Genetic Evolution of Symbolic Signal Models

Ken C. Sharman & Anna E. Esparcia-Alcazar[1]
Department of Electronics & Electrical Engineering
The University of Glasgow
Glasgow, G12 8QQ
41-339-8855 (x4902)
ken@music.gla.ac.uk

Abstract

This paper reports on a novel method of signal modelling that employs a variable model structure as opposed to the fixed model structure used in conventional modelling methods. The functional form of the model along with any required numerical parameters are simultaneously estimated from the signal sequence to be modelled. This is accomplished by defining the model functional and its parameters in terms of structured lists of symbols, and using an estimation algorithm that can infer symbol lists from the given data. Motivated by recent work in cellular coding and evolutionary computation, we use genetic programming (GP) to evolve high quality model structures. This is based on a coding the model in terms of an expression tree in polish form which can then be manipulated and optimised using standard genetic algorithm (GA) techniques. In conjunction with the model structure evolution, we use simulated annealing to optimise the numerical parameters of the model and a set of production rules to minimise the model order. The paper discusses how these three processes can be combined to yield a powerful general purpose modelling system.

1. Introduction and Overview

The aim of this paper is to report on a new technique of adaptive signal modelling. Our approach involves a novel combination of genetic evolution and simulated annealing to yield an algorithm that simultaneously estimates a model structure as well as the numerical parameters of the chosen model. This leads to a general purpose modelling algorithm with a wide range of applications including system identification, signal coding, signal detection, classification, spectral analysis, and signal synthesis. The method presented in this paper attacks a problem domain that is somewhat more general than domain covered by the current generation of modelling techniques. These are normally only concerned with finding of a set of numerical parameters for a fixed model structure that has been chosen *a-priori.*

In this work, the model structure will be estimated from the given data. The aims of this are several. Firstly, we hope to be able to obtain lower error measures than classical procedures for a given model size. A second aim is to widen the class of signals that can be used with an estimator, and finally, we seek model structures that have small numbers of parameters.

The approach we take is to use a combination of genetic algorithms (GA) and simulated annealing to evolve and optimise both the model structure and the model's numerical parameters. Motivated by recent work in genetic programming [Koza 90], we represent the signal model as a genotype, which is a list of symbols that maps onto an expression tree defining the functional structure of the model. A population of these genotypes is allowed to evolve under the control of a GA using the mean square error between the signal estimates and the observations as the basis of a fitness measure. Running in parallel with the GA is a fast simulated annealing algorithm that

---

[1] On leave from Dept. de Ingenieria de Sistemas, Computadores y Automatica, Universidad Politecnica de Valencia, Spain. (annabel@aii.upv.es)

implements learning of the numerical parameter values of each model in the population. In addition, we also employ other processes in the modelling procedure, namely a set of reduction rules which remove redundancy in the genotypes so as to minimise the model order.

2. Signal Modelling

2.1. The Classical Approach

Finding values for the parameters of a model for an observed signal is one of the most common and fundamental operations carried out in signal processing. It is a procedure that is at the heart of a wide variety of signal processing systems and algorithms. The main point of signal modelling is, loosely speaking, to simplify the signal in some way so that in can be represented using a smaller number of parameters than the original sequence of samples. These model parameters can then be used as an alias for the signal, with a view to reducing the complexity of many important operations such as detection, classification, storage, etc.

One way of looking at signal modelling is to consider it as a process whereby an *estimate* of the signal is synthesised from the model structure and its parameters. Let $\boldsymbol{X} = \{x_n,\ n = 1 \ldots N\}$ be an observed sequence that is to be modelled, $\boldsymbol{M} = \{m_i,\ i = 1 \ldots P\}$ represent a set of model parameters, and $\hat{\boldsymbol{X}} = \{\hat{x}_n\}$ be the estimated signal samples. Using F to denote the function that computes the estimated signal from its model, we can express the estimate as,

$$\hat{x}_n = F(\boldsymbol{M}, n)$$

The problem addressed by classical signal modelling is to find the parameters $\boldsymbol{M}$ from a set of data $\boldsymbol{X}$ using a fixed model structure defined by F.

