# Evolutionary neural networks: models and applications

Bryn Vaughan Williams

Doctor of Philosophy

The University of Aston in Birmingham

May 1995

The University of Aston in Birmingham

# Evolutionary neural networks: models and applications

Bryn Vaughan Williams

Doctor of Philosophy

May 1995

## Abstract

The scaling problems which afflict attempts to optimise neural networks (NNs) with genetic algorithms (GAs) are discussed. A novel GA-NN hybrid is introduced, based on the bumptree, a little-used connectionist model. As well as being computationally efficient, the bumptree is shown to be more amenable to genetic coding than other NN models. A hierarchical genetic coding scheme is developed for the bumptree and shown to have low redundancy, as well as being *complete* and *closed* with respect to the search space. When applied to optimising bumptree architectures for classification problems the GA discovers bumptrees which significantly out-perform those constructed using a standard algorithm.

The fields of artificial life, control and robotics are identified as likely application areas for the evolutionary optimisation of NNs. An artificial life case-study is presented and discussed. Experiments are reported which show that the GA-bumptree is able to learn simulated pole balancing and car parking tasks using only limited environmental feedback. A simple modification of the fitness function allows the GA-bumptree to learn mappings which are multi-modal, such as robot arm inverse kinematics.

The dynamics of the 'geographic speciation' selection model used by the GA-bumptree are investigated empirically and the *convergence profile* is introduced as an analytical tool. The relationships between the rate of genetic convergence and the phenomena of speciation, genetic drift and punctuated equilibrium are discussed.

The importance of genetic linkage to GA design is discussed and two new recombination operators are introduced. The first, *linkage mapped crossover* (LMX) is shown to be a generalisation of existing crossover operators. LMX provides a new framework for incorporating prior knowledge into GAs. Its adaptive form, ALMX, is shown to be able to infer linkage relationships automatically during genetic search.

Key words: genetic algorithm, bumptree, control, linkage

# Acknowledgements

This work was made possible by funding provided by the EPSRC and by British Telecom. I am particularly grateful to Michael Gell for his hospitality during my visits to BTRL, and for arranging the funds which made it possible for me to attend WCCI '94 to present some of this work. My thanks also to David Bounds, my supervisor, for welcome constructive criticism and for allowing me the freedom to pursue my own interests during the course of this project.

I would like to thank my colleagues and friends in the Neural Computing Research Group at Aston, both for making my time as a postgraduate extremely enjoyable, and for nobly putting up with my tendency to use far more than my fair share of the available computing resources. I am particularly indebted to Phil Barrett for technical support above and beyond the call of duty.

Part of the work described in chapter 3 of this thesis was undertaken in collaboration with Richard Bostock, and Chris Bishop kindly provided both the inspiration and the data for some of the experiments described in chapter 6. Thanks also to Edmund Rolls for making me aware of the work of Parisi *et al.* described in chapter 5. Jarmo Alander was kind enough to send me the graph shown in figure 2.1, and Alexander Linden, Philip Pratt and Brian Schack all generously offered the use of their pole-balancing code for the simulations in chapter 6.

Finally, I'd like to thank Sonja, who had to live with me while I was writing up and, for some reason, still seems to like me.

*To my mother.*
*Too many thanks, not enough time.*

# Contents

# List of tables and figures

# Chapter 1

# Introduction

## 1.1 Introduction

This thesis is concerned with an area of research which has spawned a plethora of publications in recent years: the application of genetic algorithms, optimisation methods based on simulated evolution, to the field of neural network design.

As research in this area has progressed it has become apparent that the most popular neural network models happen to be particularly unsuited to optimisation with genetic algorithms (the reasons for which are discussed in the following chapter). This unfortunate fact is reflected in the literature – there are now literally hundreds of papers describing the evolution of neural networks for toy problems, but very few which report success on larger 'real world' applications. This observation provided the motivation for the central theme of this thesis, the identification of a little-used neural network model which is shown to be considerably more amenable to evolutionary optimisation than more established networks, while offering comparable performance. The model in question is the recently-introduced *bumptree* network, and a large part of the thesis is devoted to the development and evaluation of GA-bumptree hybrids.

Genetic algorithms are weak optimisation methods which make very few assumptions about the function being optimised (although some of the assumptions they do make turn out to be rather subtle, as will be seen in chapter 7). Unlike stronger gradient-based methods, genetic algorithms have the advantage that they do not require the objective function to be smooth, continuous or differentiable. This generality has a price, however – genetic search is extremely explorative, and hence computationally intensive, which makes the method often impractical except in domains where stronger (more exploitative) methods cannot readily be applied. If the function to be optimised is smooth and differentiable, for example, a genetic algorithm is a clumsy and inefficient approach in comparison with a gradient-based method such as hill climbing. This observation was one of the main conclusions of a recent study of evolutionary neural networks (Hancock 1992), and provides the motivation for the second main theme of the present work: the need to concentrate on application areas in the field of neural networks where the computational overhead of genetic search is likely to be worthwhile, as opposed to those (such as training neural networks for well-specified classification tasks) where established, stronger, methods (such as backpropagation) are adequate.

Finally, a principle which underlies much of the work described in these pages, and to which chapter 7 of the thesis is entirely devoted, is the notion that the performance of any GA depends critically on how well the genetic coding scheme and recombination operators reflect any dependencies which exist between the parameters

to be optimised. The bumptree coding scheme and crossover operator, described in chapter 3, were designed with this in mind, and chapter 4 introduces a method for quantifying the dependencies between parameters in a GA.

## 1.2 Structure of the thesis

The following chapter reviews the field of evolutionary computation in detail, focusing in particular on the theoretical foundations of genetic algorithms (GAs) and identifying a number of principles for GA design which are applied throughout the remainder of the thesis. After a necessarily brief introduction to neural networks (NNs), previous efforts to apply genetic algorithms in the neural network field are reviewed, and certain scaling problems inherent to the approach are identified and discussed. As an illustration of the principles described, some experiments are presented in which a genetic algorithm is applied to optimising topology for a simple neural network. These experiments investigate a criticism made by Belew, McInerney, *et al.* (1991) of earlier work by Miller, Todd, *et al.* (1989), and the results support the claims of the original authors.

Chapter 3 introduces a novel GA-NN hybrid based on the bumptree, a little-used connectionist model. The GA-bumptree is shown to have considerably better scaling properties than previous GA-NN hybrids. The hierarchical genetic coding scheme developed for the bumptree is shown to have the desirable properties of *completeness*, *closure*, *continuity*, *isomorphism* and *low redundancy*.

Two sets of experiments are described in which the GA-bumptree is tested on classification problems. In the first set of experiments the GA optimises the topology of the bumptree only, and is able to discover bumptrees which significantly outperform those found by an alternative constructive algorithm. In the second set of experiments the GA is extended to optimise both the architecture and weights of the bumptree.

The GA used in the bumptree experiments uses a 'geographic speciation' model for selection and replacement. Chapter 4 presents a detailed examination of the dynamics of this GA, based on an extensive set of experiments. Methods for measuring the degree of genetic diversity in the population are discussed, and the *convergence profile* is introduced as an analytical tool. The results of this study illustrate the relationships which exist between the rate of genetic convergence in a GA and the phenomena of *speciation*, *genetic drift* and *punctuated equilibrium*.

In chapter 5 attention turns to the issue of identifying problem domains in which an evolutionary approach to neural network design is likely to be worthwhile because stronger methods cannot readily be applied. The emerging field of artificial

life (A-life) is introduced, and a case study is described in which a GA-NN hybrid is applied to an A-life problem.

Chapter 6 continues to examine applications for the GA-bumptree, focusing on tasks, drawn from the domains of control and robotics, which are difficult or impossible to learn using standard NN training algorithms. Experiments are described in which the GA-bumptree is able to learn simulated pole balancing and car parking tasks which involve the problem of *temporal credit assignment*. A further set of experiments shows how the GA-bumptree can be applied to learning mappings which are multi-valued, such as the inverse kinematics mapping for a simple robot arm.

Some of the experimental results reported in chapter 6 reveal the limits of the present GA-bumptree approach. These results motivated the final study undertaken during this project, a re-examination of the role of recombination in GAs. In chapter 7 it is shown that different crossover operators make different implicit assumptions about genetic linkage which can profoundly affect the performance of the GA. Two new generalised recombination operators, *linkage mapped crossover* and *adaptive linkage mapped crossover*, are introduced. The new operators are tested on two variants of a 'Royal Road' function (Mitchell, Forrest, *et al.* 1991) and their performance is compared against standard 1-point and uniform crossover operators.

Finally, chapter 8 summarises the main conclusions drawn from the project as a whole.

# Chapter 2

# Evolutionary connectionism

## 2.1 Introduction

This chapter reviews the literature in the field of evolutionary connectionism, the intersection between the growing fields of evolutionary computation (EC) and neural networks.

The genetic algorithm is undoubtedly the most popular model of evolutionary computation, and all the experimental work reported in this thesis has involved GAs of one form or another. Section 2.2 reviews the development of genetic algorithm theory and practice up to the time of writing. The conclusions drawn from this review have shaped the design of the GAs described in the remainder of the thesis. Section 2.2.5 briefly describes the most popular of the alternative evolutionary computation algorithms, and shows how each relates to the GA.

A thorough treatment of the connectionist field would require several theses the size of this one, so no attempt is made to exhaustively review the neural network literature here. Instead, section 2.3 outlines the field in broad strokes, with attention focused on the multi-layer perceptron, since this is the model which has been the subject of the overwhelming majority of evolutionary connectionist experiments.

With the necessary foundations in place, the remainder of the chapter is devoted to reviewing the literature of evolutionary neural networks. In addition to the review, some experiments are described in section 2.4.2 in which a genetic algorithm is used to optimise topology for a simple neural network.

## 2.2 Genetic Algorithms

The field of evolutionary computation has its roots in the mid 1950s and early 1960s when several researchers, working independently, began to experiment with modelling evolutionary processes as an approach to computational problem solving. While no two authors are ever in agreement as to who was EC's founding father, researchers such as Fraser, Friedberg, Box and Bremmermann seem the most popular candidates (Goldberg 1989; Fogel 1993; De Jong 1994; Schwefel 1994).

By the 1970s, three main schools of evolutionary computation had emerged and begun to develop in isolation from one another. These were the fields of genetic algorithms, evolutionary programming (EP) and the German *evolutionstrategie* (ES), and they remained independent until 1990, when the first "Parallel Problem Solving from Nature" workshop (Schwefel and Männer 1991) brought them together.

This thesis is concerned with just one of these fields, the field of genetic algorithms. While ES and EP have matured steadily since the '70s in the hands of a relatively small core of practitioners, the GA has recently enjoyed an enviable

explosion of research interest. Figure 2.1, taken from Alander's (1994) recent bibliography of evolutionary computation, illustrates this growth. Out of more than 2500 references in Alander's bibliography, fewer than 200 are in the fields of ES and EP.

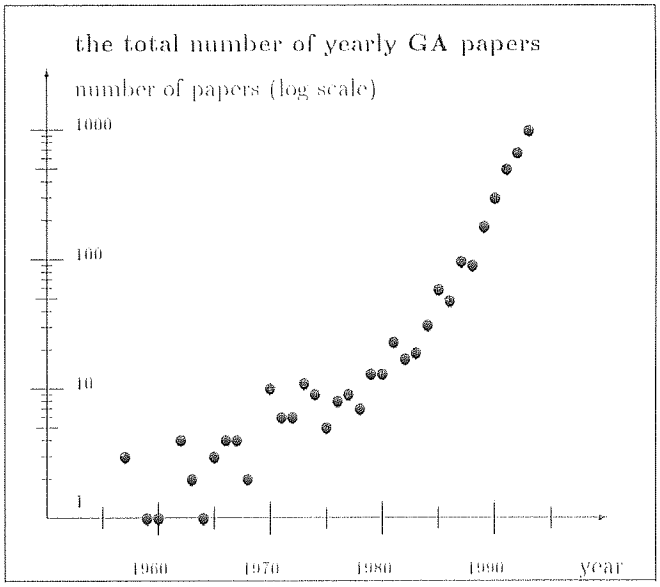



Figure 2.1: The number of yearly publications in the field of evolutionary computation (after Alander 1994).

All the experiments reported in this thesis have been based on GAs of one form or another. Although EP and ES are described only briefly, however, it is noted that the distinctions between the various EC algorithms are somewhat blurred, and seem to be becoming more so with each new publication. The GA described in the following chapter, for example, has many features generally associated with EP or ES.

### 2.2.1 Genetic algorithm fundamentals

While evolutionary computation itself may be of uncertain parentage, there is no doubt in the GA community as to who was the father of genetic algorithms. In 1975 John Holland published *Adaptation in Natural and Artificial Systems*, an attempt to identify the essential properties of adaptive systems and cast them in a formal framework (Holland 1975). Holland's seminal example of adaptation in the natural world was evolution, and the computational model he introduced to illustrate the validity of the formalism was the genetic algorithm. Since Holland's formal beginnings, authors such as Goldberg (1989) and Davis (1991) have introduced the GA to a wider audience, and attention has shifted to the GA as a computational tool for optimisation, more than as a general model of adaptive processes.

When seen as algorithms for parameter optimisation, there are three features of GAs which distinguish them from other methods such as hill climbing. Firstly, GAs operate on a coding of the parameter space rather than directly on the space itself. Secondly, GAs sample new points according to probabilistic, rather than deterministic, transition rules. Finally, GAs maintain a population of points, effectively searching from many places at once. This is not simply a case of applying a local heuristic many times in parallel – the essential principle of the GA is that it is the population as a whole which searches, guided by global information distributed among all the points within it.

As an illustration of the basic genetic algorithm, consider an optimisation problem in which the task is to tune a number of parameters in order to maximise some reward. In the traditional GA, the first step is to code each parameter as a binary word of as many bits as are necessary to represent the range of values over which the parameter is allowed to vary. The coded parameters are then concatenated to form a binary string. A population of many such strings is generated, in such a way that each string has a random pattern of 1s and 0s. Thereafter, 'evolution' proceeds as follows:

- A measure of *fitness* is calculated for each member of the population, by decoding its string and evaluating the resulting set of parameters according to the reward function.

- A number of parents are selected from the population with replacement, with a string's probability of selection being proportional to its fitness relative to the rest of the population.

- A new population of strings is generated from these fit parents by the application of the *genetic operators*, which are described below.

- Some or all of the original population is replaced by the new generation of offspring, and the process is repeated.

In the basic GA there are three genetic operators: *reproduction*, *crossover* and *mutation*. Reproduction is self explanatory: it refers simply to the copying of a parent string into the next generation, verbatim. With each successive generation, the processes of selection and reproduction together are enough to ensure that the average fitness of the population will increase, as the fit are reproduced at the expense of the poor. In the absence of other operators the population will converge on the fittest of the original strings, i.e. the best of the original random solutions.

New strings are generated by crossover, which combines information from two parents to produce a novel string. In simple *one-point crossover*, a single locus is selected at random to be the *crossover point*, and a new string is created by beginning at the first locus and copying the bit pattern from one parent as far as the crossover point, and from the other thereafter. This is illustrated in figure 2.2.

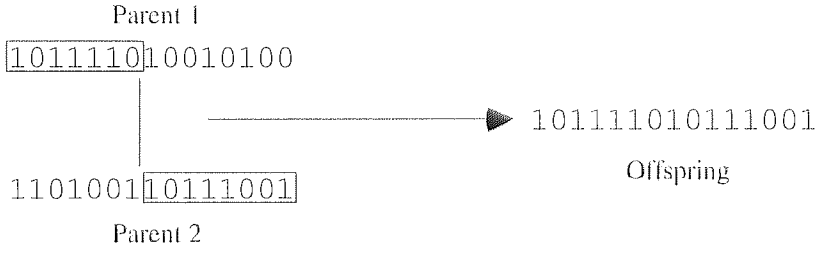



Figure 2.2: Simple one-point crossover.

Crossover is the GA's search engine, continuously recombining the information stored in the population in new ways to produce new strings for evaluation. The third genetic operator, mutation, introduces occasional 'errors' into the reproductive process by randomly flipping bits in the offspring. Mutation traditionally operates at low probability, and its purpose is not to generate novel strings for evaluation, but to safeguard against potentially useful information being lost from the population. As selection crowds out poor strings the value of a particular locus may converge, becoming uniform throughout the population. Mutation serves to re-introduce such lost information, to be tried in new contexts by crossover. The relationship between recombination and mutation in the GA is inspired by an analogous relationship in nature. The following quote, taken from a textbook on evolution (Savage 1977), illustrates the point:

> "The significance of recombination to evolution cannot be overestimated. A single mutational change may be lost or passed on without great impact on a population, but if its effect is modified and enhanced by recombination, an unending contribution to variation is begun. Variation is the raw material for evolutionary change; recombination is its principal source. Mutation alone has relatively little effect on variation without the pervasive impact of recombination."

The genetic operators are applied probabilistically. For each parent string there is a fixed probability, $p_c$, that crossover will be applied, in which case a second string with which to cross is drawn at random from the pool of parents. Otherwise, the string is simply reproduced 'asexually', so the probability of reproduction is $(1-p_c)$. Every new child string is subject to mutation, which occurs at the *mutation rate* $p_m$, where $p_m$ is the probability that any locus will mutate. An early empirical study by

De Jong (1975) suggested typical values for the operator probabilities as $p_c$=0.6 and $p_m$=0.001, and these values are still widely used.

### 2.2.2 Biological analogies

At this point it is useful to introduce some of the biological terminology which has been borrowed (but not always well looked after) by the GA community.

In biology (Whitehouse 1973; Savage 1977; Lawrence 1989), a *chromosome* is a long chain molecule of DNA consisting of many *genes* in sequence. DNA can usefully be visualised as a ladder, twisted into the familiar double-helix, where each rung is a *nucleotide base pair*, of which there are four types. According to the modern definition, a gene is a long sequence of base pairs which specifies the manufacture of a single protein[1]. If there are several viable variants of a particular gene, these are referred to as the *alleles* of that gene. All the genes which determine the genetic makeup of an organism are collectively referred to as that organism's *genotype*, while the organism itself, i.e. the expression of those genes in a particular environment, is the *phenotype*.

In GA terms, it is reasonable to refer to the strings which are manipulated by the GA as chromosomes, and this is common practice. Since a single string codes the complete set of parameters to be optimised, a particular string is also often referred to as a genotype, with the actual parameters it codes being the phenotype.

The digits which make up a string are, unfortunately, referred to as genes by GA practitioners, with the alphabet of possible values (the set {1,0} for binary strings) referred to as alleles. This is not a particularly good analogy. In a binary string, the individual bits are more akin to the base pairs in DNA; a whole sequence of bits which codes a particular parameter might more reasonably be referred to as a gene, and the possible range of parameter values as its alleles. GAs which use higher-order representations, in which each locus specifies a whole parameter, fit in better with the accepted use of the terms 'gene' and 'allele'.

Crossover is generally considered analogous to sexual recombination, since the child inherits a mixture of 'genetic material' from its two parents. Again, the terminology is unfortunate because the term *crossing over* in biology refers to a specific process which is rather unlike crossover in GAs. In biology, crossing over is the exchange of genetic material between homologous chromosomes which occurs during meiosis in the sex cells of a diploid organism[2]. The problem with the term

[1]This is a slight simplification, since some genes actually specify certain RNAs.

[2]Most sexually reproducing organisms are *diploid*, meaning that their chromosomes occur in homologous pairs. When gametes are produced, the pairs separate so that each gamete is *haploid*,

'crossover' in GAs is that biological crossing over is very much a physical consequence of diploidy, whereas the individuals in the GA are haploid.

The fact that crossing over in biology occurs within each individual parent's sex cells *prior* to sexual fusion, whereas crossover in GAs mixes alleles *between* parents during fusion, has been levelled as a criticism of the GA as an evolutionary model (Fogel 1993). The simple observation, however, that crossing over during gametogenesis serves to mix *grandparental* alleles, reveals this particular criticism as a straw man. It is the *fact* of recombination which is important, and which is captured in the GA's crossover operator: whether parents' genes are mixed in their children, or in their children's children, is something of a moot point.

While the GA's crossover operator may be something of a misnomer, there is no doubting the power of the principle it embodies. A common example, based on an evolving population, illustrates the power of recombination: suppose that there are two alleles, either of which would confer a selective advantage, but neither of which is present in any member of the population. Each of these alleles can only be introduced by a mutation of the appropriate gene, a very rare event. As the populations evolves, mutant individuals possessing one or other of the alleles will eventually occur and begin to pass them on to their offspring. If the individuals in the population reproduce asexually, producing an individual with *both* alleles requires another rare mutation in an individual which already has one of the alleles. In a sexually reproducing population, producing an individual with both alleles requires only that two individuals, each with one of the alleles, exist in the same population and interbreed.

### 2.2.3 The theory of GAs

#### Schemata

Holland (1975) laid the theoretical foundations for GAs on the principle that it is not the processing of individuals which is important, but the processing of *schemata*. A *schema* is defined as a similarity template, a particular pattern of bits shared by some individuals in the population. For example, the schema #11#01, where # is a 'don't care' symbol matching either 0 or 1, matches the strings {011001, 011101, 111001, 111101}. Formally, for a population of binary strings of length $k$, the schema $h \in \{0,1,\#\}^k = (h_1, h_2, \cdots, h_k)$ matches the subset of strings given by

$$H = \left\{ x \in \{0,1\}^k = (x_1, x_2, \cdots, x_k) \mid \forall j: h_j \neq \#,\ x_j = h_j \right\}, \quad j=1,2,\ldots,k.$$

---

containing one chromosome from each pair. This ensures that when two parental gametes fuse the diploid condition is restored.

For convenience, the *subset* defined by the schema $(h_1,h_2,\cdots,h_k)$ is labelled $h_1h_2\cdots h_k$, whereas a single *element* in the population, $(x_1,x_2,\cdots,x_k)$, is labelled $x_1x_2\cdots x_k$; hence it is natural to write that $111001\in\#11\#01$ but $101001\notin\#11\#01$, and so on. Each $j : h_j \neq \#$ is called a *defining position* in $H$.

Since any $k$-bit binary string is a member (or an *instance*) of $2^k$ distinct schemata, a given population of $n$ strings will contain instances of somewhere between $2^k$ and $n2^k$ schemata[3]. The fitness of a schema $H$ with respect to a particular population at time $t$ is defined as the average fitness of all its member strings:

$$f_t(H) = \frac{1}{m(H,t)} \sum_{x \in H} f(x), \tag{2.1}$$

where $f(x)$ is the fitness of string $x$, and $m(H,t)$ denotes the number of individuals in the population at time $t$ which are instances of $H$. Since strings, and therefore also the schemata they represent, are selected for reproduction in proportion to their fitness, it is possible to estimate the change in the number of instances of any schema after one generation of selection and reproduction. Holland's (1975) *schema theorem*, also known as the *fundamental theorem of genetic algorithms*, states that the expected number of instances of $H$ in generation $t+1$, $m(H,t+1)$, is bounded by

$$\langle m(H,t+1) \rangle \geq m(H,t) \frac{f_t(H)}{\bar{f}_t} [1 - p_d(H)], \tag{2.2}$$

where $\bar{f}_t$ is the average fitness of the population and $p_d(H)$ represents the probability that $H$ will be disrupted by the genetic operators, that is, the probability that a parent string which is an instance of $H$ will produce a child which is not an instance of $H$. In the simple GA, the probability of $H$ being disrupted by either crossover or mutation is bounded by

$$p_d(H) \leq p_c \frac{\delta(H)}{k-1} + p_m o(H), \tag{2.3}$$

where $p_c$ is the probability of applying the crossover operator to a selected parent string (as opposed to simply reproducing the string), $\delta(H)$ is the *defining length* of

[3]For large $n$, such that $n2^k > 3^k$, the number of schemata represented in the population is bounded by $3^k$, since this is the number of unique schemata of length $k$.

$H$, the number of loci between the first and last defining positions in $H$, $p_m$ is the mutation rate and $o(H)$ is the *order* of $H$, the number of defining positions[4].

It is clear that schemata with high defining lengths stand a greater chance of being disrupted by crossover. For example, a crossover applied to the string 111001 is much more likely to disrupt the schema 1####1 than the schema ##10##, since there is only one possible crossover point which could result in the second schema being disrupted (i.e. between the third and fourth positions), whereas any crossover point could disrupt the first. It is similarly clear why high order schemata are more likely to be disrupted by mutation, since every defined bit has a given probability of being mutated. These observations lead naturally to the expectation that short, low order schemata whose average fitness is higher than the population average will become increasingly represented in the population in each successive generation, increasing in frequency at a super-linear rate.

Holland's schema theorem (equation 2.2) is a lower bound rather than an exact equality because it does not account for two factors which affect the expected transmission of schemata. The first is that schemata are not only destroyed by crossover and mutation, they are also created – a child may contain schemata which neither of its parents contained. The second, and the reason for the inequality in equation 2.3, is that a crossover which falls between the two extreme defining positions of a schema does not always disrupt that schema – specifically, crossover will have no disruptive effect on a schema of which both parents are instances. Bridges and Goldberg (1987) have incorporated these factors into the schema theorem and extended equation 2.2 to an exact expression (i.e. an equality) for the expected change in proportion of a given schema from one generation to the next in a basic binary GA.

### Intrinsic parallelism

The power of the genetic algorithm, according to Holland, lies in the fact that the evaluation of a single $k$-bit string yields information about the $2^k$ schemata of which it is an instance. The GA's task can be viewed as choosing how best to allocate trials between competing schemata, a problem which Holland argues is analogous to that of allocating trials to the arms of a k-armed bandit. In the k-armed bandit problem, payoff associated with each arm is determined by an independent, unknown, probability distribution, and the goal is to allocate a finite number of trials between the arms in order to maximise total payoff. There is a trade-off between the need to

[4]The term $p_m o(H)$ is an approximation. Since each position in a schema has equal independent probability of being mutated, the probability of a schema surviving mutation is $(1-p_m)^{o(H)}$, which can be approximated by $(1-o(H)p_m)$ for $p_m \ll 1$ (Holland 1975).

sample each arm as often as possible in order to obtain a good estimate of their relative payoffs, and the desire to allocate the majority of trials to the arm for which the current estimated payoff is the greatest. Holland argues that each evaluation in a GA effectively samples many competing schemata (analogous to sampling many arms of the k-armed bandit) in parallel, and that the GA's exponential allocation of trials to the observed best schemata approaches the optimal strategy for this class of problem.

This principle of *intrinsic parallelism*[5] underpins the original choice of a binary representation for strings in the GA. The argument is simple: if each evaluation yields information on $2^k$ schemata, then maximising the degree of intrinsic parallelism is the same as maximising $k$, the number of loci in the string. To code a given amount of information, maximising $k$ is, in turn, the same as minimising the information stored at each locus. Goldberg (1989) refers to this as the *principle of minimal alphabets*. Clearly, a binary representation stores the minimum information at each locus, maximising $k$ and therefore maximising the degree of intrinsic parallelism.

### Generalising schemata

The above argument in favour of the use of binary representations remains widely accepted in the GA community. However, the theory's failure to account for the empirical successes reported by a number of workers using alphabets of much higher cardinality, such as strings of integers or real values, has prompted some researchers to question the assumptions behind it.

Grefenstette and Baker (1989) examine the k-armed bandit analogy, and conclude that the analogy is not valid for schema sampling in a GA because the GA does not sample competing schemata independently. To illustrate this, they consider three schemata in a population, whose actual fitness averages (i.e. averaged over every possible string) are: $f(0\#\#\#\cdots\#)=1$, $f(1\#\#\#\cdots\#)=1/2$ and $f(111\#\cdots\#)=2$. In the first generation, a GA correctly allocates twice as many trials to the schema 0###···# as 1###···#. However, these trials are not allocated independently; fitness-proportionate selection ensures that the trials allocated to 1###···# are biased towards the subspace represented by the high-fitness schema 111#···#. This leads to an overly high estimate of the average fitness of 1###···# which, in turn, leads to an increasing allocation of trials to 1###···# in favour of 0###···#, even though the latter schema has the higher actual average fitness.

Goldberg (1991) seeks to explain the empirical success of high-cardinality codings in terms of his *theory of virtual alphabets*. This is the theory that a real-coded GA rapidly selects a virtual alphabet for each locus, a subset of relatively low

[5]later called *implicit parallelism* by Goldberg, for reasons unknown.

cardinality drawn from the space of real values, and searches strings defined over these alphabets thereafter. While there is no doubt that a real-valued GA with a finite population will tend to converge to a reduced set of alleles at each locus, the argument that this convergence somehow increases the intrinsic parallelism does not hold water. By definition, the number of schemata processed when a string is evaluated depends on the number of loci in the string – for $k$ loci it is simply $2^k$, since each locus can take its actual value or the # 'don't care' symbol – not the cardinality of the alphabet at each locus. The reason that low cardinality alphabets are associated with the greatest intrinsic parallelism is that lower cardinality representations require longer strings *to code a given search space*, and longer strings mean more schemata. The convergence of loci to reduced virtual alphabets during a GA run has no effect on the string length and, therefore, no effect on the number of schemata processed by each string; this convergence simply focuses the GA on a particular subspace of the original search space. In fact, since the total number of possible schemata is the product of the number of alleles (plus the #) at each locus, reducing the cardinality of each locus by selecting a reduced virtual alphabet actually reduces the maximum possible number of schemata in the population.

Rather than trying to reconcile the success of high cardinality codings with the existing schema theory, Antonisse (1989) takes the crucial step of re-examining the original argument for binary codings. Antonisse reasons as follows: what is important is that the GA processes subsets of strings which are similar in some sense, and discovers which patterns of similarity are correlated with high fitness. Counting the number of schemata processed under different representations is misleading, because Holland's schemata are only appropriate as similarity templates when a binary representation is used. For higher cardinality alphabets, the single 'don't care' symbol, #, is not sufficient to express all possible patterns of similarity at each locus[6]. As an example, Antonisse proposes that schemata for strings defined over the alphabet {0,1,2} should be defined over the augmented alphabet $\{0,1,2,\#_{00},\#_{01},\#_{02},\#_{12},\#_{012}\}$, where $\#_{01}$ means 'this locus may be either 0 or 1', and so on. This simple change in viewpoint admits a far greater number of schemata to higher cardinality representations, and effectively reverses the counting argument in favour of binary codings.

Since Antonisse, Vose (1991) and Radcliffe (Radcliffe 1991a; Radcliffe 1991b; Radcliffe 1991c; Radcliffe 1992; Radcliffe and George 1993; Radcliffe 1994) have taken the final conceptual step and adopted a completely general view in which any

[6]Indeed, as Radcliffe (1992) points out, even for binary representations, schemata do not necessarily capture all similarities which might be expected to influence fitness. Coding an integer as 4 bits, for example, the representatives of 7 and 8, 0111 and 1000 respectively, share membership of no schema but ####.

arbitrary subset of strings constitutes a 'generalised schema'. Although these generalised schemata, termed *predicates* by Vose, and *formae* by Radcliffe, still satisfy the principle of Holland's schema theorem, they repudiate the counting arguments which favour one representation scheme over another. In the general view, so long as the coding space is isomorphic to the search space, the coding scheme is irrelevant to the degree of intrinsic parallelism; it is possible to define the same number of subsets over a given search space however it is coded. Mason (1993) sums up the development of schema theory since Holland thus:

> "Both Holland and more recently Goldberg...have argued that binary coded chromosomes allow the greatest number of schemata to be represented and thus processed. Most GA research has consequently focused on binary codings. The consideration of generalised schemata results in all codings admitting the same number of schemata. There is now no justification for a continuance of this bias towards binary codings."

### Building blocks

The success of a GA in solving any optimisation problem depends critically on the existence of *building blocks*. In terms of Holland's original schema theorem, building blocks are short, low order schemata of high fitness which can be combined by crossover to form higher order schemata of greater fitness. In terms of generalised schemata, building blocks are arbitrary subsets of strings of above average fitness which satisfy two conditions: first, the average fitness of strings which fall in the intersection of two such subsets should be greater than the average fitness of either one, and second, recombining two strings drawn from separate subsets should yield a string which is a member of their intersection.

The problem with applying GAs to practical optimisation tasks is that it is generally non-trivial to ensure that useful building blocks exists. The traditional approach is to use a binary representation and standard genetic operators and hope that the mapping between the search space and the binary representation space is such that schemata do capture meaningful partial solutions. The more modern approach, championed by Davis (1991) and Radcliffe (1992), is to design both the representation and the genetic operators according to the particular optimisation problem at hand, so that known regularities in the search space are preserved and exploited as much as possible.

Problems arise when fit, low order schemata (or formae) combine to form less fit higher order schemata. This effect has been termed *deception* by Goldberg (1987), and is caused by epistatic interactions between loci; a problem may be *deceptive* if the contribution to fitness attributable to a particular allele at a particular

locus (or a particular set of alleles at different loci) depends on the values of alleles at other loci.

Several GA theorists have attempted to build analytical tools to quantify the degree of deception exhibited by a particular combination of objective function, representation and operators. Goldberg (1987) introduces the 2-bit *minimal deceptive problem* in order to investigate the schema frequency dynamics of a simple binary GA attempting to optimise a deceptive function. Goldberg later outlines an algorithm for the analysis of the degree of deception in a given GA application (Goldberg 1989a; Goldberg 1989b), based on a method of calculating schema average fitness values using Walsh functions pioneered by Bethke (1981). Unfortunately, the algorithm involves first transforming the objective function with a Walsh transform, which has computational complexity of higher order than enumerating over the entire search space. This rather limits its practical usefulness.

Mason (1993) introduces the *crossover non-linearity ratio* as a measure of the degree of deception in a GA, based on the change in average fitness when two strings are crossed over to produce two children – put simply, a large change in average fitness indicates that the fitness contribution attributable to the set of genes exchanged by crossover depends strongly on the rest of the string, and so these genes do not constitute a good building block.

Finally, Mitchell, Forrest and Holland (Mitchell, Forrest, *et al.* 1991; Forrest and Mitchell 1993; Mitchell, Holland, *et al.* 1994), instead of studying deceptive problems, take the complementary approach and investigate GA performance on a class of objective functions which are designed to be maximally 'GA-friendly'. These *Royal Road* functions are constructed from a hierarchy of explicitly defined building blocks with no epistatic interactions between them. Both Mason's crossover non-linearity ratio and Mitchell's Royal Road functions are considered in detail in chapter 7, where they are applied to analyse the performance of a new crossover operator.

### 2.2.4 Selection and replacement strategies

During the time that the relatively small core of GA theorists have been advancing the theoretical understanding of GAs, a large number of practitioners have been developing and refining all aspects of GA implementation and studying the effects of various representations, operators, and selection and replacement strategies empirically. This and the following section briefly review this work.

### Preserving good solutions

An unfortunate feature of the canonical GA is that it is possible that the best solution found will be lost during a run. This is clearly undesirable, and the problem has been addressed in various ways.

One source of such losses lies in the stochastic nature of selection. The selection process is often likened to spinning a gambler's 'wheel of fortune'[7], where strings are laid out around the circumference of the wheel such that each string has a slot whose size is proportional to its fitness. In the limit, repeatedly spinning the wheel will select strings in proportion to each string's fitness relative to the average. Unfortunately, selecting a finite number of parents introduces stochastic sampling errors (as an extreme example, there is a small but non-zero probability that the worst string will be selected every time) and rounding errors (since it is impossible to select a fraction of a string), which become more significant as the population size becomes smaller. Baker (1987) addresses this problem by introducing *Stochastic Universal Sampling* (SUS), which is a simple variant of normal selection which minimises sampling error. SUS is analogous to a gambler's wheel with a number of pointers equally spaced around the wheel's circumference, one pointer for each parent to be selected, which is spun only once.

A more direct approach to ensuring that the best solution is never lost is simply to copy the best string into the next generation verbatim before selecting parents for the remaining offspring in the normal way. This process, introduced by De Jong (1975) and known as *elitism*, has enjoyed a good deal of empirical success and is widely used.

Taking the principle of elitism further, De Jong experimented with GAs in which generations overlapped by varying degrees, so that the majority of the existing strings were preserved and only a few (a proportion which he referred to as the *generation gap*) were replaced each generation (De Jong 1975). This method finds its extreme in GAs such as Whitley's GENITOR, in which only a single string is replaced each 'generation' (Whitley 1988). Whitley referred to this as *one-at-a-time* reproduction, while Syswerda (1989) later described GENITOR as a *steady state* GA. The latter phrase has proved more popular in the GA community, perhaps understandably.

Finally, Whitley has experimented with deliberately introducing bias into the selection process, in favour of those strings which are not only fit themselves, but which have shown a tendency toward producing fit offspring. This scheme, known as *reproductive evaluation*, proved useful in problems where linkage effects and redundancy in the genetic code meant that the performance of schemata was highly

[7]This is often referred to as 'roulette wheel' selection.

context dependent, that is, dependent on the makeup of the population as a whole. If there is more than one evolutionary road to a solution, reproductive evaluation helps the population to traverse a single path, rather than attempting to set off in many different directions at once, generating partial solutions which conflict.

### Maintaining diversity

In ensuring the survival of the best solutions, elitist strategies unfortunately tend to increase the number of duplicate strings in the population, accelerating the loss of genetic diversity. This is ironic because the only possible benefit of having duplicate strings in the population is in a non-elitist GA, where duplication may be regarded as a useful 'insurance policy' against losing a good solution – in elitist GAs, duplicate strings are just dead weight. It is not surprising that various strategies for minimising duplication have evolved in parallel with the emergence of elitism.

Goldberg (1989) cites Cavicchio's *preselection* scheme (Cavicchio 1970), in which offspring may only replace their parents, as one of the earliest efforts to preserve diversity by preventing duplication. Maintaining diversity was a serious issue for Cavicchio, since he was working with strings of over 3000 bits, but was obliged by available computing resources to use a population size of less than 20. More recently, a similar replacement scheme has been referred to by Mason (1993) as PC-elitism (Parent/Child-elitism).

*Crowding* is a replacement scheme introduced by De Jong (1975), whereby a number of strings (a proportion of the population called the *crowding factor*) are chosen at random, and the one most similar, genetically, to the new child is replaced by the child. De Jong was working with binary strings, and used Hamming distance as the measure of similarity. Whitley's GENITOR algorithm prevents duplicates by the simple expedient of discarding any offspring which is identical to an existing member of the population. Stadnyk (1987) describes an improved crowding algorithm which uses inverse-fitness proportionate selection to choose the crowd of candidates for replacement, thus biasing in favour of replacing strings of low fitness. Mahfoud (1992) compares the performance of several variations of crowding on simple multi-modal problems, and favours a simple variant of preselection which he calls *deterministic crowding*, which offers high performance while being computationally inexpensive. Mahfoud's results also favour phenotypic rather than genotypic comparisons, based on the difference between decoded parameters rather than the Hamming distance between strings. In Mahfoud's multi-modal test functions, only the 7 most significant bits out of 30 yield useful information as to which peak a point is on; as far as crowding is concerned, the other 23 bits are just noise (Mahfoud 1992).

Like crowding, *sharing* (Goldberg 1987; Deb and Goldberg 1989) is a mechanism which adjusts a string's chance of survival according to its similarity to the other members of the population. Unlike crowding, sharing operates at the selection phase rather than during replacement. The principle is that each peak on a multi-modal fitness landscape is a resource which must be shared by those strings which ascend it. Thus a string's fitness is reduced by a function of the number and closeness of other strings which are sufficiently similar to it to fall within a pre-determined neighbourhood. As with Mahfoud's algorithms, the neighbourhood function is a distance metric typically defined over the actual parameter space (*phenotypic sharing*) rather than over the coded strings (*genotypic sharing*). On simple one-dimensional multi-modal test problems a GA with sharing maintains search on several peaks throughout the run, while an ordinary GA quickly converges to a single peak. It is worth noting, however, that the effectiveness of sharing depends on a choice of neighbourhood size which separates different peaks into different neighbourhoods. This choice is simple for one-dimensional test functions, but not necessarily so for real problems where the topology of the fitness landscape is unknown. Sharing is also $O(N^2)$ computationally complex, since every string's distance from every other string must be recalculated each generation.

As well as modified selection and replacement strategies, various modifications made to the genetic operators also prevent undue duplication in the population. Booker's *reduced surrogate crossover* (Booker 1987) and Eshelman's *HUX* operator (Eshelman and Schaffer 1991), both described in the following section, are examples of operators which prevent crossover from exchanging identical portions of similar parent strings and thus creating duplicates. Eshelman also introduces *incest prevention*, a simple scheme whereby two parents will not cross over if they are too similar (Eshelman and Schaffer 1991). Similarity, in this case, is determined by a threshold Hamming distance which is reduced during the course of a run. Finally, Mauldin (1984) uses a tailored mutation operator to prevent duplication, in which new children are repeatedly mutated until they are sufficiently *unique*, that is, until their Hamming distance from every other string is sufficiently great. The required *uniqueness* is reduced during the course of a run, so the GA samples strings over an increasingly fine lattice, in a process similar to cooling in simulated annealing.

