. The individual components of the

vector (i.e. the way the perturbation is distributed among the different nodes) are random and their distribution is independent of the temperature. This vector is then added to the node gain vector, $\vec{g}$.
The algorithm is as follows:

1. Calculate the amount of perturbation, $p = f(T)$
2. Generate random values for $q_i$ (e.g. from $N(0,1)$ )
3. Calculate $q^2 = \sum_i q_i^2$
4. Find $\beta$ such that $\beta = \frac{p}{q}$
5. Scale $\beta$ by the length of the vector
6. Calculate the individual components of the vector: $p_i = \beta \cdot q_i$
7. The new gain vector is $\vec{g} = \vec{g} + \vec{p}$

Therefore if a specific node gain is affected by a big perturbation ("stretching") then elsewhere in the tree other nodes are affected by small ones ("shrinking"), hence the denomination "elastic".

Although no statistical analysis has been carried out so far to compare both perturbation schemes, the elastic one seems to provide a more successful annealing.

# Appendix B

# Statistical analysis for Chapter 4

## B.1 Two sample *t*-test

### Assumptions

The data consists of two random samples of sizes $m$ and $n$. Usually, the two samples are supposed to come from normal distributions, but because the size of both samples is 30 the Central Limit Theorem can be applied (Devore, 1995, p. 234).
Assuming both samples have different variances, a variant of the *t*-test called the *Smith–Satterthwaite test* of approximate level $\alpha$ can be employed.

### Hypotheses

$H_0$: The difference between the two populations'[1] means is $\Delta_0$, $\mu_1 - \mu_2 = \Delta_0$

$H_a$: 1. $\mu_1 - \mu_2 > \Delta_0$

2. $\mu_1 - \mu_2 < \Delta_0$

3. $\mu_1 - \mu_2 \neq \Delta_0$

### Test statistic

The test statistic value is calculated as

$$t' = \frac{\bar{x} - \bar{y} - \Delta_0}{\sqrt{\frac{s_1^2}{m} + \frac{s_2^2}{n}}}$$

where $\mu_1$ and $\mu_2$ are the sample means and $s_1$ and $s_2$ are the sample standard deviations

[1] Here, methods

for populations 1 and 2, respectively. The degrees of freedom are given by

$$\nu = \frac{\left(\frac{s_1^2}{m} + \frac{s_2^2}{n}\right)^2}{\frac{\left(\frac{s_1^2}{m}\right)^2}{m-1} + \frac{\left(\frac{s_2^2}{n}\right)^2}{n-1}}$$

### Decision rule

Reject $H_0$ and accept the alternative hypothesis $H_a$ when

1. $t \geq t_{\alpha,\nu}$
2. $t \leq -t_{\alpha,\nu}$
3. either $t \geq t_{\frac{\alpha}{2},\nu}$ or $t \leq -t_{\frac{\alpha}{2},\nu}$

where $t_\alpha$ is the $\alpha$ quantile of the $t$ distribution with $\nu$ degrees of freedom.

## B.2 *t*-tests for fitness

Two sets of 30 runs each are realised. First, the $t$-test is performed for the fitness values. The results of the test for the first set of runs are given in table B.1.

Table B.1: $t$-Test for fitness: Two-Sample Assuming Unequal Variances

| | node gain GP | standard GP |
|---|---|---|
| Mean | 0.5728 | 0.8978 |
| Variance | 0.2210 | 0.0594 |
| Observations | 30 | 30 |
| Hypothesised Mean Difference | 0 | |
| Degrees of freedom | 44 | |
| $t$ Statistic | -3.3616 | |
| P($T \leq t$) one-tail | 0.0008 | |
| t Critical one-tail | 1.6802 | |
| P($T \leq t$) two-tail | 0.0016 | |
| $t$ Critical two-tail | 2.0154 | |

Since the statistic $t < -t_{critical}$ for one and two tails, the conclusion of the test is to accept $H_a$, i.e. the mean of the fitnesses for node gain GP is lower than for standard GP.