A fundamental point is the *quality* with which a signal can be described by a model. This can be quantified in various ways, one of the more important being the weighted mean squared error between the observed signal and its estimate,

$$e(\hat{\boldsymbol{X}}, \boldsymbol{X}) = \frac{1}{N} \sum_{n=1}^{N} w_n \, (x_n - \hat{x}_n)^2$$

where $\{w_n\}$ are a set of weighting coefficients. The main tasks required of a modelling procedure are therefore twofold: (1) to effect good data reduction, $P < N$, and (2) to yield as low an error measure e as possible for any signal in the class of signals to be applied to the modelling algorithm. Additional properties to be considered include the computational cost of finding the parameters $\boldsymbol{M}$ from $\boldsymbol{X}$, and perhaps the number of different types of signal that can be successfully modelled (the *generality* of the procedure).

The majority of signal modelling algorithms in use today are really just estimators for the numerical values for a structure that has been chosen a-priori. For instance, many types of stationary signals can be represented as the output of a linear filter driven by white noise, leading to the well known autoregressive (AR), moving average (MA), and combined (ARMA) signal models, along with their derivatives and variants. In these methods, the function of the modelling algorithm is to find numerical values for the filter coefficients. Although these methods can work very well, their performance is always dependent on the matching of the model structure to the given signal. In a sense, you must know what you are looking for, or at least have a good idea of what it looks like. If the chosen model is not matched to the observed signal, parameter estimates will be meaningless, and other problems, like unstable estimation algorithms, can arise.

The procedures developed below are not restricted to simple linear models but are capable of generating highly non-linear structures.

2.2. The Evolutionary Approach to Signal Modelling

2.2.1 Parameterising the Model Structure

The focus of the method presented herein is to treat the task of finding a signal model structure as part of the estimation problem. This is illustrated by way of a simple example. Consider a deterministic signal - a damped exponentially decaying sinusoid,

$$\hat{x}_n = 10\, e^{-0.1\, n} \sin(0.4\, n)$$

Here the model parameters and structure could be defined as,

$$\boldsymbol{M} = \{A, \alpha, \beta\} \text{ and } F(\boldsymbol{M}, n) = A\, e^{\alpha n} \sin(\beta\, n)$$

The key to using a variable model structure as opposed to a fixed one is to represent it in a parameterised form with a coding that is able to define a wide range of different structures and at the same time have parameters whose values can be reliably and efficiently estimated. Introduce a second set of parameters, $\boldsymbol{G} = \{g_i, i = 1 \ldots Q\}$, and a functional mapping, $F_{\boldsymbol{G}}$, defining the model structure function F from $\boldsymbol{G}$. We seek a coding for $\boldsymbol{G}$ which can represent a variety of different model functions $F_{\boldsymbol{G}}$, using as few parameters Q as possible, along with efficient algorithms that can estimate $\boldsymbol{G}$ from $\boldsymbol{X}$ and evaluate $F_{\boldsymbol{G}}$ from $\boldsymbol{G}$.

2.2.2 Expression Trees

The model structure $F_{\boldsymbol{G}}$ is parameterised in terms of $\boldsymbol{G}$ by describing it in terms of a *symbolic expression tree*, which in turn is represented as an ordered list of symbols and numerical values in $\boldsymbol{G}$. An expression tree is a point rooted graph whose nodes are the primitive component functions of the model. The ordering of the graph dictates the sequence in which these primitives are executed and how their output values are combined to provide the estimated signal sequence. The root of the tree yields the estimated sequence $\{\hat{x}_n\}$, while the leaves are the external arguments to the modelling function F (see figure 1).

Expression trees of this form can be written using a syntax not unlike that used in LISP programmes. To be precise, the expression tree represents F using a *polish* notation. The damped sinusoid model given above could be described by the expression tree shown in figure 1, which is written in polish notational form as,

$$\hat{x}_n = (* \; A \; (* \; (\sin \; (* \; \beta \; n)) \; (\exp \; (* \; n \; \alpha))))$$

An expression tree is completely defined by the triplet $\{\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{G}\}$, where $\boldsymbol{T} = \{t_i, i = 1 \ldots N_T\}$ is a set of *terminal nodes*, $\boldsymbol{N} = \{N_i, i = 1 \ldots N_N\}$ is a set of *non-terminal nodes*, and $\boldsymbol{G}$ is a coded description of the connections between the nodes within the tree. The terminals in $\boldsymbol{T}$ are symbols representing the leaves of the tree. These are the arguments (inputs) that are supplied to the model structure function F. The non-terminals are symbols which label the internal nodes where computations are performed. The example from above uses a terminal set of $\boldsymbol{T} = \{A, \alpha, \beta, n\}$ and non-terminal set consisting of the multiplication operation along with the *sin* and *exponentia*l functions, i.e., $\boldsymbol{N} = \{*, sin. exp\}$. The third component in the tree definition is the parameter list,