### Selection pressure

Selection pressure can be considered as the differential between the most and least fit members of the population. If selection pressure is too high, a finite population will quickly fill with duplicates of the high performing strings, and the resulting loss of diversity will stall the search; this is known as *premature convergence*. On the other

hand, selection pressure must be high enough to overcome the stochastic errors in selection, otherwise there will be no tendency for better strings to be selected more often than poor ones, and the population will wander randomly along a fitness contour; this is known as *genetic drift*.

If strings are evaluated according to a fixed scale there is a tendency for both of these effects to occur. Initially, a few good strings seize the lion's share of the available reproductive trials, and later in the run selection pressure wanes as the optimum is approached and the difference between the population average and the fittest string decreases. Like the frog which tries to cross the pond by always leaping half the remaining distance, the GA often quickly locates a near-optimal solution but never quite makes it to the optimum. To try to prevent this, a string's absolute fitness (according to the objective function) is often passed through a *fitness scaling* function, which compresses or expands the range of fitness values present in the population so that the difference between fittest and least fit, and thus the selection pressure, remains constant. The use of fitness scaling functions is widespread in the GA community, with the earliest example, according to Goldberg (1989c), being Bagley's (1967) adaptive game-playing program.

An alternative method of maintaining constant selection pressure, and one which departs further from Holland's original fitness-proportionate selection, is *ranking*. Introduced by Baker (1985), ranking simply involves sorting the population in order of descending fitness, and allocating a number of reproductive trials to each string according to a fixed scale. For example, the fittest string might always be selected five times, the second fittest four times, and so on. Ranking maintains constant selection pressure at the expense of ignoring the relative magnitudes of string fitnesses, and therefore also of schema fitnesses. Theoretically, this would seem a high price to pay; the fact that ranking schemes have proved highly successful underlines the gap which exists between GA theory (which often rests on assumptions such as a population size tending to infinity) and GA practice. Whitley's popular GENITOR algorithm uses ranking, and the advantages of rank-based selection, and its relationship to Holland's schema theorem, are discussed in (Whitley 1989). Montana and Davis (1989) report empirical success with a variation of ranking which allocates the number of trials to each rank according to a geometric scale. Ranking methods are particularly useful when the population is small in comparison to the size of the search space. For large populations, where premature convergence tends to be less of a problem, the extra computational overheads associated with ranking (ranking requires sorting the population, unlike roulette-wheel selection) may become significant.

Another selection method which has enjoyed empirical success despite lacking firm theoretical foundations is *tournament selection*, introduced by Brindle (1981).

The algorithm is simple: a number of strings are chosen at random, and the one with the highest fitness is selected for reproduction. This process is repeated until all parents have been chosen. Selective pressure depends on the number of strings which compete each time – the *tournament size*. Clearly, the larger the tournament size, the higher the selective pressure. Tournament selection has several desirable features: it is simple to implement, fairly computationally efficient for large populations (since only a small number of strings are compared each time), and it ensures that selection pressure is maintained towards the end of the run – any advantage in fitness, however small, is rewarded. Tournament selection seems to be particularly favoured among the *genetic programming*[8] community, e.g. (Reynolds 1994).

Goldberg and Deb (1991) have undertaken an analytical comparison of the expected convergence rates of roulette-wheel selection, ranking, binary tournament selection and the GENITOR algorithm (ranking with steady-state reproduction). Their analysis suggests that GENITOR typically exhibits the most rapid proliferation of fit strings (which may lead to premature convergence), followed by linear ranking and binary tournament selection, which display similar characteristics, with roulette-wheel selection being the slowest.

### Subpopulations and speciation

The selection and replacement strategies described so far have been population-wide, or *panmictic*. An alternative approach to balancing exploration and exploitation is to divide the population into subpopulations, or *demes*. In this approach, selection, mating and replacement almost always occur within a subpopulation, but occasional matings occur between subpopulations. The subpopulations act like mini-GAs, each converging on a particular evolutionary path on the fitness landscape. The formation of subpopulations is also referred to as *niche-formation* or *speciation*.

The most obvious way of creating subpopulations is by running several small GAs independently, and occasionally transferring strings between them. This approach, which has the advantage that it maps very well onto parallel hardware such as a transputer array, has been taken by Muhlenbein and Gorges-Schleuter in their ASPARAGOS GA, and has yielded impressive performance on various hard problems (Gorges-Schleuter 1989; Mühlenbein 1989; Mühlenbein and Kindermann 1989). Similar subpopulation schemes include the *partitioned* and *distributed* GAs of Tanese (1989) and Whitley and Starkweather's GENITOR II (Whitley and Starkweather 1990).

[8]Genetic programming is a growing branch of evolutionary computation pioneered by John Koza (Koza 1992), in which tailored GAs are used to evolve LISP programs to solve arbitrary problems.

Deb and Goldberg form subpopulations by incorporating a *restricted mating* scheme into their sharing mechanism (Deb and Goldberg 1989), whereby crossover only occurs between strings which are members of the same neighbourhood. This has the effect of forming different subpopulations on different peaks of their multi-modal test function. Because mating doesn't occur between subpopulations, this greatly reduces the generation of 'lethal' strings which fall into the regions of low fitness between peaks.

In Deb and Goldberg's scheme, the subpopulation to which a string belongs is uniquely determined by its associated point on the fitness landscape; neighbouring points on the fitness landscape are grouped into the same subpopulation. Other researchers have implemented subpopulations in a more general manner, by imposing a separate metric to determine subpopulation membership. This approach has the advantage that similar points on the fitness landscape need not be members of the same subpopulation. Moreover, the same point can be a member of several subpopulations. This allows the GA to explore several evolutionary paths from the same starting point, and can help escape local optima. An example of this more general approach is the work of Spears (1994), in which each string has several tag bits appended which label it as belonging to a particular subpopulation. When the population is initialised, a string's tag bits are assigned randomly along with the other bits in the string, so the initial distribution of points on the fitness landscape into subpopulations is random. Mating is only allowed between strings which share the same tag bits, so the only possible exchange of genetic material between subpopulations occurs when a tag bit is mutated, an event which Spears refers to as *diffusion*[9].

Another approach to creating subpopulations is to define an arbitrary spatial relationship between strings in the population, and only allow mating between nearby strings. In this model of *geographic speciation*, used in the work of Manderick and Spiessens (1989), Davidor (1991) and Collins and Jefferson (1991), the population is mapped onto a 2-D toroidal lattice so that each string is assigned a unique cell. Note that neighbourhood on the lattice is unrelated to neighbourhood on the fitness landscape; as with Spears' GA, the same point on the fitness landscape might be duplicated in several different cells. Davidor's algorithm is a steady-state GA, and selection begins by choosing a cell on the lattice at random. Two parents are then selected in separate stochastic tournaments among the 9-cell neighbourhood of the chosen cell, and the new child is replaced into the same neighbourhood. Manderick and Spiessens use a similar selection procedure, but their GA is generational rather than steady-state, replacing every cell in each generation. Collins' selection procedure

[9]Diffusion was actually unimplemented in the paper referenced here, since mutation of tag bits was not allowed.

is slightly different, in that a parent is chosen by a tournament among strings encountered on a short random walk across the lattice from the cell to be replaced. In all three algorithms, subpopulations of similar strings form in local regions on the lattice, with a limited exchange of genetic material occurring at the borders between them.

### 2.2.5 Representations and operators

#### Binary representations and operators

A popular variation of the traditional binary coding of parameters is the use of Gray codes. Gray codes are binary representations of integers which have the property that the representations of consecutive integers differ by only one bit. With normal binary coding, for example, a *Hamming cliff* exists between the binary representations of the integers 7 and 8, represented by 0111 and 1000 respectively, since a unit change in the integer requires a much greater change in its representation; every bit must be mutated simultaneously in order to change between the binary representations of 7 and 8. Hollstein (1971) first used Gray codes to relieve this problem in GAs. However, as Hancock (1992) points out, there are arguments against the use of Gray codes. While a unit change in an integer always produces a single bit change in its Gray coded representation, the reverse does not necessarily apply. A single mutation of the Gray coded binary 0000, representing 0, is enough to produce 1000, representing 15 – a maximal change in the integer. Thus Hamming cliffs exist in Gray codes as well. Despite the rather mixed results of empirical comparisons between Gray codes and normal binary coding, e.g. (Caruana and Shaffer 1988; Hancock 1992), Gray codes remain a popular choice among GA practitioners.

Many different crossover operators have been tried in place of Holland's original 1-point crossover. These include 2-point and $n$-point crossovers, where parents exchange several short sections of chromosome, and Syswerda's (1989) *uniform crossover*, which exchanges an arbitrary subset of genes, chosen with uniform probability along the length of the chromosome. Although there appears to be some evidence that 2-point crossover is generally slightly superior to 1-point crossover regardless of the function being optimised (Schaffer 1989), empirical comparisons of crossover operators have not favoured any one decisively. For example, uniform crossover works well in combination with steady-state selection and replacement (Syswerda 1989), but can be too disruptive for non-elitist GAs (Schaffer 1991).

Various crossover operators exist in which crossover points are chosen non-randomly. Booker's (1987) reduced surrogate crossover aims to increase efficiency

by always choosing crossover points at loci where the parent strings differ, to ensure that a new combination of genes is produced. Eshelman and Schaffer's (1991) HUX operator is a combination of reduced surrogate crossover and uniform crossover, in which parents exchange a subset of the genes which differ between them. Schaffer's (1987) *punctuated crossover* uses a binary mask to determine which loci may be crossover points. Each string has its own crossover mask appended to it, so the masks themselves adapt as the strings evolve.

Levenick, inspired by a feature of biological chromosomes, experimented with the insertion of *introns*[10] into strings, sequences of bits which are not decoded, but simply serve as padding between coded parameters (Levenick 1991). Levenick found that the insertion of introns improved performance by reducing the disruptive effect of crossover without affecting its ability to assort building blocks. This assumes, however, that any crossover point which falls within an intron will result in a non-disruptive crossover, that is, introns must separate known low-order building blocks. While this condition was fulfilled in Levenick's test function, the benefit of inserting introns in GAs for practical applications has yet to be demonstrated.

Efforts have been made to improve GA performance by varying operator probabilities on-line. Fogarty (1989) describes two approaches to varying the mutation rate in a binary GA. The first of these is a standard GA in which the mutation rate is decreased exponentially during the course of a run. In the second approach, the mutation rate differs for each bit according to its significance relative to the integer parameter it codes, so that the more significant the bit, the less frequently it is mutated. Unfortunately, Fogarty's results were rather negative. The only experiments where either adaptive mutation method produced a statistically significant increase in performance over a standard GA were those in which the population was initialised so that every string had a zero at every locus. Given that the initial mutation rates for the adaptive GAs were several orders of magnitude greater than the constant mutation rate used for the standard GA, this result is hardly surprising.

Since choosing the best genetic operators for non-trivial problems remains something of an art, Davis (1989) supports the approach of maintaining a pool of many different genetic operators which compete for the chance to be applied during reproduction. Those operators which more frequently produce fit offspring are rewarded by giving them a higher probability of being chosen. Montana and Davis (1989) have successfully used this method for balancing the probabilities of several specialised operators in a GA for optimising neural network weights.

---

[10]See (Gilbert 1978; Dorit, Schoenback, *et al.* 1990; Doolittle and Stoltzfus 1993).

### Higher cardinality alphabets

A great many of the practical applications of GAs have used alphabets of higher cardinality than binary. Bramlette uses an integer representation (which he calls a *vector representation*) for several benchmark function optimisation problems (Bramlette 1991), and a real-world aircraft design task (Bramlette and Cusic 1989; Bramlette and Bouchard 1991). Bramlette uses 1-point crossover and a mutation operator which perturbs a gene by a random quantity, rounded to an integer and biased to make small mutations more probable than large ones. Mansour reports empirical success using integer codings and tailored operators for multicomputer task allocation (Mansour and Fox 1991), and Hancock has found that integer representations compare favourably to binary for various neural network optimisation problems, e.g. (Hancock and Smith 1990; Hancock 1992).

The majority of non-binary GAs use real-valued (floating-point) genes. Real-valued codes are frequently used in the optimisation of neural network parameters, discussed in detail later in this chapter. As well as neural network optimisation, example applications of real-valued GAs include function optimisation (Radcliffe 1991b; Wright 1991), optimisation of reactive control parameters for a simulated robot controller (Pearce, Arkin, *et al.* 1992), and various problems in analytical chemistry (Lucasius and Kateman 1989), among many others. Janikow and Michalewicz (1991) compare floating-point and binary representations on a complex dynamic control problem, and find that the floating-point GA significantly and consistently performs better. They attribute the improvement to the fact that the floating-point representation is more natural for this problem, and allows powerful problem-specific operators to be used.

While the various crossover operators used in binary GAs are equally applicable to GAs of higher cardinality, additional crossover operators have been introduced which are specific to real-valued codings. Radcliffe (1991c) proposes *flat crossover*, in which each gene in the offspring takes a value randomly chosen in the range defined by the equivalent genes in its parents. In order to offset this operator's tendency to continuously narrow the population's range of values at each locus, a tailored *extremal mutation* operator, which inserts extremal values into the gene pool, is also used. Wright's *linear crossover* (Wright 1991) takes a rather different approach, treating whole parent strings as points in parameter-space and generating offspring which lie on the line between them.

Davis (1991) describes various mutation operators for use with real valued codings. *Real number mutation* is the high-cardinality analogue of bit-flipping, in that it simply replaces a gene with a randomly chosen real value. *Creeping* mutation is more popular, and works by perturbing a gene's value by a small random quantity,

chosen uniformly over a particular range. The magnitude of the perturbations will affect the GA's performance; small changes are desirable to facilitate hill-climbing on smooth problem surfaces, but larger mutations may be necessary to maintain diversity and prevent premature convergence. Davis recommends using several creeping mutation operators, applied with different probabilities, each operating with a different range (Davis 1991).

### Other representations and operators

Certain classes of optimisation problem do not lend themselves to coding as a list of parameters, and attempts to apply GAs to these problems have spawned a plethora of highly non-standard representations and operators. Radcliffe (1992) cites the class of permutation-based problems such as the TSP (Travelling Salesman Problem) as a prime example.

The most popular coding scheme for the TSP is the simple order-based representation introduced by Goldberg and Lingle (1985) in which each chromosome is an ordered list of cities representing a tour. To recombine these strings without producing illegal tours, Goldberg introduces *partially mapped crossover* (PMX). Grefenstette examines various more complex representations and operators for the TSP, including an *adjacency representation* and a crossover operator which assorts edges (Grefenstette, Gopal, *et al.* 1985; Grefenstette 1987), and obtains promising solutions to TSPs of up to 200 cities. Whitley, Starkweather, *et al.* (1989) introduce *edge recombination*, a more general variant of Grefenstette's heuristic crossover. Many other, lesser known, operators have been applied to the TSP (Liepins, Hilliard, *et al.* 1987; Oliver, Smith, *et al.* 1987; Suh Jung and Van Gucht 1987; Gorges-Schleuter 1989), and problem-specific order-based representations and operators, including PMX and edge recombination, have also been applied to the related problem of scheduling (Davis 1985; Whitley, Starkweather, *et al.* 1989; Bagchi, Uckun, *et al.* 1991; Syswerda 1991; Whitley, Starkweather, *et al.* 1991).

The many other types of problem to which non-standard codings and operators have been applied are too numerous and too varied to be considered in detail here. Examples are the use of *diploidy*, *dominance* and *latent genes* for non-stationary function optimisation (Goldberg and Smith 1987; Dasgupta and McGregor 1992), *variable length chromosomes* (Hintz 1989; Goldberg, Korb, *et al.* 1990; Goldberg, Deb, *et al.* 1991), *matrix* and *multiple chromosome* representations (Juliff 1992; Cartwright and Harris 1993) and *tree-structured chromosomes* (Jones 1992; Koza 1992). A tree-structured representation forms the basis of the *GA-bumptree*, a novel GA-neural network hybrid introduced in the following chapter.

### 2.2.6 Other evolutionary algorithms

The field of genetic algorithms is the largest of three separate computational paradigms inspired by the natural process of evolution. The others, evolution strategies (ES) and evolutionary programming (EP) are collectively known as *evolutionary algorithms* (EAs), to distinguish them from GAs. Although similar in many respects, historically, GAs and EAs have two fundamental differences:

> "Evolutionary algorithms emphasize phenotypic adaptation, while genetic algorithms emphasize genotypic transformations. In addition, evolutionary algorithms treat evolving structures holistically, in contrast with genetic algorithms which assume the reductionist perspective of assembling desired structures from the bottom-up." (Fogel 1993)

Evolution strategies (Bäck, Hoffmeister, *et al.* 1991; Bäck and Schwefel 1993; Rechenberg 1994) manipulate strings of real-valued parameters, and firmly emphasise mutation as the principal search mechanism. Recombination, so fundamental to the GA, is rarely used; when it is, it is seen as a useful heuristic and plays a supporting role. ESs come in various flavours, distinguished by the notation $(p[+,]o)$, where $p$ is the size of the parent population, $o$ is the number of offspring generated each generation and '+' and ',' denote two alternative approaches to selection. Selection is purely deterministic, based on rank, and competition occurs either among the new offspring (,) or the whole population, parents and offspring (+). In roughly historical order, the development of ESs can be summarised as follows (Bäck 1994):

- (1,1) : Random walk – search from a single point in a succession of small random steps.

- (1+1) : Minimal 'real' ES – generate a new point by mutating the existing one, and keep the better of the two. This simple stochastic hill-climbing is referred to by GA practitioners as RMHC (Random Mutation Hill Climbing) or bit-climbing (Mitchell, Holland, *et al.* 1994).

- (μ+1) : Keep a population of μ points. Each generation, generate a new offspring at random by mutating an existing point, and discard the worst out of the μ+1 points. This is analogous to a steady-state GA with no recombination.

- $(1+\lambda)$ : 'sphere model' – search from a single point. Each generation, generate $\lambda$ mutant offspring within a fixed-radius sphere (or hypersphere) centred on the current point, and keep the best. There is no real GA equivalent.

- $(\mu+\lambda)$ : Similar to an elitist real-valued GA, but with emphasis on mutation rather than recombination.

- $(\mu,\lambda)$ : State of the art ES – as above, but purely generational rather than elitist, and with a self-adapting mutation operator.

In an ES, mutation generates a new point in the region of an existing one according to a multi-dimensional Gaussian probability distribution over the parameter space, centred on the parental point. The variances and covariances of this distribution are different for each individual, and are stored as additional parameters appended to each string. The parameters governing mutation are therefore subject to evolution themselves, and are adapted by mutation and recombination. Recombination typically operates differently on each part of a string: the objective function parameters are exchanged discretely, while the mutation parameters are mixed to form intermediate values.

Evolutionary programming (Fogel 1994) is virtually the same as a $(\mu+\lambda)$-ES, with $\mu=\lambda$. In modern EP, mutation operates in a similar manner to that in ES, with mutation parameters also appended to individuals' strings. Selection is, however, stochastic rather than deterministic: in GA terms, EP uses fitness-proportionate selection as opposed to ranking. Unlike modern ESs, EP uses no recombination at all (Fogel 1993).

As the field of evolutionary computation has itself evolved, the use of real-valued codes and operators in GAs and the incorporation of GA-like recombination operators into ESs have blurred the distinctions between GAs and EAs. There is now a growing tendency in the literature for the entire spectrum of evolutionary algorithms to be grouped under the banner of 'genetic algorithms'.

### 2.2.7 Summary

The preceding sections have presented a broad review of the field of genetic algorithms. Three general conclusions emerge naturally from this literature survey, and these have formed the basis of the GAs used in the experimental work described elsewhere in this thesis.

Firstly, there is no longer any theoretical reason to prefer binary codings. The generalisation of schemata by Radcliffe and Vose (section 2.2.3) lends support to the

approach of designing higher cardinality representations and operators tailored to each particular application. The empirical success of real-valued GAs and the various practical applications of ESs and EP offer strong evidence to support searching in the space of real parameters wherever appropriate. In addition, the success of phenotypic (as opposed to genotypic) comparisons for crowding and sharing algorithms (section 2.2.5) underlines the fact that binary representations don't generally preserve continuity in the fitness landscape.

The second conclusion concerns the design of crossover operators. The important fact is that different crossover operators make different implicit assumptions about the degree and nature of genetic linkage (i.e. the nature of the building blocks) in the population. The performance of a particular operator will depend on how justified these assumptions are for the problem being tackled. The relationship between genetic linkage and the crossover operator is a critical one, and is examined in greater detail in chapter 7.

The third conclusion concerns the choice of selection and replacement strategies. For finite-population GAs applied to non-trivial optimisation problems it is apparent that simple fitness-proportionate selection and generational replacement are invariably associated with premature convergence. Of the many methods discussed in section 2.2.5 for dynamically balancing selection pressure, the most promising seems to be the local mating model, where selection and replacement are constrained to a limited neighbourhood on a lattice defined over the population. This 'geographical speciation' approach embodies the principle of crowding, that offspring should replace similar parents, without the need to calculate a distance metric for every possible pair of strings. It also incorporates ranking, in that selection pressure is maintained throughout the search without fitness scaling, but does not require the population to be sorted. Finally, local mating leads to subpopulation formation, which may help the GA to escape local optima on multi-modal fitness landscapes (section 2.2.3). Since the local mating model offers many of the advantages of more complex selection and replacement strategies, while being computationally inexpensive and simple to implement, it has been the basis of the GA used in the work reported in chapters 3, 4 and 6 of this thesis.

## 2.3 Artificial neural networks

Artificial neural networks (hereafter referred to simply as 'neural networks', or NNs) are parallel, distributed machine learning architectures. They were originally inspired by, and are loosely based on, biological nervous systems. The field is massive and growing, so only a very brief and broad overview is given here; the reader unfamiliar

with neural networks will find good introductions in (Lippmann 1987; Dayhoff 1990; Hush and Horne 1993).

A neural network consists of a number of processing units, or neurons, joined to each other with weighted connections. In general, the number of inter-neuron connections is very much greater than the number of neurons. Although each neuron computes a very simple function of its inputs, the behaviour of the network as a whole can be extremely complex. The precise function computed by the network is determined by the particular combination of values of all its connection weights. Because of this, neural networks are also referred to as *parallel distributed processing* (PDP) or *connectionist* architectures. In neural networks, 'learning' refers to the process of adapting the individual weights until the network as a whole computes some desired function.

Neural networks can be divided into two broad categories: supervised and unsupervised learning architectures. Unsupervised learning networks self-organise according to data presented to them, and are typically applied as alternatives to statistical clustering algorithms. In these systems the weights are usually initialised randomly and, thereafter, neurons compete to respond to each input pattern. The winner (the neuron with the highest activation for that pattern) is reinforced by modifying its input weights to make it even more likely to win for future occurrences of similar patterns. In this way the network self-organises, so that different neurons respond to input patterns falling into different classes. These classes (or clusters) are not determined *a priori*, but emerge as the network captures regularities in the data. Unsupervised learning networks include simple winner-takes-all competitive systems (Rumelhart and Zipser 1986) and the Kohonen feature map (Kohonen 1988).

This thesis is concerned with the second kind of neural networks, those which perform supervised learning. A supervised learning network is trained by example to implement a particular mapping from its input space to its output space. Training typically involves repeatedly presenting known input/output pairs to the network, and applying a learning rule to modify the connection weights until the network produces the desired output in response to each input pattern. When training is complete, the network will exhibit some degree of *generalisation* (interpolation) ability, the ability to produce an appropriate output when presented with an input pattern which was not part of the training data. Typical applications for supervised learning networks are pattern recognition, classification and function approximation.

The earliest real examples of supervised learning neural networks were the *perceptron* (Rosenblatt 1962) and the *adeline* and *madeline* (Widrow and Hoff 1960; Widrow and Stearns 1985). These are arrays of simple artificial neurons, each of which computes a weighted sum of its inputs and produces an output if this sum exceeds a certain threshold. These systems are limited by the fact that they only have

a single layer of adaptive weights and are, as a consequence, only capable of learning linearly-separable mappings (Minsky and Papert 1969). In 1986, Rumelhart, Hinton, *et al.* introduced the *multi-layer perceptron* (MLP), a generalised supervised learning network which has one or more *hidden layers* of adaptive neurons between the input and output layers, and derived an associated training algorithm called *back-error propagation* (or simply *backpropagation*). Unlike the earlier models, the MLP can approximate any function (Funahashi 1989), and it has become the most commonly used and widely studied neural network model.

An important alternative to the MLP is the *radial basis function* network, or RBF (Powell 1985; Broomhead and Lowe 1988; Moody and Darken 1988; Moody and Darken 1989). Unlike the MLP, the neurons in the RBF's single hidden layer do not compute a simple weighted sum and threshold. Instead, each responds only to input which falls within a small localised region of the input space. Hidden unit activation is typically a Gaussian kernel function, so a particular hidden unit only generates a significant non-zero response for input points which fall close to its function's centre. Output layer units compute a linear combination of the hidden layer functions. The *bumptree* network, described in detail in the following chapter, uses similar local activation functions to the RBF.

## 2.4 Evolving neural networks

There is now a large body of literature in which GAs have been applied to the optimisation of NNs, and reviews of much of this research may be found in (Rudnick 1990; Weiss 1990; Yao 1992; Jones 1993). In the literature, the neural network model optimised by the GA is almost invariably based on the MLP, with the GA being used either to optimise the connection weights, the architecture, or other network parameters. The existing work in this area is summarised in the following sections.

### 2.4.1 GAs for optimising network parameters

One possibility is to apply a GA to search for optimal control parameters for backpropagation, such as learning rate or momentum terms. This approach has been taken by Marshall and Harrison (1991) and Belew, McInerney, *et al.* (1991). In the latter study the GA produced unusually large values for both learning rate and momentum. This result is not, however, as controversial as it first appears, and can be attributed to the fact that their networks were only allowed a small number of training epochs before evaluation. The optimisation of learning rate and momentum terms has also been combined with attempts to evolve network architecture, e.g.

(Harp, Samad, *et al.* 1989; Harp, Samad, *et al.* 1990; Harp and Samad 1991; Robbins, Plumbley, *et al.* 1993).

Another area where GAs have been applied is in feature selection (input dimensionality reduction) for classification problems, e.g. (Chang and Lippmann 1991; Wong and Tan 1992). On a speech recognition task a GA was able to reduce the number of input features for a KNN classifier from 153 to only 15. This was better than a conventional feature selection method, which selected a subset of 33 features for the same problem and resulted in higher test-set classification error than the GA (Chang and Lippmann 1991). In the same paper, the authors report successful applications of GAs to feature generation (discovering useful polynomial combinations of input features) and choosing an optimal set of KNN reference exemplars from a large training set.

Finally, some authors have experimented with using a GA to discover learning rules for neural networks. This approach was suggested by Bengio and Bengio (1990) and applied by Chalmers (1990). In the latter work a GA 'discovers' the delta rule by searching different combinations of parameters such as pre- and post-synaptic activity. Dasdan and Oflazer (1993) have recently reported analogous work in which a GA constructs a weight update rule for a Kohonen network.

### 2.4.2 GAs for training neural networks

#### GA vs. backpropagation

One possibility which has been widely explored is the use of a GA to optimise the weights of an MLP in order to minimise training error. When compared to backpropagation, the GA offers certain advantages. Perhaps the most significant is that, unlike backpropagation or similar gradient-descent methods, the GA does not require the error surface to be differentiable with respect to the weights in the network. This makes it a suitable training method when gradients are difficult or impossible to calculate, for instance in recurrent networks (Torreele 1991), deeply layered networks, or problems in which the error signal is only realised after many training epochs (temporal credit assignment problems) such as the artificial life scenario which will be described in chapter 5. GA-based network training also allows arbitrary node activation functions, not just those which are continuously differentiable.

Another attraction of the GA as a training method is that it offers new scope for tackling the common problem of overfitting noisy training data. *Weight decay* (Hanson and Pratt 1989), *weight elimination* (Weigend, Rumelhart, *et al.* 1990) and *soft weight sharing* (Nowlan and Hinton 1992a; Nowlan and Hinton 1992b) are

examples of *regularisation* methods, modifications to backpropagation which are designed to reduce the complexity (that is, the information capacity) of a network during training by limiting the variance in the weights. This is equivalent to limiting the number of degrees of freedom in the network's model of the training data, and has the effect of forcing the network to learn general trends underlying the data instead of fitting the minutiae. This results in networks with improved generalisation performance. Because the GA does not require a differentiable cost function, there are no constraints on the nature or complexity of training criteria; it is trivial to incorporate additional criteria such as regularisation parameters into GA-based NN training, simply by adding extra terms to the fitness function. There is no need to worry about calculating derivatives of the regularisation terms with respect to each weight in the network.

One big disadvantage of GA-based neural network training is training time. Kitano (1990) has shown that GAs typically take many times longer to reduce training error than backpropagation, which is itself often considered slow and rather inefficient. This is to be expected since, as a stochastic global search method, the GA is inherently less greedy than gradient descent and explores a far greater number of points on the error surface. While this means that GAs are typically less sensitive to initial conditions and less susceptible to becoming caught in local minima than gradient descent methods, it also makes them slow to fine-tune solutions.

Bos and Weber (1991) report an empirical comparison of the generalisation abilities of networks trained by GA and backpropagation for two x-ray fluorescence spectrometry problems, and find that the GA is outperformed by backpropagation on one problem, and the two algorithms perform similarly on the other. It is not clear if any general conclusions can be drawn from these rather inconclusive results, however, because their implementations of both the network model and the GA are somewhat non-standard.

### Genetic representations and operators for NN training

Neural network weights may be coded genetically in various ways. A simple representation is to code each network weight as a binary word and concatenate these to form the genome. This approach typifies the early work in combining GAs and neural networks, and reflects the GA community's deep-rooted faith in binary representations and standard operators such as one-point crossover and point mutation (see section 2.3). Examples of problems for which neural networks have been trained using this approach include path planning (Smith, Pitney, *et al.* 1987), artificial life (Ackley and Littman 1991) and medical diagnosis (Marshall and Harrison 1991), as well as various simple and well-known NN benchmark problems such as

XOR, ADDER, ENCODER and so on (Caudell and Dolan 1989; Whitley 1989; Whitley and Hanson 1989; Whitley, Starkweather, *et al.* 1990; Marshall and Harrison 1991).

Unfortunately, the use of a low-order representation and traditional bitwise genetic operators introduces discontinuities into the effects of the operators on the phenotype. A single point mutation which happens to occur to the most significant bit of a particular weight, for example, will have a much greater effect than one which happens to mutate the least significant bit. In addition, crossover will tend to split words, disrupting the binary representations of the weight values and producing offspring with new weight values which did not belong to either parent. Partly as a consequence of these flaws, this approach, while able to find good weight sets for simple learning problems and benchmarks such as XOR, has not been shown to scale to larger problems.

Another approach is to use a high-order representation, in which each gene represents a whole real-valued weight. This constrains crossover to its proper role of exchanging existing weight values, rather than generating new values by mixing up component bits. Various complex crossover operators have been employed, some of which exchange sets of weights corresponding to whole units at a time (Menczer and Parisi 1990) or whole paths from input to output layer (Paredis 1991). Applications of networks trained using a high-order representation include control (Hoptroff, Hall, *et al.* 1990; Schoenauer, Ronald, *et al.* 1993; Ronald and Schoenauer 1994; Saravanan and Fogel 1994; Schoenauer and Ronald 1994), classification (Montana and Davis 1989; Fogel, Fogel, *et al.* 1990a; Paredis 1994), x-ray spectrometry (Bos and Weber 1991)[11] and artificial life (Cecconi and Parisi 1990; Parisi, Nolfi, *et al.* 1991; Paredis 1994).

### 2.4.3 GAs for optimising network architecture

#### Why optimise MLP architecture?

In the MLP, the number of hidden units, the manner in which they are organised into layers and the presence or absence of connections between them all influence the network's final ability to learn and generalise, e.g. (LeCun 1989). A network with too few weights, for instance, will have too few degrees of freedom to model the data. Unfortunately, it is also possible to have too many weights: if the network is over-specified and the data is noisy, over-fitting can occur (Baum and Haussler 1990). In real problems, where the number of degrees of freedom in the system

[11]This work actually uses elements of both binary and real-valued representations: weights are coded as real values, but crossover operates at the bit level.

which generated the training data is unknown, choosing the number of weights in the network tends to be something of a hit-and-miss affair. Hand-crafted networks are typically over-specified, and additional measures to attempt to prevent over-fitting, such as those mentioned in the previous section, are employed.

Various attempts have been made to automate the choice of network architecture (Wynne-Jones 1991). One method is to begin with a network which is known (or expected) to be over-specified, and then remove weights and/or whole nodes after training. This is referred to as network *pruning*. Mozer and Smolensky (1989) describe one such process which they call *skeletonization*, in which they calculate a measure of the *relevance* of each of the nodes in a trained network, an approximation of each node's contribution to the network's performance. The least relevant node is deleted and the network is re-trained, and this process repeats until the minimum architecture has been found which can still model the data to an acceptable degree of accuracy. LeCun, Denker, *et al.* (1990) describe a very similar method for removing individual weights which they refer to as *optimal brain damage*.

The complementary approach to pruning is to begin with a very small network and apply a constructive algorithm to add weights or units until the network is able to model the data. One example of such an algorithm is *cascade correlation* (Fahlman and Lebiere 1990), which maintains a pool of candidate nodes from which to build the network and adds the one which most reduces training error whenever learning stalls. Other constructive algorithms include Frean's (1990) *upstart algorithm* and Wynne-Jones' *node splitting* (Wynne-Jones 1993). Bostock (1994) describes an algorithm which incorporates elements of both construction and pruning.

### Why use a GA?

Many researchers have investigated using the GA as a tool to search the space of possible network architectures for a particular learning problem. The arguments for and against this approach are similar to those already discussed for GA-based network training. Since a constructive or pruning method describes a single trajectory through 'architecture space', the final architecture reached is dependent on the particular initial choice of network architecture and weights. As a global search method, the GA is less sensitive to initial conditions and should, consequently, be less prone to generating solutions which represent local optima. Another advantage, as with training, is the simplicity of tailoring the fitness function to optimise for complex criteria. As well as generalisation performance, the fitness function might include constraints imposed by hardware, such as some maximum number of nodes or layers, or more subtle criteria such as reliability of convergence during training. In

the experiments described below, for example, networks are optimised for minimum training time using backpropagation.

The big disadvantage associated with using a GA for network architecture design is, predictably, the computing time required. While optimising MLP architecture using a GA may not actually be the most computationally expensive algorithm ever devised, it must rank highly. Every fitness evaluation involves training a network from scratch, which may require thousands of passes through the training set using backpropagation, and a typical GA run may sample tens of thousands of different networks. These severe computational overheads are one reason why most of the work in the literature to date has been with small networks and small training sets.

### A simple binary representation for network architecture

As with network training, various approaches to representing topology can be taken. The simplest approach is to constrain the network to some maximum number of nodes and define the topology with a binary connection matrix, as shown in figure 2.3. If the network is to be purely feed-forward (a necessary condition for training with backpropagation), only the shaded area of the matrix need be coded.

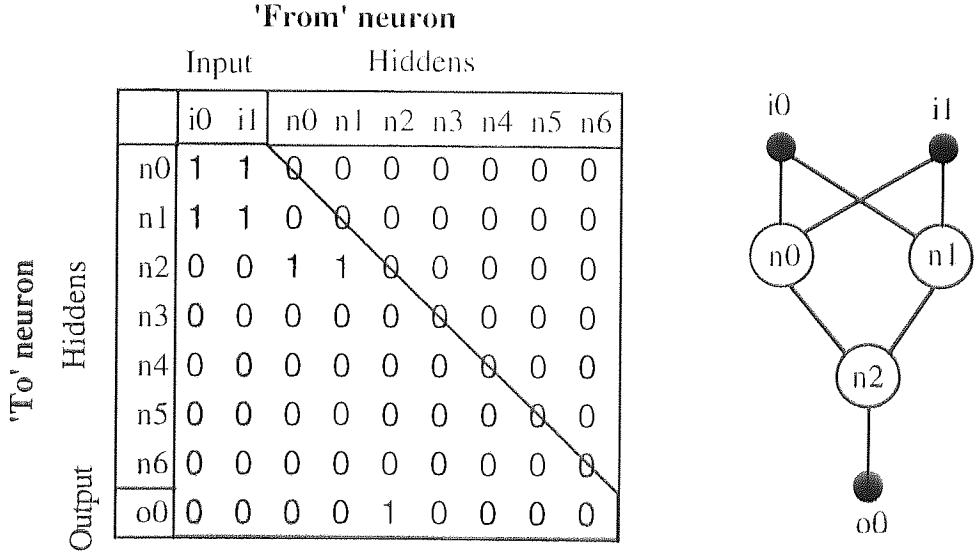



Figure 2.3: Example of connection matrix representation of MLP topology for networks of up to 6 nodes. A 1 specifies the presence of a connection, a 0 the absence.

This representation has the advantage that any possible organisation of the nodes can be represented by the genome. Since the genome is binary and of fixed length, a standard GA and traditional binary operators may be used. This representation scheme has been used recently by Robbins, Plumbley, *et al.* (1993) to generate small networks for the XOR and 'two spirals' benchmark problems.

### Experiments: optimising MLP architecture for fast learning of XOR

Some of the earliest experiments performed by the author, based on the work of Miller, Todd, *et al.* (1989), investigated the use of a simple GA using the connection matrix representation to generate networks which could learn the XOR mapping reliably. In the first experiment, a network's fitness was inversely proportional to the number of training epochs required to learn the mapping. Although XOR can be mapped with only a single hidden unit, the object was to see if the GA could make use of extra nodes or connections to generate networks with improved training speed, and so a maximum of 10 nodes was allowed. A population size of 20 was used, combined with a steady-state replacement strategy which discarded offspring if they were less fit than the poorest of the current population, or if they were duplicates of any members of the population. Up to 6000 training epochs were allowed, after which training was terminated. The results shown in figure 2.4 demonstrate that the GA is able to generate networks which learn XOR in progressively fewer training epochs.

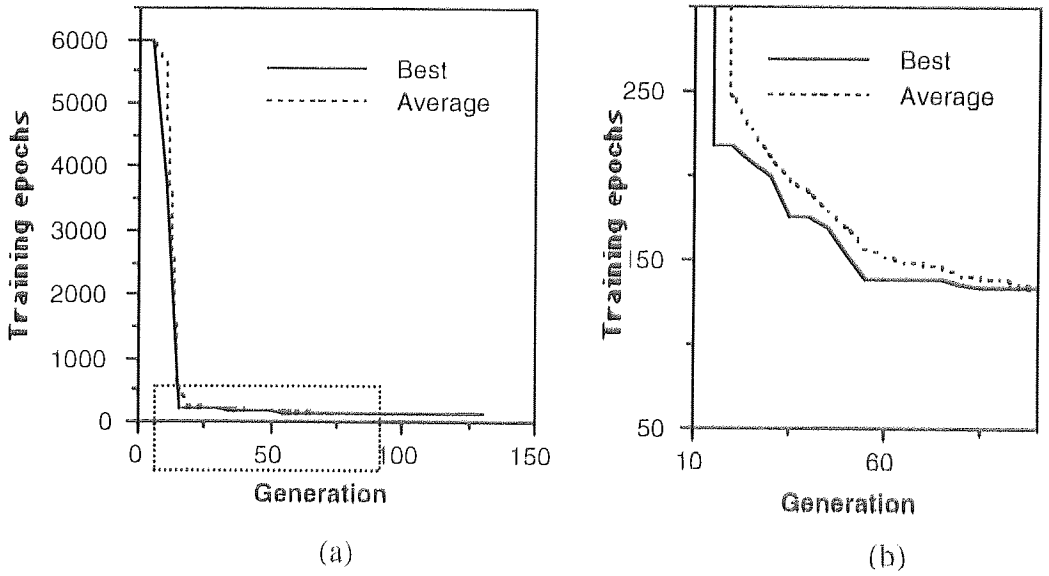



Figure 2.4: Best and average XOR training times for GA generated networks, per generation. (b) is an enlargement of the boxed area of the graph in (a).

This experiment differs from those of Miller, Todd, *et al.* (1989) in two respects: firstly, Miller *et al.* used a generational, rather than a steady-state, GA and secondly, they used a problem-specific crossover operator which exchanged whole nodes, whereas simple 1-point crossover was used in the experiment reported here.

One observation made by Miller *et al.* was that the networks produced by the GA tended to have direct connections from the input layer to the output unit, which is not a feature typically found in hand-crafted networks. This was also true of the networks generated in the experiment described here, as can be seen in figure 2.5(b). Miller *et al.* suggested that these direct connections may improve training speed on this task. However, Belew, McInerney, *et al.* (1991) point out that, since there is no selection pressure to remove connections which do not *adversely* affect training time, the presence of input-output connections is not necessarily evidence that they confer any advantage. The network shown in figure 2.5(b) demonstrates this fact – many of the networks evolved by the author's GA contain nodes and connections which are clearly non-functional.