$$E(ngGP) < E(stdGP) \tag{B.1}$$

Table B.2: $t$-Test for fitness: Two-Sample Assuming Unequal Variances

| | node gain GP | standard GP |
|---|---|---|
| Mean | 0.6222 | 0.8129 |
| Variance | 0.1774 | 0.1068 |
| Observations | 30 | 30 |
| Hypothesised Mean Difference | 0 | |
| Degrees of freedom | 55 | |
| $t$ Statistic | -1.9594 | |
| P($T \leq t$) one-tail | 0.0276 | |
| $t$ Critical one-tail | 1.6730 | |
| P($T \leq t$) two-tail | 0.0551 | |
| $t$ Critical two-tail | 2.0040 | |

For the second set of runs, the test is shown in table B.2. Here $t < -t_{critical}$ for one tail and $>$ for two tails, so the test is inconclusive.

## B.3 *t*-tests for length

The same procedure is followed here for the solution lengths. In this case and for both sets of runs $t > t_{critical}$ for one and two tails, so the conclusion is to accept $H_a$, i.e. the average length of the solutions obtained by node gain GP was greater than those obtained by standard GP.

Table B.3: *t*-Test for length: Two-Sample Assuming Unequal Variances

| | node gain GP | standard GP |
|---|---|---|
| Mean | 93.9 | 37.7333 |
| Variance | 4953.9552 | 879.5126 |
| Observations | 30 | 30 |
| Hypothesised Mean Difference | 0 | |
| df | 39 | |
| $t$ Statistic | 4.0279 | |
| P($T \leq t$) one-tail | 0.0001 | |
| $t$ Critical one-tail | 1.6849 | |
| P($T \leq t$) two-tail | 0.0003 | |
| $t$ Critical two-tail | 2.0227 | |

Table B.4: $t$-Test for length: Two-Sample Assuming Unequal Variances

| | node gain GP | standard GP |
|---|---|---|
| Mean | 81.7667 | 42.4667 |
| Variance | 8199.6333 | 1391.7057 |
| Observations | 30 | 30 |
| Hypothesised Mean Difference | 0 | |
| Degrees of freedom | 39 | |
| $t$ Statistic | 2.1979 | |
| P($T \leq t$) one-tail | 0.0170 | |
| $t$ Critical one-tail | 1.6849 | |
| P($T \leq t$) two-tail | 0.0340 | |
| $t$ Critical two-tail | 2.0227 | |

# Appendix C

# Statistical analysis for Chapter 5

## C.1 The Kruskal-Wallis test

The data consist of $k$ random samples of possibly different sizes (Conover 1980, pp. 229–231). Denote the $i_{th}$ random sample of size $n_i$ by $X_{i_1}, X_{i_2}, \ldots, X_{i_{n_i}}$.

Let $N$ denote the total number of observations,

$$N = \sum_{i=1}^{k} n_i \tag{C.1}$$

Assign rank 1 to the smallest of the totality of $N$ observations, rank 2 to the second smallest and so on, to the largest of all $N$ observations, which receives rank $N$. Let $R(X_{ij})$ represent the rank assigned to $X_{ij}$. Let $R_i$ be the sum of the ranks assigned to the $i^{th}$ sample.

$$R_i = \sum_{j=1}^{k} R(X_{ij}) \tag{C.2}$$

Compute $R_i$ for each sample.

### Assumptions

1. All samples are random samples from their respective populations.
2. In addition to independence within each sample, there is mutual independence among the various samples.
3. Either the $k$ population distribution functions are identical, or else some of the populations[1] tend to yield larger values than other populations do.

[1]Here, methods

### Hypotheses

$H_0$: All of the $k$ population distributions are identical.

versus

$H_a$: At least one of the populations tends to yield larger observations than at least one of the other populations.

The alternative hypothesis can be also stated as follows:

$H_a$: The $k$ populations do not all have identical means.

### Test statistic

This is defined as

$$T = \frac{1}{S^2}\left(\sum_{i=1}^{k}\frac{R_i^2}{n_i} - \frac{N(N+1)^2}{4}\right) \tag{C.3}$$

where

$$S^2 = \frac{1}{N-1}\left(\sum_{\text{all ranks}} R(X_{ij})^2 - \frac{N(N+1)^2}{4}\right) \tag{C.4}$$

### Decision rule

The approximate quantiles may be obtained from the chi-square ($\chi^2$) distribution with $k-1$ degrees of freedom. Reject $H_0$ at the level $\alpha$ if $T$ exceeds the $1-\alpha$ quantile thus obtained.