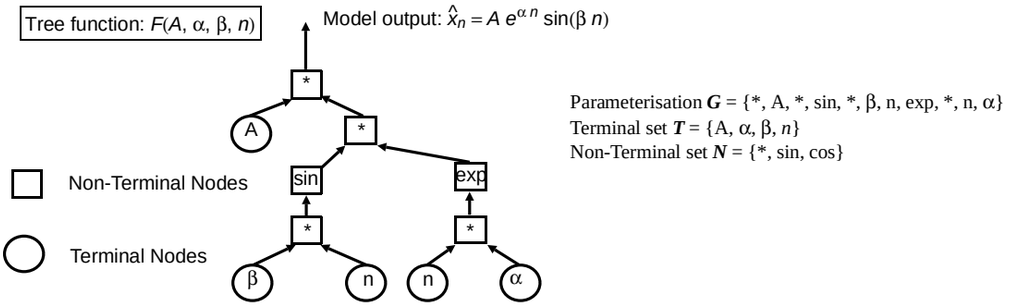

Figure 1 - A Symbolic Expression Tree for a Damped Sinusoid

Label	Action
X	Data sample x_n
n	Time sample index n
1	Constant number (e.g. 1) etc.
R	Random number - Gaussian (0,1)
V_i	Get value from relative node i
A_i	i'*th* argument of a function definition

Note 1: Some nodes (e.g. V_i, A_i, G_i) require a numerical alrgument. This is considered to be part of the symbol in the ***G*** list.

Note 2: A saturation limit is applied to the output of each non-terminal cell to prevent numerical overflow problems. In addition, cells with restricted range arguments (e.g. *log)*, will have their inputs restricted as is appropriate.

Note 3: In the table of non-terminals, the *cell function* column lists the function that calculates the cell output from the cell inputs which are labelled i_1, i_2 etc., starting from the left hand node.

Table 1 - Terminal Node Types

Label	Description	Cell Function	Branches
+	Addition	$i_1 + i_2$	2
-	Subtraction	$i_1 = i_2$	2
/	Division	i_1 / i_2 or 0 if $\lvert i_2 \rvert < \varepsilon$	2
*	Multiplication	$i_1 \times i_2$	2
+1	Increment by 1	$i_1 + 1$	1
-1	Decrement by 1	$i_1 - 1$	1
*2	Multiply by 2	$i_1 \times 2$	1
/2	Divide by 2	$i_1 / 2$	1
exp	*Exponential*	e^{i_1}	1
sin	*sine* function	sin(i_1) - i_1 in radians	1
cos	*cosine* function	cos(i_1) - i_1 in radians	1
log	logarithm	ln(i_1)	1
sqr	square	$i_1 \times i_1$	1
root	square root	$\sqrt{i_1}$ or 0 if $i_1 < 0$	1
if	Selection	i_2 if $i_1 \le 0$ otherwise i_3	3
U	Until loop	exexute i_2 until $i_1 \le 0$	2
D	Unit time delay	i_1 from previous call to this cell	1
G_i	definition tree i	$F_i(i_1 \ldots i_{Q_i})$	variable

Table 2 - Non-terminal Cell Types

G = {g_i, $i = 1 \ldots Q$}. This is an ordered list of Q symbols with $g_i \in \mathrm{N} \cup \boldsymbol{T}$. The ordering of ***G*** is such that its first element labels the root node. Following this are listed the symbols for the nodes supplying the inputs to this node starting from the leftmost branch and applying this rule recursively down each branch. Each node in the tree can now be identified with a unique integer index that is its position within ***G***.

An expression tree of this form can realise a large number of model structures using a relatively small set of non-terminals. In fact, the basic arithmetic functions and a single (non-zero) constant numerical value, ***N*** = {+, -, *, /, 1, n}, are sufficient to define any function F if the size of ***G*** is unlimited. One of our aims though is to minimise the number of model parameters. This is carried out by judicious choices of the terminal and non-terminal node types. For instance, although a **sine** function could be in theory be approximated to any desired degree of precision by a power series involving only addition and multiplication operations, it would be far more efficient to include the **sine** function in the set of non-terminal node types. Other primitives that we have found to be useful are those listed in tables 1 and 2. The functions of some of these cells will be described in more detail later in the paper.