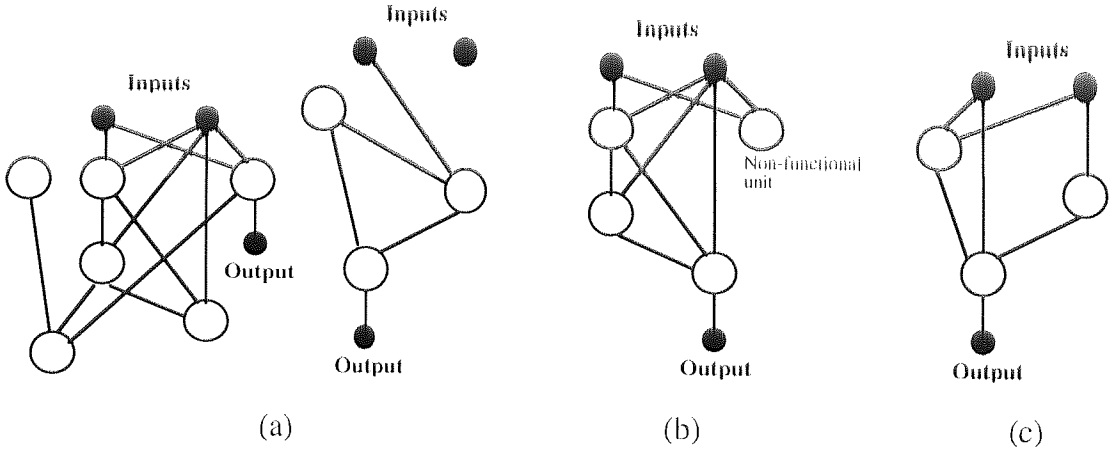



Figure 2.5: Examples of XOR network architectures produced by the GA. (a) shows two randomly generated networks from generation 0, before any evolution. (b) shows a final evolved network from the first experiment, with no connection penalty term, and (c) shows a final network when excessive connections are penalised.

In a second experiment, a penalty term was added to the fitness function for each connection present in the network, in order to encourage the evolutionary pruning of unnecessary connections. This did not adversely affect final learning speed, but caused non-functional units and connections to be removed from the networks. It was noted, however, that direct input-output connections still existed in the final networks – see figure 2.5(c). This result answers the criticism of Belew *et al.* and supports the hypothesis that these connections are not simply redundant, but do in fact speed learning, presumably by producing an error surface which can be more rapidly and reliably descended by backpropagation. In a complementary experiment, Whitley, Starkweather, *et al.* (1990) have applied a similar GA to prune

a pre-trained network, and their results lend further support to the assertion that direct input-output connections do beneficially influence training (in their case, re-training) speed.

### More complex representations

There are several reasons why the simple binary connection matrix coding for network topology does not scale well to anything other than toy problems. The first is that the length of the genotype grows quadratically with the number of nodes, so even a moderately sized network requires a massive binary string to represent its connection matrix. A second problem is that it is possible to represent, and therefore for the GA to generate, many networks which are fundamentally flawed, such as the second network in figure 2.5(a), which lacks any connection to one of the inputs. As network size increases, the proportion of strings which represent such *lethal* networks rises, and the GA may waste a large amount of time generating worthless solutions.

Harp *et al.* describe a coding for neural network architecture which is based on a Gray-coded binary string, but which doesn't represent a connection matrix directly (Harp, Samad, *et al.* 1989a; Harp, Samad, *et al.* 1989b; Harp, Samad, *et al.* 1990; Harp and Samad 1991). Instead, their coding consists of a variable number of blocks of parameters, each of which defines a group of nodes in the network. Parameters determine the number of nodes in the group, their organisation, and the nature of the connections to nodes in other groups. Since the coding is variable-length, and even the size of parameter blocks may vary, highly specialised genetic operators are used to manipulate the strings. Even so, the generation of lethal networks is still a problem. Since Harp *et al.* only report results on small problems (XOR, a sine-function and a simple digit-recognition task), the scaling properties of this complex representation are unclear.

A similar, but more constrained, representation scheme has been used by Hancock (Hancock 1990; Hancock and Smith 1990). Since there are fewer degrees of freedom in the representation, Hancock's GA is less prone to generating lethal structures than that of Harp *et al.* (Hancock 1990). Unfortunately, the GA was unable to improve on hand-crafted networks for a face-recognition problem. Mandischer (1993) has used a similar representation scheme, and reports some improvement over hand-crafted networks on test problems.

Kitano (1990) introduces a rather more abstract method of coding neural network architecture, in which the genetic string codes a number of graph rewriting rules which are applied iteratively to 'grow' a connection matrix in a process akin to cell division. A stochastic variant of this method has shown promise when applied to

generating deep networks for problems which are hierarchical and self-similar, such as high-dimensional parity problems (Voigt, Born, *et al.* 1993). Along the same lines, a developmental scheme inspired by the biological process of embryogenesis was proposed by Vico and Sandoval (Vico and Sandoval 1991; Vico and Sandoval 1992). Gruau has also developed a similar genetic coding, based on a number of growth rules for Boolean networks, which has recently shown promising results (Gruau 1992; Gruau 1994).

### Optimising architecture and weights simultaneously

A natural extension of the various GA-NN hybrids already described is to code both the architecture and the connection weights genetically and attempt to evolve a complete 'finished' network model for a particular data set. From the GA point of view, this approach has an immediate practical advantage when compared to optimising architecture alone: it removes backpropagation from the evaluation phase of the GA, and so considerably reduces processing overheads. For problem domains such as artificial life it is appealing for its 'biological plausibility' (Collins and Jefferson 1991; Collins and Jefferson 1991; Todd and Miller 1991; Wood 1991), and natural applications for the approach also exist in robotics and control problems where backpropagation isn't readily applicable (Cliff, Harvey, *et al.* 1992; Cliff, Husbands, *et al.* 1992; Harvey, Husbands, *et al.* 1992). From the pure NN viewpoint, this approach seems to offer something of a Holy Grail: a general method for simultaneously optimising both architecture and weights which places no constraints on network connectivity, node activation functions or optimisation criteria.

Unfortunately, the Holy Grail remains elusive. Despite the many papers in which GAs have been used to generate neural network architecture and weights simultaneously, and the large number of different representation schemes and genetic operators which have been tried, there is little evidence so far of empirical success on practical classification problems. Instead, studies have concentrated on simple neural network benchmarks such as XOR, PARITY, ADDER, ENCODER and the like. Examples which illustrate the diversity of approaches and the millions of CPU cycles which have been applied to modelling XOR include: networks designed using *genetic programming* (Koza and Rice 1991); Dasgupta and McGregor's *structured genetic algorithm* (Dasgupta and McGregor 1992); a massively parallel GA implemented on a SIMD computer (Penfold, Kohlmorgen, *et al.* 1993); a rather strange GA which operates on a population of individual nodes, and constructs networks a layer at a time (Anderson and Tsoi 1994); and Bornholdt and Graudenz' genetic recurrent networks (Bornholdt and Graudenz 1991).

### 2.4.4 The permutation problem

The emphasis on toy problems in the literature, in spite of the many different approaches taken, points to severe and fundamental scaling problems with applying GAs to find network architecture and/or weights. It is now widely recognised that the critical issue is finding a genetic representation for neural networks which will allow crossover to have the desired effect. In the case of the MLP, the problem lies in the fact that the nodes in a layer form an order-less set, whereas genetic codes are ordered strings. Since the order in which hidden units are coded makes no difference to the network's functionality, any particular MLP maps to many different genetic strings, one coding each possible permutation of its nodes. This redundancy in the genetic code leads to a search-space which is unnecessarily large, into which the space of real networks is mapped many times, and means that recombination of fit parents from different regions of the space will duplicate some selective traits and omit others. Figure 2.6 illustrates this: each hidden unit in network 2.6(a) is equivalent in function to a unit in 2.6(b) but, since the units are ordered differently on the coded strings, recombination by crossover produces an inferior network, 2.6(c).

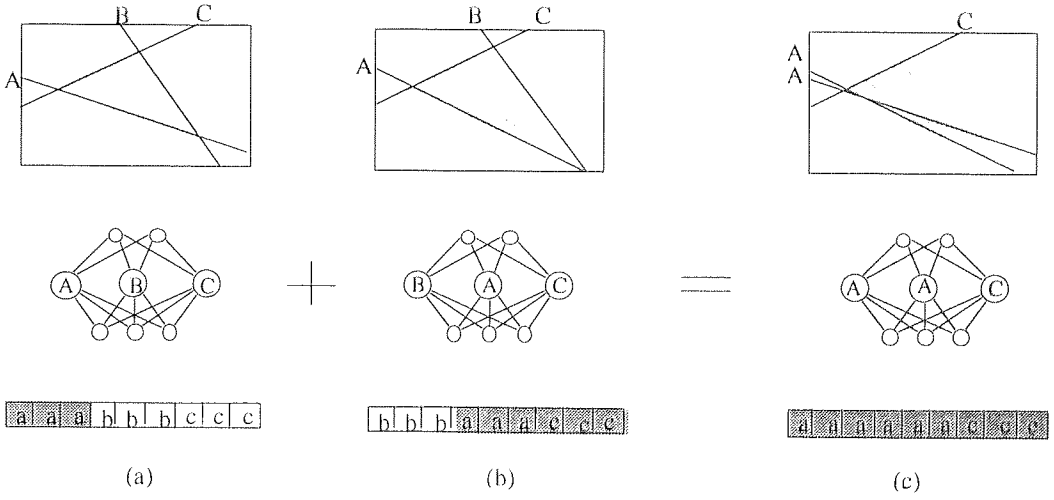



Figure 2.6: The permutation problem. The figure beneath each network represents the network's genetic code and the figure above represents the hyperplanes associated with its hidden units. Each hidden unit in network (a) is functionally equivalent to a unit in network (b). Because their coded order differs, however, crossover results in duplication, and generates an inferior offspring (c).

It is clear that the permutation problem exists whether the aim is to optimise network topology, weights or both. In fact, Belew, McInerney, *et al.* (1991) and Hancock (1992) point out that the situation is worsened by other redundancies, more subtle than simply permuting nodes. One example is that, because of the symmetrical

nature of the MLP's node activation function, it is possible to change the sign of all connection weights to any hidden unit without affecting the network's behaviour.

Many authors have simply avoided the permutation problem by using evolutionary algorithms which rely heavily or entirely on mutation to effect the search (Cecconi and Parisi 1990; de Garis 1990; Fogel, Fogel, *et al.* 1990a; Fogel 1990; Parisi, Nolfi, *et al.* 1991; Wood 1991; Saravanan and Fogel 1994). Menczer and Parisi (1990) describe an artificial life scenario where simple crossover does appear to offer some advantage over mutation alone, but only when applied at low probability - mutation is still the dominant operator. This emphasis on mutation effectively reduces the GA to a population of parallel instances of gradient descent, and not necessarily very efficient gradient descent at that. By avoiding the representation issue, however, even those practitioners who claim philosophical objections to the use of recombination as a search operator – see e.g. (Fogel 1994) – are searching a space which is unnecessarily large; even parallel stochastic gradient descent would benefit from non-redundant coding of the search space.

Some efforts have been made towards directly addressing the permutation problem illustrated in figure 2.6. Montana and Davis (1989) have experimented with the use of various specialised crossover operators, applied with different relative probabilities. One such operator attempts to identify which hidden units in the parents are performing equivalent functions by applying a subset of training inputs to each parent and comparing their units' response. Although this operator does appear to improve performance at the start of a run, when population diversity is high, its effect on final performance is less significant (Montana and Davis 1989). Radcliffe suggests a labelling algorithm which identifies networks which are identical except for the permutation of their nodes, but concludes that the algorithm is impractical (Radcliffe 1990).

Hancock (1992) considers the permutation problem in depth and discusses various approaches to tackling it. One possibility is to add asymmetric connections between hidden units so that they are no longer freely permutable. Another is to attempt to constrain the GA to searching within a single symmetry sub-space, i.e. a region of the search space in which all networks share the same node ordering. Hancock implemented the latter approach by coding hidden units in order of increasing bias weight, but this did not appear to improve performance (Hancock 1992).

More recently, Radcliffe (1993) has launched a vigorous attack on the permutation problem, focusing on the problem of evolving connectivity for a feedforward network with $j$ input units and $k$ outputs separated by a single hidden layer. Radcliffe treats the hidden layer as a multiset, the elements of which are binary vectors of $j+k$ bits. Each vector represents a particular set of input and output

connections, and so uniquely defines one of the $2^{j+k}$ possible kinds of hidden unit. Radcliffe's recombination operator constructs a new network by sampling nodes from the union of the parental multisets, with nodes which are common to both parents (i.e. those where the parents' multisets intersect) sampled with higher probability than those which are not. A binary crossover is also occasionally applied at the vector level, in order to sample hidden node types which may not be represented in the current population.

Based on the observation that Radcliffe's recombination operator differentiates crudely between those nodes which are common to both parents and those which are not, Hancock (1992) has introduced an operator which matches nodes according to a continuous scale of similarity, defined by a bitwise overlap metric. Each node in one parent is paired with the most similar node in the other, and a child is constructed by sampling one node from each such pair. Hancock compares this *Sort* operator with other variants of Radcliffe's recombination operator in a series of experiments in which a GA attempts to construct networks to match some hand-crafted target architecture. Although the results for this artificial task tend to favour the Sort operator, the fact that naïve uniform crossover also does remarkably well, outperforming some of the more complex recombination operators, suggests that there is still some way to go before the permutation problem is fully understood (Hancock 1992). Hancock draws one optimistic conclusion from these results, that the GA has a natural tendency to converge on a particular network permutation and so avoid the problem. While his results bear out this observation, reliance on this process would seem to be at odds with the accepted view that maintaining population diversity is of prime importance, especially with search spaces of large size and complexity (section 2.2.4). Indeed, as Radcliffe (1993) points out, relying on the population to converge on a particular permutation cannot be a solution when diversity-maintaining schemes such as subpopulation formation are employed.

Finally, it is worth noting that, although the preceding discussion of the permutation problem has focused on the MLP (which reflects a similar emphasis in the literature), precisely the same problem applies to the genetic coding of other multi-layer network models such as the RBF network.

## 2.5 Conclusions

This chapter has presented an extensive review of the literature of genetic algorithms and their application to optimising neural networks. Section 2.2 summarised current thinking on GA design and drew three broad conclusions which have underpinned the design of the GAs applied in the following chapters:

- There is no theoretical justification for preferring binary codes as a matter of principle; wherever possible, the coding scheme should reflect any structure known to exist in the search space.

- Recombination operators make implicit assumptions about relationships (linkage) between parameters. The degree to which these assumptions are valid will determine the performance of any particular operator.

- When designing a GA for difficult problems, preventing premature convergence must be a prime concern. To this end, the local subpopulation model appears to offer a good compromise between power and computational efficiency.

The conclusions of the review of existing GA-NN hybrids motivated much of the experimental work described in the remainder of this thesis. Experiments such as those described in section 2.4.3 demonstrate the potential of the GA for discovering useful novel network architectures, in this case networks with direct input-output connections, even for problems which have been as well studied as XOR. Neural network application areas such as control and robotics pose many learning problems which are not well suited to backpropagation, where GA-based network training offers considerable promise.

The scaling properties of present GA-MLP hybrids, however, are not encouraging. It is apparent that the multiset nature of MLP (and similarly RBF) architecture poses an extremely difficult coding problem. Despite recent efforts, the problems of finding a non-redundant genetic code and an associated recombination operator for the MLP remain to be solved.

A more mundane, but equally significant, obstacle in the way of applying GAs to optimising neural networks for real problems is the computational cost involved. This is especially true when the GA is applied to optimising network topology, since every network generated must be trained by backpropagation (or an equivalent method) before it can be evaluated.

In light of these conclusions, the following chapters introduce and develop a novel evolutionary neural network model based on the *bumptree*. The bumptree offers learning and generalisation performance comparable to that of the MLP or RBF, but is shown to be considerably more amenable to genetic optimisation.

# Chapter 3

# The genetic bumptree

## 3.1 Introduction

The main conclusion of the previous chapter was a frustrating one: while there are many potential applications for a global optimisation method such as the GA in the field of neural networks, the most widely used neural network models appear to be particularly unsuited to evolutionary optimisation. The work described in this chapter was motivated by this conclusion.

The following section introduces the bumptree network, a connectionist model based on the original bumptrees of Omohundro (1991) and examined in detail by Bostock (1994). The bumptree is attractive from the GA point of view because it appears not to suffer from the two fundamental problems posed by the MLP: computational expense and the permutation problem. The bumptree is very fast in comparison to the MLP, especially in terms of training time. Furthermore, because it is a more inherently ordered structure than the MLP, representing the bumptree as an ordered genetic code does not introduce the same degree of redundancy.

Section 3.3 describes experiments in which a GA is applied to optimising the architecture of the bumptree. The GA is described in detail and an appropriate genetic coding scheme is introduced and examined. The results indicate that the GA is able to discover superior architectures for the classification problems studied.

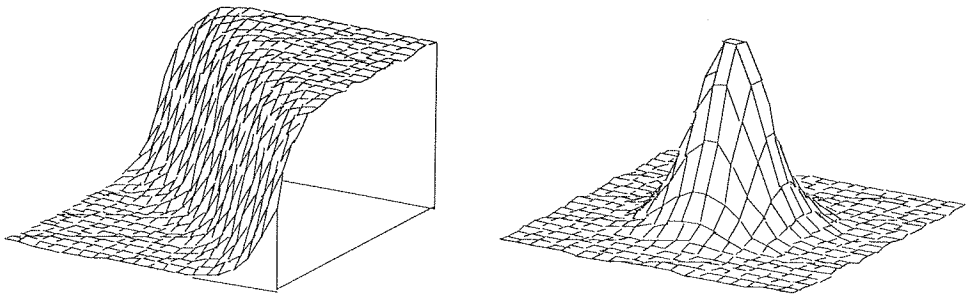
Section 3.4 builds on this work and extends the GA to optimise both the architecture and the weights of the bumptree network. This is a necessary step if the method is to be applied to problems where the bumptree's normal training rule cannot be used. Optimising architecture and weights simultaneously requires modifications to the coding scheme, and turns out to be a considerably harder problem than optimising architecture alone.

## 3.2 The bumptree network

In a typical neural network classifier, based on the MLP, each node $i$ computes an output activation, $y_i$, as a function of its inputs:

$$y_i = f\left(\sum_j w_{ij} x_j\right) \tag{3.1}$$

where the $j$th node input has activation $x_j$ and connection weight $w_{ij}$, and $f$ is a non-linear thresholding function, typically sigmoidal, as in figure 3.1(a). Each such unit defines a decision surface in the form of a hyperplane in its input space and, in the trained network, these hyperplanes model actual class boundaries in the training data.



(a) (b)

Figure 3.1: Sigmoid (a) and Gaussian product (b) activation functions for 2 input dimensions

The bumptree network is somewhat different. In the bumptree network, each node computes a more complex activation function, such as:

$$y_i = \prod_j \left( \frac{1}{\sqrt{2\pi}\sigma_j} e^{\frac{(x_j - \mu_j)^2}{2\sigma_j^2}} \right) \tag{3.2}$$

Here, $y_i$ is a product of Gaussians, where $\mu_j$ and $\sigma_j^2$ are the mean and variance, respectively, of a Gaussian associated with the $j$th input. In two input dimensions the activation function is a 'bump', as shown in figure 3.1(b), defining a circular decision surface. In higher-dimensional spaces, the decision surface is a hyper-ellipse enclosing a local region of the node's input space.

The bumptree itself is a binary tree of such nodes with the defining constraint (Omohundro 1991) that:

> ". . . each interior node's function must be everywhere larger than each of the functions associated with the leaves beneath it." (3.3)

Figures 3.2(a) and 3.2(b), adapted from (Omohundro, 1991), show the bumptree structure. Node A in figure 3.2 represents the region of input space in which the tree is constructed; for classification purposes, the top level of the tree is the first division of the input space into two regions (nodes B and C in the figure). At each successive level, each region is further divided into two smaller sub-regions. The purpose of the bumptree is to divide the input space into successively smaller regions until, at the leaf layer, the classification task has been reduced to a set of linearly-separable sub-tasks.

Associated with each leaf node is a 'local expert', a single layer network which forms a linear map between network inputs and outputs, as shown in figure 3.2(c). Each expert is responsible for the classification of points which fall within its leaf node's active region. When a point $y$ is input to the bumptree, a descent of the tree,

choosing the more active branch at each interior node, provides efficient access to the appropriate leaf node, $l$. The point is then classified according to the most active of the outputs $o_i$ of the expert associated with node $l$, where:

$$o_i = \sum_j w_{ij}^l y_j \tag{3.4}$$

with $w_{ij}^l$ being the weights connecting classifier input units $j$ to output units $i$.

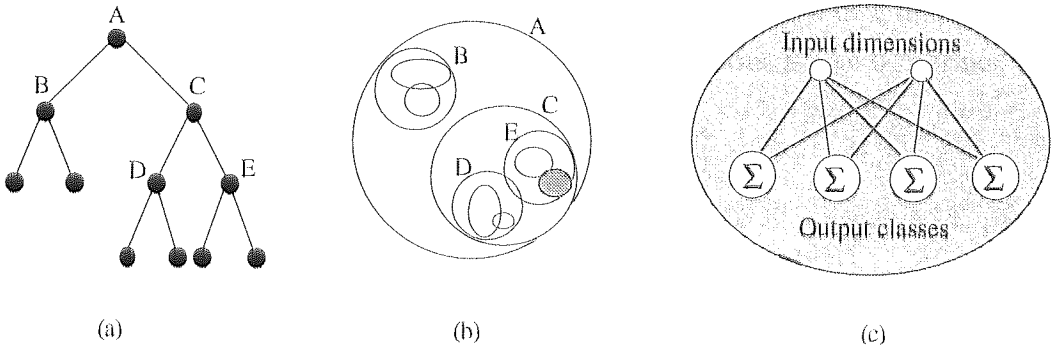



Figure 3.2: The bumptree structure (a), an example of partitioning in a 2-D space (b) and a local expert (c).

### 3.2.1 Constructing the bumptree

To build a data model with the bumptree network, the first step is to construct the tree. Bostock (1994) has investigated various constructive algorithms in which the tree is built from the top down, beginning with just two nodes. In a single pass through the training set, each expert's weights are optimised (see section 3.2.2 below) to minimise classification error on its particular subset of training points. After presentation of the training set, any leaf-layer node whose expert has an unacceptably high error, according to some pre-defined threshold, grows two new nodes below it, thus further dividing its input space between two new experts. As the process repeats, new experts are added until every expert has a suitably low error.

As each new node is added to the tree, the choice of its function's centre and radii influences the final distribution of local models over the input space; choosing the best tree to partition the space is obviously critical to the classification performance of the network. In the orthodox top-down constructive algorithm, local clustering (*e.g.* k-means) is typically used to position each new node function according to the local distribution of training data. Section 3.3 investigates the use of a genetic algorithm to optimise tree topology.

### 3.2.2 Training the local models

As well as choosing the best partition of the space, it is obviously necessary to fit each local model to its particular subset of the training data. Since the local experts are simple linear models (single layer networks) it is not necessary to use an iterative gradient descent method such as backpropagation in order to train them. Following Renals and Rohwer (1989), the weights associated with each expert network may be solved for in a single pass through the training set, as follows. For a linear network with weights $w_{ij}$, the error, $E$, on a subset of training vectors, $p$, can be defined as:

$$E = \tfrac{1}{2}\sum_{ip}\left(\sum_{j} w_{ij} y_{jp} - Y_{ip}\right)^2 \tag{3.5}$$

where, for training pattern $p$, $y_{jp}$ is the $j$th component of the input vector and $Y_{ip}$ is the $i$th target output. To find a set of weights for which $E$ is minimised, take the partial differential with respect to each individual weight $w_{kl}$. For each weight:

$$\frac{\partial E}{\partial w_{kl}} = \sum_{j} w_{kj}\left(\sum_{p} y_{jp} y_{lp}\right) - \sum_{p} Y_{kp} y_{lp} \tag{3.6}$$

At the minimum of $E$, each $\partial E/\partial w_{kl} = 0$, giving a set of simultaneous equations which may be expressed in matrix form as $\mathbf{WM}=\mathbf{T}$ giving $\mathbf{W}=\mathbf{TM}^{-1}$. Here, $\mathbf{W}$ is the unknown weight matrix, $\mathbf{T}$ is the target matrix and $\mathbf{M}$ is the correlation matrix of the network inputs. The matrices $\mathbf{T}$ and $\mathbf{M}$ can be constructed in a single pass through the training set, and the inverse $\mathbf{M}^{-1}$ can be calculated using singular value decomposition (Press, Teukolsky, *et al.* 1992), which gives a best estimate in the case where $\mathbf{M}$ is singular. The weights are then calculated in a single matrix multiplication. This method is equivalent to the standard Moore-Penrose pseudo-inverse method for solving linear systems (e.g. Golub and Van Loan 1989).

### 3.2.3 Comparison with other connectionist models

At this point it is worth briefly putting the bumptree in perspective in relation to other neural network classifiers.

The bumptree partitions its input into a number of local regions, each of which is then modelled individually. To this end, each node in the tree has a highly localised receptive field and responds only to a small region of the input space. This approach contrasts with the MLP, in which each hidden unit's response is defined over the

whole input space[1], and is more in keeping with models such as the RBF network (section 2.3) and the *Gaussian mixtures model* (Brindle and Cox 1991).

One example in the literature which bears certain similarities to the bumptree is the work of Wynne-Jones (1993), in which a hierarchical constructive rule (*node splitting*) is used to generate Gaussian mixtures models by repeatedly subdividing a small initial number of receptive fields. A similar approach was taken by Hanson (1990) in his *meiosis networks*. Also relevant is the recent work of Gentric and Withagen (1993) in which a decision tree is constructed on top of a pre-trained RBF network. This improves classification speed because it allows those nodes which are unlikely to respond to a particular input point to be quickly pruned away, significantly reducing the number of response functions which need to be calculated.

The principal difference between the bumptree classifier and the models described above is the fact that the bumptree models the data in a piecewise linear manner, rather than as a continuous approximation. In models such as the RBF or MLP, a single output layer forms a linear combination of the activations of all the receptive fields. In the bumptree, a collection of local experts are distributed into the input space, and each is responsible for modelling its own region of the data. The tree determines the manner in which the space is partitioned and allows efficient selection of the appropriate expert for a given input point, but each point is ultimately classified according to a single linear model of its local neighbourhood – see figure 3.3.

At this point it is worth noting that in the bumptree described here the system response is discontinuous over neighbourhood boundaries, although this need not necessarily be the case – smooth interpolation between neighbouring regions could be achieved by mixing the outputs of their experts, weighting according to the relative activation strengths of the associated leaf functions.

The interested reader is referred to (Bostock 1994) for a more detailed examination of the bumptree: for the present work it is sufficient to note that Bostock has performed extensive empirical comparisons in which the bumptree appears to offer classification performance comparable to that of other neural network models (Bostock 1994).

[1]The global response of hidden units in the MLP can lead to problems with fine-tuning the model, since a small adjustment of the weights of a hidden unit intended to improve the model in one region of the space may disrupt the model in some other region. This is well illustrated in (Wynne-Jones 1993).

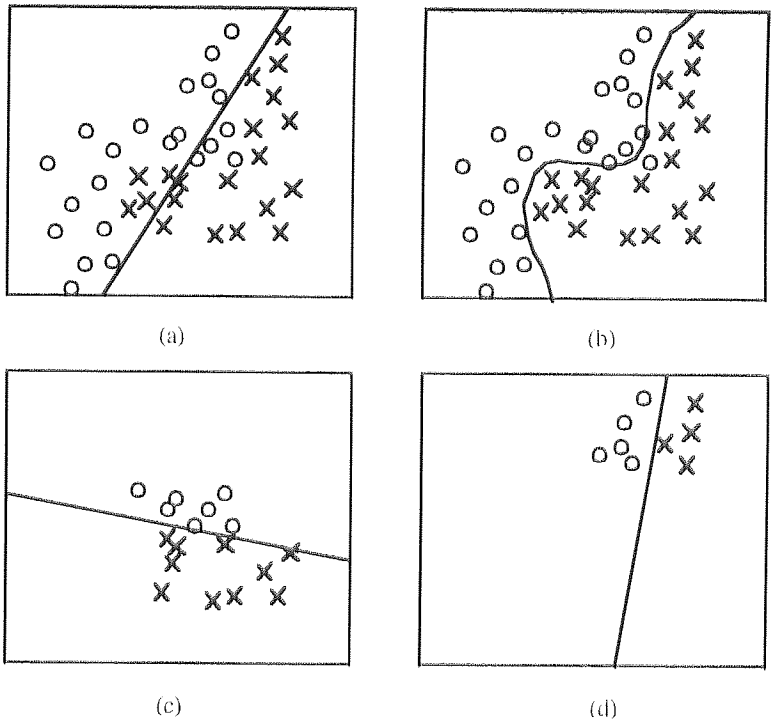



Figure 3.3: How different models separate two overlapping (i.e. linearly inseparable) classes in a 2-D input space. A single layer perceptron (equivalent to linear discriminant analysis) fits a single linear decision boundary (a) and some points are misclassified. Multi-layer networks such as the MLP or RBF form complex nonlinear decision boundaries (b) and can completely separate the classes. Each expert in the bumptree is a simple perceptron, but only 'sees' a localised region of the input space within which the classes are linearly separable (c,d).

From the GA point of view, a significant advantage of the bumptree over other neural network models is its speed, both in terms of training time and in access of the trained network. Training a bumptree of given topology only requires a single pass through the training set, in contrast to iterative gradient descent methods such as backpropagation which typically require hundreds or even thousands of passes. This makes genetic optimisation of bumptree topology computationally tractable, even for large networks or large data sets. Access of the trained network is also very fast. To classify an input point, a descent of the tree, requiring the evaluation of only two node functions at each layer, provides efficient access to the appropriate expert, after which a simple summation gives the classification. This is in contrast to networks such as the MLP or RBF, in which every node function must be evaluated in order to classify each point.

These speed advantages alone, which scale up with increasing network size and size of training set, make the bumptree a much more attractive prospect for hybridisation with a GA than other neural networks such as the MLP. Because the network forms the evaluation function for the GA, it operates in the innermost loop of the algorithm. Any increase in speed of the evaluation function is multiplied by every

chromosome evaluated in every generation. In GA experiments, the kind of speedup offered by the bumptree over the MLP can make the difference between hours of computing time and days.

## 3.3 Evolving bumptree topology

The potential advantages offered by an evolutionary approach to optimising neural network architectures have already been discussed for the case of the MLP (section 2.4.3). The advantages are the same for the bumptree: the GA offers a robust global search in the space of possible tree topologies which is not sensitive to initial conditions and which can easily be tailored to any optimisation criteria. An example is that when the bumptree is constructed from the top down, classification error influences *whether* new nodes should be added but not necessarily *where* their functions are placed – this is determined by the input data distribution. In the GA-optimised bumptree described here, the fitness function is simply the total classification error over all training points, after the expert weights have been calculated. This approach has the advantage that the optimisation of tree topology is driven entirely by the total classification error and, as a consequence, the bumptrees generated by the GA show improved classification performance over those generated using the top-down constructive algorithm. In the following sections the genetic algorithm implementation is described in detail.

### 3.3.1 Coding bumptree topology

In chapter 2 it was stated that a GA's effectiveness depends critically on the choice of an appropriate coding scheme and suitable genetic operators for the problem at hand. The literature review also showed that devising an appropriate genetic coding scheme for a neural network model is not a trivial task. In the case of the MLP, the fact that the nodes in any layer form an order-less set, whereas GAs typically work with ordered strings, leads to the crippling 'permutation problem'. A major attraction of the bumptree network is that it is possible to devise a coding scheme which does not introduce anywhere near the same degree of redundancy as typical genetic codes for the MLP.

In order to apply the GA, bumptree topology is coded as a chromosome of fixed length, built up of 30 blocks of real-valued genes as shown in figure 3.4. Each block defines a single node in the tree, with each gene in a block coding a separate parameter for that node, such as a function radius. The largest bumptree that this coding can represent therefore has 30 nodes, i.e. is a complete tree with 4 layers. This limit is rather arbitrary and is intended to keep the chromosome to a manageable

length without severely limiting the expressive power of the trees produced. The figure of 30 nodes was based on an initial examination of the size of trees typically produced by the constructive algorithm for the test problems. The results given later confirm that this limit is not overly restrictive: the largest trees produced by the GA contain around 20 nodes, and none of the GA runs produced a tree with 30 nodes.

The first two blocks on the chromosome define the two root-level nodes, which are always present in the decoded tree. Each node has a single gene which determines whether it is a terminal (leaf) node, or has two descendants. Each node is coded at a fixed position on the chromosome, so if the node defined by block $n$ is non-terminal its two descendants are defined by blocks $2n+1$ and $2n+2$ (see figure 3.4).

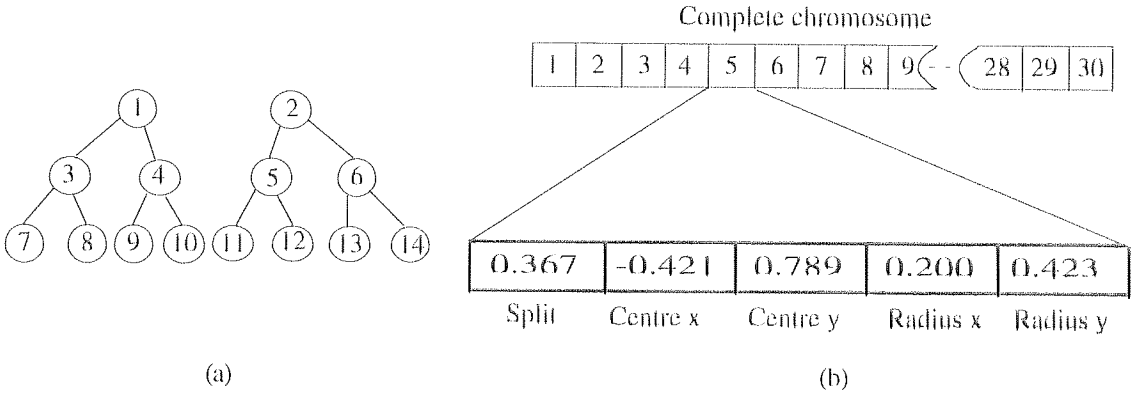



Figure 3.4: Genetic representation of bumptree parameters. (a) shows how bumptree nodes are labelled (3 layers only are shown), and (b) shows the chromosome structure, with the parameters for each node represented by normalised real-valued genes. This example shows a tree built in two input dimensions, x and y.

The problem with coding structural parameters for the bumptree is that strong dependencies exist between the parameters. By definition 3.3 (section 3.2), a node function must always be contained within its parent's volume. Thus, the limits on a function's radii and centre co-ordinates are not known *a priori*, but depend on the centre and radii of its parent node. A GA operating with a naïve coding scheme, where each function's centre and radii are coded in a global co-ordinate system, will be unlikely to generate legal bumptree structures. Even given an initial population of legal bumptrees, recombination or mutation of chromosomes would inevitably produce children which violate definition 3.3. This situation is analogous to the production of 'lethal' networks in MLP experiments such as those described in section 2.4.3, and is clearly highly undesirable.

The genetic coding has been tailored to address this problem. To code the bumptree every gene is a single real value constrained to the interval [-1,1]. The process of translating a chromosome into a bumptree is represented diagrammatically in figure 3.5, and is as follows. First, it is necessary to define the volume in which

the bumptree is to be constructed, i.e. some region of the input space, as shown in figure 3.5(a). This volume may be conveniently defined as a hyper-ellipse, centred on the mean of the training data and large enough in each dimension to enclose all the data points in the training set. This volume defines a co-ordinate system in which the origin is at the volume's centre and, in each dimension of the space, a unit offset from the origin is equal to the volume's radius in that dimension – see figure 3.5(b). The genes which code the centre co-ordinates of each of the two top-level functions are expressed in this co-ordinate system. In each dimension, the value of the gene which codes a top-level function's radius is mapped onto the region between the function's centre and the perimeter of the volume within which the bumptree is contained. Having constructed the two top-level functions, each interior node function is constructed in a similar manner, a function's co-ordinate system being defined by the volume enclosed by its parent, as shown in figures 3.5(c) and 3.5(d).

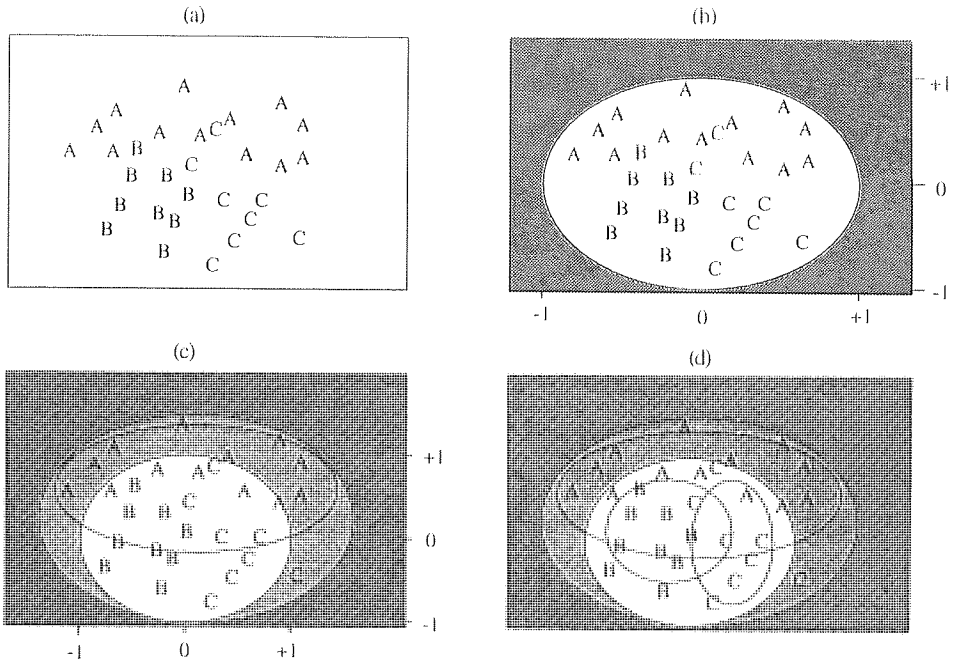



Figure 3.5: Translation of normalised chromosome into bumptree structure.

It can be shown that this genetic coding scheme has the desirable fundamental properties defined by Gruau (1992) of *completeness* and *closure*. The coding scheme also has certain properties which support meaningful mutation and recombination operators, which may be referred to as *continuity* and *isomorphism* respectively. Finally, the coding has the important quality of *low redundancy*.

- *Completeness.* Given the constraint that some maximum number of nodes is to be allowed in the tree, this coding scheme is clearly able to represent all possible bumptrees within a given volume of discrete space, so the representation is effectively complete.

- *Closure.* An ideal coding should be closed with respect to the space of possible phenotypes. For the bumptree, this means that every possible genetic string must map to a legal tree structure. Here, functions' centres and radii are coded as proportions of the volume enclosed by their parents, and this ensures that the integrity of the tree structure is inherent to the coding. It is impossible to represent an illegal, 'lethal' bumptree structure, so the representation is closed.

- *Continuity.* In a real-valued coding, 'creeping' mutation (Davis 1991) combined with selective pressure leads to genetic hill-climbing, fine-tuning solutions and improving the genetic stock. For this hill-climbing to occur there must be continuity in the fitness landscape, which is equivalent to saying that a small change in the genotype should cause a small change in the phenotype. In the present coding there is a generally smooth and continuous mapping between the space of chromosomes and the space of bumptrees. As the coding is hierarchical, a small change in the dimensions of one function in the tree will be transmitted down through its descendants, stretching or compressing them slightly and preserving structural relationships. Another important property is that all genes are real values mapped to the same interval, [-1,1], so no gene is any more sensitive to perturbation by a given amount than any other. This is an improvement on most binary codings, in which some bits are more significant than others, or real-valued codings which are not normalised. The coded search space has high continuity, and so creeping mutation is expected to be beneficial.

- *Isomorphism.* Crossover must be able to assort and recombine meaningful building blocks. It has already been argued (section 2.5) that any structure or modularity known to exist in the problem should be preserved in the genetic code, otherwise useful information is discarded and the GA's efficiency is reduced. For the bumptree, the minimum functionally relevant component is an individual node. On a larger scale, whole subtrees also represent meaningful functional units. As node parameters map to constant positions on the chromosome, it is simple to restrict crossover points to node boundaries. It is also simple to implement higher level crossover operators such as the subtree crossover described in section 3.3.3 below. Finally, recall that crossover in a real-valued coding only recombines values which exist in the population. In a binary coding, crossover typically splits binary words,

effectively causing random implicit mutation of parameters coded at the crossover boundaries. This coding has the desirable property of isomorphism, since it incorporates reasonable assumptions about regularities in the search space (that nodes and subtrees are relevant functional components) and allows the design of operators which perform meaningful recombination of bumptree building blocks.