### Multiple comparisons

If and only if the null hypothesis is rejected, the following procedure may be used to determine which pairs of populations tend to differ. It can be said that populations $i$ and $j$ seem to be different if the following inequality is satisfied

$$\left|\frac{R_i}{n_i} - \frac{R_j}{n_j}\right| > t_{1-\frac{\alpha}{2}}\left(S^2\frac{N-1-T}{N-k}\right)^{\frac{1}{2}}\left(\frac{1}{n_i}+\frac{1}{n_j}\right)^{\frac{1}{2}} \tag{C.5}$$

where $R_i$ and $R_j$ are the rank sums of the two samples, $t_{1-\frac{\alpha}{2}}$ is the $1-\frac{\alpha}{2}$ quantile of the $t$ distribution with $N-k$ degrees of freedom. This procedure is repeated for all pairs of populations. The same $\alpha$ level is used here as in the Kruskal-Wallis test.

## C.1.1 Kruskal-Wallis test for LC1

The critical region of approximate size $\alpha = 0.01$ corresponds to values of $T$ greater than the 0.99 quantile of the chi-square random variable with $k-1=3$ degrees of freedom, which can

be obtained from tables and is equal to 11.34.
The number of runs per method, $n$, is 50, which yields a total number of observations, $N$, of

$$N = 200$$

The value of $S^2$ is

$$S^2 = 3349.997$$

This yields a $T$ of

$$T = 43.37$$

which clearly leads to a rejection of $H_0$. Now the multiple comparison procedure can be used.

The value of the $t_{1-\frac{\alpha}{2}}$ quantile can be obtained from a $t$-distribution table. The degrees of freedom can be calculated as

$$N - k = 200 - 4 = 196$$

The table does not provide this value, so the quantile will be approximated using an infinite number of degrees of freedom, which yields the following value for $t_{1-\frac{\alpha}{2}}$

$$t_{0.995} = 2.576$$

Because all the $n_i$'s are the same, equal to $n$, the right hand side of inequality C.5 remains the same for all comparisons, its value being 26.57. The values of the left hand side of C.5 for each method and the results of the comparison are given in table C.1.

Table C.1: Multiple comparison for LC1. Critical value: 26.57

| Methods | | Statistic | Result | Conclusion |
|---|---|---|---|---|
| Darwinian | Lamarckian | 0.68 | < | same |
| Darwinian | RGNL | 16.71 | < | same |
| Darwinian | NGNL | 54.79 | > | different |
| Lamarckian | RGNL | 17.39 | < | same |
| Lamarckian | NGNL | 54.11 | > | different |
| RGNL | NGNL | 71.5 | > | different |

### C.1.2 Kruskal-Wallis test for NLC

The value of $n$ is 24 which results in an $N$ of 96.The value of $T$ is

$$T = 17.15$$

For the same level of confidence, $\alpha = 0.01$, the quantile is the same as before, 11.34, which results in the rejection of $H_0$.

The number of degrees of freedom for the comparison can be calculated as

$$N - k = 96 - 4 = 92$$

The table provides values for 60 and 120 degrees of freedom, which are

$$t_{0.995,60} = 1.671$$
$$t_{0.995,120} = 1.658$$

Of these, we will take 1.671 as the more restrictive. The values of the statistic for each method and the results of the comparison are given in table C.2.

Table C.2: Multiple comparison for NLC. Critical value: 1.671

| Methods | | Statistic | Result | Conclusion |
|---|---|---|---|---|
| Darwinian | Lamarckian | 9.89 | < | same |
| Darwinian | RGNL | 23.23 | < | different |
| Darwinian | NGNL | 30.45 | > | different |
| Lamarckian | RGNL | 13.33 | < | different |
| Lamarckian | NGNL | 20.56 | > | different |
| RGNL | NGNL | 7.23 | > | different |

### C.1.3 Kruskal-Wallis test for LC1 → LC2

The value of $n$ in this case is 51 which yields an $N$ of 204. The computed value of T is

$$T = 55.544$$

and the quantile for $\alpha = 0.01$ is the same as in the previous section, 11.34. This results again in the rejection of $H_0$.