Note that the terminal and non-terminal node sets must be chosen *a-priori* ; they are not estimated from the data to be modelled. The modelling problem is to find an ordered list of symbols (***G***) from the alphabet defined by ***N*** and ***T***.

2.2.3 Expression Trees with Branch Value Scaling Factors

An extension to the basic tree structure is to let the branches forming the connections between nodes in a tree have numerical values that scale the output of a processing node before it is supplied to the input of the node at

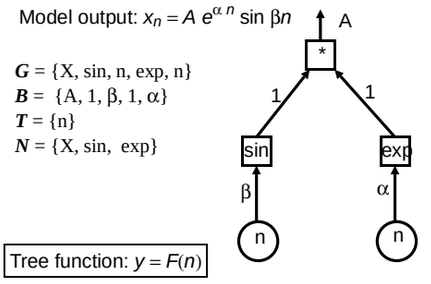

Figure 2 - Branch Scaled Tree for Damped Exponential

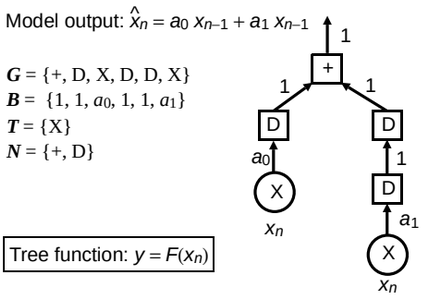

Figure 3 - Expression Tree for AR(2) Process

the end of the branch. Let $\boldsymbol{B} = \{b_i, i = 1 \ldots Q\}$ be the set of branch values for a tree, where b_i is the scale factor for the output of the cell g_i . The model structure and its parameters are now described by the set $\{\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{G}, \boldsymbol{H}\}$. Including the parameters in $\boldsymbol{B}$ might at first sight appear to increase the number of model parameters. In fact the reverse is true, and the number of parameters will usually be smaller because the branch value scaling factors reduces the number of multiplication nodes required (see figure 2).

2.2.4 Stochastic Signals

Classical stochastic signal models can be described using a unit time delay operation (D) (non-terminal) with the observed data $\{x_n\}$ in $\boldsymbol{T}$. The D cell uses an internal storage register. When the output of a D cell is requested it returns the value of this register and stores its input value in the register ready for the next call.

For example, the linear prediction model of a second order AR process is,

$$\hat{x}_n = a_0\, x_{n-1} + a_1\, x_{n-2}$$

which has an expression tree and branch values described by the lists, (see figure 3).

$$\boldsymbol{G} = \{+, \mathrm{D}, \mathrm{X}, \mathrm{D}, \mathrm{D}, \mathrm{X}\}, \qquad \boldsymbol{B} = \{1, 1, a_0, 1, 1, a_1\}$$

where X is the symbol for a terminal node supplying the samples of the AR process, $\{x_n\}$. As an aside, it is pointed out that the expression tree parameterisation is far from unique; this process could equally well be modelled using,

$$\boldsymbol{G} = \{+, \mathrm{D}, \mathrm{D}, \mathrm{X}, \mathrm{D}, \mathrm{X}\}, \qquad \boldsymbol{B} = \{1, 2, 0.5, a_1, 1, a_0\}$$

2.2.5 Recursion

Recursion is a powerful tool that can simplify the description certain types of computational tasks. It can be implemented in an expression tree by using the special terminal node labelled V_i. The action of this is cell is to return the current output value of the node located in the $\boldsymbol{G}$ list at a distance of i units from the current node. The *jump* cell V_i is labelled with an integer number specifying the jump distance. To prevent infinite recursions, it is implicitly assumed that a unit delay cell is placed in the path between the output of the node pointed at by the i parameter of the V cell and the value returned by the cell. If i points to a cell outside the limits of the $\boldsymbol{G}$ list, it will be clipped at the start (or end) of $\boldsymbol{G}$.