- *Low redundancy.* Permutation problems arise if two identical bumptrees can have quite different genotypes, for instance if both have the same genetic information but coded in a different order on their chromosomes. Recombination then tends to duplicate some selective traits, and omit others (section 2.4.4). This affect has crippled efforts to apply GAs to optimising neural networks. The bumptree, however, is more strongly structured than the MLP or RBF networks, in that its internal nodes cannot be re-ordered arbitrarily without affecting its functionality. For an MLP (or RBF) with $n$ hidden nodes, the number of equivalent permutations (and hence the number of different chromosomes which map to identical networks) is $n!$. For the bumptree the situation is different. Each non-terminal node has two descendants which may be transposed without affecting the bumptree's functionality. The total number of ways of ordering the bumptree without affecting its functionality is the product of the number of permutations at each layer. For a complete tree with $n$ terminal nodes, this product is $2^{n-1}$, so there are $2^{n-1}$ different chromosomes for each possible bumptree. Clearly this is not an ideal one-to-one mapping, but it is a significant improvement on the situation with the MLP. Aside from the trivial result that $2^{n-1} \leq n!$ for all positive $n$, the amount of coding redundancy for the bumptree only scales geometrically with increasing network size, as opposed to the factorial growth with the MLP. Using an order-based coding, a bumptree with 8 units in the leaf layer can be represented by 128 different chromosomes. For an equivalent MLP, with 8 hidden units, the figure is 40320. For 10 units the figures rise to 512 for the bumptree, compared to more than 3.6 million for the MLP.

### 3.3.2 Selection

In accordance with the conclusions drawn in chapter 2, the GA uses a local subpopulation model for selection, reproduction and replacement, similar to those described by Davidor (1991) and Collins and Jefferson (1991). Associated with each chromosome in the population is a unique co-ordinate on a 2-dimensional toroidal lattice. The selection process is illustrated in figure 3.6. A random cell on the lattice is chosen, represented by the black square in each picture in figure 3.6. The first parent selected is the fittest chromosome encountered during a short random walk

across the lattice from this initial cell (five steps in this example). Another parent is selected similarly during a second random walk of the same length from the original starting point. These two parents are then recombined (section 3.3.3 below) to produce a new child. The new child is replaced into the population, overwriting the least-fit chromosome encountered on a third random walk from the original cell. This whole process is repeated with new random starting points until a certain proportion of the population has been replaced, after which time all new chromosomes are evaluated, and this completes one generation of the GA.

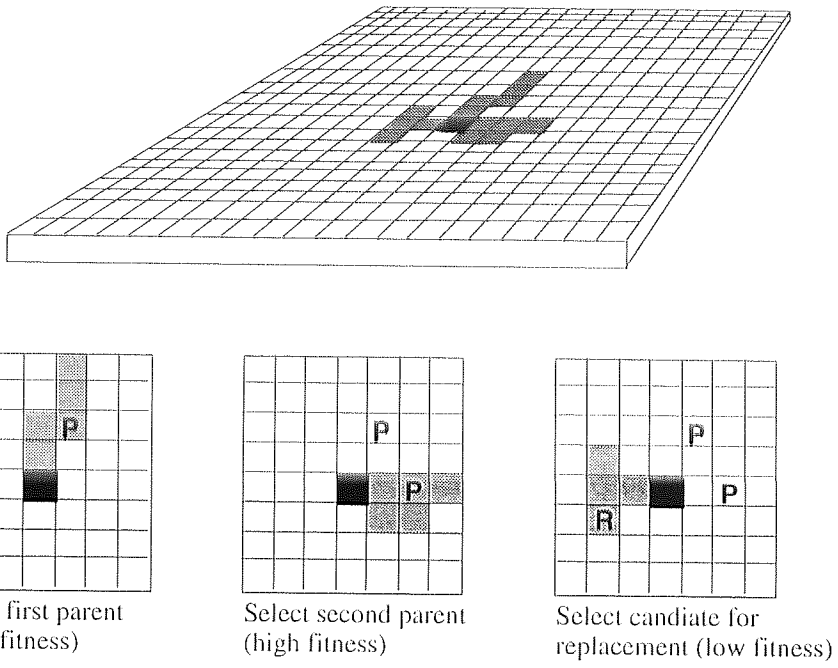



Figure 3.6: Local selection and replacement.

Several advantages associated with this selection strategy have already been outlined (chapter 2). The first is a tendency to preserve genetic variety in the population; the spread of genetic material is limited by the geography of the lattice, and this helps to prevent the whole population from converging prematurely on a local optimum, encouraging local speciation instead. In the following chapter a detailed examination of the convergence and speciation properties of the GA is presented, and it is shown that the local selection scheme does appear to slow convergence and encourage speciation for these problems. Another advantage, common to other successful alternatives to roulette-wheel selection such as rank-based (Whitley 1989) and tournament selection (Brindle and Cox 1991), is that there is an implicit rescaling of the fitness function over time. As chromosomes are always measured against their near neighbours, fitness is a relative, rather than an absolute measure, so selection pressure does not wane as the population converges to a set of solutions with similar

absolute performance. In addition, the local selection procedure is a steady-state or generation gap model (section 2.2.4) and so incorporates elitism; in each generation, only a small proportion of the population is replaced so the best performing strings are guaranteed survival into the next generation. Finally, this selection strategy is less computationally expensive than most alternatives and more amenable to implementation on parallel hardware, as it does not rely on global comparisons or sorting of the whole population.

### 3.3.3 Crossover

In the bumptree chromosome, a single floating-point gene does not represent a meaningful building block for crossover. A node's activation function is defined by the interaction of a number of genes and a single gene coding, for example, a centre co-ordinate in a particular dimension, does not represent a useful phenotypic attribute when taken in isolation. A simple one- or multi-point crossover operator applied to the bumptree chromosome will tend to mix up the genes coding particular functions, swapping pieces of chromosome that do not represent useful partial solutions. A minimum practical crossover operator might be restricted to only crossing over at node boundaries, thus only exchanging whole functions. In this work a higher-level *subtree* crossover is used, which exchanges genetic material corresponding to complete subtrees between parents. A similar subtree crossover is used extensively in the emerging genetic programming paradigm spearheaded by Koza (1992).

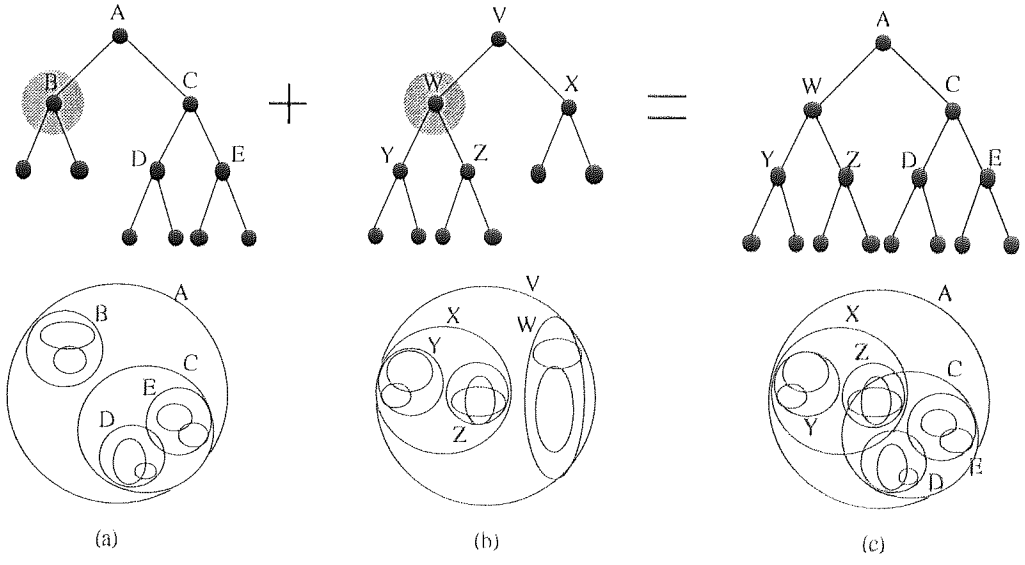



Figure 3.7: The subtree crossover operator exchanges equivalent subtrees between two parent bumptrees (a) and (b) to produce the child (c).

The subtree crossover is shown diagrammatically in figure 3.7, and works as follows. Initially, the child chromosome is created by duplicating the first parent. One of the 30 nodes is then chosen at random as the crossover site, and the block of genes representing that node is copied from the second parent into the child, replacing the genes that were inherited from the first parent for that node. The blocks of genes representing the node's two descendants, their descendants and so on are also copied from the second parent, until the complete subtree from the crossover site down has been copied over. Note that the crossover includes non-expressed genes – if node 10 is selected as the crossover site, the blocks coding nodes 21 and 22 (its descendants) will also be copied regardless of the value of the gene which specifies whether node 10 is subdivided in the phenotype.

### 3.3.4 Mutation

As the GA uses a real-valued coding, it is appropriate to use a 'creeping' mutation operator (Davis 1991). The mutation operator is applied to every new chromosome produced, and works by slightly perturbing the values of 10% of the genes, sampled with uniform probability along the length of the chromosome. A mutated gene is modified by a random quantity uniformly distributed in the interval [-0.2,0.2], with clipping if necessary to keep its value within the normalised range [-1,1]

### 3.3.5 Results

Genetically-optimised bumptree networks were tested against orthodox bumptrees, built using a top-down constructive algorithm, on three problems. The first is the well-known "Iris" problem. The task is to classify an iris flower into one of three possible varieties, according to four descriptive parameters such as stamen length and the like. The data consists of a training set of 75 examples and a generalisation test set, also of 75 examples. Iris was chosen primarily because it is a useful problem on which to test code during development: the data set is small, so results can be obtained relatively quickly; it is a well known neural network benchmark, so plenty of results exist to indicate what performance might be expected; and it is sufficiently simple that it is difficult to get wrong – poor performance on Iris suggests either a bug in the code or a fundamental flaw in the algorithm.

The second problem studied, Parity-6, is the six-bit parity mapping, i.e. a six-dimensional version of the binary XOR function. This problem was chosen because it appeared to be extremely difficult for the orthodox bumptree, presumably because classes are highly inseparable. There is no generalisation set for Parity-6: the problem is simply to learn the mapping as completely as possible.

The final test problem is the recognition of spoken vowels, and is based on the data described in (Chang and Lippmann 1991; Ng and Lippmann 1991). For this task there are 10 classes, each representing a different vowel sound. Data is gathered from speakers uttering words starting with 'h' and ending with 'd', with different vowels in between ("head", "hid", "hod", "had", "hawed", "heard", "heed", "hud", "who'd" and "hood"). Speech input is pre-processed to extract the first two formant frequencies of the vowel, determined by spectrographic analysis (Chang and Lippmann 1991; Ng and Lippmann 1991). Each utterance is, therefore, represented by a point in a 2 dimensional input space, and the problem is to separate these points into the 10 appropriate vowel sounds. The data for this problem is shown in figure 3.8. This problem was chosen because it is a difficult 'real world' learning task: the data is noisy and many classes overlap significantly. In order to compare results, the division of the data into training and test sets is the same as that used by Bostock (Bostock 1994). The data is split into a training set and a test set, each containing 320 example patterns, and the two inputs are normalised. In all these experiments the fitness function for the GA is the inverse of the mean square error (MSE) across the bumptree's output units, across all examples in the training set.

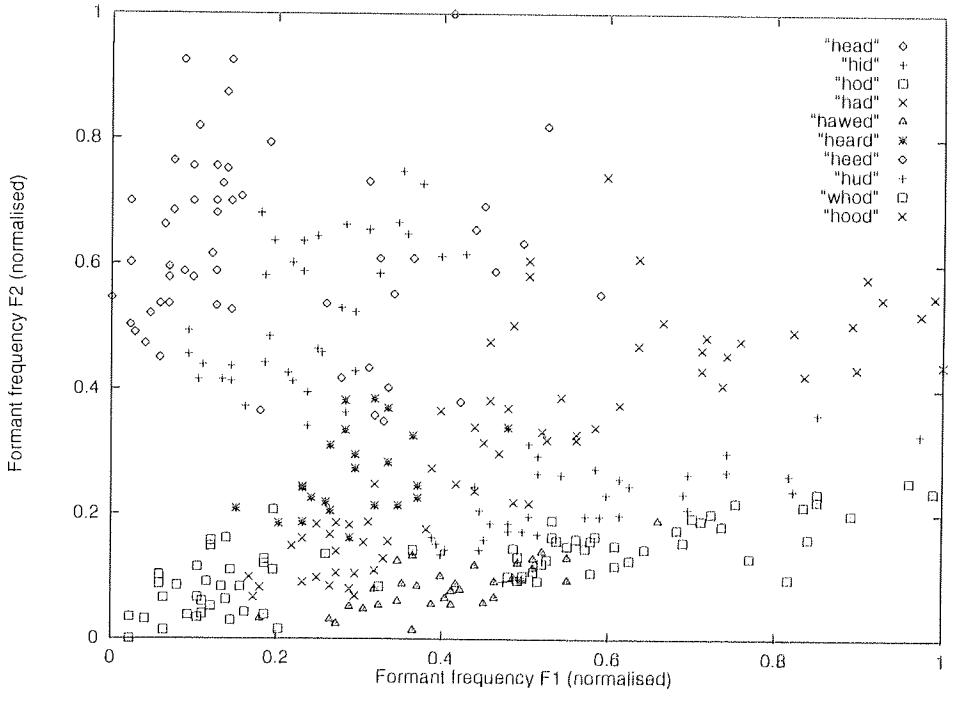



Figure 3.8: The vowel recognition data (training set only shown).

The choice of other experimental parameters was based on preliminary experiments. A population size of 400 was used, mapped onto a 20x20 toroidal lattice. Candidates for selection and replacement were the fittest and least-fit, respectively, encountered during random walks of 5 cells from a randomly selected

initial point on the lattice. Experiments were run for a maximum of 20000 trials, i.e. 500 generations, and all results are averaged over 10 runs.

| | Iris | | |
|---|---|---|---|
| | Training % | Test % | # nodes |
| BT | 95.5 (3.2) | 92.1 (3.4) | 15.2 (7.9) |
| GA | 99.7 (0.5) | 96.3 (0.8) | 9.0 (4.9) |

| | Parity-6 | |
|---|---|---|
| | Training % | # nodes |
| BT | 56.9 (6.0) | 9.4 (6.7) |
| GA | 77.2 (8.4) | 9.0 (3.6) |

| | Vowel recognition | | |
|---|---|---|---|
| | Training % | Test % | # nodes |
| BT | 72.0 (7.1) | 65.6 (6.1) | 52.6 (40.2) |
| GA | 78.4 (1.5) | 75.0 (2.4) | 18.4 (2.9) |

Table 3.1: Comparison of results for orthodox bumptree (BT) and GA-bumptree (GA) for IRIS, parity-6 and vowel recognition data sets. Figures in brackets are standard deviations.

Table 3.1 compares the performance of bumptrees evolved by the GA against that of bumptrees built using the orthodox top-down constructive algorithm. Figures show the percentage of patterns correctly classified after training was terminated. For both the top-down algorithm and the genetic algorithm the same metric was used to terminate runs when generalisation ability peaked, to prevent over-fitting.

These results indicate that the GA is able to discover tree topologies which give significantly increased classification performance on these problems when compared to the trees produced by top-down construction. It is also interesting to note that, with the exception of the Parity-6 problem, trees produced by the GA are significantly smaller than those produced by the orthodox bumptree algorithm.

### 3.3.6 Evaluating the GA

Although the above results are encouraging, two questions immediately present themselves. The first is a question which should be asked routinely of any directed search algorithm, and is especially relevant to one which requires as many evaluations

as the GA: could solutions of similar or higher quality have been attained by sampling the same number of points in a blind random search? If the answer turns out to be yes, then the algorithm is either so inefficient as to be worthless, or is being misled by assumptions about the search space which are not justified. In GA terms, if the coding and operators have introduced deception (see section 2.2.3) it is quite possible for the GA to become trapped sampling a region of the search space far from the optimum, and random search would eventually find better solutions.

The second question is more specific: is the local selection and replacement scheme of any benefit in this problem, or would a more simple GA have performed just as well?

These questions were answered by further experiments which compared the performance of the local selection GA against three alternatives: a purely random search, a simple generational roulette-wheel GA, and a generation-gap roulette-wheel GA in which only the worst 10% of the population was replaced in each generation, the same proportion as was replaced in each generation of the local selection GA. These results, averaged over 10 runs on the vowel recognition data, are shown in figure 3.9.

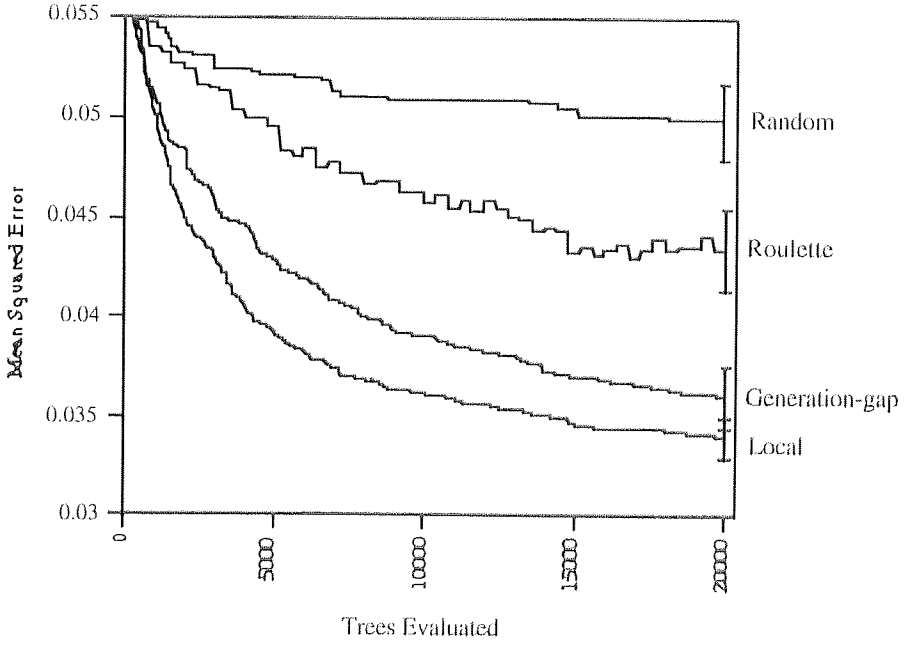



Figure 3.9: Learning performance of bumptrees found by GAs with various selection strategies, compared with that of trees found by random search. Curves shows the quality of solutions found by each algorithm plotted against the number of trees evaluated. Error bars show one standard deviation.

The first thing which is clear from the figure is that all three GAs comfortably out-perform random search, as would be hoped. This is a vindication of the

suitability of the hierarchical coding scheme and the subtree crossover and creeping mutation operators for this problem, and implies that they have not introduced any unexpected deceptive effects. A second conclusion is that the two generation-gap GAs significantly out-perform the generational GA on this task: with no fitness scaling or elitism, the generational roulette-wheel GA appears to be losing good solutions which the other two GAs are able to maintain in the population. Finally, the fact that the local subpopulation GA was consistently better than the others is encouraging, and suggests that its choice was well motivated. Although the generation-gap GA with roulette-wheel selection occasionally finds solutions of comparable quality, the local GA is more consistent, and its improvement over the generation-gap GA is significant throughout the search.

### 3.3.7 Summary

The preceding sections have described experiments which establish the feasibility of applying a GA to optimising the architecture of the bumptree. On the three classification problems investigated, the GA is consistently able to discover bumptrees which are smaller than, and which give a significant improvement in classification performance over, trees built using the orthodox top-down construction algorithm. The experimental results also support the decision to use a local selection and replacement strategy, originally based on the review of the GA literature reported in chapter 2.

Because of the bumptree's structure, the genetic coding described in section 3.3.1 introduces significantly less redundancy into the search space than is typical with GA-NN hybrids, and has better scaling properties. This makes the GA-bumptree considerably less susceptible to the serious permutation problem discussed in section 2.4.4. As well as this *low redundancy*, the coding scheme also has the desirable properties of *closure*, *completeness*, *continuity* and *isomorphism* with respect to the problem space.

## 3.4 Evolving topology and weights

Although the results thus far have been encouraging, the GA-bumptree hybrid still relies on the original training algorithm to optimise the weights of the local experts. This is fine for the classification problems described in the last section, but it rules out applying the bumptree to many other classes of problem – control problems, for instance, where there is no well defined training signal for every input vector. It also clashes with one of the original motivations for taking an evolutionary approach to network design: the freedom to define arbitrary optimisation criteria. So far, no

matter what fitness function is used to tune the tree architecture, the expert weights are still optimised to minimise training set MSE[2].

If the full potential of an evolutionary approach to network design is to be realised, it is necessary to include every parameter of the model in the genetic code. This gives the GA free rein to globally search the space of complete bumptrees for models which satisfy any criteria or constraints which are appropriate. As a consequence, it opens the door to applications which are beyond the scope of the original bumptree training algorithm. The remainder of this chapter is devoted to extending the GA to optimise both the topology of the tree and the experts' weights simultaneously.

### 3.4.1 Coding structure and weights: first attempt

There are various ways in which the genetic code described in section 3.3 could be extended to incorporate the experts' weights as well as the structure of the tree. As usual with GAs, finding the best coding is not trivial.

One thing which is clear is that the experts cannot be optimised independently. The contribution to fitness made by a given expert is not just a function of its weights; it depends on the region of the input space in which it operates, which is determined by the structure of the tree above it. Experts 'belong' to particular leaf nodes. When leaves are exchanged by crossover, their experts must be carried with them.

An added complication is the fact that it is not known beforehand which nodes in the tree are going to be the leaves, or how many leaves (and therefore experts) there are going to be. Because each node has a gene which determines whether or not it is split, the GA is free to switch different subtrees on and off during the search. A particular node in one tree in the population may be a leaf, with an associated expert, while its equivalent in another tree may be have a subtree below it, modelling the same region with several experts.

One possible coding scheme which deals with these consideration is to make a weight matrix an integral part of every node's genetic definition. Any node which turns out to be a leaf then expresses its weight genes to define its associated expert. In nodes which are not leaves, the weight genes are simply ignored. This scheme was the basis of a first attempt at evolving structure and weights simultaneously.

In this coding, each of the blocks of parameters in the genetic coding (see figure 3.3) is extended to incorporate a complete weight matrix for a local expert network. This significantly increases the size of the search space when compared to

[2]One class of problems where MSE is not an appropriate error measure is finding useful models of systems where the mapping between inputs and outputs is one-to-many. Some examples of these problems are examined in detail in chapter 6.

optimising structure alone. Table 3.2 compares the length of the genetic codes required by each coding scheme for the three classification problems already described.

In keeping with the rest of the genetic code, weights are coded as normalised real values in the range [-1,1]. Translating a particular gene into a weight value is simply a matter of multiplying it by a scalar which determines the range of the weights. In these experiments this range was simply [-1,1] so the genes represent weight values exactly.

| Problem | Architecture only | Architecture and weights |
|---|---|---|
| Iris | 270 | 720 |
| Parity-6 | 390 | 600 |
| Vowel recognition | 150 | 1050 |

Table 3.2: Chromosome length (number of real-valued genes) required to code architecture only vs. that for coding architecture and weights.

### 3.4.2 Results

The classification performance of trees discovered by a GA using this coding is shown in table 3.3. These results are not encouraging. For the more difficult problems (Parity-6 and Vowel), the GA appears to quickly converge to a very poor local optimum, and makes little progress thereafter.

| Problem | Training % | Test % | # nodes |
|---|---|---|---|
| Iris | 99.1 (1.1) | 95.7 (1.3) | 10.2 (5.3) |
| Parity-6 | 47.3 (8.1) | n/a | 7.8 (4.7) |
| Vowel | 40.3 (9.3) | 40.4 (8.4) | 7.8 (5.0) |

Table 3.3: Classification performance of trees found by the GA when both architecture and weights are optimised genetically. Results are averaged over 10 runs, and figures in parentheses are standard deviations.

### 3.4.3 Coding structure and weights: second attempt

The poor results shown in table 3.3 point to serious problems with this first attempt at coding expert weights. One fact which stands out is the large increase in chromosome length which has resulted from adding the weights (table 3.2). Even for the simple Iris problem, adding the weight matrices has given the GA an extra 450

real parameters to optimise. For the vowel recognition problem, adding the weights has increased the dimensionality of the search space by a factor of 7.

Coding a complete weight matrix for every node is extremely inefficient. For trees of up to 30 nodes, 30 complete weight matrices are coded on the chromosome because any node has the potential to be a leaf node. However, the maximum possible number of leaves in a binary tree of 30 nodes is only 16. Even if every node is present in the final tree, 14 out of the 30 weight matrices are redundant, genetic deadwood. The dilemma is that it is not known beforehand which nodes are going to end up as the leaves, and therefore which ones should have weight matrices coded for them.

This problem can be alleviated by removing one degree of freedom from the GA. In this second attempt to optimise architecture and weights simultaneously, every node is required to be present in the final tree. The GA now searches the space of complete trees of 4 layers, so every model has the maximum 16 local experts. Preventing the GA from 'switching off' subtrees removes the uncertainty about which nodes are leaves and which are not. Since the leaf layer now always consists of nodes 15 to 30, only these nodes need to have weight matrices associated with them. Experiments suggest that restricting the search to the space of complete trees does not adversely affect the quality of the solutions found by the GA. The final trees do not have to 'use' all 16 experts because the GA can position the node functions so that some branches of the tree will never be active on the data. This effect is seen in the results reported in chapter 6, where the GA finds trees to model simple functions which use only a small number of experts. In this second coding scheme the 16 weight matrices are appended to the end of the original chromosome. The subtree crossover is modified to ensure that whenever a leaf node is crossed over, its weight matrix is crossed with it.

At the time of this work, Bostock (1994) reported the result that classification performance is improved if the constraints on the structure of the tree are relaxed. Empirical results seem to favour simplified trees in which node functions are allowed to span any region of the input space, not just the regions spanned by their immediate predecessors in the tree. Bostock also reported another simplification which he found to improve performance on some problems: using a constant radius for all node functions (Bostock 1994). Both of these modifications to the model were incorporated into the coding scheme at this point.

Table 3.4 shows the effect of the modifications described in this section on the chromosome length for each of the three classification problems.

| | | Architecture and weights | |
|---|---|---|---|
| Problem | Architecture only | First attempt | Second attempt |
| Iris | 270 | 720 | 360 |
| Parity-6 | 390 | 600 | 292 |
| Vowel recognition | 150 | 950 | 540 |

Table 3.4: Chromosome lengths for two methods of coding architecture and weights compared to those for the original architecture-only code.

### 3.4.4 Results

The performance of trees produced by the GA using the second coding scheme for architecture and weights is shown in table 3.5. Note that all trees have 30 nodes, since the coding scheme can no longer represent incomplete trees. Performance on the Iris and Parity-6 problems is good, but the performance on the Vowel data is still fairly poor (compare with the results in table 3.1).

| Problem | Training % | Test % | # nodes |
|---|---|---|---|
| Iris | 100.0 (0.0) | 95.9 (1.4) | 30 |
| Parity-6 | 90.3 (3.4) | n/a | 30 |
| Vowel | 55.7 (6.7) | 56.2 (5.1) | 30 |

Table 3.5: Classification performance of trees found by the GA using second coding for architecture and weights. Results are averaged over 10 runs and figures in parentheses are standard deviations.

### 3.4.5 Summary

This section has extended the GA-bumptree model to make the GA responsible for optimising both tree structure and expert weights. By replacing the original error minimisation algorithm for the weights with genetic search, the possible range of applications for the GA-bumptee is broadened to include problems such as the artificial life simulation described in chapter 5 and the control and robotics problems considered in chapter 6.

With the new coding scheme the results for vowel classification are poor, both when compared to other authors' published results on the same data (Chang and Lippmann 1991) and when compared to the results for the original GA-bumptree (table 3.1). This illustrates the limitations of using a weak stochastic optimisation

method to tune the expert weights: 'stronger' optimisation methods such as the error minimisation algorithm described in section 3.2.2 are obviously a better choice whenever the error signal is sufficiently informative for them to be applied.

## 3.5 Conclusions

This chapter has introduced the bumptree, a neural network model which appears to be much more suitable for genetic optimisation than the MLP. From the GA point of view, the principal attractions of the bumptree are its speed, and the fact that it is possible to code the bumptree as an ordered genetic string without introducing the same degree of redundancy as is typical with GA-MLP hybrids.

Three variations of the GA-bumptree have been tested on three classification problems. In the first GA-bumptree, the GA was responsible for optimising tree topology only. For the test problems, the GA was able to find trees which gave a significant increase in classification performance over those generated by a top-down constructive algorithm.

When the approach was extended so that the GA was responsible for optimising both topology and weights, classification performance on the difficult vowel problem declined considerably. This illustrates the limitations of a weak global search algorithm such as the GA when compared to an analytical solution, and suggests that trying to fine-tune the expert weights genetically is unlikely to be worthwhile for classification problems. Instead, the advantage of optimising architecture and weights together with the GA is that it opens the door to domains, such as control and robotics, where the error signal is not sufficiently informative for the normal error minimisation algorithm (or other network training algorithms such as backpropagation) to be used. Chapter 6 investigates whether the GA-bumptree can offer useful performance on these kinds of problems.

Finally, the experimental results in this chapter appear to vindicate the use of a geographic selection and replacement model for the GA. This geographic model is the subject of the following chapter, which takes a closer look at the dynamics of the GA, and at how the various control parameters affect the nature of the search.

# Chapter 4

# Evolutionary dynamics of the GA-bumptree

# 4.1 Introduction

This chapter summarises the results of an extensive set of experiments which were undertaken with the GA-bumptree hybrid. The purpose of these experiments was not to find better classifiers, but to investigate certain aspects of the GA, particularly the local selection scheme. Specifically, the experiments were intended to answer the following five questions:

1. To what extent does the local selection scheme help to maintain genetic diversity?

2. In particular, how does the choice of neighbourhood size (i.e. the length of the random walks taken across the lattice during selection and replacement) affect the GA's rate of convergence?

3. Is 'speciation' actually happening, and if so, to what extent?[1]

4. How does the neighbourhood size affect the degree of speciation?

5. How sensitive is the GA to small changes in the mutation rate and generation gap parameters?

The first two questions are addressed in section 4.3. The *convergence profile* is introduced as a general tool for analysing convergence rates in real-valued GAs, and the relationships between neighbourhood size, convergence rate and performance are examined for the GA-bumptree. Section 4.3 goes on to investigate speciation in the GA-bumptree, addressing the third and fourth of the above questions. The final question is answered in section 4.2 below.

# 4.2 The experiments

There are many parameters which might be expected to affect the dynamics of the GA described in the previous chapter. Examples include the choice of mutation rate,

[1]It has already been noted, in section 2.2.2, that the GA community has a fondness for hijacking biological terminology wherever is convenient. In the discussion in this chapter, the use of terms such as 'species' and 'speciation' is consistent with their use in the GA literature, which is not necessarily the same as their use among biologists. Here, the term 'species' refers simply to a subset of genetically similar strings in the population, defined by some arbitrary similarity metric such as that introduced in section 4.3.4. The term 'speciation' refers to the formation of many such subsets within the same population.

crossover rate, generation gap, population size and the local neighbourhood size, as well as more general choices such as the representations and operators used. Unfortunately, the computing overheads involved when working with GAs rule out any attempt to examine the effects of varying all these parameters. For the GA-bumptree, a single run of 20000 tree evaluations on the Iris data, distributed on a network of workstations, takes between two and four hours depending on machine availability, cpu and network loading, and so on.

Because of the time involved, it was deemed necessary to focus these experiments on a subset of just three parameters, each of which was to be varied over three different values, giving 27 different combinations in total. Even this modest set of experiments is extremely computationally intensive. If 10 runs are performed for each of the 27 experiments, something of a bare minimum in order to get statistically meaningful results, this gives 270 runs in total. If every run were to take the minimum of 2 hours, and experiments were run back-to-back with no breaks in between, this would give 540 hours, or more than 3 weeks, of solid computing. In the real world of high daytime machine loads, network congestion, operating system upgrades and hardware failures, these experiments actually took over two months to complete.

### 4.2.1 The experimental parameters

Since the main purpose of these experiments was to investigate the local selection strategy, the three parameters which were examined were:

- *Walk length.* This parameter determines how many cells on the lattice are traversed in each random walk during selection and replacement (see section 3.3.2). The setting of this parameter is expected to be fundamental to the rate of convergence and the degree of speciation. If this parameter is small, selection and replacement always occurs among a few strings sampled from a small local neighbourhood. This means that population-wide selection pressure is low: even the best strings in the population will only get the opportunity to reproduce if the selection routine happens to visit their cells, and the worst strings in the population may be similarly overlooked for replacement. The ability of good genetic material to spread through the population is also lessened if this parameter is small, since children only ever replace strings in a small local region around their parents.

If the walk length is very large relative to the size of the lattice, the effects are reversed. Each random walk now visits a large proportion of the cells, so selection pressure is very high, much more so than with a conventional roulette-wheel GA: there is a high probability that one of a few good strings in the population will be

found each time. Similarly, poorer strings will be mercilessly found and replaced. It is expected, therefore, that very large values of this parameter would lead to rapid premature convergence.

Two extreme values and one intermediate value were chosen for this parameter. At one extreme, a walk length of 5 was chosen to minimise the spread of genetic material across the lattice. Because the walks are random, a particular walk may traverse the same cell more than once. Random walks of 5 steps visit an average of only 4 unique cells, limiting selection to a tournament between 4 strings, or 1% of the population. It was felt that reducing the walk length, and hence the tournament size, any further would introduce an unacceptable degree of noise into the selection procedure. A value of 30 was chosen as the other extreme. A random walk of length 30 can reach any point on the lattice from any starting point, allowing genetic material to spread across the lattice unchecked. Walks of 30 steps visit around 18 unique cells on average so tournaments occur among about 4.5% of the population, which leads to increased selection pressure. Figure 4.1 shows the relative probability of visiting each cell on the lattice associated with random walks of lengths 5 and 30 from the same starting point. The third value tested for this parameter was 14. Walks of length 14 visit around 10 unique cells. This value was intended to slow the migration of genetic material across the lattice, without ruling out the possibility of choosing any cell from any starting point.

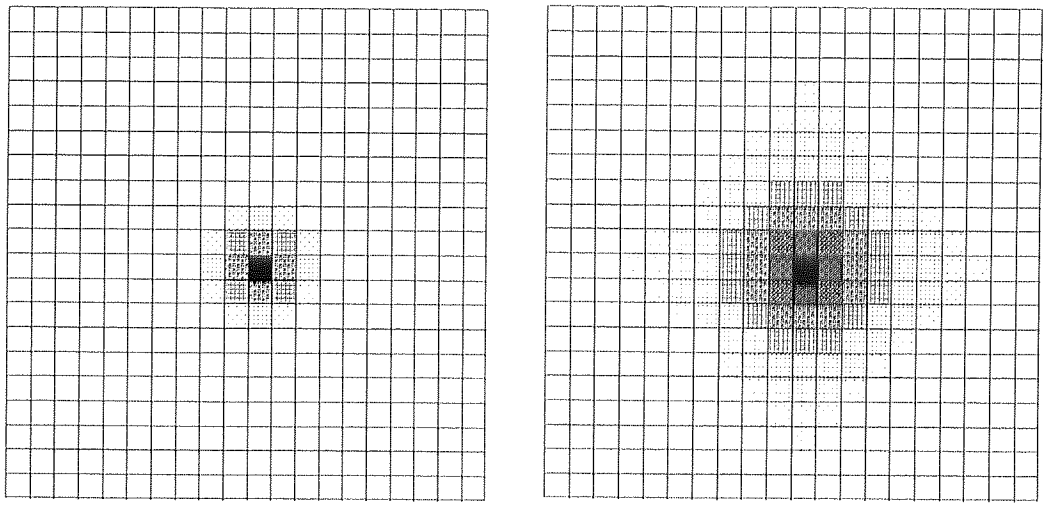



Figure 4.1: Probability of visiting each cell on the lattice during a random walk of (a) 5 cells and (b) 30 cells from the same initial starting point.

- *Generation gap.* This is the proportion of the population which is replaced in each generation. A value of 100% for this parameter indicates a purely generational GA in which the whole population is replaced at the end of each generation. Decreasing the parameter from 100% introduces increasing degrees of elitism

(automatic preservation of the fitter strings from one generation to the next) until, at the opposite extreme, only a single string is replaced each generation, giving a pure steady-state GA.

Like walk length, this parameter was chosen because it seems intimately related to the GA's convergence rate. In a generational GA, there needs to be sufficient selection pressure to ensure that the current best solutions reproduce before they are overwritten at the end of each generation. If the population is too diverse, recombination may disrupt the best strings, in which case the information which they contained is lost. As the generation gap decreases, the better strings are protected, and can only be overwritten by strings which are better still. The GA can afford to maintain a more diverse population and perform a more explorative search, because there is no danger of replacing good parents with poorer offspring. Since the whole point of local selection is to maintain diversity, a small generation gap should be preferred. This is borne out by the results shown previously in figure 3.9. For these experiments values of 5%, 10% and 20% were chosen for the generation gap.

- *Mutation rate.* The final parameter chosen was the mutation rate, the percentage of genes in each new child which were subject to mutation. Unlike the previous two parameters, mutation rate is not related to selection and replacement. However, it does clearly affect the level of genetic diversity maintained in the population. A high mutation rate will lead to high diversity in the population, but it will also be disruptive, tending to reduce the GA to pure random search. Too little mutation, on the other hand, is likely to result in the search stagnating. The mutation rate must be sufficient to re-introduce useful genetic material which has been lost, and to facilitate some degree of stochastic hill-climbing. For these experiments values of 2%, 5% and 15% were chosen for the mutation rate.

| Parameter | Values | | |
|---|---|---|---|
| Walk length | 5 | 14 | 30 |
| Generation gap | 5% | 10% | 20% |
| Mutation rate | 2% | 5% | 15% |

Table 4.1: Summary of variables for the experiments.

The three variable parameters and their values are summarised in table 4.1. All other experimental parameters were fixed for all 27 sets of runs, as follows. A population size of 400 was used, mapped onto a 20x20 lattice. All runs were terminated after 20000 evaluations. With a population size of 400, this is equivalent

to either 1000, 500 or 250 generations for generation gaps of 5%, 10% or 20% respectively.

In all these experiments the GA optimises both architecture and weights. The coding scheme used is the first of the two schemes for architecture and weights described in the previous chapter. The reason for this is simple: this set of experiments was started some time before the second coding scheme was developed.

Because the focus of these experiments was to investigate the dynamics of the local selection scheme, the fitness function was considered to be arbitrary, so long as the same measure was used across all experiments. The Iris data was chosen as the test problem, for two reasons. First, the initial experiments (chapter 3) had indicated that GA could converge reliably on this data. This is an important consideration because the GA's performance needs to be sufficiently consistent that meaningful averages can be obtained in only 10 runs. From the point of view of examining the nature and rate of convergence it was considered better to choose a problem for which the GA was known to converge reliably to a good solution for at least some parameter settings, rather than a problem on which the GA was known to be unreliable. Secondly, the Iris data set is small, with just 75 patterns in the training set. Since the experiments were to involve 270 separate runs, run time was a prime consideration. This excluded the use of the vowel recognition data, for instance, which has more than four times the number of data points and typically involves considerably longer runs.

### 4.2.2 Results: final fitness scores

The final fitness scores produced by the GAs in these experiments are summarised in table 4.2. The first thing which is clear from the table is that the level of fitness achieved by the GA does not seem to be particularly sensitive to the changes made to the chosen parameters. This is not to say that the parameters do not affect the GA – in the following sections it is shown that there are significant differences between the rate at which the various GAs converge, and even the manner in which they search. Instead, it is necessary to conclude that the Iris problem is too simple to reveal any variation in the searching power of the various GAs: despite their differences, all 27 GAs have discovered solutions of similar quality.

Although some particular combinations of parameters stand out – the combination of a walk length of 14, 5% mutation and a 5% generation gap appears to be especially bad, for example – it is not immediately obvious if any general trends exist in these results. To investigate this, it is useful to examine the contribution of each parameter in isolation. Each graph in figure 4.2 shows the effect on performance of varying one of the three parameters. In each graph, a point is an

average taken over all the runs for which that particular parameter was set to a particular value.