To obtain the value of the $t_{1-\frac{\alpha}{2}}$ quantile, the degrees of freedom can be calculated as

$$N - k = 204 - 4 = 200$$

As before, the quantile will be approximated for the $t$ distribution with an infinite number of degrees of freedom, which yields the same value for $t_{1-\frac{\alpha}{2}}$,

$$t_{0.995} = 2.576$$

The right hand side of inequality C.5 is equal to 25.85. The values of the left hand side for each method, together with the results of the comparison, are given in table C.3.

Table C.3: Multiple comparison for LC1→LC2. Critical value: 25.85

| Methods | | Statistic | Result | Conclusion |
|---|---|---|---|---|
| Darwinian | Lamarckian | 0.67 | < | same |
| Darwinian | RGNL | 29.78 | > | different |
| Darwinian | NGNL | 74.92 | > | different |
| Lamarckian | RGNL | 30.45 | > | different |
| Lamarckian | NGNL | 75.59 | > | different |
| RGNL | NGNL | 45.14 | > | different |

# Appendix D

# Recursive Least Squares algorithm for adapting Finite Impulse Response filters

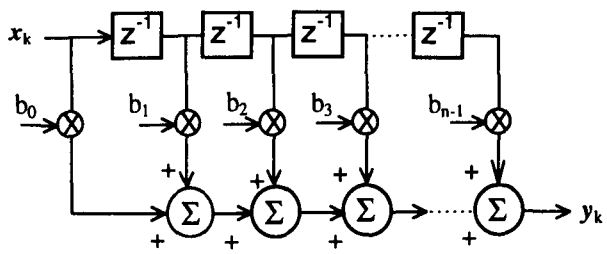


Figure D.1: An $n$-tap FIR digital filter. The filter coefficients are represented by $b_i$ and $z^{-1}$ represents a unit time delay.

The estimate of the transmitted symbol at time $t$ is given by (Kechriotis et al., 1994)

$$\hat{x}_t = \sum_{k=0}^{N-1} c_k \hat{y}_{t-k} = c^T \hat{y}_{e,t} \tag{D.1}$$

where:

$$c = [c_0 \; c_1 \; \cdots \; c_{N-1}]^T \tag{D.2}$$

and

$$\hat{y}_{e,t} = [\hat{y}_t \; \hat{y}_{t-1} \; \cdots \; \hat{y}_{t-N+1}] \tag{D.3}$$

During the adaptation period, at every time instant $t$, the filter's error

$$e_t = x_t - \hat{x}_t \tag{D.4}$$

is calculated along with the Kalman gain vector $k_t$ and the inverse of the correlation matrix $P_t$, via the recursive equations:

$$k_t = \frac{P_{t-1}\hat{y}^*_{e,t}}{\omega + \hat{y}^T_{e,t}P_{t-1}\hat{y}^*_{e,t}} \tag{D.5}$$

$$P_t = \frac{1}{\omega}\left[P_{t-1} - k_t\hat{y}^T_{e,t}P_{t-1}\right] \tag{D.6}$$

where $0 < \omega < 1$ is the *forgetting factor*, "*" denotes here the complex conjugate and "T" denotes the transpose of a vector.

Finally the filter's coefficients are updated via:

$$c_t = c_{t-1} + k_t e_t \tag{D.7}$$

# Appendix E

# Sample initialisation file

```
[Environment Manager Parameters]
Action(evolution/execution): evolution
PopulationSize: 500
TournamentSize: 7
CreateRandomPopulation(yes/no): yes
ReadPopulationFromFile: ini.pop
CreationType(variable/grow/rampedhalf/rampedvar/rampedgrow/unilength): VARIABLE
MaxDepthForCreation:  4
MaxLengthForCreation:   30
MinLengthForCreation: 3

LimitDepthAtCrossover:  no
MaxDepthForCrossover:  9
LimitLengthAtCrossover: yes
MaxLengthForCrossover:  30

ADFs: 0
NodeMutationProbability: 1
GainMutationProbability: 1

UseOperations:  + - * % +1 -1 *2 /2  X Y 1  ;
GainNodes:  X Y 1 ;
MaximumInputDelays:  3
MaximumOutputDelays:  2
MaximumStackNodes:  5
MaximumConstants:  7
MaximumConstantValue:  2
```

```
MaximumGainValue:  2
HigherBetaValue:  10
LowerBetaValue:  0.1

MaximiseSpeed:  yes
RunSilent: yes
TerminationCriterion(fitness/time/births/nodeEvals/ber/any): births
StopWhenFitness:  0.9
TimeLimit:  30
BirthLimit:  100000
MaxNodeEvaluations: 50000000000
ReportingInterval(births):  500

FitnessType(MSE/CMA/BER/MABS):  MSE
CalculateBitErrorRate:  no
StopWhenBitErrorRate:  0
VarianceAnalysisOfOutput(YES/NO): no
KurtosisAnalysisOfOutput(YES/NO): no
TransientDiesAfterSample: 5

DataSource(generate/read):  generate
GenerateDataForChannel:  f:/anna/signal~1/channe~1/galesia.chn
SignalLengthForGeneration:  15
SNRForGeneration(dB):  30
NoiseType(white/coloured/none): WHITE
NoiseColouringFilter:  f:/anna/expts/more/noisecolour.chn
DelayReferenceBy(samples):  1
ReadInputToGPfrom:  c:/gp97/coloured/dirty10.txt
ReadDesiredOutputFrom:  c:/gp97/coloured/orig10.txt
SaveActualOutputIn:  gpout.txt
SaveConstantTableIn:  constab.txt
ChangeDataDuringRun: no
ChangeDataAfter(births): 200000
ChangeDataAfter(NodeEvals): 100000000
GenerateDataForSecondChannel:  basic2.chn
ReadNewInputToGPfrom: dirty1e3.txt
ReadNewDesiredOutputFrom: orig1e3.txt
```

```
SaveInitialPopulationIn: c:/temp/ini.pop
SaveFinalPopulationIn:   c:/temp/fin.pop
SaveBestIndividualIn:    c:/temp/best.txt
SaveEvolutionDataIn:     c:/temp/evol.txt
SaveBirthsData:     yes
BirthsLogFile:  c:/temp/births.txt

MaxCrossoverTrials: 1

NodeGainsAreRandom:  yes
DefaultGainValue:  1
InheritNodeGains:  no
ConstantTableIsRandom:  no
C0= 0.1 C1= 0.5 C2= 2 C3= 10 C4= -1 C5= 3 C6= 0.7

LearningMethod(anneal/hillclimb/none):  none
UseGrantScheme:  no
LearningProbability:  1
InitialLearningProbability:  1
PerturbationType(GAUSS/CAUCHY):  CAUCHY
PerturbationScheme(ELASTIC/INDEPENDENT):  ELASTIC
StartingTemperature:  3
TemperatureDecay:  0.99
PerturbationScalingFactor:  0.2
LearningIterations:  100
TrialsPerTemperature:  5
MaxInvariantFitnessTrials:  20
AnimateLearning:  no
```

# Bibliography


Ackley, DH and ML Littman (1994). A case for Lamarckian evolution. In: *Artificial Life III, SFI Studies in the Sciences of Complexity, Proceedings Vol. XVII* (CG Langton, Ed.). pp. 3–10. Addison-Wesley.

Al-Mashouq, KA and IS Reed (1994). The use of neural nets to combine equalization with decoding for severe intersymbol interference channels. *IEEE Trans. on Neural Networks* **5**(6), 982–988.

Anderson, RW (1995). Learning and evolution: A quantitative genetics approach. *Journal of Theoretical Biology* **175**, 89–101.

Andre, D (1994). Automatically defined features: the simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them. pp. 477–494. In: Kinnear (1994).

Angeline, PJ (1996). An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover. In: Koza *et al.* (1996). pp. 21–29.

Angeline, PJ, GM Saunders and JB Pollack (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Trans. on Neural Networks.*

Atmar, Wirt (1994). Notes on the simulation of evolution. *IEEE Trans. on Neural Networks* **5**(1), 130–147.