The non-terminal labelled U implements a useful loop control function. A U node has two input branches. Its action is to execute its right branch a number of times specified by the left branch. The V and U cells can be combined to yield powerful constructions. For instance, a function returning the sum of squares of an input sequence evaluated over a window length of N samples is, (see figure 4),

$$\boldsymbol{G} = \{\mathrm{U}, \mathrm{N}, +, \ \mathrm{sqr}, \mathrm{X}, \ V_{-2}\} \ \text{ and } \boldsymbol{B} = \{1, 1, 1, 1, 1, 1\}$$

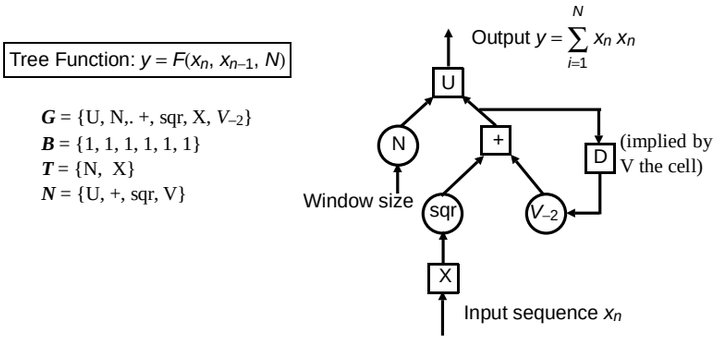

Figure 4 - Expression Tree for Sum of Squares Calculation

2.2.6 Hierarchical Model Structures

Hierarchical tree structures involve more than one expression tree. The additional trees are used to define non-terminal functions for use in the main model evaluation tree, effectively allowing it to be composed of cells whose functions are not defined in advance, but are themselves defined by the additional expression trees.. The model is now coded by a list of N_G tree descriptions, $\{\boldsymbol{G_1}, \ldots \boldsymbol{G_{N_G}}\}$ along with corresponding lists of branch values, terminal, and non-terminal sets, which may be different for each tree. The non-terminal cell labelled G_i indicates a cell whose function is defined by the tree $\boldsymbol{G_i}$, and the inputs to a G_i cell are special terminal cells in $\boldsymbol{T_i}$ labelled as A_k. These terminals implement the dummy arguments to the function $F_{\boldsymbol{G_i}}$ defined by $\boldsymbol{G_i}$, where there the qualifier k in the A_k cell is a small integer number indicating an argument number in a list of arguments to $F_{\boldsymbol{G_k}}$.

To avoid infinite computational loops, a strict hierarchy is imposed,

$$G_k \notin \boldsymbol{N_{k+1}}, \ldots \boldsymbol{N_{N_G}} \quad k = 1 \ldots N_G - 1$$

2.2.7 Reduction of the Model Order

One of our aims is to minimise the number of model parameters. This is accomplished using a set of reduction rules that transforms a tree into a simpler form with a minimum number of nodes. The reduction algorithm recursively descends a tree and employs the fundamental laws of algebra and arithmetic to eliminate redundant nodes and branches. Space does not allow us to more fully describe this procedure, but to illustrate its flavour, the reduction algorithm would replace the tree, $\boldsymbol{G}$ = {+, X, X} $\boldsymbol{B}$ = {2, 3, 4} with the simpler tree, $\boldsymbol{G}$ = {X} $\boldsymbol{B}$ = {14}. (i.e. $2\,(3\,x + 4\,x) = 14\,x$).

3. Genetic Evolution of Expression Trees

3.1. Genetic Algorithms

To return to the main modelling problem, our objective is: given sets of terminal and non-terminal primitives $\boldsymbol{T}$ and $\boldsymbol{N}$, and a sequence to be modelled $\boldsymbol{X}$, find an expression tree $\boldsymbol{G}$ and branch values $\boldsymbol{B}$ that best model the data. This is an non-trivial optimisation problem, essentially requiring a search over the space of all functions that can be defined by expression trees and over all possible sets of branch values for each tree. This is a ridiculously huge search space, but is one whose structure is ideal for a GA implementation. The GA operates by manipulating the $\boldsymbol{G}$ lists only; it is not used to estimate the branch values $\boldsymbol{B}$, which are optimised by a simulated annealing algorithm running in parallel with the GA. Thus we have separated estimation of the model structure from its numerical parameters. This is because we believe the GA to be better at optimising symbol lists and not so good at numerical lists, whereas simulated annealing appears to be more suited to numerical problems than symbol ones. Further comments on this are presented in section 4.1 where we describe the simulated annealing algorithm.

In this section we outline a GA that evolves a population of models with a view to finding a best fit to the given data. The GA is based on ideas originally developed by Koza [Koza 92] who pioneered the use of genetic programming as a means of evolving function definitions and computer programmes.