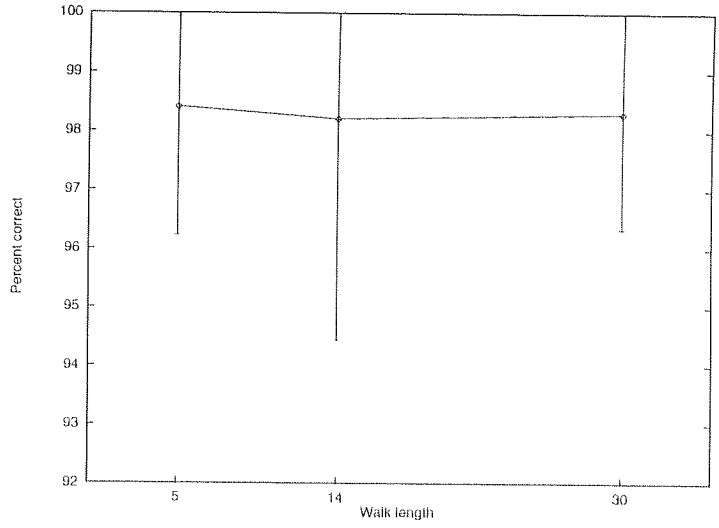
| Walk length | Mutation rate (%) | Generation gap (%) | | |
|---|---|---|---|---|
| | | 5 | 10 | 20 |
| 5 | 2 | 97.3 (3.3) | 98.4 (2.1) | 97.5 (2.5) |
| | 5 | 98.0 (2.1) | 98.0 (2.1) | 98.7 (1.3) |
| | 15 | 99.6 (0.9) | 99.7 (0.5) | 98.5 (2.0) |
| 14 | 2 | 98.5 (0.9) | 98.4 (1.2) | 96.9 (2.8) |
| | 5 | 95.2 (9.6) | 99.2 (0.9) | 98.4 (2.0) |
| | 15 | 99.6 (0.9) | 99.6 (0.6) | 98.0 (1.8) |
| 30 | 2 | 97.6 (1.6) | 97.6 (2.0) | 98.4 (1.2) |
| | 5 | 98.3 (1.8) | 97.2 (2.7) | 98.8 (1.3) |
| | 15 | 98.7 (2.1) | 98.5 (2.4) | 99.6 (0.6) |

Table 4.2: Summary of final fitness scores for all 27 combinations of parameter values. Each result is the percentage of patterns correctly classified by the best bumptree found by the GA, averaged over 10 runs. Figures in brackets are standard deviations.

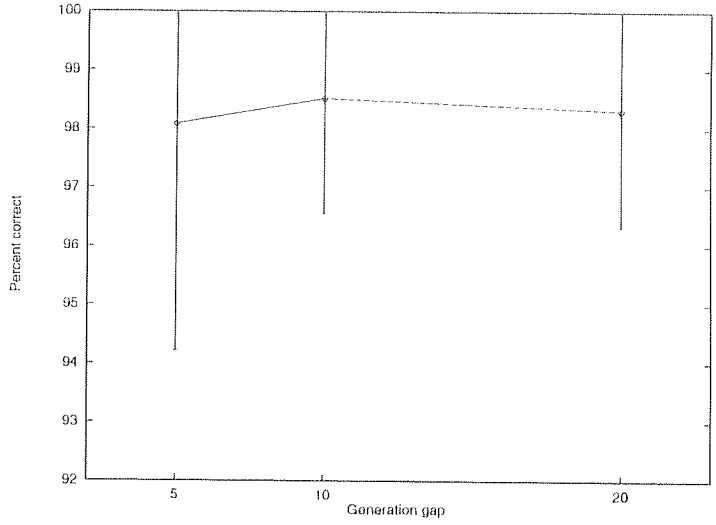
It is clear from figures 4.2(a) and 4.2(b) that no general conclusion can be drawn from these results as to what is the best overall setting for either walk length or generation gap, since there are no statistically significant differences between the average scores associated with particular values for these parameters. This invariance can be attributed, at least in part, to the simplicity of the Iris problem. One other possible conclusion is that these parameters are related and cannot be tuned independently. Given that both parameters are expected to affect the same properties of the GA – the rate of convergence and the selection pressure – this seems likely to be the case.

The results shown in figure 4.2(c) are a little more conclusive. There is a small but statistically significant[2] increase in the average performance of runs where the mutation rate was 15% when compared to those where it was either 2% or 5%. Although these results do not suggest an upper bound on the best setting for mutation rate, it is possible to conclude that rates of 5% or below are too low for this problem.

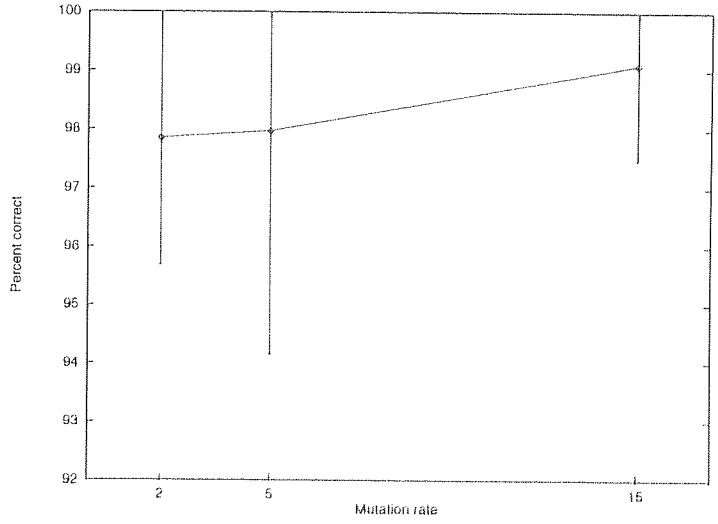
[2]Two-tailed test, 5% confidence limit.




(a)




(b)




(c)

Figure 4.2: Effect of (a) generation gap, (b) walk length and (c) mutation rate on final performance, averaged over all runs. Error bars show one standard deviation.

## 4.3 Genetic convergence

The main purpose of these experiments was not to spend two months producing the best classifier for the simple Iris problem, but to learn more about the dynamics of the local selection and replacement strategy. A central aim of this study was to examine to what extent the convergence rate of the GA could be controlled by tuning the chosen parameters, in particular the walk length.

### 4.3.1 Measuring convergence

In a given generation it is possible to measure the amount of variation in a particular gene from the frequency of its different alleles in the population. In a binary GA there are only two alleles, 0 and 1, so a useful measure of variation is simply the ratio of 0s to 1s. If the frequencies are equal, the gene in question has maximum diversity. If all members of the population have the same allele at that position, the gene has completely converged. This kind of measure was used by Tanese (1989) to investigate the effect of using local subpopulations in a binary GA.

Increasing the cardinality of the representation (i.e. the number of possible alleles for each gene) complicates the issue, since different genes may converge to different subsets of the possible alleles. The extreme case is a real-valued GA, where a gene can take any value from a continuous range. This makes the number of different alleles for each gene theoretically infinite. One way to estimate the degree to which a particular real-valued gene has converged is to measure the total range of values which exist for that gene in the population. The problem with this is that a gene whose values are distributed between two small but distant clusters will appear more diverse than one which has a much broader spread of contiguous values, as illustrated in figure 4.3(a).

A better measure of the degree to which a real-valued gene has converged can be obtained by dividing its maximum possible range of values into a number of discrete bins, and counting the number of bins for which representative values exist in the population. This measure is illustrated in figure 4.3(b).

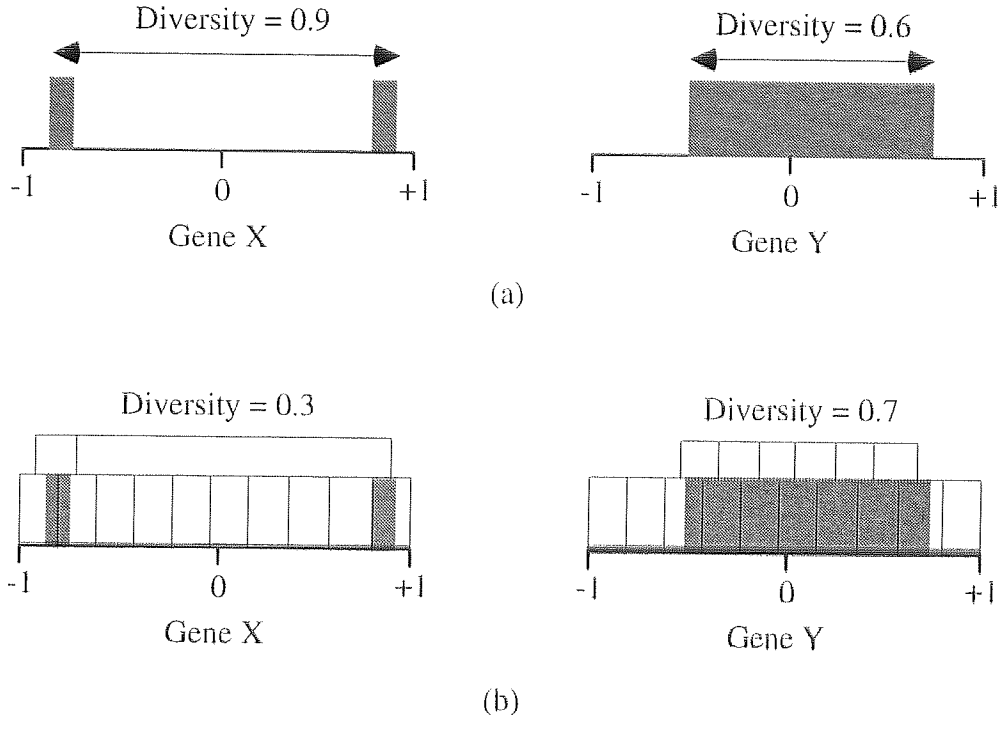


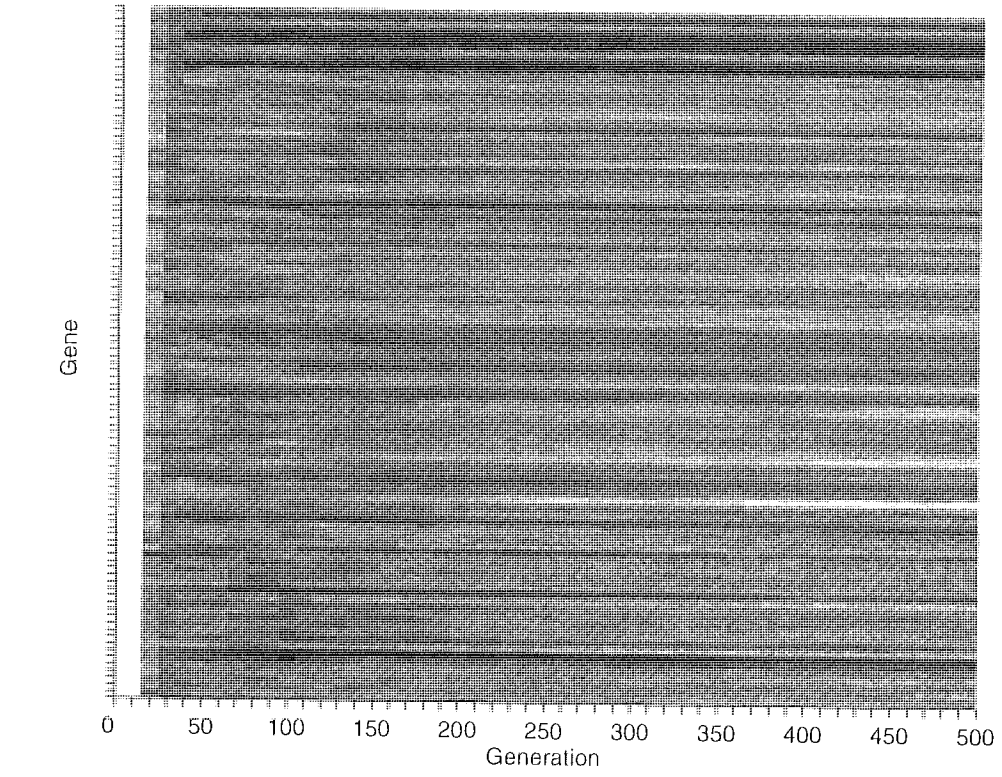

Figure 4.3: Population-wide distributions of values for two hypothetical normalised real genes, X and Y. If the total range of values is used as a measure of diversity (a), gene X is considered to have greater diversity than gene Y. A better measure (b) is obtained by dividing the range into bins and counting the number of filled bins, in which case Y is seen to be more diverse than X. The diversity score is expressed as a proportion of the maximum possible diversity in both cases.

For the purpose of measuring genetic convergence in these experiments, the second of the methods described above is used. The range [-1,1] is divided into 100 equal bins, each of width 0.02. A useful property of the normalised genetic coding used in the GA-bumptree is that every gene has the same possible range of values, meaning that the same measure can be used to estimate the convergence of any gene. To calculate the degree of convergence for each gene, each of the 400 different values for that gene in the population (one for each string) is placed in the appropriate bin. Dividing the number of non-empty bins by 100 (the total number of bins) then gives a diversity measure in the range [0,1], as in figure 4.3(b). The closer this value is to 0, the closer the gene is to complete convergence.

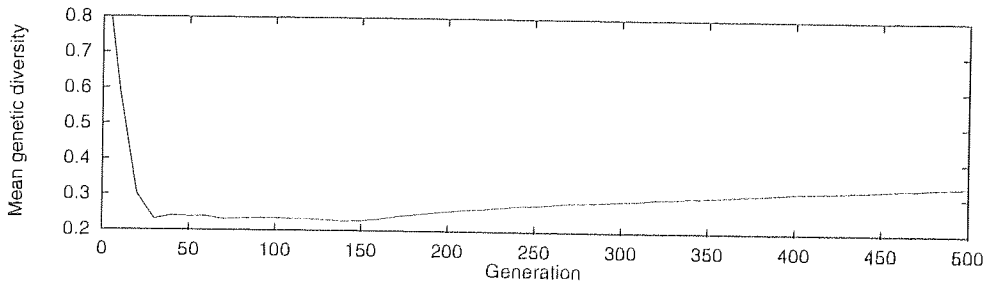
### 4.3.2 Convergence and function

Tracing the change in diversity of individual genes over the course of a GA run can reveal a wealth of information about the nature of the search space and also about the GA itself. One way of visualising this information is to plot a *convergence profile*. Figure 4.4(a) shows the convergence profile of a GA-bumptree run in which the walk length, generation gap and mutation rate parameters were set to 30, 10% and 15%

respectively. The genes are laid out along the vertical axis, while evolutionary time increases from left to right in steps of 10 generations. The shade of grey in each cell indicates the diversity for that gene at a particular generation, calculated in the manner already described, with white indicating maximum diversity and black minimum. Tanese (Tanese 1989) has used similar figures to compare genetic diversity in binary GAs. Figure 4.4(b) shows the change in mean genetic diversity, i.e. the average of each column in 4.4(a), from generation to generation.




(a)




(b)

Figure 4.4: Genetic convergence in a GA-bumptree run with high selection pressure. The convergence profile (a) shows the change in diversity of each individual gene (for clarity, only the first 200 genes are shown) as the population evolves. Figure (b) shows the mean genetic diversity, equivalent to averaging each column in (a).

The leftmost column in figure 4.4(a) represents the initial state of the population. Because every string initially has a random value for every gene, diversity is high and this column is completely saturated white. The 30 step walk length, however, means that selection pressure is intense, and advantageous genetic material is able to quickly spread through the population. This causes diversity to plummet in the first 30 generations as most of the initial alleles are selected out. In Goldberg's (Goldberg 1991a; Goldberg 1991b) terms, this is the period during which the GA is selecting a 'virtual alphabet' for each gene, a small subset of the original alleles.

After around 50 generations the average genetic diversity reaches a minimum and remains constant for the next 100 or so generations (figure 4.4(b)). During this period every gene has converged to only a few alleles. This does not mean, however, that the GA has discovered the best alleles for every gene – far from it. With high selection pressure and little to impede the spread of genetic material, the early stages of the search are driven by those genes for which advantageous alleles happen to exist in the initial population. Strings carrying these few beneficial alleles receive the lion's share of the reproductive trials, and the population is quickly swamped by their descendants. Of course, it is not just the good alleles which these strings pass on to their many offspring, it is their whole genotype. This results in every gene converging to a small range of values, in a process known as hitch-hiking – the indifferent genetic material hitch-hikes along with the beneficial alleles (e.g. Forrest and Mitchell 1993; Harvey 1993).

After around 150 generations, genetic drift begins to occur. Mean diversity creeps up (see figure 4.4(b)) as mutations accumulate in those genes which are not making a significant contribution to fitness. Mutations will also affect the beneficial genes, of course, but these will be quickly selected out. The pattern which emerges as the population continues to evolve is that diversity steadily increases in the 'indifferent' genes, while remaining low in the important genes due to the selection pressure against changes in these genes. Towards the end of the run it is possible to clearly identify those genes for which the GA has found good alleles. These are represented by the black lines across the convergence profile in figure 4.4(a), which stand out against the grey backdrop of the original 'hitch-hikers' which have diverged steadily in the latter stages of the run.

The fact that the amount of variation in a gene is a reflection of its contribution to fitness has been exploited by biologists for some time. Hendriks, Leunissen, *et al.* 1987) used this principle to establish that a gene coding for the protein αA-crystallin, which is normally associated with lens formation in the vertebrate eye, had some survival value even for the blind mole rat *Spalax Ehrenbergi*. Hendriks *et al.* compared the observed rate of mutation in the αA-crystallin gene with the rate of

mutation in other genes which were known to be irrelevant introns (non-expressed portions of DNA, also called *pseudogenes*), and found that the αA-crystallin gene accumulated mutations at a much lower rate. Since the basic mutation rate is assumed to be the same for all genes, they were able to conclude that many mutations to this gene were being selected out.

Perhaps surprisingly, there only seems to have been one application of this idea in the GA field to date, that of Ackley and Littman. Inspired by Gould's (1989) review of the work of Hendriks *et al.*, Ackley and Littman used a similar method to study the relative importance of two different groups of genes in their binary ERL GA (described briefly in the next chapter). The convergence profile offers a general method for evaluating the relative contribution of genes in a real-valued GA, and would seem to be a useful addition to any GA practitioner's toolbox.

In many parametric optimisation problems, the relative sensitivity of the system to the parameters being optimised may be static, but unknown. In this case the convergence profile can yield useful information about the system itself. In the GA-bumptree there are many epistatic interactions between genes, so the importance of any particular gene is not fixed. A gene's contribution to fitness may change in every different run of the GA, according to the particular evolutionary path which is followed. As an example, a node which is critical to classification in the best trees found in one run might never be used in another. The genes which code for this node will be extremely important in the first instance, and irrelevant in the second. In this case, the convergence profile reveals more about the particular evolutionary path followed by the GA than about the objective function. Figure 4.5 illustrates the relationship between convergence at the genotype level and functionality at the phenotype level for the same GA-bumptree run which generated the convergence profile in figure 4.4. The histogram in figure 4.5(a) shows the final mean genetic diversity for each block of genes which corresponds to a single node in the tree. In this case there is a striking difference between the diversity of genes coding for the two top-layer nodes. The node 1 genes have much lower variation than those for node 2, suggesting that the former node is playing the major role in classification. This suggestion is reinforced by examining the nodes in the next layer: the genes coding nodes 3 and 4, the descendants of node 1, are significantly more converged than those for nodes 5 and 6. In fact, the best tree in the final generation actually consisted of just nodes 1,2, 3 and 4, as shown in figure 4.5(b). The low diversity in nodes 7 and 8, particularly the latter, indicates that these nodes must also have been important towards the end of the run, suggesting the tree in figure 4.5(c) as a recent ancestor to the eventual solution.

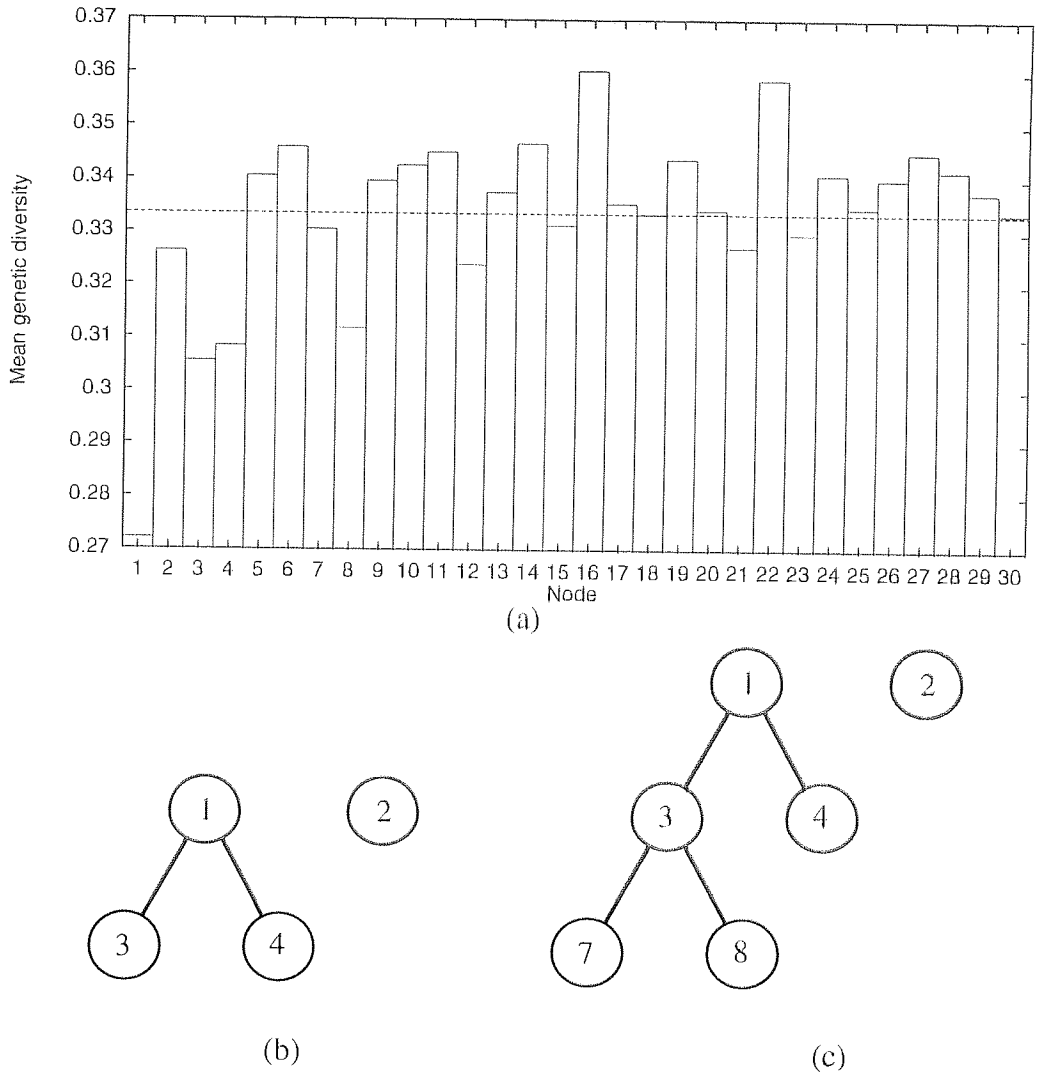



Figure 4.5: (a) Mean genetic diversity for each block of genes in the last generation of a GA-bumptree run. Each block of genes codes for one node. The dotted line shows the overall mean diversity. Also shown are the best tree found by the GA (b) and one of its recent ancestors (c).

### 4.3.3 Convergence and fitness: phyletic gradualism vs. punctuated equilibrium

So far only a single GA run has been examined, in order to introduce the methods used to measure convergence. Two questions remain: how does the walk length parameter affect the rate of convergence, and how does the rate of convergence affect the way the GA searches?

#### The effect of walk length on genetic convergence

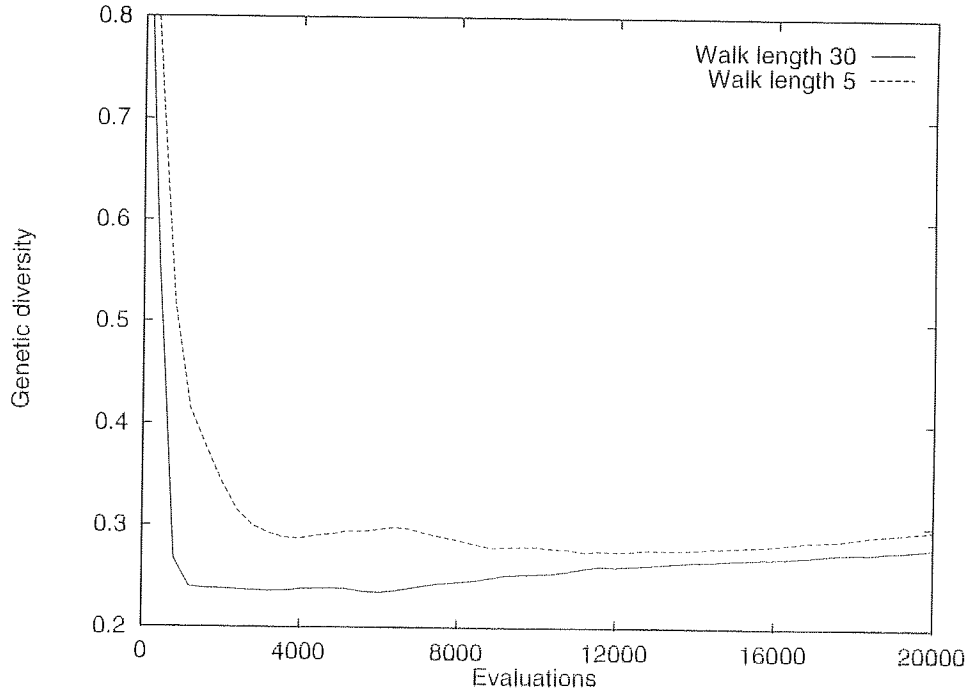
Figure 4.6 compares results from two experiments which are equivalent except for the walk length parameter, which is set to 30 in one and 5 in the other. In the following discussion, these settings for walk length are referred to as WL30 and WL5

respectively. In these experiments the mutation rate and generation gap are 15% and 10% respectively, as in the run described in the previous section. Unlike before, these are not results from individual runs, but are averaged over all 10 runs for each experiment.

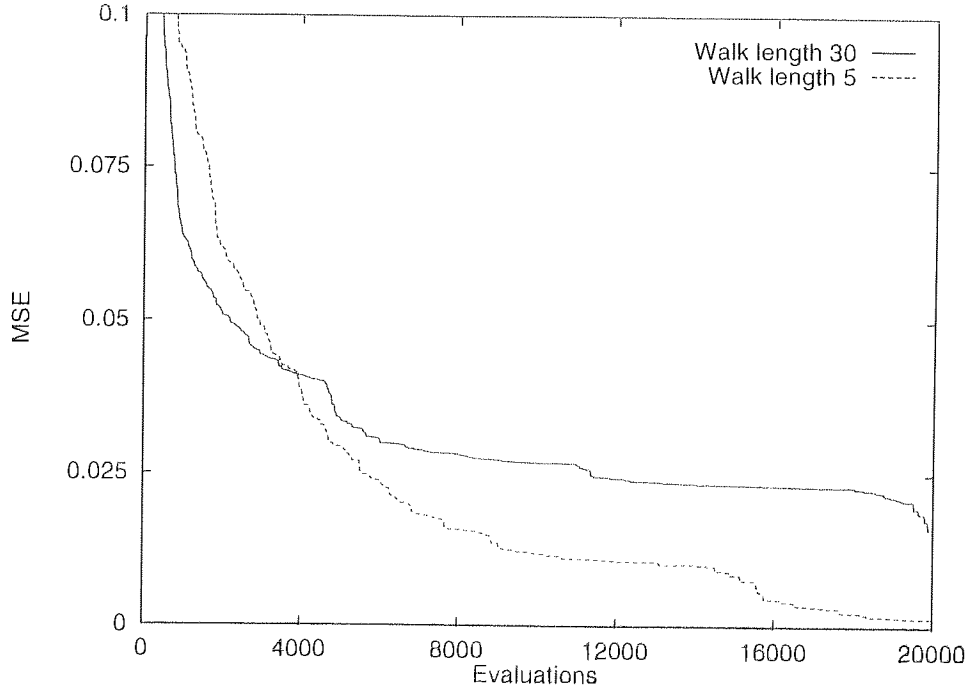
For WL30 the mean genetic diversity (figure 4.6(a)) follows the expected pattern: there is rapid loss of diversity in the first 2000 evaluations[3] (50 generations), followed by a period of low diversity lasting around 4000 evaluations, before diversity begins to rise again due to genetic drift. In WL5, however, the short walk length means that selection pressure is reduced and the spread of genetic material through the population is impeded. Genetic diversity drops much less rapidly in the early generations of WL5, giving the GA more time to explore a diverse range of alleles. Diversity levels off after around 4000 evaluations, but remains high in comparison to WL30. Unlike in WL30, the diversity in WL5 does not show a tendency to rise towards the end of the search, since the GA never converges to the same extent.

Figures 4.7(a) and 4.8(a) show the same results for two different pairs of runs, and are included to illustrate that these are general trends, not specific to a particular choice of parameters. In figure 4.7 the generation gap remains the same but the mutation rate is lowered to 5%. The only noticeable effect this has on diversity is to reduce the rate at which diversity increases due to genetic drift in the later stages of the WL30 runs. In figure 4.8 the mutation rate is 15% and the generation gap is reduced to 5%. This increases selection pressure and leads to a reduction in the overall level of diversity for both WL30 and WL5. The diversity curves retain their characteristic shapes, however, with diversity in WL30 dropping quickly and then rising towards the end of the search while the curve for WL5 drops more slowly before levelling off.

[3]The horizontal axis on these graphs is labelled in evaluations rather than generations in order to compare runs with different generation gaps.

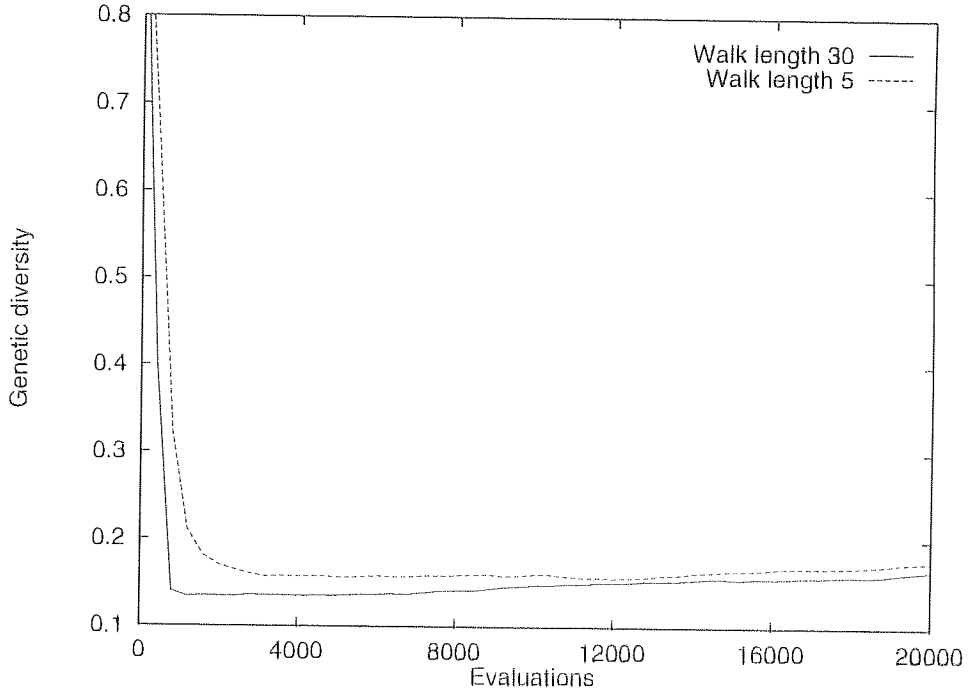



(a)

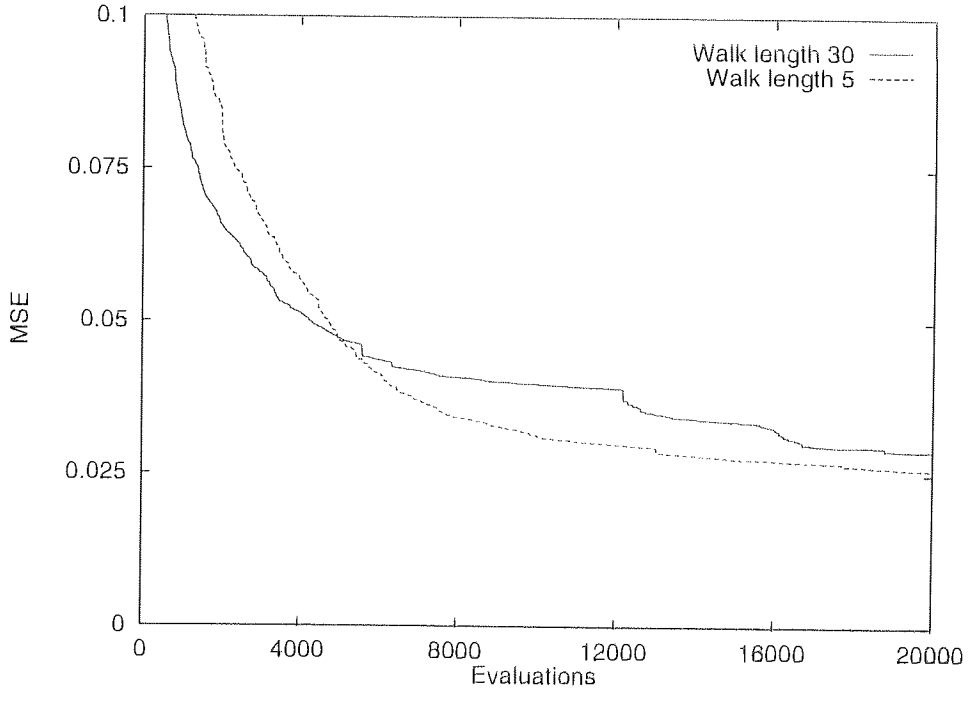



(b)

Figure 4.6: Convergence (a) and fitness (b) curves for walk lengths of 30 and 5 cells. Mutation rate = 15%, generation gap = 10%. All curves are averaged over 10 runs.

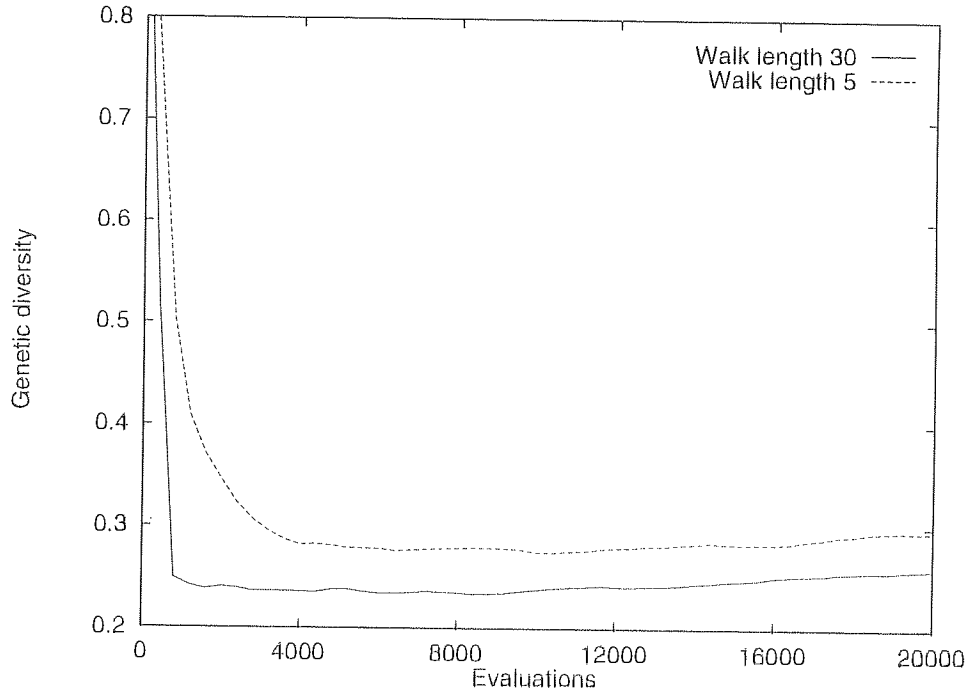



(a)

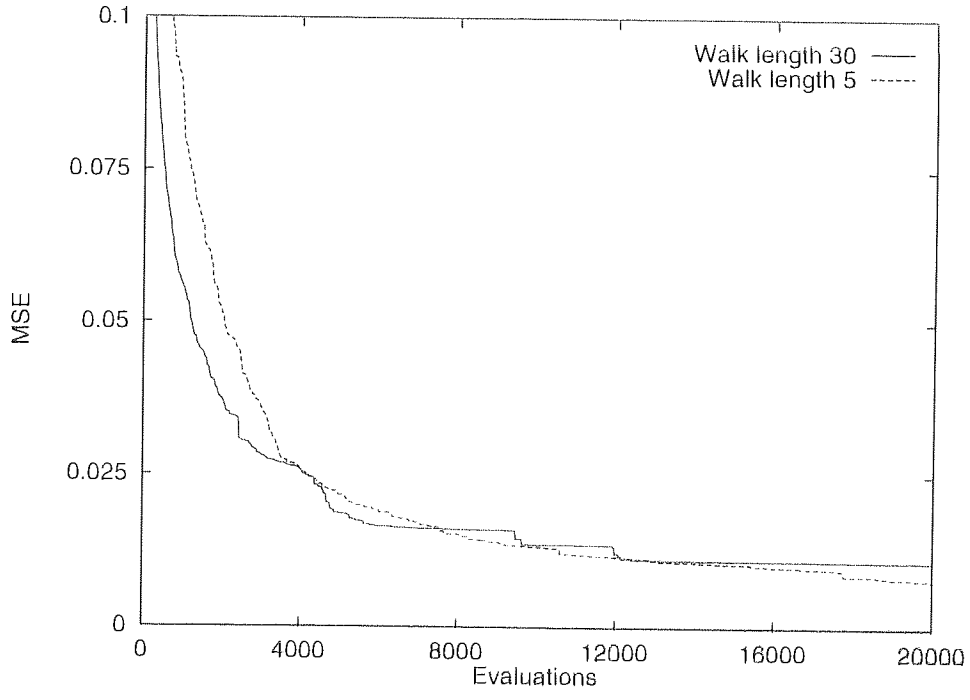



(b)

Figure 4.7: Convergence (a) and fitness (b) curves for walk lengths of 30 and 5 cells. Mutation rate = 5%, generation gap = 10%. All curves are averaged over 10 runs.




(a)




(b)

Figure 4.8: Convergence (a) and fitness (b) curves for walk lengths of 30 and 5 cells. Mutation rate = 15%, generation gap = 5%. All curves are averaged over 10 runs.

### Walk length and fitness

Figures 4.6(b), 4.7(b) and 4.8(b) show the fitness curves for these experiments. With the particular combination of 15% mutation rate and 10% generation gap shown in figure 4.6, WL5 shows a marked improvement in final fitness when compared to WL30, although this is not generally the case. What is more interesting is the difference in the shapes of these curves. In each case (and in general, although only these three examples are included here to illustrate the point), performance in the WL5 runs appears to improve continuously and gradually throughout, while the WL30 runs are characterised by periods of stasis, during which no improvement takes place, punctuated by sudden bursts of rapid improvement. These curves are reminiscent of two traditionally opposed views of evolution in the natural world, the schools of *phyletic gradualism* and *punctuated equilibrium*.

The first school of thought, phyletic gradualism, is the traditional interpretation of evolution as a process of continuous gradual refinement of a lineage through a sequence of small changes. Proponents of this view cite examples such as the horse lineage, in which the fossil record appears to document a continuous progression of form from an ancestral dog-like, browsing creature with padded feet (*Hyracotherium*) through a multitude of intermediate stages spread over some 58 million years, to the large, hooved, grazing animal known today (Savage 1977). A criticism of the gradualist perspective is that such continuous sequences of gradually changing fossil forms turn out to be the exception, rather than the rule. This fact has traditionally been attributed to the inherently fragmented and incomplete nature of the fossil record – fossilisation is a rare event, it is argued, even on geological timescales, and so the fact that a certain intermediate form has not been found need not be taken as evidence that it didn't exist. Darwin himself wrote:

> "Why then is not every geological formation and every stratum full of such intermediate links? Geology assuredly does not reveal any such finely graduated organic chain; and this, perhaps, is the most obvious and serious objection which can be urged against the theory [of evolution]. The explanation lies, as I believe, in the extreme imperfection of the geological record." (Darwin 1859, chapter X)

In 1972, Eldredge and Gould proposed a slightly different viewpoint of evolution as a process of punctuated equilibrium. This view emphasises stasis, not gradual change, as the norm, and suggests that evolutionary change occurs intermittently in rapid (in geological terms) bursts. The basic principle is that a species which is adapted to its ecological niche represents an evolutionary local optimum, such that any genetic change is deleterious. Such a species will remain in stasis indefinitely. Evolutionary change only occurs when the environment changes or, more typically, when some members of the species migrate to a new environment.