Bang, SH and BJ Sheu (1992). A neural-based digital communication receiver for intersymbol interference and white gaussian noise channels. In: *Proceedings of the IEEE International Symposium on Circuits and Systems.* pp. 2933–2936.

Belew, RK (1990). Evolution, learning and culture: computational metaphors for adaptive algorithms. *Complex Systems* **4**, 11–49.

Benedetto, S and E Biglieri (1983). Nonlinear equalization of digital satellite channels. *IEEE Journal on Selected Areas in Communications* **SAC–1**(1), 57–62.

Biglieri, E, A Gersho, R D Gitlin and TL Lim (1984). Adaptive cancellation of nonlinear intersymbol interference for voiceband data transmission. *IEEE Journal on Selected Areas in Communications* **SAC–2**(5), 765–777.

Boers, EJW, MV Borst and IG Sprinkhuizen–Kuyper (1995). Evolving artificial neural networks using the baldwin effect. In: *Artificial Neural Nets and Genetic Algorithms. Proceedings of the International Conference* (DW Pearson, NC Steele and RF Albrecht, Eds.). Springer Verlag.

Chelapilla, K (1997). Evolutionary programming with tree mutations: evolving computer programs without crossover. In: Koza *et al.* (1997*a*). pp. 431–438.

Chelapilla, K, DB Fogel and SS Rao (1997). Gaining insight into evolutionary programming through landscape visualisation: an investigation into IIR filtering. In: *Evolutionary Programming VI* (PJ Angeline, RG Reynolds, JR McDonnnell and R Eberhart, Eds.). Springer–Verlag, Berlin,Germany.

Chen, S, GJ Gibson, CFN Cowan and PM Grant (1990). Adaptive equalization of finite non-linear channels using multilayer perceptrons. *Signal Processing* **20**, 107–119.

Clark, AP, LH Lee and RS Marshall (1982). Developments of the conventional nonlinear equaliser. *IEE Proceedings part F* **129**(2), 85–94.

Conover, W (1980). *Practical Non-Parametric Statistics.* 2nd ed.. John Wiley & sons.

Dekker, AJ (1958). *Solid State Physics.* Macmillan & co. ltd.

Devore, JL (1995). *Probability and Statistics for Engineering and the Sciences.* 4th ed.. Duxbury Press.

Esparcia-Alcázar, AI (1997). An investigation into a genetic programming technique for adaptive signal processing. Technical report. Department of Electronics and Electrical Engineering, Glasgow University. Second Year Report.

Esparcia-Alcázar, AI and KC Sharman (1996). Some applications of genetic programming in digital signal processing. In: *Late Breaking Papers at the GP'96 conference.* Stanford University, USA. pp. 24–31.

Esparcia-Alcázar, AI and KC Sharman (1997*a*). Evolving recurrent neural network architectures by genetic programming. In: Koza *et al.* (1997*a*). pp. 89–94.

Esparcia-Alcázar, AI and KC Sharman (1997*b*). Learning schemes for genetic programming. In: *Late Breaking Papers at the Genetic Programming 97 Conference.* Stanford University, USA. pp. 57–65.

Etter, DM, MJ Hicks and KH Cho (1982). Recursive adaptive filter design using an adaptive genetic algorithm. In: *Proceedings of the 1982 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP82.* pp. 635–638.

Falconer, DD (1978). Adaptive equalization of channel nonlinearities in QAM data transmission systems. *The Bell System Technical Journal* **7**, 2589–2611.

Fogel, DB (1991). *System identification through simulated evolution: A machine learning approach to modeling.* Ginn Press. Needham, MA.

Fogel, DB (1994). *Evolutionary Computation: Toward a new philosophy of machine intelligence.* IEEE Press. New York.

Fontanari, JF and R Meir (1990). The effect of learning on the evolution of asexual populations. *Complex Systems* **4**, 401–414.

French, RM and A Messinger (1994). Genes, phenes and the Baldwin effect: Learning and evolution in a simulated population. In: *Artificial Life IV* (R Brooks and P Maes, Eds.). pp. 277–282. MIT Press. Cambridge, MA.