The key components of a GA are, [Goldberg 89]:

Phenotype

> The phenotype is the *object* that "lives" and interacts with the *environment* defined by the GA. Here, the phenotype is a particular signal model structure and numerical parameters, and the environment is the data sequence to be modelled. i.e. the phenotype is the modelling function $F_{\boldsymbol{G}}(\boldsymbol{X})$, in an environment $\boldsymbol{X}$.

Genotype

> This is an encoding which describes the phenotype. In our case the genotype is the expression tree and its branch values, ($\boldsymbol{G}$, $\boldsymbol{B}$), or in the case of a hierarchical modelling algorithm, lists of tree definitions, $\{\boldsymbol{G_i}\}$, $\{\boldsymbol{B_i}\}$, $i = 1 \quad N_G$.

Fitness Function

A performance measure of the phenotype induced by a genotype. This is a function of the genotype as well as environmental factors (i.e. the data to be modelled, $\boldsymbol{X}$). We will denote the fitness function as $f_{\boldsymbol{G}}(\boldsymbol{X})$., (the dependence of f on $\boldsymbol{B}$ is not shown, but is implied). The aim of the GA is to evolve a phenotype that maximises the fitness function. Using an appropriate definition of fitness, this translates into finding the best model structure and parameters for the given data.

Selection Criteria

These are the criteria used to select individuals[1] for reproduction and creation of new individuals. It is common to select pairs of parents on the basis of fitness so that the probability of an individual being chosen for reproduction is proportional to its fitness value.

Breeding Operators

The breeding operators take individuals chosen by the selection criteria and use these as "parents" to create new offspring individuals. Breeding operators are discussed in more detail later.

Population stabilisation

The offspring created by the breeding operators are added to the population, increasing its size. To keep the population size stable, some members must be deleted. This is carried out on the basis of fitness by choosing members at random from a distribution that favours those with the lowest fitness values.

3.2. Fitness Function

The purpose of the fitness function is to measure the quality of the signal model defined by a given genotype. A suitable definition for our modelling problem is,

$$f_{\boldsymbol{G}}(\boldsymbol{X}) = \frac{1}{1 + e_{\boldsymbol{G}}(\boldsymbol{X})}$$

where $e_{\boldsymbol{G}}(\boldsymbol{X})$ is the mean squared error between the signal and its estimate obtained from the model defined by genotype $\boldsymbol{G}$,

$$e_{\boldsymbol{G}}(\boldsymbol{X}) = \frac{1}{N}\sum_{n=1}^{N} \left| x_n - F_{\boldsymbol{G}}(n) \right|^2$$

This definition yields a fitness value between 0 and 1, with higher values corresponding to better modelling performance. The fitness function can also be used to implement other constraints. Minimisation of the model order may be included as an objective by attaching a monotonically increasing function of the model size, $m(N_G)$,

$$f = \frac{1}{1 + e + m(N_G)}$$

3.3. Genetic Operators

The two main evolutionary operators used in GA's are crossover and mutation. These operate on individuals chosen from the population using the selection criteria (i.e. biased towards models with higher than average fitness value). The objectives of the genetic operators are to create new individuals that may possibly have higher fitness values, and also to increase the diversity of genetic material in the population, which is essential to maintain evolution towards good solutions.

1. An individual is a single phenotype in a population of phenotypes.

3.3.1 Crossover

The crossover operator employs a pair of selected individuals as parents to create child expression trees that inherent characteristics from each parent. The key to successful crossover is to produce syntactically and semantically correct child expression trees using components from both parents. The method adopted here is to select a random node in each parent tree and then to swap the sub-trees between the parents at these chosen nodes. This creates a pair of child expression trees guaranteed be fully terminated and computable (see figure 5). Either one or both of the child trees can be added to the population to continue evolution. From the point of view of a genotype, crossover can occur at any locus, but the length of the sub-tree at the locus must be determined to ensure that complete sub-trees are exchanged during crossover, producing meaningful children. In the case of a hierarchical system using more than one tree, the crossover operator is constrained to operate only between parents at the same level in the hierarchy.

3.3.2 Mutation

Mutation is used to create new genetic material, and thus main genetic diversity. Mutation is sometimes applied after a new individual is created by crossover. It operates by selecting a node in the child tree at random, deleting the sub-tree at the selected node, and then replacing it with a newly generated random sub-tree, which may or may not have the same depth (i.e. the maximum number of nodes from the sub-tree root to a terminal). The maximum depth of a mutated sub-tree is limited by a constant chosen by the user of the GA, as is the probability with which mutation is applied to newly created trees.