In this case, selection pressure to adapt to new demands drives rapid evolutionary change, and the migrant population evolves into a new species – a new local optimum. If long periods of stasis are the rule and evolutionary change the exception, then the rate of change, when it occurs, must be far greater than the apparent average rate of change. This makes the gaps in the fossil record seem less of a puzzle; instead of expecting to find intermediate forms neatly laid out in consecutive strata, the punctuationist expects them to be compressed into a period of time which may not even be measurable on the palaeontological scale[4].

A great deal of controversy has surrounded the theory of punctuated equilibrium since its proposal. This has been largely due to its being misinterpreted by many and, according to Dawkins (1986), deliberately misrepresented by its proponents, as something radically different from traditional Darwinism[5]. In reality, Darwin would not have found punctuated equilibrium in the least bit controversial, since he himself suggested the basic principle in chapter X of *The Origin of Species*:

> "Hence, when the same species occurs at the bottom, middle, and top of a formation, the probability is that it has not lived on the same spot during the whole period of deposition, but has disappeared and reappeared, perhaps many times, during the same geological period. Consequently if it were to undergo a considerable amount of modification during the deposition of any one geological formation, a section would not include all the fine intermediate graduations which must on our theory have existed, but abrupt, though perhaps slight, changes of form."

The main contribution of punctuated equilibrium is not, therefore, a new theory of evolution to rival Darwin's. Instead, it represents a significant insight into the circumstances which catalyse evolutionary change, and the rate at which it occurs. Seen as such, the theory has been gaining evidence steadily since its inception, an example being the discovery that some members of the horse lineage which were previously thought to represent a series of transitional forms were actually contemporary species which remained static over millions of years (Gould and Eldredge 1993).

Returning to the GA-bumptree fitness curves in figures 4.6(b), 4.7(b) and 4.8(b), both of these different views of evolutionary change, phyletic gradualism on the one hand and punctuated equilibrium on the other, are in evidence. The question

---

[4]In fact, Eldredge and Gould go further than this. They suggest a final step in which the migrant population, now a new species, returns to its original niche, where it competes with and replaces the ancestral species. The palaeontologist digging at the ancestral site may find fossils of both the original and the new species, but has no chance of finding an intermediate form, because the evolutionary change occurred elsewhere.

[5]Punctuated equilibrium has been misinterpreted both as a theory of saltation – evolution as a series of large jumps (macromutations) such as a *hyracotherium* giving birth to a modern horse (to take an extreme example) rather than an accumulation of small changes – and also as evidence of creationism, with divine intervention providing each new species.

is, why should all the WL5 runs appear to support the gradualist view of evolution while the WL30 runs exhibit punctuated equilibrium? If the punctuationists' view is accepted, that stasis is the norm in stable environments and that evolutionary change only generally occurs in response to environmental change, a possible explanation presents itself.

It has already been shown that many epistatic interactions exist between the genes in the GA-bumptree – a gene which is beneficial to one string may have a deleterious effect when recombination substitutes it into a different string. In the GA, the environment which determines a string's chance of reproductive success is, therefore, a combination of two factors. The first is the fitness function itself, which will determine how often the string has the opportunity to reproduce, and how long it is likely to survive before being replaced. The second factor is its 'genetic environment', the genetic makeup of the strings with which it is likely to mate (i.e. those strings which surround it on the lattice).

In the WL5 runs, the short walk length restricts the spread of genetic material across the lattice. This prevents the whole population from converging on a particular evolutionary path, instead allowing different species to form in different local regions, each evolving along its own genetic trajectory (this is illustrated empirically in the following section). Evolutionary change within a particular species will be limited, since members of the species face a fairly static environment: the fitness function does not change with time, and all the strings within the same species will be very similar genetically. However, migration between species will occur through recombination across species borders. A genotype which migrates from one species to another faces a significant change in its genetic environment, and it is these environmental changes which drive evolutionary change. The picture that emerges is one of improvements occurring in overlapping bursts of evolutionary activity associated with migration and speciation events on different parts of the lattice. The fitness curves in figures 4.6, 4.7 and 4.8 show a continuous gradual improvement because they chart the progress of the population as a whole, rather than the rise and fall of individual species.

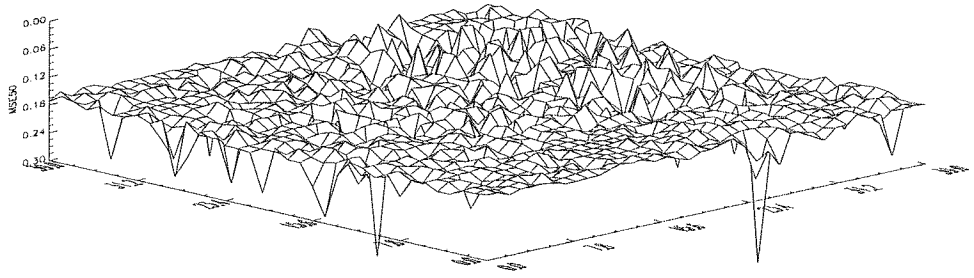
As the walk length parameter is increased, genetic material flows more freely across the lattice, and the number of different species which the lattice can support is reduced. In the WL30 runs, the lattice soon becomes dominated by a single species: the whole population effectively converges to a single evolutionary path, and follows this path to a single local optimum. Once this local optimum has been reached, the environment stabilises and the population enters a period of stasis. Although recombination and mutation will continue to suggest different evolutionary paths, all will lead away from the local optimum. Because there is little geographical restriction on selection, mating and replacement, such sub-optimal strings will be efficiently purged from the population, and will only rarely gain a foothold. The occasional

bursts of evolutionary change which punctuate the graphs in figures 4.6(b), 4.7(b) and 4.8(b) represent those rare occasions when a new species has survived long enough to explore sufficiently far along a new evolutionary path to escape the current local optimum and discover solutions of higher fitness. When this occurs, the new species will quickly spread across the lattice, upsetting the previously stable environment and catalysing a period of intense evolutionary activity. Within a few generations the whole lattice is invaded by the new species, the population converges to a new local optimum, and stasis is restored.

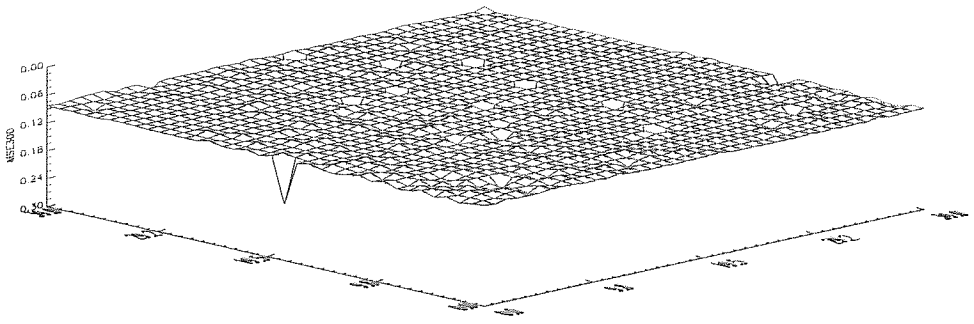
### 4.3.4 Convergence and species formation

The preceding arguments rest on one assumption which has not yet been justified empirically: the emergence of distinct species in different local regions on the lattice during the course of a GA run, the number of which decreases as the walk length is increased. In order to test this assumption, it is first necessary to decide exactly what constitutes a species in the context of the GA-bumptree.

One method used by Davidor (1991) is to plot a contour map of fitness values in the population, on the basis that different species may be identified by regions of different levels of fitness. In general, however, this approach is not very useful. A useful definition of species should group together strings which represent similar solutions, and similar fitness only implies similar solutions if the coding scheme is completely non-redundant and non-epistatic. In the case of the GA-bumptree (and in GAs in general), two strings may be quite different genetically, even to the extent of being unable to produce viable offspring through recombination, yet still have very similar fitness. This point is illustrated in figure 4.9, which shows the way the distribution of fitness values across the lattice changes during the course of a typical GA-bumptree run. After 300 generations it can be seen that there is little variation in fitness across the lattice, but the figure reveals nothing about the nature of the solutions which exist in the population. It is not clear whether the whole population has converged to one type of solution, or if different solutions of similar quality are being explored in different regions of the lattice.



(a)

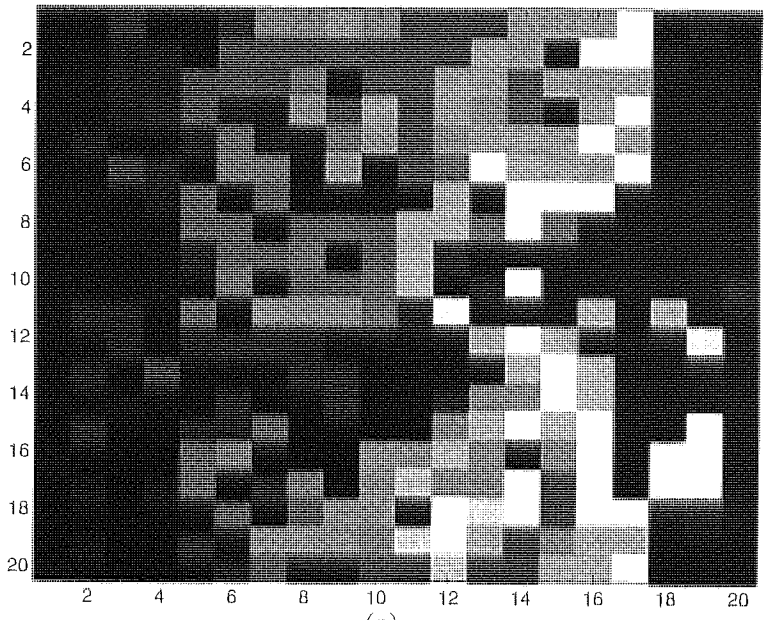


(b)

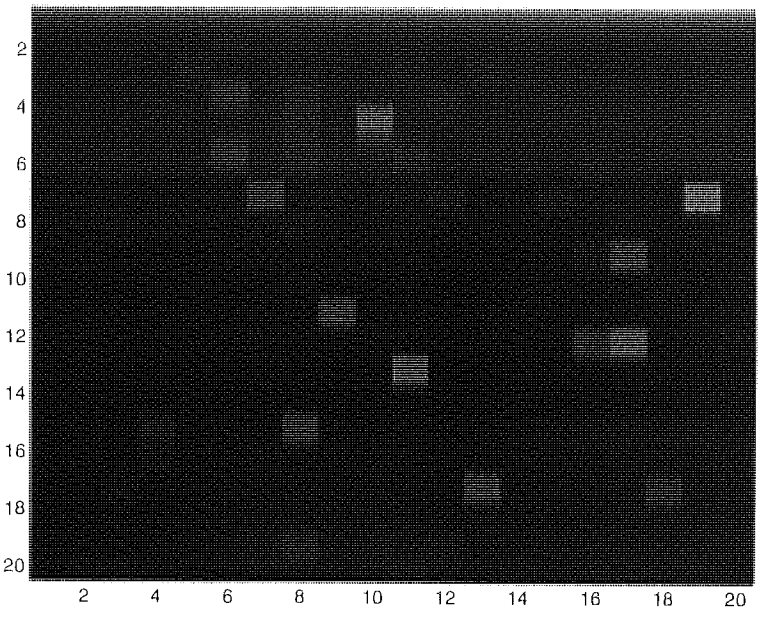
Figure 4.9: Population fitness distribution for a GA-bumptree run after (a) 50 and (b) 300 generations.

A more appropriate way of separating strings into species is to compare them at the level of the genotype, rather than the phenotype. This kind of comparison was also suggested by Davidor (1991), although it is not clear whether Davidor ever implemented the idea[6]. The principle that species should group together strings of similar genotype was applied to analysing the GA-bumptree, and the results are shown in Figures 4.10(a) and 4.10(b). These figures show the distribution of species across the lattice after 100 generations of each of two GA runs, one in which the walk length was 5 cells, and one in which it was 30.

[6]Figure 3 of (Davidor 1991) shows 'genotypically similar islands' as differently shaded regions of a square representing the lattice, but appears to be slightly contrived. The fact that different regions end abruptly at the edges of the square, despite the fact that the lattice is toroidal, suggests that this figure is intended to illustrate a point rather than to show actual experimental results.




(a)




(b)

Figure 4.10: Distributions of genotypically different species on the lattice after 100 generations for walk lengths of (a) 5 and (b) 30 cells.

To produce each of these figures, the GA is stopped and every cell on the lattice is examined in turn. The string in the first cell is allocated species number 1. Thereafter, each string is compared to those which have already been examined, and allocated the same species number as the string which it most resembles, with 'resemblance' being inversely proportional to Euclidean distance in this case. If a string does not bear sufficient resemblance ('sufficient' being a pre-defined threshold

distance) to any string previously examined, it is considered to represent a new species, and allocated a new number. In figure 4.10 each different species number is represented with a different shade of grey.

Obviously, the particular choice of threshold parameter will directly determine how many 'species' are identified – if the threshold is too large then all strings will be classed as the same species, if it is too small then each string will be considered a different species. So long as the same threshold is used in both instances, however, a useful comparison may be made between two runs. In the case of the GA-bumptree, figure 4.10 appears to support the assumptions which were made about the effect of the walk length on the number of species which the lattice can support. After 100 generations, the population for which the walk length was 30 cells has become fairly uniform across the lattice, while a considerable amount of variety exists between different regions of the lattice for the population for which the walk length was 5.

## 4.4 Conclusions

This chapter has summarised the results of an extensive series of experiments intended to explore the sensitivity of the GA-bumptree hybrid to three basic parameters: walk length, generation gap and mutation rate. In particular, these experiments were focused on investigating the evolutionary dynamics of the local selection and replacement model, and the effect of varying the walk length on the rate of genetic convergence.

Section 4.2 began by comparing the final fitness scores achieved by the various runs. The rather negative conclusion was that these results reveal little about the overall impact of each of the three parameters on the quality of solutions discovered by the GA, since all the runs discovered solutions of similar quality. The most likely reason for this uniformity of performance is the simplicity of the classification problem chosen. Although the iris problem served admirably for the main purpose of these experiments, as a basis to study the local selection scheme, it appears, with hindsight, that more information could have been gleaned from this study if a slightly more challenging problem had been chosen. In the event, the only general conclusion that can be drawn is that mutation rates below 5% appear to be too low.

Section 4.3 shifted attention to the real focus of these experiments, investigating the effectiveness of the local selection scheme at controlling genetic convergence. A method was proposed for measuring the degree of convergence of individual genes in a real-valued GA, and this was extended to the concept of the convergence profile, a graphical representation of the changes in genetic diversity during the course of a GA run. A particular run was examined, in which high selection pressure caused the GA to converge completely after a relatively small

number of generations. Under these conditions, three distinct phases of evolution can be identified: an initial period of rapid convergence, during which a virtual alphabet is selected and hitch-hiking occurs; a period during which diversity remains at a stable minimum; and a period of genetic drift towards the end of the run as mutations accumulate in non-essential genes. Examining the convergence profile during this latter stage reveals which genes are functionally relevant and which are not. The usefulness of the convergence profile as an analytical tool is not limited to these experiments – it offers a general method for analysing any real-valued GA.

The relationships between the walk length parameter, the overall rate of convergence, and the overall performance of the GA were investigated by looking at two extremes for the walk length, 5 cells and 30 cells. Based on these results, several conclusions may be drawn about the dynamics of the local selection model GA. Choosing the shorter walk length allows different species to form on different parts of the lattice. From the point of view of the population as a whole, this means that a certain level of genetic diversity is maintained throughout the run, and the GA exhibits a continuous gradual improvement in performance from each generation to the next. This gradualist picture of evolution emerges as a result of multiple overlapping migration and speciation events occurring in different regions of the lattice. When the walk length is increased to the longer extreme, the overall picture changes. The whole population now behaves like a single species. After an initial period of convergence, evolution proceeds in a process of punctuated equilibrium – periods of stasis punctuated by occasional bursts of evolutionary activity associated with rare speciation events.

What is initially surprising is the fact that these two quite different modes of evolution yield such similar overall performance. Premature convergence is generally considered to be the death of genetic search, since recombination is the GA's primary search mechanism, and variety is the grist to recombination's mill. Since reducing the walk length increases the level of diversity in the population, it was also expected to improve the performance. However, although the shorter walk length runs often showed a tendency to find slightly better solutions (the experiment shown in figure 4.6 produced the most pronounced difference), the difference overall is not statistically significant. One reason for this might be that the subtree crossover operator is not sufficiently powerful, and that the search is relying more on mutation that is expected: if stochastic hill climbing, rather than recombination, is the dominant search mechanism then genetic diversity becomes less of an issue. There is a danger, however, of looking for reasons where none exist – it has already been observed that the simplicity of the iris problem is likely to be the main cause of the overall similarities in performance.

# Chapter 5

# Learning problems I: artificial life

## 5.1 Introduction

In the closing chapter of his Ph.D. thesis, Hancock (1992) drew the following conclusion concerning the application of GAs in the field of neural networks:

> "GAs are most relevant to problems for which there is no gradient information available, for instance because of discontinuities in the search space. It thus appears unlikely that they will out-perform gradient descent techniques for training the weights of nets... A more fruitful area of interaction between GAs and NNs may be the emerging field of 'artificial life'. Even a simple simulated animal looking for food cannot readily be trained by gradient descent techniques, since the target outputs are not specified... Such systems will provide a rich testbed for exploring coding strategies within nets, and GAs remain the obvious way of adapting them."

The work described in this chapter was motivated by this observation. The intention was to investigate whether applications existed within the artificial life field for a genetic approach to optimising both the architecture and weights of a neural network model such as the bumptree.

The chapter begins with a broad overview of the field of artificial life, or A-life, in which four main themes are identified. The fourth of these, the study of interactions between the processes of learning and evolution, is described in greater detail than the others, as it is particularly relevant to evolutionary neural networks. The remainder of the chapter describes a series of A-life experiments in which a GA is used to evolve a neural network controller for an artificial ant performing a simple foraging task.

## 5.2 GAs, connectionism and A-life

The field of artificial life (Langton 1989) is a melting-pot of ideas drawn from such diverse disciplines as evolutionary biology, ethology, computer science, psychology and robotics. The field is catalysed by the ever-increasing availability of computing power. A characteristic of much A-life research is the use of the computer as a research tool to gain new insights into the natural world, through complex simulations which have not previously been computationally tractable. As well as attempting to answer nature's questions, the other main thrust of A-life research is towards the construction of artificial autonomous systems, with nature providing the inspiration.

Such a diverse field, woven as it is from previously barely related disciplines, is not easy to categorise. Accepting the fact that some overlap is inevitable, the following sections identify four fundamental branches of A-life. Only a

representative sample of the literature associated with each branch is given, since a through review of the A-life literature would be beyond the scope of this thesis.

### 5.2.1 From simple rules to global complexity

A recurrent theme in A-life research is the observation that a collection of simple agents, each operating independently and obeying a few basic behavioural rules in response to its local environment, can exhibit very complex and often apparently co-ordinated global group behaviour. Powerful computer simulations have made possible the empirical study of these kinds of complex systems, and this has led to models which can account for previously poorly-understood phenomena. Some aspects of group behaviour in social animals, for instance, can be accounted for without the need to postulate sophisticated communication mechanisms or global knowledge on the part of individuals. Camazine (1993) has successfully applied this approach to produce computer simulations modelling pattern-formation on the combs of honey bee colonies, collective nectar foraging behaviour of honey bees and brood sorting in ants. Other authors are applying the same reasoning to investigations of group behaviour in a wide range of social animals, from group vigilance behaviour in wild boars (Quenette 1993) to roosting and dispersal behaviour in flocks of gulls (De Schutter and Nuyts 1993), among many others.

Studies such as these have also served to inspire a branch of A-life which might be termed *collective robotics*, in which the eventual aim is to construct simple autonomous robots which can solve complex tasks by working collectively. Some potential advantages of this approach are readily apparent: it is inevitably going to be cheaper and easier to build many very simple, identical, robots than a few complex ones, and such robot 'teams' would have a degree of fault-tolerance built in, benefiting from the kind of graceful degradation of performance under failure which other parallel distributed processing systems, such as real and artificial neural networks, exhibit. An early example of this kind of work may be found in the 'ant-like robots' of Deneubourg *et al.* (1991), simple simulated robots which exhibit global sorting behaviour. Other simulations have investigated other aspects of collective behaviour such as co-operation, flocking, trail following, homing and so on (Goss, Deneubourg, *et al.* 1993; Mataric and Marjanovic 1993; Shibata and Fukada 1993), and the approach is beginning to be extended from simulation to the construction of actual teams of real robots (Ali Cherif 1993; Hess 1993; Mataric and Marjanovic 1993).

Finally, this section would not be complete without mention of the fascinating work pioneered by Tom Ray, in which a virtual 'primordial soup' environment is seeded with a simple self-replicating computer program. Imperfect replication,

coupled with competition for finite resources, leads to the evolution of ever more efficient replicators, and to the emergence of social phenomena such as symbiosis and parasitism (Ray 1991a; Ray 1991b; Ray 1991c; Maley 1993).

### 5.2.2 Neuroethology

Neuroethology is the study of the neural mechanisms underlying the generation of a creature's behaviour (Cliff, Husbands, *et al.* 1993). Through computer simulations of biological neural networks, A-life offers an empirical approach to testing hypotheses in this field. An example of this is the work of Brunn *et al.* (1993), in which a simple backpropagation neural network is used to model the neural mechanism of leg placement in the stick insect, in order to test the hypothesis (based on physiological experiments) that the stick insect uses an approximate algorithm for leg placement. Beer and Chiel (1990) take a similar approach, drawing on work on the neural basis of feeding in the marine mollusc *Aplysia* to construct an artificial neural network which models motivated feeding behaviour in a simulated insect. The same authors have since extended their artificial neural model to other simple insect behaviours such as wandering and edge-following (Beer and Chiel 1991).

Through the use of genetic algorithms, A-life is also able to offer insights into the evolution of the neural circuitry underlying behaviour in animals. In the crayfish, for instance, the neural circuitry controlling the 'tailflip' escape reaction contains a paradoxical 'useless' synapse. Lending support to the 'pre-adaptation' explanation of this anomaly (Dumont and Robertson 1986), simulated evolution experiments have reproduced a similar redundant synapse in an artificial neural circuit originally evolved for swimming and later adapted for flipping (Stork, Jackson, *et al.* 1991). This kind of simulation serves to underline the greedy nature of evolutionary change: in evolution "elegance of design counts for little" (Dumont and Robertson 1986), as was demonstrated by the simple experiments described in section 2.4.3. This is also well illustrated in the work of Cliff *et al.* (1993) in which artificial neural networks, evolved using a GA for a robot control task, are analysed using techniques analogous to those used in the study of biological neural networks. In this work, two networks which evolved in identical simulated environments are almost indistinguishable at the behavioural level, but are found to be strikingly different at the morphological level.

### 5.2.3 Modelling behaviour

A-life offers an empirical approach to testing existing analytical models of behaviour. For instance, Collins and Jefferson (1991) describe a microanalytic computer simulation built to investigate the evolution of sexual selection and female choice, and

in particular the paradoxical prevalence in nature of secondary sexual characteristics which are exaggerated to the point of being maladaptive (Fisher 1958; Kirkpatrick and Ryan 1991). By simulating the evolution of a population of simple individuals with sexual selection, it was possible to verify Kirkpatrick's (1982) existing analytical model, test the robustness of the model by relaxing certain simplifying assumptions (necessary to make the analytical model mathematically tractable, but not necessary for the microanalytic model) and study the dynamics of the evolving system.

Instead of using simulations to validate or extend existing behavioural models, some A-life researchers have taken the complementary approach, drawing inspiration from established models and pioneering the field of *behaviour based robotics*, exemplified by the work of Dorigo *et al.* (Dorigo and Schnepf 1991; Dorigo and Sirtori 1991; Dorigo and Schnepf 1992). In this work a model of behavioural organisation in animals, the Tinbergen model (Tinbergen 1966), provides inspiration for robot controllers based on classifier systems. In a similar vein, Riolo (1991), inspired by early experiments in 'latent learning' in rats (Walker 1987), describes a learning classifier system, used to control an agent in a maze environment, which is extended to incorporate learning even in the absence of a reward signal.

### 5.2.4 Interactions between learning and evolution

#### The Baldwin effect

The idea that acquired learning can influence the course of evolution has been termed the 'Baldwin effect', after one of its first proponents (Baldwin 1896). It is an idea which has sometimes been treated with suspicion by biologists, partly because it smacks of Lamarckism, and partly because it is a difficult hypothesis to test (Maynard Smith 1987).

The basic principle can be illustrated as follows. Consider a population of individuals whose behaviour is, in some sense, sub-optimal for their environment. Individuals which possesses the ability to learn, that is to improve their behaviour during their lifetime and so increase their chance of survival, will tend to leave more descendants than those who do not. So long as inherited behaviour is sub-optimal, learning ability will be advantageous. Moreover, in a stable environment, where the optimum behaviour remains constant, those individuals who inherit behaviour which is closer to optimal have more chance of achieving the optimum through learning, and so will be favoured by selection. In this way, learning can guide evolution so that traits which were once learned afresh by each generation become specified genetically.

Hinton and Nowlan (1987) have performed an elegant experiment to test the validity of the Baldwin effect, based on a computer simulation of the processes of evolution and learning. In their experiment, a GA searches a space of simple Boolean networks. A network consists of 20 connecting weights, each of which may take the value 0 or 1, so that a network may be represented completely by a 20-bit binary word. Of the $2^{20}$ possible networks only one, the network 1111...1, is taken to be the correct network, all others being invalid. This is a 'needle in a haystack' search problem, since the optimum is a single impulse on an otherwise flat fitness landscape, and so no algorithm can be expected to perform better than exhaustive search.

For the GA, a network is coded as a chromosome of 20 genes, each with three alleles, 0, 1 and ?. Genes with the value ? represent connections which are not congenitally hard-wired, but which can be varied by learning. The learning model is extremely simple: to evaluate a network, a succession of 1000 trials are made in which random settings are tried for all its variable connections. If this process should hit upon the correct network (obviously individuals 'born' with one or more connections hard-wired to 0 are doomed from the outset) the individual is assigned a fitness score which is inversely proportional to the number of trials taken to reach the solution.

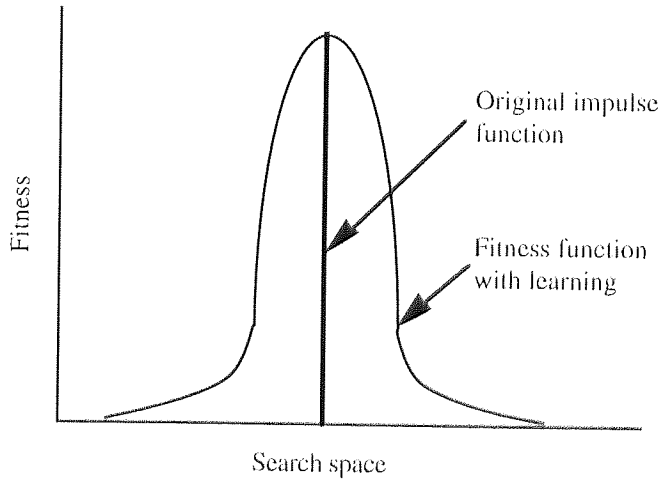



Figure 5.1: How learning transforms the search space, adding 'shoulders' to an impulse function.

The incorporation of the learning model transforms the original 'needle in a haystack' fitness surface. Instead of a single impulse, there is now a smooth region of increasing fitness leading to the optimum. Hinton and Nowlan represent this in terms of 'shoulders' added to the original impulse (figure 5.1), while Maynard Smith (1987) describes it as "like searching for the needle when someone tells you when you are getting close." Without learning, discovering the correct solution takes around 1000 generations of the GA (with population size 1000). Moreover, with

sexual reproduction the solution, once discovered, tends to be lost again due to disruption by crossover (Maynard Smith 1987). With evolution, the population frequency of correct alleles is high after 20 generations, and learning is rapid (Hinton and Nowlan 1987).

If the evolutionary model in Hinton and Nowlan's experiment is impoverished, then the learning model is positively destitute. However, the strength of the experiment is in its simplicity, and it elegantly demonstrates the viability of the *principle* of the Baldwin effect, without claiming to answer questions about the complexities of evolution and learning in the biological world. Authors since Hinton and Nowlan have extended the analysis of the experiment, focusing in particular on the slowness of fixation of the final few connections and the effect of genetic drift (Belew 1989; Harvey 1993).

Other researchers, primarily interested in combining GAs and neural networks to address practical problems, have also examined the Baldwin effect. Keesing and Stork (1991) use a GA to optimise perceptron networks for pattern recognition, allowing some supervised learning before the fitness evaluation. Their view of the Baldwin effect is slightly different from that of Hinton and Nowlan, although the two viewpoints are isomorphic. Instead of viewing learning as a function which transforms the fitness landscape, they perceive it as a 'local hill-climbing' operation on the original landscape[1], as shown in figure 5.2.

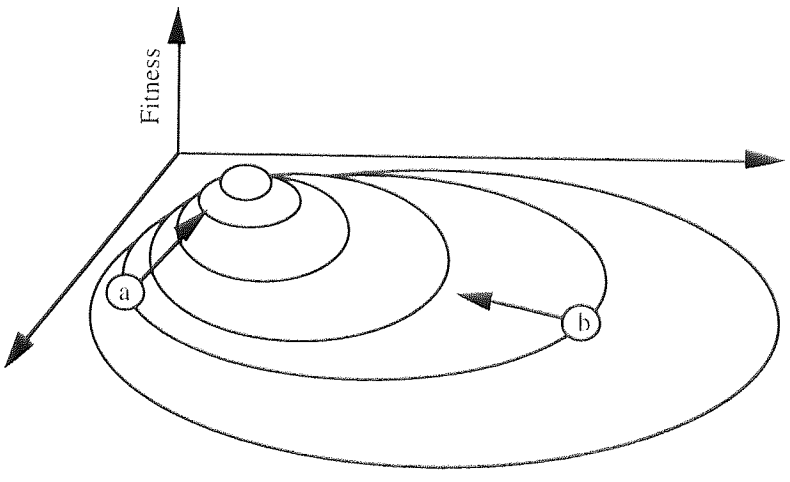



Figure 5.2: Another representation of the Baldwin effect (after Keesing and Stork). Points **a** and **b** represent two neural networks which have equal fitness at 'birth', and so would be allocated equal numbers of reproductive trials by a normal GA. The addition of learning constitutes local hill-climbing on the fitness landscape, represented by the arrows. After learning, **a** has higher fitness and so would be selected in favour of **b**. Learning therefore biases evolution towards points in the region of **a**, and progress towards the optimum is accelerated.

[1]The incorporation of local hill-climbing into genetic search has been examined by various authors (Ackley 1987; Grefenstette 1987; Mansour and Fox 1991), with hill-climbing typically treated as an additional genetic operator.

Keesing and Stork report an empirical study of the effect of varying the number and distribution of learning trials on the rate of evolution and find that, while a little learning is a dangerous thing, too much slows evolution since it reduces pressure to improve the genotype.

### Related work

In the experiments described above, the reward signal which drives learning during the evaluation phase is a function of two things: the environment, i.e. the problem to be solved, and the individual's response to that environment, i.e. the current solution. Some researchers have experimented with a more complicated scenario in which the reward signal is also a function of some additional parameters which are genetically determined.

Nolfi and Parisi describe the evolution of 'auto-teaching' networks (Nolfi and Parisi 1993), neural networks which control simple agents performing a foraging task in a simulated environment[2]. Each individual consists of two neural networks of identical topology, specified by two weight matrices coded as a single chromosome. The first, the action network, maps sensory input to motor output and directly controls an agent's behaviour. The second, the teaching network, maps sensory input to 'desired' motor output. During an agent's lifetime, its action network is trained by continuously backpropagating the difference between its output and that of the teaching network.

The principal objection to the claim that this rather bizarre scenario might exhibit the Baldwin effect lies in the fact that what is 'learned' is already fixed in the genotype: learning simply reduces the discrepancy between the action network and the teaching network, making one of the networks seemingly redundant. If anything, the adaptation which occurs during an agent's lifetime might be better described as 'development' rather than learning, since it is the gradual expression of information which is stored genetically, but not immediately exhibited.

A similar approach is taken by Ackley and Littman in their Evolutionary Reinforcement Learning (ERL) model (Ackley and Littman 1991). As in the above work, artificial agents consist of two neural networks which evolve in parallel, an action network and an evaluation network. The evaluation network maps sensory input to a scalar reward signal, on the basis of which the action network is trained by a reinforcement learning algorithm. As before, the claim that these experiments serve as a good demonstration of the Baldwin effect is undermined by the fact that each individual is born effectively 'knowing' what it intends to learn during its life. The

[2]The environment is practically identical to that described in the following section.

learning process does not so much constitute a local exploration of the problem surface as a progression along a set behavioural trajectory which is fixed in advance by the genotype.

## 5.3 Case study: the artificial ant

### 5.3.1 Introduction

Parisi, Nolfi and Cecconi (1991) report a study of the relationship between learning, behaviour and evolution in a simple two dimensional world populated by animats. In these experiments an animat is an artificial ant which performs a simple food gathering task. The animat's nervous system is a small feedforward neural network of fixed topology, the weights of which are specified by a genetic blueprint. A GA generates new animats from the more successful members of the animat population, and neural networks which are well adapted to the food gathering task evolve.

The reason for studying these experiments was to investigate the practicality of an evolutionary neural network approach to A-life, and to see whether this particular simulation could be used as a testbed for comparing different algorithms, for instance the GA-bumptree and a GA/MLP hybrid. The food gathering task is appealing, in this respect, for two reasons. First, the problem is sufficiently simple and abstract for it to be unlikely to introduce any bias into a comparison of algorithms. Second, it involves a problem of temporal credit assignment. This is the kind of problem, prevalent in real-world applications such as control and robotics, where researchers interested in training neural networks with GAs might usefully concentrate their efforts. It is the sort of problem where the overhead of GA-based training may be worthwhile, since many supervised learning algorithms such as backpropagation are unsuitable.

A further motivation for this work stemmed from the puzzling observation by Parisi, Nolfi and Cecconi that:

> "Learning can accelerate the evolutionary process both (1) when learning tasks correlate with the fitness criterion, and (2) when random learning tasks are used. Furthermore, an ability to learn a task can emerge and be transmitted evolutionarily for both correlated and uncorrelated tasks."

Statement (1) in the above does not appear to be particularly controversial; it simply describes the Baldwin effect, which was discussed in the previous section. The second statement, however, is rather more polemic. The assertion that network performance on a particular task can be improved by training on an *uncorrelated* task seems counter-intuitive. Indeed, if true, it would have consequences for the design of

artificial neural networks for practical applications since it would imply that a network trained on one task should have useful performance even on other uncorrelated tasks.

### 5.3.2 The experimental scenario

Following Parisi, Nolfi and Cecconi, each animat lives in a separate two dimensional environment containing randomly placed pieces of 'food'. Initially, both the food items and the animat are randomly located in cells within a 10x10 grid. During its lifetime an animat moves around its world, sometimes landing on a food cell and eating the food. Ultimately, those individuals that are most successful at finding food are more likely to reproduce.

An animat consists of a feedforward neural network that receives sensory input from the environment, in this case the angle and distance to the nearest cell that contains food. According to its input, the network generates an output action which results in the animat either moving forward by one cell, turning left or right through 90 degrees, or staying still. The neural network architecture, which does not change during an experiment, is shown in figure 5.3. It has four input units and two output units, fully connected to seven hidden units. Two of the four input units receive the angle and distance to the closest food cell, and the remaining two receive the values previously output by the network for the most recent action. The output units are thresholded to produce an action choice coded as two binary digits: 00 = halt; 01 = turn right; 10 = turn left; 11 = advance.

In any single experiment, all animats have the same network architecture and neuron functions; they differ only by having different sets of weight values. Each animat's weights are coded as a chromosome of floating point numbers. Initially, a population of 100 animats is created, each with a random set of connection weights. Each individual is then allowed an existence of 20 lives, a life consisting of 50 actions from a random starting point, in 5 different environments (i.e. with different food distributions). All animats are then assessed and the 20 individuals which have eaten the most food are selected as parents for the next generation. Each of these is reproduced 5 times, and each offspring is subjected to mutation by perturbing 5 weights, selected at random, by a random real value between $\pm 1.0$. This produces a population of 100 new animats. This process of evaluation, selection, reproduction and mutation represents one generation, and the original experiments were carried out for 50 generations. In the experiments described here, runs consisted of 70 generations since it was not clear that a plateau of performance had been reached with fewer generations.

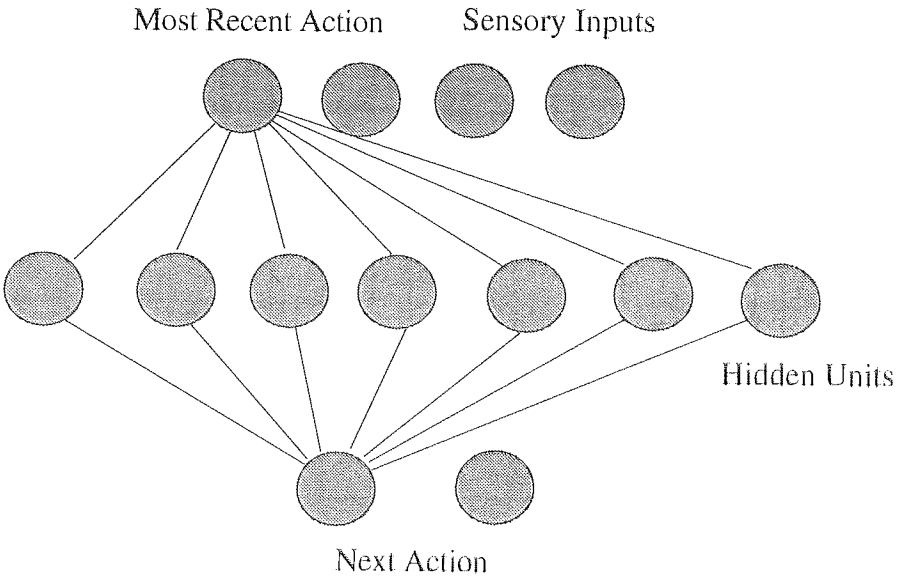



Figure 5.3: The animat network. Only some connecting weights are shown; the network is fully connected between layers.

The animats of the first generation exhibit purely random behaviour; because their network weights are random they only consume food by chance. However, by selecting those individuals that have consumed the most food, and introducing slight mutations into the reproduction of selected individuals as described above, the population evolves and individuals in successive generations become more effective at finding food as useful sets of network weights are discovered. Figure 5.4 shows typical foraging paths for animats from generations 0 and 49. The difference between the random behaviour of generation 0 and the apparently 'purposeful' behaviour of generation 49 can be clearly seen.

In a second set of experiments, individuals are also allowed to learn during their lifetimes, by adapting their weights using standard backpropagation. This introduces an interesting possibility. The evolutionary adaptation is purely Darwinian since it is the initial weights at the beginning of an individual's life that are encoded on the chromosome. No modifications made to these weights during the lifetime of an individual are passed on to the next generation directly. However, learning may increase the chance of an individual being selected if it improves its ability to find food, and so guide evolution through the Baldwin effect. In this second experiment what evolves is not necessarily just the ability to seek food efficiently; it is also the ability to learn effectively during life.

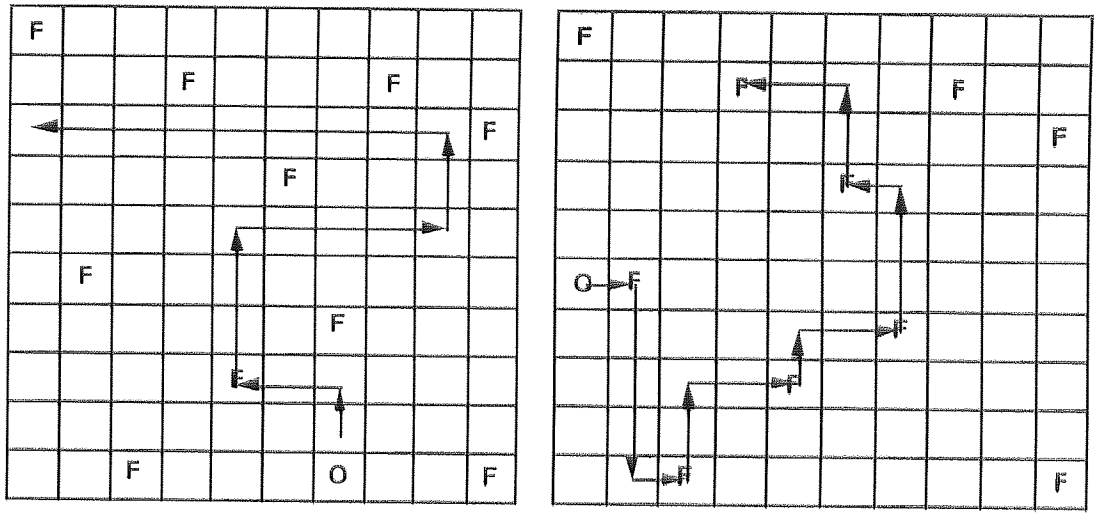



Figure 5.4: Examples of paths followed by animats from (a) generation 0 and (b) generation 49 (After Parisi *et al.*).