Gibson, GJ, S Siu and CFN Cowan (1989). Multilayer perceptron structures applied to adaptive equalisers for data communications. In: *Proceedings of the 1989 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP89.* pp. 1183–1186.

Gibson, GJ, S Siu and CFN Cowan (1991). The application of nonlinear structures to the reconstruction of binary signals. *IEEE Trans. on Signal Processing* **39**(8), 1877–1884.

Goldberg, David E (1989). *Genetic Algorithms in search, optimization and machine learning.* Addison–Wesley.

Gould, SJ (1980). *The panda's thumb.* W.W. Norton & Co.

Gray, GJ, T Weinbrenner, DJ Murray-Smith, Y Li and KC Sharman (1997). Issues in nonlinear model structure identification using genetic programming. In: *Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA.* Glasgow, Scotland. pp. 308–313.

Gruau, F (1994). Genetic micro programming of neural networks. pp. 495–518. In: Kinnear (1994).

Gruau, F and D Whitley (1993). Adding learning to the cellular development of neural networks: evolution and the baldwin effect. *Evolutionary Computation* **3**, 213–233.

Haykin, S (1996). *Adaptive Filter Theory.* 3rd ed.. Prentice Hall.

Hiden, H, M Willis, B McKay and G Montague (1997). Non-linear and direction dependent dynamic modeling using genetic programming. In: Koza *et al.* (1997*a*).

Hinton, GE and SJ Nowlan (1987). How learning can guide evolution. *Complex Systems* **1**, 495–502.

Holland, JH (1975). *Adaptation in natural and artificial systems.* University of Michigan Press. Ann Arbor.

Howard, LM and DJ D'Angelo (1995). The GA-P: A genetic algorithm & genetic programming hybrid. *IEEE Expert* pp. 11–15.

Iba, H, H de Garis and T Sato (1994). A numerical approach to genetic programming for system identification. *Evolutionary Computation* **3**(4), 417–452.

Kechriotis, G, E Zervas and ES Manolakos (1994). Using recurrent neural networks for adaptive communication channel equalization. *IEEE Trans. on Neural Networks* **5**(3), 267–278.

Kinnear, K, Ed. (1994). *Advances in Genetic Programming.* MIT Press.

Kirkpatrick, S, CD Gelatt and MP Vecchi (1983). Optimization by simulated annealing. *Science* **220**(4598), 671–679.

Kitano, H (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems.*

Koza, JR (1992). *Genetic Programming: On the programming of computers by means of natural selection.* The MIT Press. Cambridge, Massachusetts.

Koza, JR (1994). *Genetic Programming II: Automatic discovery of reusable programs.* The MIT Press. Cambridge, Massachusetts.

Koza, JR, Deb, K, Dorigo, M, Fogel, D, Garzon, M, Iba, H and Riolo, RL, Eds. (1997*a*). *Genetic Programming 1997: Proceedings of the Second Annual Conference.* Stanford University, USA.

Koza, JR, FH Bennett III, D Andre, MA Keane and F Dunlap (1997*b*). Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Trans. on Evolutionary Computation* **1**(2), 109–128.

Koza, JR, Goldberg, D, Fogel, D and Riolo, RL, Eds. (1996). *Genetic Programming 1996: Proceedings of the First Annual Conference.* Stanford University, USA.

Lucas, Simon (1996). Evolving neural network learning behaviours with set-based chromosomes. In: *Proceedings of European Symposium on Artificial Neural Networks (ESANN '96).* pp. 291 - 296.

Ma, Q and CFN Cowan (1996). Genetic algorithms applied to the adaptation of IIR filters. *Signal Processing* **48**, 155–163.

Marenbach, P, KD Bettehnausen and S Freyer (1996). Signal path oriented approach for generation of dynamic process models. In: Koza *et al.* (1996). pp. 327–332.

McKay, B, M Willis, G Montague and G Barton (1996). Using genetic programming to develop inferential estimation algorithms. In: Koza *et al.* (1996). pp. 157–165.

Metropolis, N, AW Rosenbluth, MN Rosenbluth, AH Teller and E Teller (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics* **21**(6), 1087–1092.