A summary of the complete GA for evolution of model structure is shown in table 3.

4. Simulated Annealing of the Branch Values

4.1. Learn from your Parents

The GA presented above is concerned only with evolving tree structures; the branch value parameters are not affected. These are optimised during the lifetime of an individual by a simulated annealing method [Kirkpatrick 83]. In a sense, we are likening the branch values to knowledge which is accumulated during the lifetime of an individual by its interaction with the environment. The annealed branch values are inherited by the child trees created in the GA, thus providing a passage of knowledge from parents to children. Note how we have separated the evolution of the genetic material used by crossover and mutation from the branch values learned through simulated annealing. This is a deliberate move to try and ape natural evolution where current thinking concludes that experience gained during the lifetime of an individual is not passed on to children through the parent's genes.

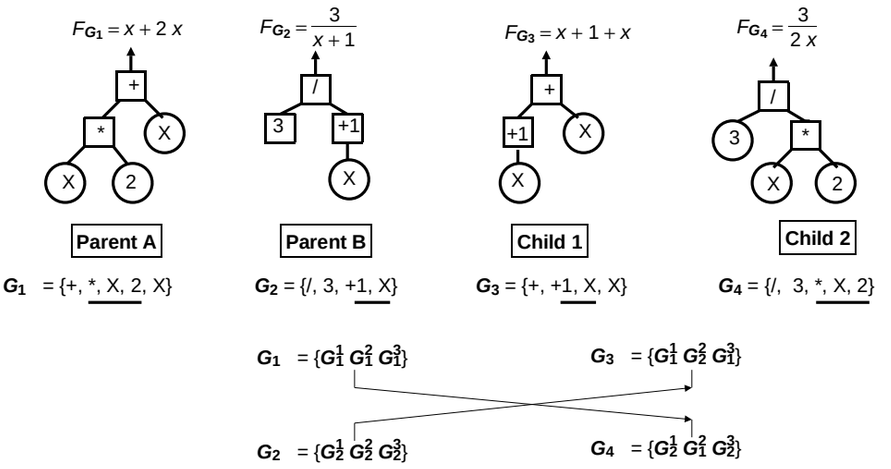

Figure 5 - Crossover of Expression Trees

However, learning and experience are, in higher animals, inherited through the *cultural* processes of observation and education. Recent work in the area of neural network systems [Muhlenbein 89, Grau 93] has employed similar ideas to good effect.

4.2. The Simulated Annealing Parameter Optimiser

During evolution of the population, a "background" annealing process operates in tandem with the evolutionary operators of selection, crossover and mutation. The annealing algorithm provides optimisation of the branch values through an adaptive trial and error procedure. This learning ability is decreased over time as the individual ages, and it is controlled by a *temperature* variable, unique to each individual and dependent on its age [1]. The temperature is initialised to a predefined maximum value when an individual is created, and is slowly reduced using an *annealing schedule*. Denoting the temperature for the i'th individual at age k as $t_i(k)$, the annealing schedule we use is,

$$t_i(k+1) = \gamma\, t_i(k),\ \gamma < 1.$$

When the annealing process is activated, its action is to select an individual from the GA population at random, and then to select a particular branch in the expression tree of the individual, also at random. The branch value at the chosen locus is randomly perturbed, by an amount dependent on the temperature. Denoting $b_{i,j}(k)$ as the branch value at locus j in the i'th individual at age k, the perturbed branch value is,

$$\bar{b}_{i,j}(k) = b_{i,j}(k) + \delta(t_i(k))$$

where $\delta(t)$ is an infinite variance Cauchy distributed random variable, obtained from a zero mean, unit variance Gaussian distributed variable, θ, by the transformation,

$$\delta(t) = K \tan^{-1} \theta$$

(K is a scale factor).The use of the Cauchy distribution is based on the work of Szu [Szu 86]. The long tails of this probability distribution will occasionally yield large perturbations, while the main lobe ensures that most perturbations are in the neighbourhood of the current branch value. This results in mainly local foraging around the neighbourhood of the current value, with only occasional excursions to points distant from home. The fitness of the phenotype using this perturbed branch value is computed. If an increase in fitness is obtained, then the perturbation is accepted and the branch will continue to use the perturbed version. Otherwise, the perturbed parameter will only persist with a probability given by the Boltzman distribution at the current temperature,

$$\text{Prob}\,[b_{i,j}(k+1) = \bar{b}_{i,j}(k)] = \exp\left(\frac{\bar{f} - f}{T(k)}\right)$$

where f and $\bar{f}$ are the fitnesses of the original phenotype and its perturbed version respectively.