In order to use backpropagation to train any network, an error signal on the output units at each time step is needed. However, for this task there is a temporal credit assignment problem since the payoff for finding food may occur many steps after any given move. To avoid this problem, Parisi, Nolfi and Cecconi generate an error signal from what they consider to be a related task: predicting the sensory consequences of the animat's most recent action. Two new output units are added to the network (figure 5.5) and these units are used to predict what the sensory input will be on the succeeding move. The error signal is the difference between this prediction and the actual sensory input after the move is made. In this way it is possible to generate an error signal at each time-step which can be backpropagated through the network, modifying all weights except those between the hidden units and the two original (action) output units.

The experiments reported here follow those of Parisi, Nolfi and Cecconi wherever they gave explicit details. The following choices were made where details were not given:

- The 'angle to the closest food cell' input is measured clockwise relative to the animat's current facing direction, and normalised so that food directly in front of the animat always has an angle of 0.0, food directly to the right an angle of 0.25, and so on.

- If the closest food cell is equidistant to one or more other food cells, then the one at the greatest angle is detected by the animat.

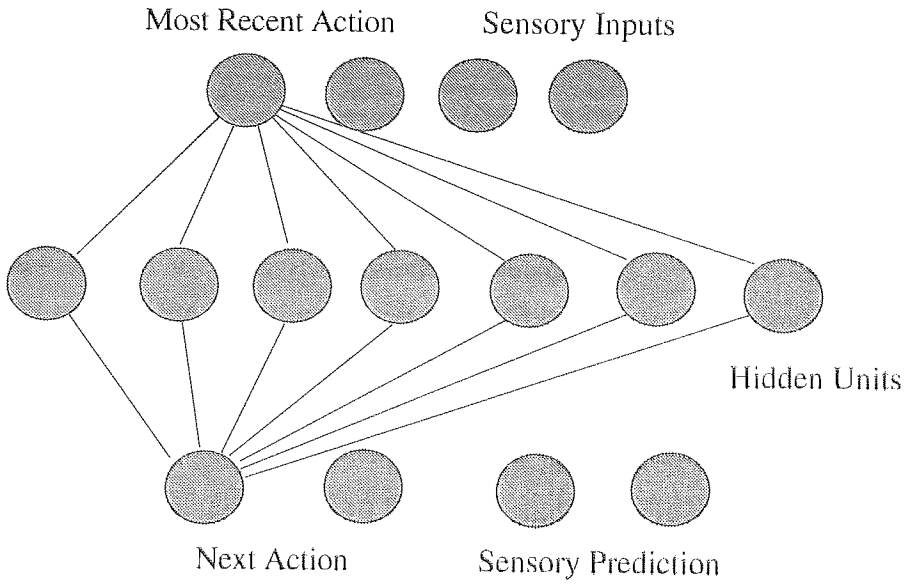



Figure 5.5: The animat network with predictive outputs.

- The distance input is normalised such that the diagonal distance across the 10x10 grid world is approximately 1.0.

- The starting weights for the animats of generation 0 are random floating point numbers in the range ±1.0.

- In the experiments involving backpropagation the most basic algorithm was used, with no momentum term, weight decay, or other convergence improvement strategy. A learning rate of 0.6 was chosen for all the runs reported here.

- At the start of each experimental run, 5 separate worlds with different, random food distributions were created. These worlds remained fixed for the duration of that experiment. In each generation, the animats were each allowed 20 lives of 50 moves in all 5 worlds. At the beginning of each life the world was restored to its initial state (i.e. every food cell was replenished) and the animat was placed in a randomly selected empty cell.

### 5.3.3 Experiments: evolution and learning

All experiments were repeated 16 times with different initial random weights and different worlds in order to obtain statistically meaningful results. All the figures in this section show average results over the 16 runs.

In addition to the evolutionary experiments already described, two benchmark runs were performed in which the animats' foraging strategy was hard-coded. In the

first of these (BM1), the animat simply made a random action choice at each time step. Tested for 20 lives in each of 500 randomly generated worlds, this animat consumed an average of 0.63 food cells in each one. The second benchmark strategy (BM2) was designed to represent an effective foraging behaviour, and consisted of the following rules: if the nearest food lies within a 40° arc in front of the animat, the chosen action is to advance; if the food lies outside this arc and to the animat's left, the animat turns to the left; similarly, if the food is outside the arc and to the animat's right, the animat turns to the right; if the food is directly behind the animat, it turns right by default. An animat employing this strategy, tested for 20 lives in each of 500 worlds, consumed an average of 9.74 food cells in each one. Note that this is lower than the theoretical maximum of 10 foods consumed per life. One reason for this is that an animat's limited sensory information prevents it from planning any kind of optimal tour around the food cells – the greedy algorithm of going directly towards each nearest food cell in turn will result in a sub-optimal tour.

Figure 5.6 shows the mean performance over the population, together with the performance of the most fit individual (peak), as a function of generation, where no learning takes place during the animats' lifetimes. The performance measure is the average number of food items eaten in a lifetime. The peak performance appears to be close to a plateau at around 8.8 food cells consumed. Note that the performance of networks generated after even a small number of generations is much better than the random walk benchmark.

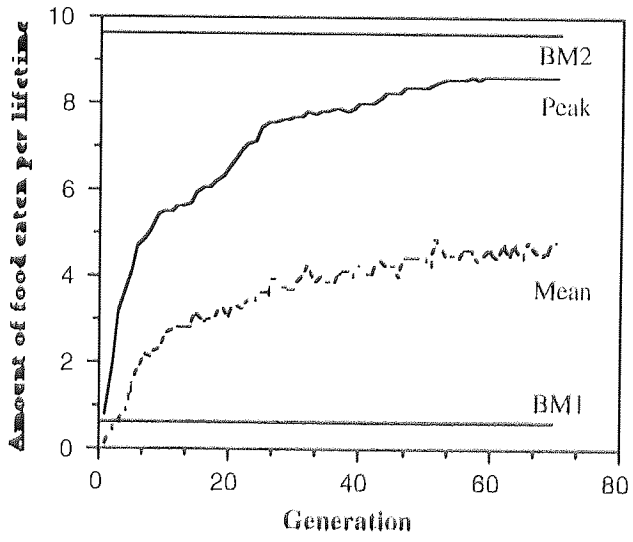



Figure 5.6: Animat performance with no modification of network weights during lifetime.

The effect of allowing the animats to learn during their lifetime can be seen in figure 5.7, which shows the population mean and fittest individual performance as a function of generation.

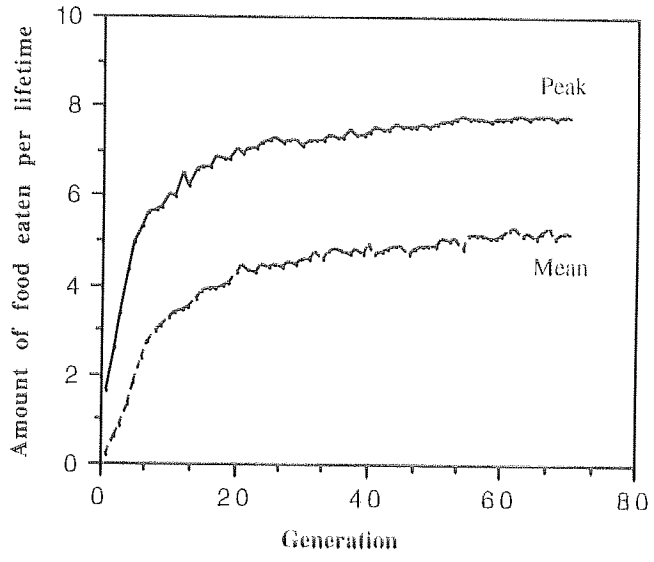
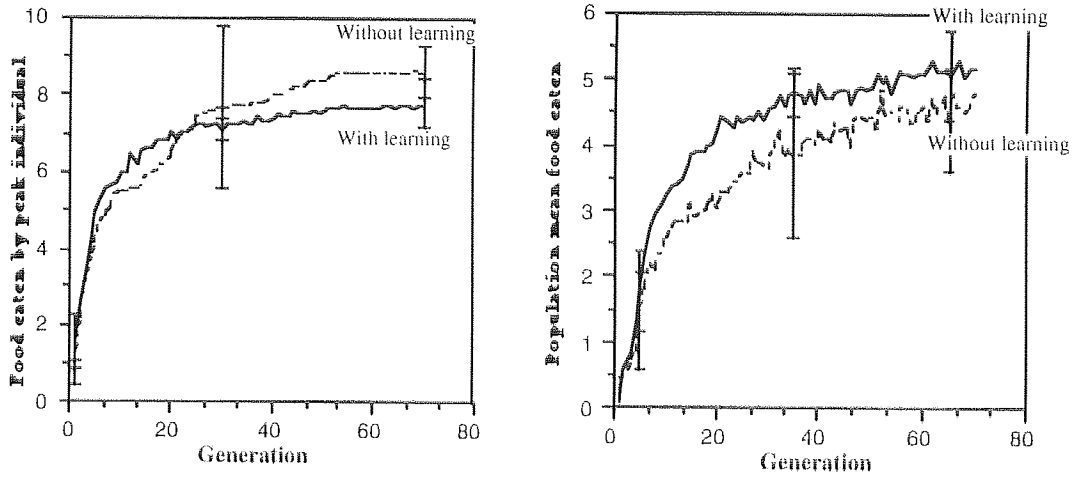



Figure 5.7: Animat performance with predictive learning during lifetime.

Figure 5.8 compares the performance with and without learning for both the peak individual (a) and the population mean (b). Representative error bars (one standard deviation) are shown only at some generations in order to avoid cluttering the picture. In neither case is there a significant benefit resulting from learning during life; if anything, learning appears to be a handicap rather than a benefit. This is in stark contrast to the results obtained by the original authors, who observed a significant increase in performance when learning was included. This difference remains unexplained, but in view of the large number of experiments which have been performed, it seems unlikely that further experiments would reverse these conclusions.

Is it reasonable to expect to see an increase in food-gathering performance as a result of incorporating predictive learning during the animats' lives? It is useful to consider the fitness landscape around a point in weight space (i.e. a possible animat). A point may be considered to have a good surrounding landscape if most of its adjacent points are higher than it is, that is to say a small mutation is likely to lead to an improvement in fitness.




(a) (b)

Figure 5.8: Comparison of (a) peak and (b) mean performance with and without predictive learning

Parisi, Nolfi and Cecconi suggest that, as the predictive task seems correlated to the food-gathering task, learning to predict constitutes a local exploration in weight-space, biasing selection between points of similar fitness in favour of the one with the better surrounding landscape. This causes learning to guide the evolutionary process via the Baldwin effect, as described in section 5.2.4.

It is instructive to consider the learning task itself. The network attempts to learn the relationship between 'action' and 'resultant change in sensory input'. In a world with only one food cell, this relationship is fairly trivial – the change in angle and distance to the food is a simple and consistent function of movement. However, in an environment with many food cells the mapping may become discontinuous due to the sensory inputs only ever detecting the single closest food cell. A step which results in a new food cell becoming the closest will result in a totally unpredictable (from the animat's point of view) change in sensory input. During the early stages of evolution, when the animats' movements are largely random, such steps will occur frequently as the animat moves. This effect is reduced as behaviour evolves to allow efficient movement towards food cells, but the same situation will always occur every time a food cell is found and consumed. This discontinuity in the training data for the predictive task casts doubt over how much useful predictive learning the animat might be able to acquire during its life.

Given the simple nature of the original task, the results reported here suggest that this increased complexity may actually add an unnecessary layer of abstraction between the GA and the actual task being optimised, disrupting the search space and actually hindering the search, as the GA by itself (figure 5.6) seems powerful enough to find near-optimal weight sets in a relatively small number of generations.

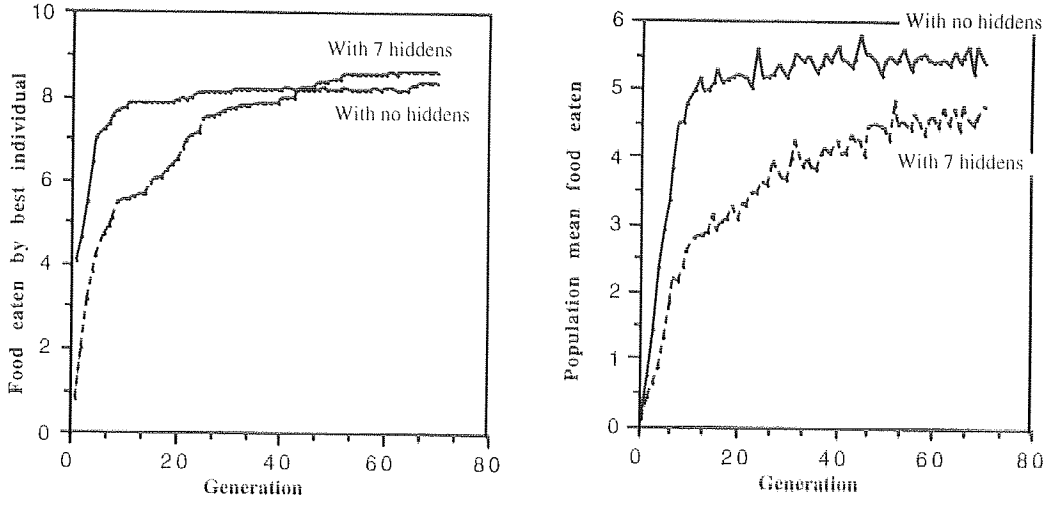
### 5.3.4 Reducing the number of parameters in the network

Parisi, Nolfi and Cecconi performed a further experiment in which the animats' networks were trained on an arbitrary task (the XOR problem) while undergoing evolution, as before, on the basis of their food gathering ability. They report that performance, even on this task, becomes related to fitness for animats which are trained on the task during their lives. They suggest that the learning is biasing evolution towards regions of weight-space in which the performance surfaces for the two tasks (XOR and food gathering) are similar, so that an ascent on one surface would imply an ascent on the other.

For such regions of the space to exist, if the two tasks really are uncorrelated, the number of free parameters in the network must be large enough for the network to encapsulate both learning tasks at the same time. A network has a finite capacity for storing information; by definition, if a network consists of a *minimum* set of weights adapted for one task, then further training of these weights for another, uncorrelated task, can only decrease performance in the original task. Learning the XOR problem during the animat's lifetime only guides the evolution of food gathering ability in as much as it forces the GA to find networks which are not only good networks for food gathering, but are also tolerant of those weight changes caused by training on XOR. This tolerance is only possible if the network architecture contains excess free parameters.

This motivated an investigation into whether the animats' network architecture did indeed contain more parameters than are required for the food gathering task. A set of experiments were performed in which the hidden layer was removed altogether from the animats' networks. Figure 5.9 compares the performance of animats with no network hidden layer with those with the original architecture of seven hidden units. Figure 5.9(a) compares the performance of the most fit individuals, and figure 5.9(b) the performance of the population means, as a function of generation. In this experiment no learning takes place during the animats' lifetimes.

It can be seen from figure 5.9 that the removal of the hidden layer makes no significant difference to the final ability to perform the food gathering task. In fact, the animats with the simpler architecture display a much greater ability in the early stages of the search, and evolve more rapidly, than those with a hidden layer. This can be attributed to the much smaller weight space that the GA has to search. This result demonstrates that the original animat architecture, with seven hidden units, has an excessive number of free parameters for this simple food gathering task. Furthermore, as the simple perceptron (single layer) architecture can learn this task, the food gathering task is likely to be linearly separable (Minsky and Papert 1969).

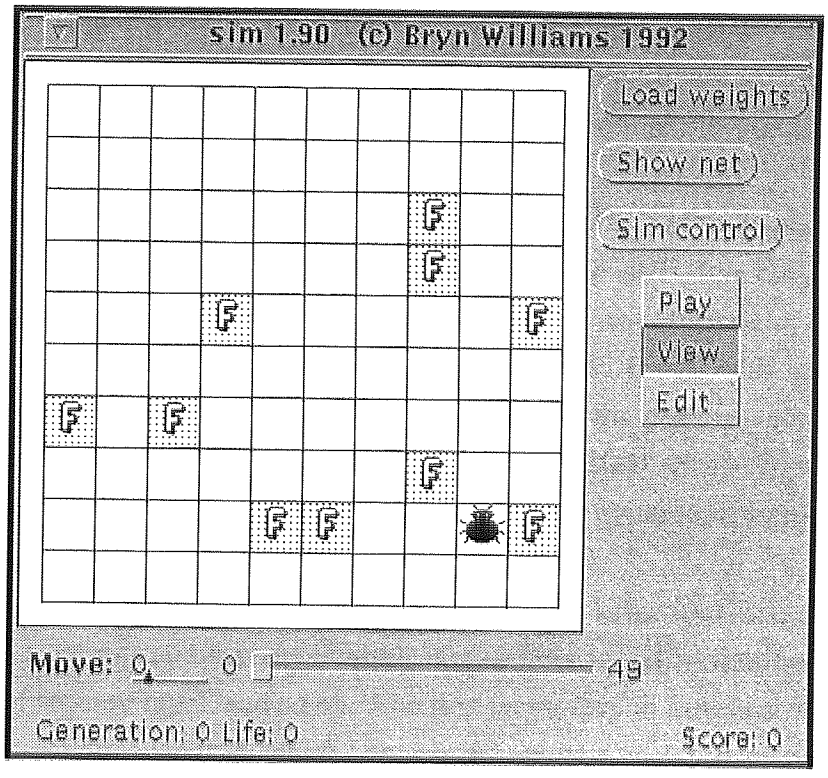



(a) (b)

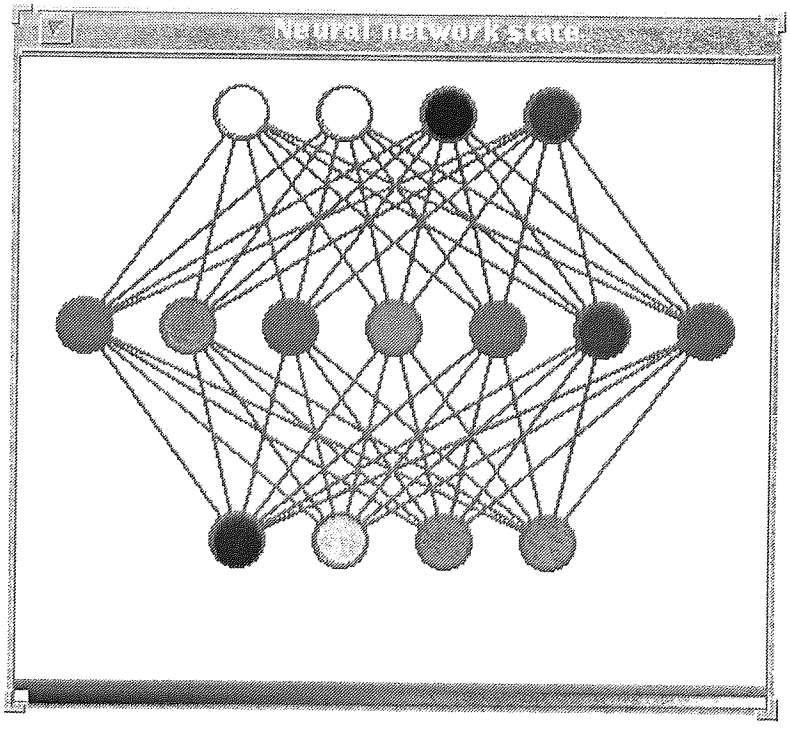
Figure 5.9: Comparison of (a) peak and (b) mean performance with and without a network hidden layer

### 5.3.5 Observed behaviour

The final stage of this work was an examination of the phenomena that Parisi, Nolfi and Cecconi observed and originally explained in terms of 'behavioural self-selection of stimuli'. In their example, a given high-performing animat (i.e. one from a late generation) was found to be much more likely, for example, to encounter food at relatively small angles (i.e. to its right) than at large angles (to its left), as it moved around its world. They also observed that the same animat was more likely to make an appropriate action choice, i.e. one which moved or oriented it closer to the food, if the food was detected at a small angle than if it were detected at a large angle. They concluded that, as the animat influences its sensory input by its movements, food gathering ability had evolved in two parts. On the one hand, the animat had learned to respond effectively to a reduced set of input stimuli, i.e. when the food is on the right, and on the other, it contrived to move in such a way as to encounter situations from this 'known' set of stimuli more often than other situations (such as the food being on the left). As further evidence of this process at work, it was observed that animats which were replaced at a random position after each move were much more likely to make inappropriate action choices, as they were being deprived of the benefit of self-selecting their input stimuli through their movements. This self-selection process was considered to be evidence of the influence of behaviour on the course of evolution.




(a)




(b)

Figure 5.10: The A-life simulation, written for X-windows on a Sun workstation, showing (a) an animat in its environment, and (b) the current state of its neural network 'nervous system'. The simulation proved to be a useful tool for studying animat behaviour.

To investigate the validity of these claims, a graphical software simulation was developed which allowed animat behaviour to be studied qualitatively at various points in evolutionary time and under various environmental conditions. The simulator is shown in figure 5.10.

It was observed that animats from later generations tend to display a reduced behavioural repertoire, relying on turns made in one direction only. A given population might come to contain individuals which only ever turn right, for instance, and so have to make three right turns in order to turn left. Clearly this is not optimal behaviour, and it accounts for the difference between the performance of the best evolved strategies and the hand-coded benchmark strategy BM2 (figure 3.5). An examination of animat behaviour at various stages in the evolutionary process reveals the explanation for this phenomenon.

In the early stages of evolution, every animat's behaviour is random: some animats do nothing, some wander aimlessly, others march relentlessly forward regardless of sensory input, and so on. The first major leap in fitness occurs when an animat evolves with a network which happens to capture a useful correlation between sensory input and motor output. The simplest form of this, and the one which is inevitably discovered first, can be expressed as a single rule such as "If the food is to the right, turn right". Although this rule says nothing about what to do in other circumstances (such as when the food is on the left, behind, or even straight ahead), it captures one important property of the task, which is a big step forward in evolutionary terms. Because of this, the trait confers such a selective advantage that it quickly spreads through the population, until, a couple of generations later, every animat's behaviour is descended from this simple rule. As the population continues to evolve, the basic behaviour is refined, and animats become more proficient at applying the 'right turn' rule appropriately.

The more the behaviour is refined, the less likely it becomes for a contrasting behaviour, such as left-turning, to occur by mutation. By this stage, the movement through weight-space required to reach a network with ideal behaviour, that is the ability to turn either way, is greater than that possible in a single mutation of an existing population member. Unless every intermediate step between a good right-turning individual and an 'ambidextrous' individual represents an improvement in fitness (which the experiments suggest is highly unlikely), the transition cannot occur. This accounts for the convergence of the population on the sub-optimal behaviour of turning only one way.

A close examination, therefore, reveals the observation that animats from late generations only ever turn in one direction as evidence of premature convergence, a common phenomenon with GAs (Baker 1985; Booker 1987; Eshelman and Schaffer 1991). In the experiments described here, the tendency toward premature

convergence is exacerbated by the extremely greedy selection algorithm (keep the best 20%) used by the GA.

Premature convergence in itself is sufficient to account for those observations which Parisi, Nolfi and Cecconi explained in terms of self-selection of stimuli. An individual which only ever turns right will always tend to decrease the angle of a food cell relative to itself. This would account for an observed statistical bias towards encountering small food angles during the course of the animat's life[3]. If the food is on the animat's left (i.e. the angle is large), the animat's chosen action, to turn right, may be highly inappropriate. The same action, however, becomes more appropriate with each repeated right turn. This accounts for the observation that correct action choices are associated with more commonly encountered sets of stimuli (in this example, small angles), and also the observed drop in the frequency with which appropriate actions are chosen if the animat is randomly re-positioned after each move. The principle of Ockham's Razor favours the explanation given here, that these phenomena are all evidence of premature genetic convergence, for its simplicity.

### 5.3.6 Summary

This section has presented the results of a study of the interaction between learning and evolution, based on the experimental scenario described in (Parisi, Nolfi, *et al.* 1991). There seems to be no evidence that learning has a beneficial effect on evolution within the context of these experiments.

The food gathering task in these experiments is learned more successfully by simple perceptrons than by multi-layer networks, which suggests that it is a linearly separable problem, and that the original network architecture was over-specified. Finally, a more simple explanation, that of premature convergence due to loss of population diversity, has been given for the phenomena which Parisi, Nolfi and Cecconi explain as behavioural self-selection of input stimuli.

## 5.4 Conclusions

This chapter began with the observation that the emerging field of A-life contains many natural applications for an evolutionary approach to neural network design. A simple A-life scenario was chosen as the subject of a detailed case-study, the investigation being motivated partly by the desire to identify an abstract problem domain suitable for testing the GA-bumptree. Unfortunately, a detailed study of these experiments revealed several flaws in the original experimental method. These

[3]Note that the 'always turn right' example is used here for the sake of argument. There was no observed bias towards convergence on right-turning behaviour as opposed to left-turning.

include the inappropriate implementation of 'learning' by backpropagating rather arbitrary error signals, and the fact that the network topology is vastly over-specified for the task. The fact that most of the conclusions reported here differ from those of the original authors did nothing to inspire confidence in these experiments as a foundation for further work.

In view of the conclusions of this study, it was decided to abandon the notion of designing a test-bed for the GA-bumptree around these experiments, and to concentrate instead on more 'real-world' problems, such as robotics and control. This is the approach which has been taken in the work reported in the following chapter.

# Chapter 6

# Learning problems II: control and inverse problems

## 6.1 Introduction

In this chapter the GA-bumptree is applied to a number of problems which cannot easily be solved by conventional neural network models such as the MLP or RBF. The problems examined are drawn from two distinct classes of difficult learning tasks: *temporal credit assignment* problems and *multi-valued mappings*. Examples of both of these classes of problems are prevalent in real-world applications, particularly in domains such as control and robotics. Since both classes of problems violate the conditions necessary for most conventional neural network training algorithms to be used, the object of this work is to see if the GA provides a viable alternative.

## 6.2 Temporal credit assignment problems

For supervised learning algorithms such as backpropagation, each input vector applied to the network during training is accompanied by a target vector which specifies what the network outputs should be for that input. Learning is a matter of tuning the network parameters until the desired output is produced for every input vector in the training set. This process, sometimes referred to as *learning with a teacher* (Widrow, Gupta, *et al.* 1973) is suitable for some learning tasks, such as pattern classification, but not for others. The artificial life scenario described in the previous chapter is an example of a learning task where there is a *temporal credit assignment problem* . In this case there is no teacher to specify the correct output for the network (i.e. the correct move for the animat) at each time step, so normal supervised learning is impossible. Instead, the controlling network only receives a single reward signal – the quantity of food the animat has managed to consume – after choosing a whole series of moves. The problem then becomes one of deciding which moves were good and which were bad. Because of the nature of the reward signal, this kind of learning problem is sometimes referred to as *learning with a critic* (Widrow, Gupta, *et al.* 1973).

The genetic algorithm offers a method for training standard neural networks which does not rely on a continuous reward signal. As such, the GA is well suited to adapting neural networks for control tasks where temporal credit assignment problems preclude the use of normal supervised learning algorithms.

### 6.2.1 Pole balancing

The pole balancing problem[1] is probably the most well known benchmark in the field of reactive control, and offers a classic example of a learning task which involves a temporal credit assignment problem. The object is to learn to balance a rigid pole which is attached, with a hinged joint at one end, to a wheeled cart, as shown in figure 6.1. The cart is free to move horizontally along a track of fixed length and the pole is free to rotate about the hinge in the vertical plane. The controller must balance the pole by applying a series of impulses to the cart at discrete time intervals. The magnitude of every impulse is the same; the only choice made by the controller at each time step is whether to apply the impulse to the left- or to the right-hand side of the cart. This method is referred to as 'bang-bang' control, for obvious reasons.

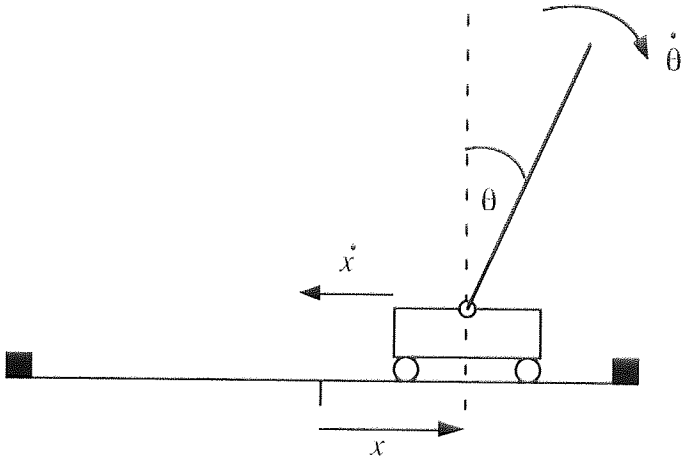



Figure 6.1: The cart and pole system and the four state variables: $x$, $\dot{x}$, $\theta$, $\dot{\theta}$.

The four state variables shown in figure 6.1 are:

$x$: the position of the cart relative to the centre of the track,
$\dot{x}$: the velocity of the cart,
$\theta$: the angle of the pole with the vertical, and
$\dot{\theta}$: the angular velocity of the pole.

At each time step the controller receives the state vector $(x, \dot{x}, \theta, \dot{\theta})$ as input and produces a single binary output specifying in which direction the impulse is to be applied. The controller initially knows nothing of the system dynamics, and does not receive any kind of training signal after each time step. The only evaluative feedback

[1] Also referred to as the *inverted pendulum* problem.

which the controller receives from the environment is a single failure signal generated if the cart hits the end of its track or if the pole falls. Under these conditions, the controller must learn to avoid failure for as long as possible.

The pole balancing problem is a popular benchmark because it represents one of the simplest examples of an inherently unstable system, and because it is simple and computationally inexpensive to simulate. Although the control problem itself is not complex – a simple linear control law (see, e.g. Cheok and Loh 1987) is sufficient to balance the pole indefinitely if the pole angle is initially small – inferring the control rule from the limited failure signal provided by the environment is a difficult learning problem.

### Neural network approaches to pole balancing

In 1983, Barto, Sutton, *et al.* developed a neural network based controller which was able to solve the temporal credit assignment problem and learn the pole balancing task. Their controller is a development of the BOXES system of Michie and Chambers (1968) and consists of a decoder and two specialised adaptive units, the *Associative Search Element* (ASE) and the *Adaptive Critic Element* (ACE). The decoder partitions the state space into 162 disjoint 'boxes', and the ASE generates the control output at each time step according to which box the system state falls into. The ACE's task is to generate a reward signal to train the ASE after each time step based on a prediction of the time remaining until failure. The ACE's predictive mechanism is adapted whenever an actual failure occurs. The ASE/ACE controller is an example of a *temporal difference* (TD) reinforcement learning method, the theory of which is developed in detail in (Sutton 1987). Barto, Sutton, *et al.* (1983) report a computer simulation in which the ASE/ACE controller was able to balance the pole for more than 25 minutes of simulated time after around 100 learning trials, where each trial started from an initial state of (0,0,0,0).

Jervis and Fallside (1992) have applied an ASE/ACE controller to learning to balance a pole on a real rig, in real time. Although this introduces various complications which are not present in simulation, their best controller was able to learn to balance the pole for more than 35 minutes after 255 learning trials. This is an encouraging result which shows that success on this task in simulation can translate to success in the real world. More recently, Moody and Tresp (1994) have described a simplified ASE/ACE controller which does away with the decoder and is able to learn the simulated pole balancing task used by Barto *et al.* after a median of only 2 learning trials, a speedup factor of over 7000 in simulated time.

## Evolutionary approaches to pole balancing

Several efforts have been made to apply evolutionary techniques to designing controllers for the pole balancing problem. Twardowski (1993) compares various reinforcement algorithms for *learning classifier systems* (LCSs) applied to the pole balancing task. LCSs are rule-based production systems in which a GA is used to generate new rules by recombining or mutating existing rules – see (Goldberg 1989) for a general description of LCSs. Whitley, Dominic, *et al.* (1991) have successfully applied a real-valued GA to optimising MLP weights for the pole balancing problem, and Saravanan and Fogel (1994) describe an application of evolutionary programming (EP) to training MLPs for more complicated systems involving balancing two poles on the same cart.

Odetayo and McGregor (1989) have applied a standard binary GA to optimising a simple controller in which the state space is divided into 54 pre-defined 'boxes'. Associated with each box is a binary value indicating whether the force should be applied to the left or to the right whenever the system state enters that box. Each chromosome, therefore, has 54 bits and represents a complete control policy. Finally, some researchers have applied evolutionary techniques to solving simplified versions of the pole balancing problem in which the environmental feedback is more informative than the basic success/failure signal. An example is the work of Karr (1991) in which a GA is used to design a fuzzy logic controller for pole balancing[2]. In this work, the objective is taken to be achieving the state (0,0,0,0) rather than simply avoiding failure, and the fitness function is a weighted sum of the magnitudes of $x$ and $\theta$ over 30 seconds of simulated time.

## Pole balancing with the GA-bumptree

The similarities between the bumptree and some of the controllers which have already been discussed should already be clear: the bumptree partitions the state space into a number of disjoint boxes (one for each leaf node), and within each box is a linear controller responsible for balancing the pole whenever the system state enters that box. For the GA-bumptree, both the partitioning of the space and the parameters of the local models are under genetic control, and adapt to the problem at hand.

The following sections describe three experiments in which the GA-bumptree was applied to the pole balancing problem. Following Barto, Sutton, *et al.* (1983), the cart and pole system was simulated using the following non-linear differential equations:

[2]For a description of fuzzy set theory see (Zadeh 1973) and for an example of a fuzzy logic controller applied to pole balancing see (Berenji and Khedkar 1992).

$$\ddot{x} = \frac{F + ml\left[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta\right] - \mu_c \,\mathrm{sgn}(\dot{x})}{m_c + m} \tag{6.1}$$

$$\ddot{\theta} = \frac{g\sin\theta - \cos\theta\left[\dfrac{F + ml\dot{\theta}^2 \sin\theta - \mu_c \,\mathrm{sgn}(\dot{x})}{m_c + m}\right] - \dfrac{\mu_p \dot{\theta}}{ml}}{l\left[\frac{4}{3} - \dfrac{m\cos^2\theta}{m_c + m}\right]} \tag{6.2}$$

where

$g$ = 9.8 ms$^{-2}$, the acceleration due to gravity,
$m_c$ = 1.0 kg, the mass of the cart,
$m$ = 0.1 kg, the mass of the pole,
$l$ = 0.5 m, half the pole length,
$\mu_c$ = 0.0005, the coefficient of friction of the cart on the track,
$\mu_p$ = 0.000002, the coefficient of friction at the hinge, and
$F$ = ±10.0 N, the force applied by the controller to the cart's centre of mass.

The system state was updated using Euler's method with a step size of 0.02s and the controller was triggered at every time step, giving a controller frequency of 50Hz.

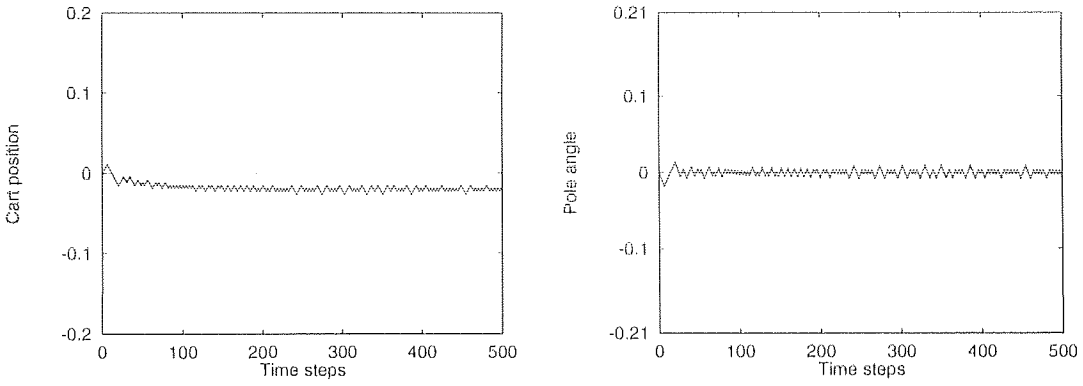
The bumptree controller has four inputs, corresponding to the four state variables, and two outputs, one corresponding to a control force of +10N and the other to -10N. At each time step, the unit with the highest activation determines the force applied to the cart. No attempt was made to optimise the parameters for the GA-bumptree; the parameters adopted in chapter 3 were retained for all the experiments described in this chapter. To recap, the GA-bumptree's control parameters were: 30 nodes in the bumptree, a population size of 400 mapped to a 20x20 grid, a walk length of 5 units, and mutation rate and generation-gap set to 10%.

### Experiment 1

In the first experiment, the controller's task was to balance the pole from an initial state of (0,0,0,0), i.e. starting with the pole vertical, the cart centred on the track, and cart and pole at rest. Failure occurred if the cart position left the interval [-2.4m, 2.4m] or if the pole angle left the interval [-12°, 12°]. These conditions follow the experimental approach of Barto, Sutton, *et al.* (1983) and are typical of many more recent pole balancing studies (e.g. Anderson 1989; Jervis and Fallside 1992; Twardowski 1993).

Each fitness evaluation was based on a single trial in which the bumptree controller attempted to balance the pole for up to 15000 time steps, equivalent to 5 minutes of simulated time. Fitness was simply the number of time steps before failure occurred or the trial was terminated.

Balancing the pole from an initial state of (0,0,0,0) is a trivial linear control problem, and this fact was reflected in the results. The GA inevitably found a bumptree which could balance the pole for 5 minutes in only a couple of generations, and solutions frequently existed in the initial (random) population. Figure 6.2 shows the performance of a typical bumptree controller found by the GA during the first 10 seconds of simulated time. Within the first second (50 time steps) the controller stabilises the system with the cart less than 2 cm from the centre of the track and the pole oscillating within 1° (≈0.02 radians) of the vertical.




(a) (b)

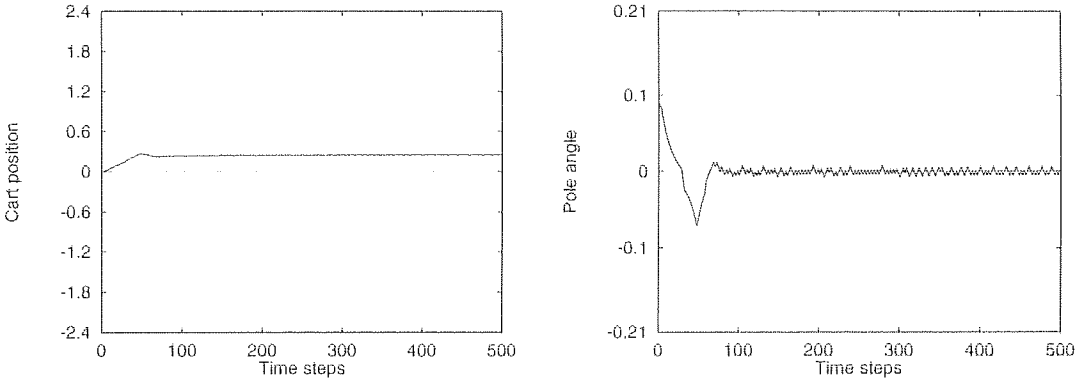
Figure 6.2: An evolved bumptree controller successfully balances the pole from the initial state (0,0,0,0). (a) shows the cart position in metres relative to the centre of the track, and (b) shows the pole angle in radians relative to the vertical. Only the first 10 seconds (500 time steps) of simulated time are shown, since the system remained stable thereafter.

### Experiment 2

The second experiment was intended to test whether the GA could find controllers which could recover if the pole was initially set at a small angle from the vertical. In this experiment each fitness evaluation consisted of two trials in which the initial state of the system was (0,0,0.09,0) for one and (0,0,-0.09,0) in the other. These are the two states which correspond to starting the pole at an angle of approximately 5° (0.09 radians) either side of the vertical, with the cart centred and the cart and pole at rest as before. Fitness was the average of the number of timesteps before failure for the two trials.