Montana, DJ and S Czerwinski (1996). Evolving control laws for a network of traffic signals. In: Koza *et al.* (1996). pp. 333–338.

Mulgrew, B (1996). Applying radial basis functions. *IEEE Signal Processing Magazine* **13**(2), 50–65.

Nair, SK and J Moon (1995). Simplified nonlinear equalizers. *IEEE Trans. on Magnetics* **31**(6), 3051–3053.

Nair, SK and J Moon (1997*a*). Data storage channel equalization using neural networks. *IEEE Trans. on Neural Networks* **8**(5), 1037–1048.

Nair, SK and J Moon (1997*b*). A theoretical study of linear and nonlinear equalization in nonlinear magnetic storage channels. *IEEE Trans. on Neural Networks* **8**(5), 1106–1117.

Nambiar, R and P Mars (1992). Genetic and annealing approaches to adaptive digital filtering. In: *Proceedings of the 26th Asilomar Conference on Signals, Systems and Computers.* IEEE Computer Society Press. pp. 871–875.

Nguyen, T and T Huang (1994). Evolvable 3d modeling for model-based object recognition systems. pp. 459–475. In: Kinnear (1994).

Ong, S, C You, S Choi and D Hong (1997). A decision feedback recurrent neural equalizer as an infinite impulse response filter. *IEEE Trans. on Signal Processing* **45**(11), 2851–2858.

O'Reilly, Una-May (1995). An Analysis of Genetic Programming. PhD thesis. Carleton University, Ottawa, Ontario.

Parisi, D and S Nolfi (1996). The influence of learning on evolution. In: *Adaptive individuals in evolving populations* (RK Belew and M Mitchell, Eds.). pp. 419–428. Addison-Wesley.

Parisi, R, ED Di Claudio, G Orlandi and BD Rao (1997). Fast adaptive digital equalization by recurrent neural networks. *IEEE Trans. on Signal Processing* **45**(11), 2731–2739.

Press, WH, BP Flannery, SA Teukolsky and WT Vetterling (1988). *Numerical Recipes in C. The Art of Scientific Computing.* Cambridge University Press.

Priestley, MB (1988). *Non-linear and non-stationary time series analysis.* Academic Press.

Proakis, JG (1995). *Digital Communications.* 3rd ed.. McGraw-Hill.

Proakis, JG and DG Manolakis (1992). *Digital Signal Processing : Principles, Algorithms and Applications.* Macmillan.

Proakis, JG, CM Rader, F Ling and CL Nikias (1992). *Advanced Digital Signal Processing.* Macmillan.

Qureshi, SUH (1985). Adaptive equalization. *Proceedings of the IEEE* **73**(9), 1349–1387.

Sharman, KC, AI Esparcia-Alcázar and Y Li (1995). Evolving signal processing algorithms by genetic programming. In: *IEE/IEEE Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA.* Sheffield, England. pp. 473–480.

Sharman, KC and AI Esparcia-Alcázar (1993). Genetic evolution of symbolic signal models. In: *Proceedings of the 2nd IEE Workshop on Natural Algorithms in Signal Processing vol 2.* Chelmsford, Essex, England. pp. 29/1–29/11.

Shynk, JJ (1989). Adaptive IIR filtering. *IEEE Acoustics, Speech and Signal Processing Magazine* pp. 4–21.

Spector, L and S Luke (1996). Cultural transmission of information in genetic programming. In: Koza *et al.* (1996). pp. 209–214.

Szu, H and R. Hartley (1987). Fast simulated annealing. *Physics Letters A* **122**(3-4), 157–162.

Theodoridis, S, CFN Cowan, CP Callender and CMS See (1995). Schemes for equalisation of communication channels with nonlinear impairments. *IEE Proceedings on Communications* **142**(3), 165–171.

van Laarhoven, PJM (1988). *Theoretical and Computational aspects of Simulated Annealing -CWI Tract 51.* Centrum voor Wiskunde en Informatica. Amsterdam.

Wolpert, DH and WG Macready (1997). No free lunch theorems for optimization. *IEEE Trans. on Evolutionary Computation* **1**(1), 67–82.