5. The Evolutionary Signal Modelling Algorithm

The complete evolutionary signal modelling algorithm is a combination of the three main processes: (1) evolution of a population of tree structures using the GA, (2) annealing of the branch values, and (3) minimisation of the model (tree) size using a reduction rule process briefly discussed in section 2. An outline of the particular scheduling and sequencing of these three algorithmic components is described in table 4.

6. Experimental Results

A selection of interesting experimental reults will be presented at the workshop.

1 Age is defined as the time since the individual was created and added to the population. It is an integer timer that increments every generation.

```
Initialisation
      Generate a population of random expres-
      sion trees and branch value sets.
      Evaluate the normalised fitness of each
      member in the population.
Repeat  {
      Increment the "age" counter.
   Selection
      Select a subset of size N_GA, of the
      population, using fitness proportional
      selection. Arrange the selection into
      parent pairs.
   Reproduction
      With probability P_r, replicate each
      selected individual.
   Crossover
      With probability P_C, crossover the
      parent pairs and create 1 or 2 children
      from each pair.
   Mutation
      With probability P_M, apply mutation to
      each node in every child.
   Life
      Evaluate the fitnesses of the children.
   Death
      Stabilise population size by selecting
      low fitness members to delete.
}
```

Table 3 - The Genetic Algorithm

```
Initialisation
   Define Resources
      Terminal and non-terminal sets.
      Population size.
      Random tree generator data.
      GA control  parameters.
      SA control parameters.
      Scheduling probabilities.

Repeat  {
      Keep track of the tree with the best
      fitness.
   Evolution
      Evolve a new generation using the GA.
   Simulated Annealer
      With probability P_SA, select an in-
      dividual at random and execute N_SA
      simulated annealing trials.
   Tree Optimiser
      With  probability P_OPT, select a node
      at random and apply reduction rules
      to simplify the sub-tree at this
      node.
}
```

Table 4 -The Complete Modelling Algorithm

7. References

[Goldberg 89] Goldberg, D. *Genetic Algorithms in Search, Optimisation and Machine Learning*. Reading, MA: Addisson-Wesley 1989.

[Koza 92] Koza, J. *Genetic Programming: On the Programming of Computers by Natural Selection*. Cambridge, MA: The MIT Press 1992.

[Koza 91] Koza, J. Genetic evolution and co-evolution of computer programmes. In Langton, C., Taylor, C., Farmer, J., Rasmussen, S. (eds), *Artificial Life II*. Redwood City, CA: Addison-Wesley. pp. 603-629. 1991.

[Grau 93] Grau, F. and Whitley, D., The cellular development of neural networks: the interaction of learning and evolution. Laboratoire de l'Informatique du Parallelisme, Ecole Normal Superieure de Lyon, France, Research report No. 93-04, 1993.

[Kitano 90] Kitano, H. Designing Neural Networks Using Genetic Algorithms with Graph Generation Systems. *Complex Systems* **4**, pp461-476, 1990.

[Montana 93] Montana, D. Strongly Typed Genetic Programming. BBN Tech. Rep. #7866, BBN Inc., Cambridge, MA, 1993.

[Whitley 90] Whitley, D, Startweather, T and Bogart, C. Genetic Algorithms and neural networks: optimising connections and connectivity. *Parallel Computing*, **14**, pp347-361, 1990.

[Szu 86] Szu, H. Fast Simulated Annealing. in J. S. Dinjer (ed), AIP Conf. Proc. 151, *Neural Networks for Computing*, Snowbird, Utah, 1986.

[Kirkpatrick 83] Kirkpatrick, S., Gellat, C., Vecci, M., Optimisation by Simulated Annealing, *Science*, **20**, pp621-680, 1983.

[Muhlenbein 90] Muhlenbein, H., Limitations of multi-layer perceptron networks - steps towards genetic neural networks. Parallel Computing, 14, pp249-260, 1990.

[Muhlenbein 89] Muhlenbein, H . and Kindermann, J. The Dynamics of Evolution and Learning - Towards Genetic Neural Networks. in Pfeifer, P et. al. (eds), *Connectionism in Perspective*, Elsevier (North-Holland), 1989.