As in experiment 1, this problem proved to be simple for the GA-bumptree, with solutions always being discovered in fewer than 5 generations. Figure 6.3

shows the first 10 seconds of simulated time for a typical bumptree controller found by the GA. The controller takes around 2 seconds (100 time steps) to recover the pole from an initial angle of 5° by moving the cart quickly to the right, and the system stabilises with the cart around 25cm from its initial starting point and the pole oscillating slightly about the vertical.




(a) (b)

Figure 6.3: An evolved bumptree controller which can recover the pole from the initial state (0,0,0.09,0). For comparison with figure 6.2, note the change in scale of the vertical axis in (a).

### Experiment 3

In the first two experiments the controllers were able to keep the pole angle small. If the pole angle is outside the interval [-12°, 12°] the control problem becomes non-linear, and thus more difficult (Whitley, Dominic, *et al.* 1991). The third experiment was intended to investigate whether the GA could find controllers which could recover if the pole was initially set to an angle outside the interval [-12°, 12°]. For this experiment the failure condition was relaxed so that failure only occurred if the cart left the track, as before, or if the pole angle exceeded 57° (≈1 radian).

In this experiment each fitness evaluation consisted of three trials, in which the magnitude of the initial pole angles were 0.1 radians (≈6°), 0.2 radians (≈12°) and 0.3 radians (≈17°) respectively. The choice of sign, i.e. whether the pole was initially angled to the left or to the right, was made randomly for each trial. The other state variables were initialised to zero in all cases. As in experiment 2, fitness was the average time before failure for the three trials.

The GA took an average (over 10 runs) of 36 generations to discover a bumptree capable of balancing the pole for the maximum 5 minutes of simulated time from all three starting positions. Figure 6.4 shows the performance of an evolved bumptree controller when the pole is initialised at the maximum angle of 0.3 radians. Unlike in the previous two experiments, the controller does not damp the oscillation of the system so that the cart and pole settle to being almost stationary. Instead, the

controller maintains the system in a relatively large-scale, but stable, oscillation. The cart and pole oscillate with a period of approximately 150 time steps (3 seconds), the cart moving back and forth in the interval [0.3m, 1.5m] and the pole angle varying over the approximate interval [-12°, 17°], i.e. outside the linear control region. As well as the first 10 seconds of simulated time, figure 6.4 shows the results for the entire run (5 minutes, or 15000 time steps), to illustrate that the system is stable. Note that keeping the pole oscillating like this is a perfectly valid solution, no less so than any other. The only criterion on which the controller is judged is its ability to avoid the failure condition, so there is no reason to prefer small oscillations to large ones, for example, so long as the system remains within the allowed bounds.

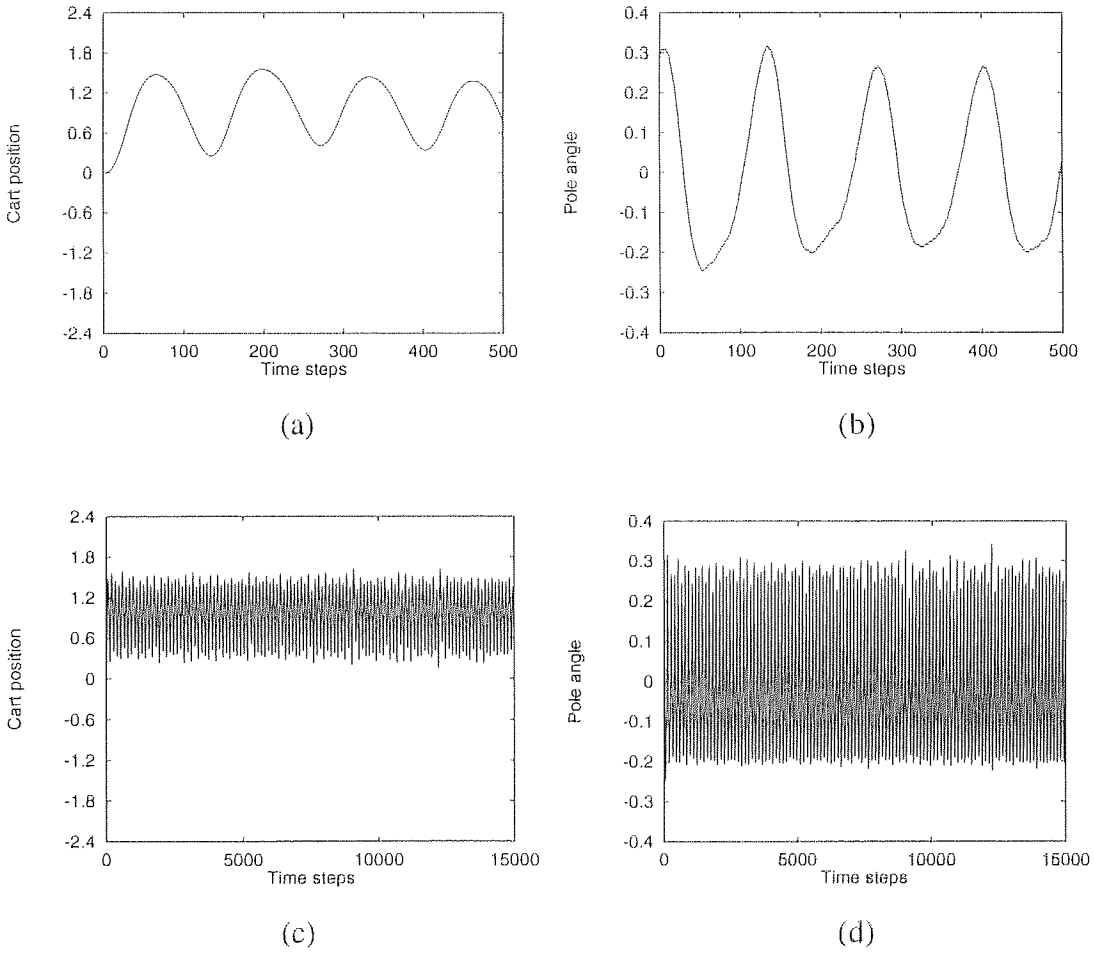



Figure 6.4: An evolved bumptree controller which can recover the pole from an initial angle of 0.3 radians. (a) and (b) show the first 10 seconds of simulated time, while (c) and (d) show the same results but for the whole run of 5 minutes (15000 time steps).

### 6.2.2 Car parking

Another example of a control task in which a temporal credit assignment problem makes learning difficult is the car parking problem described by Schoenauer, Ronald, *et al.* (1993), a simplified variation of Nguyen and Widrow's (1990) 'truck backer-upper' problem (Schoenauer and Ronald 1994). In the car parking problem, a controller must learn to manoeuvre a car from a given starting point in an empty car park into a specified parking space.

Following Schoenauer, Ronald, *et al.* (1993), the car park is defined as the square [-100, 100] x [-100, 100] such that the origin is at the centre of the car park. The state of the system is completely defined by the position and orientation of the car, given by the tuple $(x,y,\theta)$, where $(x,y)$ are the co-ordinates of the centre of the front axle and $\theta$ is the angle of the car measured anti-clockwise relative to the $x$ axis. The geometry of the car and the three state variables are shown in figure 6.5.

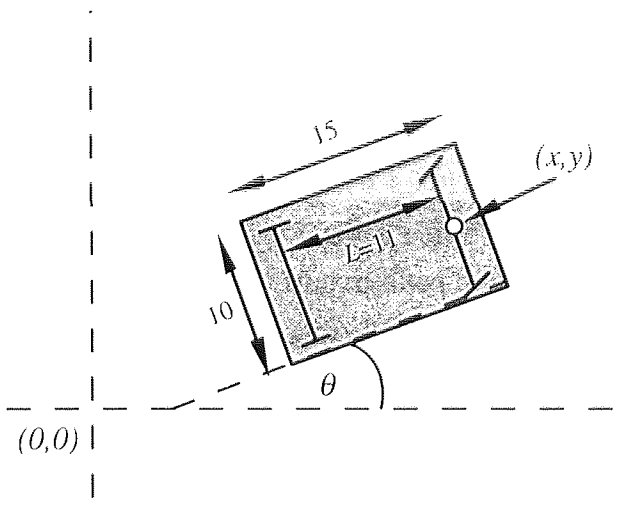



Figure 6.5: The geometry of the car, and the state variables $x$, $y$ and $\theta$. The inter-axle distance $L$ is used in the simulation, described later.

From its initial starting point, the car moves forwards at a constant speed. At each instant, the controller sees the state vector $(x,y,\theta)$ as input and must steer the car by deciding the angle $\phi$ of the front wheels relative to the long axis of the car. The value of $\phi$ may be any angle in the interval [-0.7 radians, 0.7 radians] ($\approx\pm 40^{\circ}$). The object is to park the car in as short a time as possible. In this case, parking is a matter of positioning the car as close as possible to the origin (0,0), in such a way that the car is aligned closely with the x axis ($\theta\approx 0$) and facing to the right (in the direction of increasing $x$).

Following Schoenauer, Ronald, *et al.* (1993), the car moves forward in discrete steps of size $u$=3 units, and the controller produces a new output angle $\phi$ at

each step. The system dynamics may be approximated by the following simple difference equations for the three state variables $x$, $y$ and $\theta$:

$$x' = x + u\cos(\phi + \theta), \quad y' = y + u\sin(\phi + \theta), \tag{6.3}$$

$$\theta' = \theta + \arcsin\left[\frac{u\sin\phi}{L}\right] \tag{6.4}$$

where $L$ is the distance between the front and rear axles (figure 6.5).

### Experiments

For the GA-bumptree, each fitness evaluation consisted of a single attempt to park the car from an initial position[3] of (20,10,-2). The bumptree controller was allowed to drive the car for a maximum of 500 steps, and at each step the distance $d$ from the goal state (0,0,0) was computed:

$$d^2 = x^2 + y^2 + \min(\theta^2, (\theta - 2\pi)^2, (\theta + 2\pi)^2) \tag{6.5}$$

where the expression $\min(\theta^2, (\theta - 2\pi)^2, (\theta + 2\pi)^2)$ was included, following Schoenauer *et al.*, to allow one complete rotation either way. The car was considered to have been successfully parked if any step resulted in $d^2$ becoming less than 1.0, in which case the trial was stopped. At the end of each trial, the fitness $f$ was calculated as:

$$f = \frac{1}{d^2} + \frac{1}{n} \tag{6.6}$$

where $n$ is the number of steps during the trial, 500 if the bumptree failed to park the car with sufficient accuracy, fewer if it succeeded. This fitness function rewards controllers both for accuracy and for efficiency in parking the car. Note that this is still 'learning with a critic', since the only reward signal is a single qualitative evaluation of an entire trial. The GA knows nothing of the system dynamics or the quality of individual control outputs.

[3]These are the initial conditions used by Schoenauer, Ronald, *et al.* (1993).

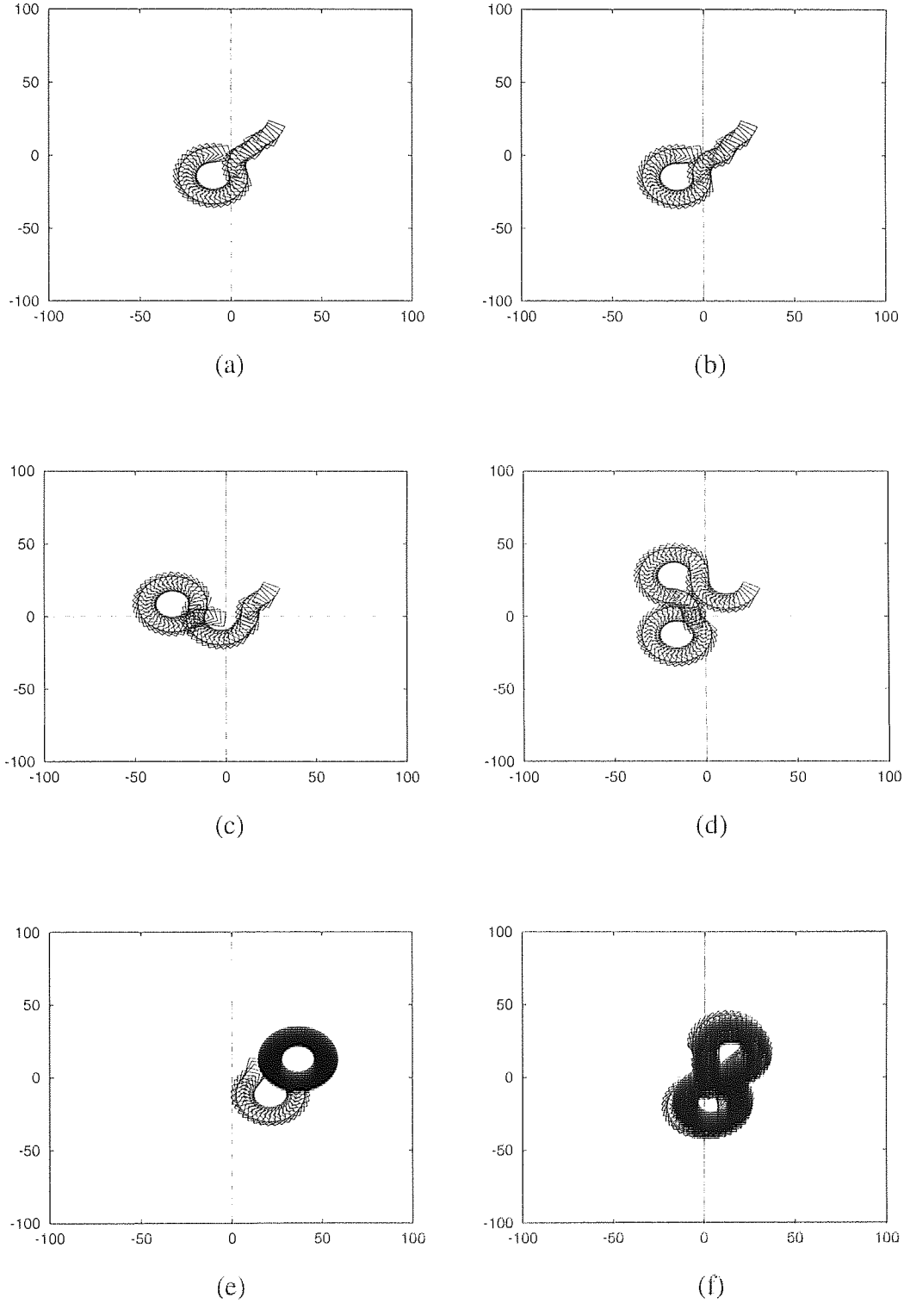



Figure 6.6: Examples of solutions found by the GA-bumptree. In the best cases, e.g. (a) and (b), the bumptree controller is able to park the car accurately and efficiently. Some controllers were accurate, but took less efficient paths, e.g. (c) and (d). Two out of the ten GA runs converged to the controllers shown in (e) and (f), which fail to park the car. Instead, each becomes stuck in a cyclic trajectory, a circular path for controller (e) and a figure-of-eight for (f).

The GA was run for 100 generations, which represents roughly the same number of trials as the GA used by Schoenauer *et al.* All other parameters for the GA-bumptree remained the same as for the pole balancing and classification experiments described earlier. Figure 6.6 shows the parking strategies used by some of the final controllers found by the GA. Out of ten runs, there were two in which the GA failed to find controllers which could park the car to the required degree of accuracy. The remaining runs discovered solutions of varying quality, the best of which are comparable to the best solutions reported by Schoenauer *et al.* Note that the coarse step size used ($u$=3) limits the parking accuracy which it is possible to achieve.

### 6.2.3 Summary

The pole balancing and car parking tasks are examples of control problems where learning is difficult because of limited evaluative feedback from the environment. The experimental results given in the previous two sections demonstrate the feasibility of applying the GA-bumptree to the design of controllers for these kinds of problems. The fact that these problems could be solved with only simple fitness functions and no effort to tune the GA parameters is encouraging.

## 6.3 Multi-valued mappings

In addition to problems where there is a delayed reward signal such as pole balancing or car parking, there are other kinds of learning task which are particularly problematic for normal supervised learning algorithms. One example is the problem of learning multi-valued mappings. There are many practical problems in control, robotics and data analysis where there may be more than one possible correct output for each input vector. A common example is the problem of learning the *inverse kinematics* of a robot arm, that is, the control signals which must be applied to the joint motors in order to move the end effector to a given point in the arm's workspace. The mapping between the co-ordinates of the end effector and the angles of the joints may be multi-valued, that is, there may be more than one combination of joint angles which results in the same co-ordinates for the end effector. An example of this situation is shown in figure 6.7.

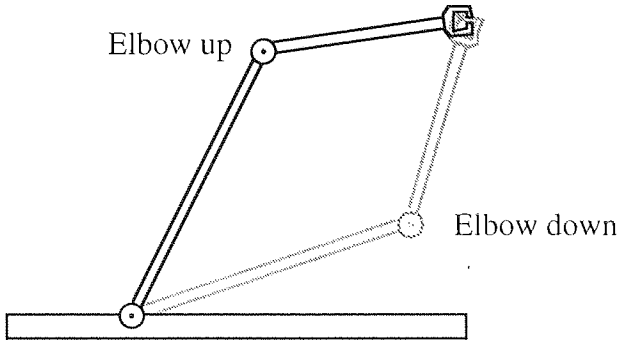



Figure 6.7: The inverse kinematics mapping of a simple two-joint robot arm is multi-valued because there may be two possible ways of reaching the same point. The two alternative configurations shown are referred to as 'elbow up' and 'elbow down' solutions, for obvious reasons.

Neural network models are generally deterministic, so that the output of a trained network is a function of its inputs: a unique output vector is assigned to every input vector. If the mapping to be learned is multi-valued then the training set will contain examples of the same input vector associated with different outputs. Clearly the network cannot learn such conflicting information. By minimising the mean squared error (MSE) over the whole training set, most supervised learning algorithms learn to map each input vector to the average of the target outputs associated with it (Rohwer, Wynne-Jones *et al.* 1994). This approach is based on the assumption that the generator of the training data is inherently uni-modal (i.e. deterministic), and that any contradictions in the data are the result of noise. If this is the case, then the standard error minimisation method yields a model which may be considered optimal (Bishop 1994). If the generator of the training data really is multi-modal, however, learning an average output for each input vector is not necessarily appropriate, since the average of several possible outputs for a given input may not itself be a possible output for that input. Figure 6.8 illustrates this point for the simple robot arm considered above. In this case, averaging the joint angles associated with the two possible arm configurations results in the arm being positioned straight out, with the end effector some distance from the target location.

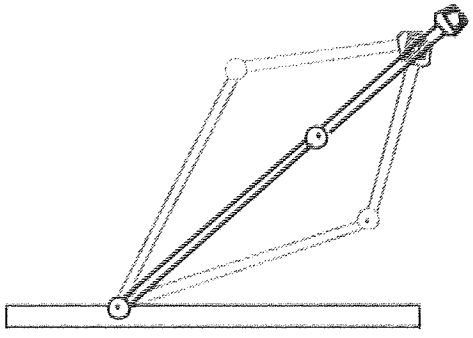


Figure 6.8: For the robot arm, the average of the two solutions to positioning the end effector at a particular point is not itself a solution.

There are two basic approaches to the problem of learning multi-valued mappings. The most general is to abandon functional models such as the MLP, and attempt to model the data generator completely with a multi-modal model. This is the approach taken by Ghahramani (1993), who describes a mixture density model for approximating multi-valued mappings. To model the multi-valued vector mapping $m:\mathbf{x} \rightarrow \mathbf{y}$, Ghahramani uses the EM algorithm to compute the maximum likelihood model of the density of the training data in joint input/output space, $P(\mathbf{x},\mathbf{y})$. For a given input $\mathbf{x}$, this estimate of the joint density can then be used to estimate the complete conditional output density $P(\mathbf{y}|\mathbf{x})$. If necessary, the conditional density can be sampled in one of several ways to give a single output value $\mathbf{y}$, for example the single most likely output for the given input (Ghahramani 1993).

Bishop (1994) takes a similar approach, introducing the Mixture Density Network (MDN), a hybrid model for multi-valued mappings. The MDN consists of an MLP coupled to a Gaussian mixture model. For a given input $\mathbf{x}$ the mixture model represents the conditional probability density of the outputs, $P(\mathbf{y}|\mathbf{x})$. Instead of trying to learn the multi-valued mapping directly, the MLP maps each input vector to a set of parameters for the mixture model, i.e. the mixing coefficients and the means and variances of the Gaussian kernels. Bishop derives a differentiable error function, based on maximising the likelihood of the mixture model, which can be backpropagated through the MLP to train the model. For the MDN, training the MLP fulfils the role played by the EM algorithm in Ghahramani's model.

The alternative approach to learning multi-valued mappings is to use a functional model such as the MLP directly, and to try to learn just a single mode of the data. One possibility is to learn a function mapping each input vector to the single most likely output for that input. Although this approach discards a good deal of information about the statistical properties of the generator, it is often sufficient for practical applications. For the robot arm, for example, what is required is to position the end effector at the desired location. The fact that there are many possible solutions is not necessarily of practical interest, so long as the controller always produces a single combination of joint angles which does the job[4].

Rohwer and van der Rest (1994) take this approach and introduce a new error measure for feedforward networks, based on description length. Their method is based on the observation that, if the data is multi-modal, it is better to learn a good fit to some training points than a mediocre fit to all the points. Unlike MSE, the *Naïve*

[4]This is only strictly true if the only goal is accurate positioning of the end effector. In real applications it is also generally desirable to minimise the amount of movement required. In this case, the best solution for reaching a certain point will not be static, but will depend on the current position of the robot arm.

*Description Length* (NDL) error measure is minimised by maximising the number of training data points which the network models to within a pre-determined distance $\varepsilon$ of the target value. For multi-modal training data, choosing an appropriate value for $\varepsilon$ causes the network to converge on the single most likely mode of the data, effectively ignoring data points generated by other modes (Rohwer and van der Rest 1994). If the data is noisy as well as multi-modal, the choice of $\varepsilon$ is critical. Ideally, the network should average out small variations due to noise, but ignore larger variations which are due to multi-modality. If $\varepsilon$ is too small then variations due to noise will be treated as different modes, and the network will model the noise (overfitting), and if $\varepsilon$ is too large, the network may fail to separate different modes, averaging them instead.

It is trivial to incorporate an error measure such as this into training the GA-bumptree, since there are no constraints on the form or complexity of the fitness function. In the following sections, the GA-bumptree is applied to the two multi-valued mapping problems described in (Bishop 1994).

### 6.3.1 A 1-D test problem

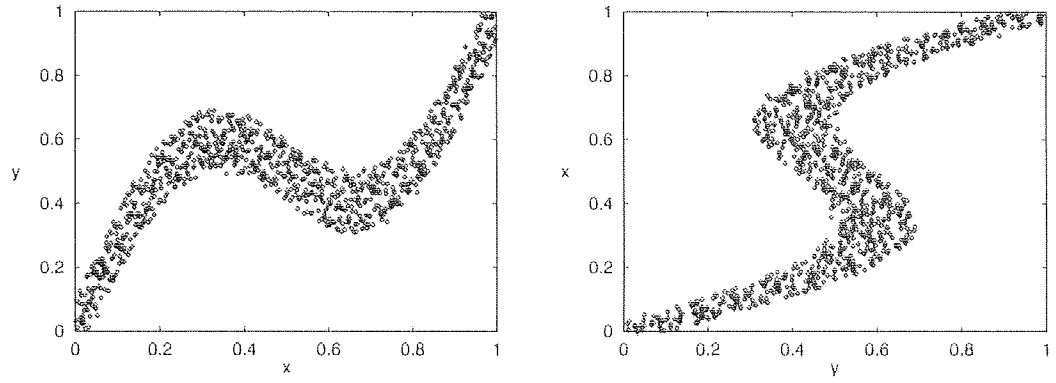
The first problem is a simple mapping between a single input and a single output. The mapping is defined by:

$$y = x + 0.3\sin(2\pi x) + \delta \tag{6.7}$$

where $\delta$ is a random variable uniformly distributed in the interval (-0.1,0.1). In the absence of noise, the forward mapping $m_F: x \rightarrow y$ is single-valued (mono-modal). The corresponding inverse mapping $m_I: y \rightarrow x$ is, however, multi-valued (multi-modal) – there are many values of $y$ which map to more than one possible $x$. The difference is clear from figure 6.9, which shows training data sets for both the forward and inverse mappings. Each data set consists of 1000 points generated by sampling (6.7) at equal intervals of $x$ in the range (0.0,1.0).

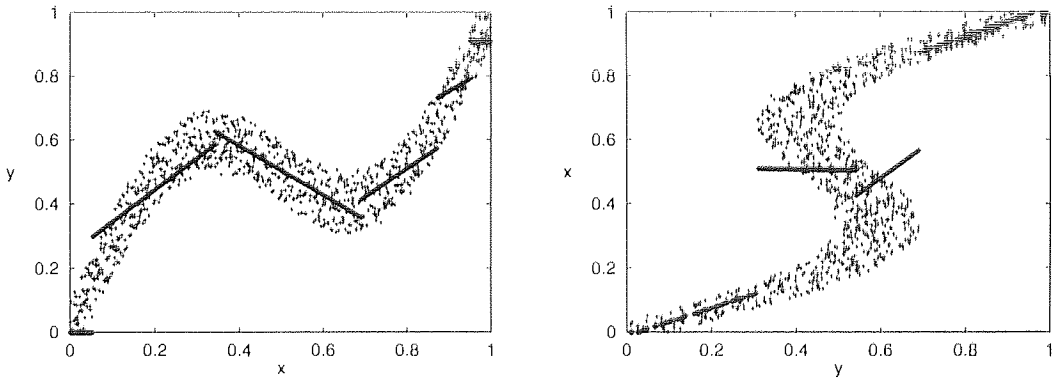
Figure 6.10 shows examples of bumptree models found by the GA for the two mappings, using the same inverse-MSE fitness measure that was used for the classification problems described in chapter 3. In both cases, selecting bumptrees with the lowest MSE has resulted in models which attempt to fit the conditional mean of the target data. In the case of the forward mapping, the resulting model is a reasonable piecewise linear approximation of the generator of the data (although far from perfect, as will be discussed later). For the inverse mapping, however, learning the mean of the targets results in a very poor model of the region in which the data is multi-valued. The bumptree model includes many points which lie completely outside the range of the training data. These results are directly analogous to those obtained

when training an MLP on these data sets using the standard least-squares error measure (see Bishop 1994).




(a) (b)

Figure 6.9: Two versions of the simple test function. Each figure shows training data for the mapping from the variable shown on the horizontal axis to that on the vertical axis. The forward mapping from $x$ to $y$ is well defined (a) but the inverse mapping from $y$ to $x$ is multi-valued (b).




(a) (b)

Figure 6.10: Examples of GA-bumptree approximations of the two test functions. In each figure the bumptree output appears as a number of solid line segments, clearly showing the piecewise linear nature of the bumptree model. For the forward mapping (a) the bumptree is using six local experts, and produces a reasonable approximation of the generator of the data. For the inverse mapping (b), four experts are used, and the approximation is poor. In the region where the mapping is multi-valued, minimising MSE forces the network to try to fit the average of the target values.

With a simple modification of the fitness function, the GA can be made to search for bumptrees which model some of the training data very well, rather than modelling all of it fairly well. The fitness of a given bumptree $X$ may be simply defined as the number of training points which the bumptree maps to within a pre-determined distance $\varepsilon$ of their target values. For the 1-D example:

$$f(X)=\sum_{p=1}^{P}\xi(p), \qquad \xi(p)=\begin{cases}1 & |y_p - T_p| < \varepsilon \\ 0 & \text{otherwise}\end{cases} \tag{6.8}$$

where $y_p$ is the network's output and $T_p$ is the target for each example $p$ out of a total training set of $P$ patterns.

Initial experiments with this simple fitness function used a value of $\varepsilon$=0.1, a threshold distance sufficiently large to accommodate the variations due to noise in the test problem, but small enough to separate the different modes of the data. The results of these experiments were encouraging, but highlighted an immediate problem: since all points which are within 0.1 of their target outputs are rewarded equally, there is no selection pressure to fine-tune the model. The bumptrees found by the GA were effective at selecting a single mode of the data, but did not average out the noise within that mode. This problem can easily be overcome by adding additional terms to the fitness function. In a second experiment, fitness was defined as the sum of the number of points correctly classified to within a series of $k$ decreasing thresholds $\varepsilon_1 \ldots \varepsilon_k$, giving:

$$f(X)=\sum_{p=1}^{P}\xi_1(p)+\xi_2(p)+\cdots+\xi_k(p), \qquad \xi_i(p)=\begin{cases}1 & |y_p - T_p| < \varepsilon_i \\ 0 & \text{otherwise}\end{cases} \tag{6.9}$$

Figure 6.11 shows a typical bumptree model found by the GA using the fitness function given in equation (6.9), with $k$=3, $\varepsilon_1$=0.1, $\varepsilon_2$=0.05 and $\varepsilon_3$=0.02. Comparing the model with that in figure 6.10(b), the improvement is clear. Instead of attempting to fit the average of the target data in the region where the mapping is multi-modal, the bumptree is now modelling a single mode of the data well.

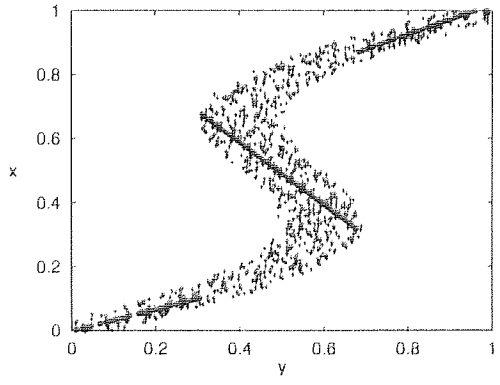



Figure 6.11: A typical bumptree model found using an improved fitness function. The model consists of three experts, and represents a good functional approximation to a single mode of the data.

### 6.3.2 Robot arm inverse kinematics

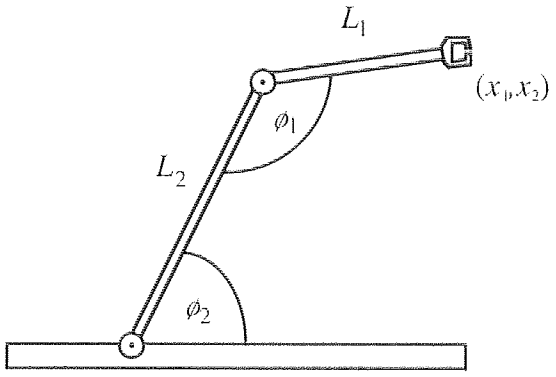



Figure 6.12: A simple two-joint robot arm. The position of the end effector $(x,y)$ is uniquely determined by the joint angles $\emptyset_1$ and $\emptyset_2$. The inverse mapping, from $(x_1,x_2)$ to $(\emptyset_1,\emptyset_2)$, is multi-valued.

The second test problem is the problem of learning the inverse kinematics mapping for the simple two-joint robot arm described at the beginning of section 6.3. For the arm shown in figure 6.12, the position of the end effector is determined by the forward kinematic equations:

$$x_1 = L_1 \cos(\phi_1) - L_2 \cos(\phi_1 + \phi_2) \tag{6.10}$$

$$x_2 = L_1 \sin(\phi_1) - L_2 \sin(\phi_1 + \phi_2) \tag{6.11}$$

Using the same approach as in the previous section, the GA-bumptree was applied to modelling the inverse kinematics of the robot arm. The particular robot configuration studied was that used by Bishop (1994), for which $L_1$=0.8, $L_2$=0.2 and the joint angles $\phi_1$ and $\phi_2$ are restricted to the ranges (0.3,1.2) and ($\pi$/2,3$\pi$/2) radians respectively. Training and test data sets each consisted of 1000 randomly generated pairs of joint angles and their corresponding end effector co-ordinates. The task was to model the two-input, two-output inverse mapping $m_I:(x_1,x_2) \rightarrow (\phi_1,\phi_2)$. Because many points can be reached equally well with the arm either in the 'elbow up' or 'elbow down' configurations, this mapping is multi-valued for a large part of the input space, as shown in figure 6.13.

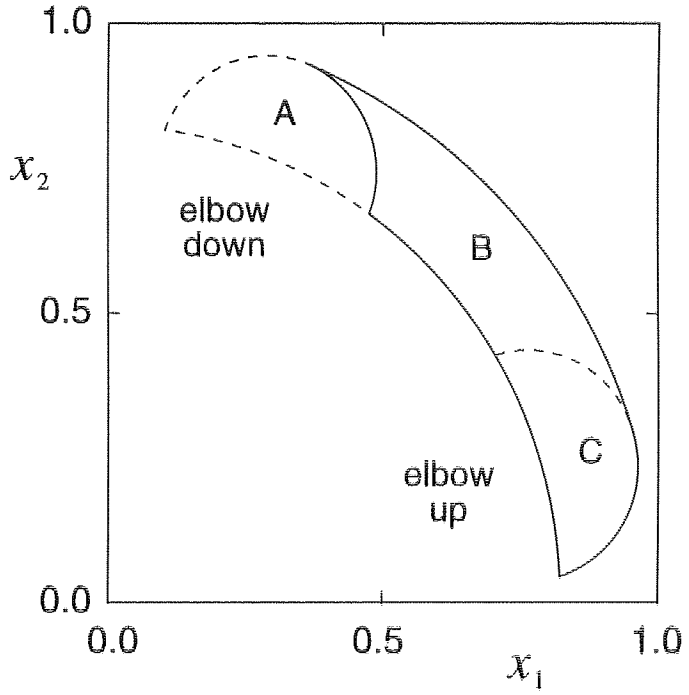



Figure 6.13: The total workspace of the robot arm, where the base of the arm is at (0,0). Points in region A can only be reached with the arm in the 'elbow down' configuration, while those in region C can only be reached in the 'elbow up' configuration. Points in region B can be reached with the arm in either configuration, so the inverse kinematic mapping is double-valued in this region (after Bishop, 1994).

In separate experiments, GAs using two different fitness functions were applied to finding bumptree models for the robot arm inverse kinematics mapping. In the first experiment the simple inverse-MSE fitness function was used, while the second used the function given in (6.9). Figure 6.14 shows typical positioning errors produced by final bumptree controllers from the two experiments.

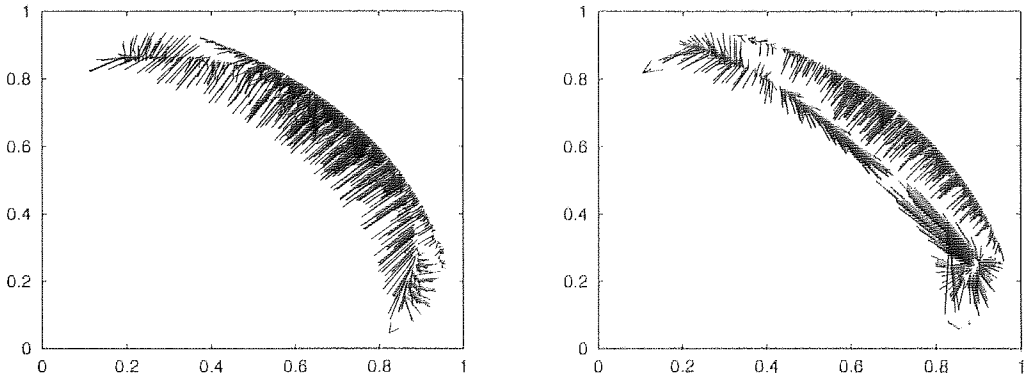



Figure 6.14: Positioning errors for bumptree controllers evolved using (a) inverse MSE and (b) sum of thresholds (6.9) fitness measures. Each line represents the positioning error for one example in the test set, and connects two points: the target point input to the controller, and the actual end effector position produced by using the joint angles output by the controller.

The results shown in figure 6.14 are poor throughout the workspace, even in the regions where the inverse kinematics mapping is single-valued. The fact that the sum of thresholds fitness measure produces a slight improvement in RMS positioning error over the MSE fitness measure offers little consolation – in practical terms, neither GA has found a bumptree capable of modelling the desired mapping. Table 6.1 compares the RMS positioning errors for the bumptree controllers found by the GA with similar results for a 10 hidden unit MLP trained using standard least squares, Bishop's (1994) Mixture Density Network and a 100 hidden unit MLP trained using Rohwer and van der Rest's (1994) NDL algorithm.

| Model | RMS positioning error |
|---|---|
| MLP, least squares | 0.0578 |
| Mixture Density Network | 0.0053 |
| MLP, NDL | 0.0064 |
| GA-BT, MSE fitness | 0.0531 |
| GA-BT, sum of thresholds | 0.0519 |

Table 6.1: Comparison of RMS positioning errors for various connectionist controllers and the two GA-bumptree (GA-BT) controllers.

It is not obvious why the GA-bumptree should perform so poorly on this task. One possibility is that the bumptree itself is unsuited to the problem, perhaps because the robot arm inverse kinematics mapping is particularly difficult to model in a piecewise linear manner. In order to test this hypothesis, a three-layer bumptree was hand-crafted so that the eight leaf-layer nodes partitioned the input space as shown in figure 6.15(a). This bumptree partitions the region reachable by the robot (its workspace, represented by the training data shown in the figure) into eight 'boxes' of roughly equal size. Each of the eight experts in the hand-crafted tree was trained using the original MSE minimisation algorithm described in section 3.2.2, and the resultant test-set positioning errors are shown in figure 6.15(b).

Even though the training algorithm is minimising MSE, and so cannot possibly capture the bi-modality of the mapping, the hand-crafted bumptree performs significantly better than those found by either GA. In the regions where the mapping is single-valued, the positional errors are small. In the multi-valued region, minimising MSE results in the controller attempting to average the 'elbow up' and 'elbow down' solutions, and the arm is always positioned straight out: for every target point in this region, the end effector moves to a point on the arc which delimits the perimeter of the workspace.

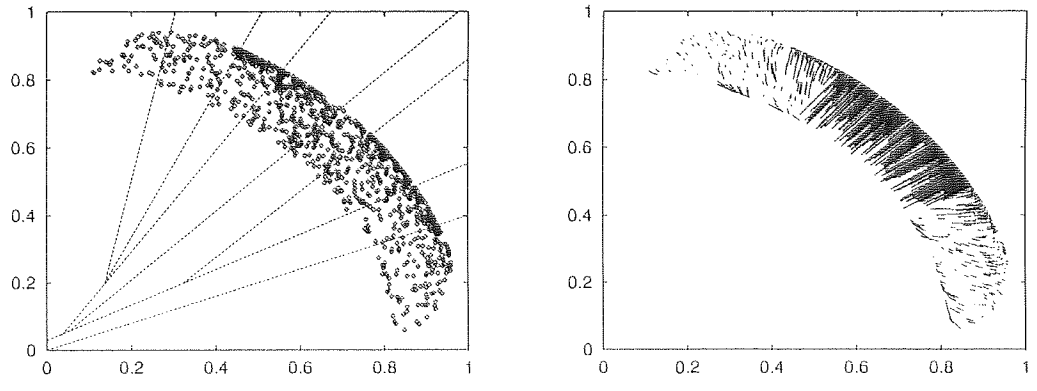



Figure 6.15: Results for a hand-crafted three-layer bumptree controller. (a) shows the partition of the input space, and (b) the positioning errors for the controller after minimising MSE for the eight linear experts.

These results are analogous to those obtained when training a standard MLP on this data (see Bishop 1994). In fact, the hand-crafted bumptree has an RMS positioning error of 0.0393, considerably better than that obtained with the MLP (see table 6.1). This suggests that there is nothing inherently difficult about the robot arm problem as far as the bumptree is concerned, and that the blame for the poor performance lies with the GA: bumptree solutions exist, the GA simply isn't finding them.

The GA must rely on the local stochastic hill-climbing effect of mutation and selection to tune the expert weights, since there is no crossover at the weight level. One possible cause of poor performance is that the fitness surface is not amenable to such hill-climbing. For a local expert in the GA-bumptree, each output node's activation function is a linear ramp in the range (0,1), and is clipped if it exceeds this range. Any value less than 0 is clipped to 0, while any value greater than 1 is clipped to 1. The effect of this clipping is to create flat plains on the fitness landscape, as illustrated in figure 6.16.

Figures 6.16 show 2-D slices through the fitness landscape associated with local experts in the hand-crafted bumptree described above. Figure 6.16(a) shows the fitness surface for one output node (the $\phi_1$ output) of a particular local expert. 'weight 1' and 'weight 2' connect to inputs $x_1$ and $x_2$ respectively. The expert in question is the one responsible for points in the top-leftmost of the eight regions shown in figure 6.15(a), a region in which the inverse kinematics mapping is single-valued. Figure 6.16(b) shows the equivalent surface for the other output node (the $\phi_2$ output) of the same expert. In each case there are large flat plains on the fitness landscape representing weight combinations which cause the node's activation to saturate, i.e. to be clipped to either 1 or 0. Stochastic hill-climbing degenerates to a

random walk on such flat plains, since there is no reward for moving towards the optimum.

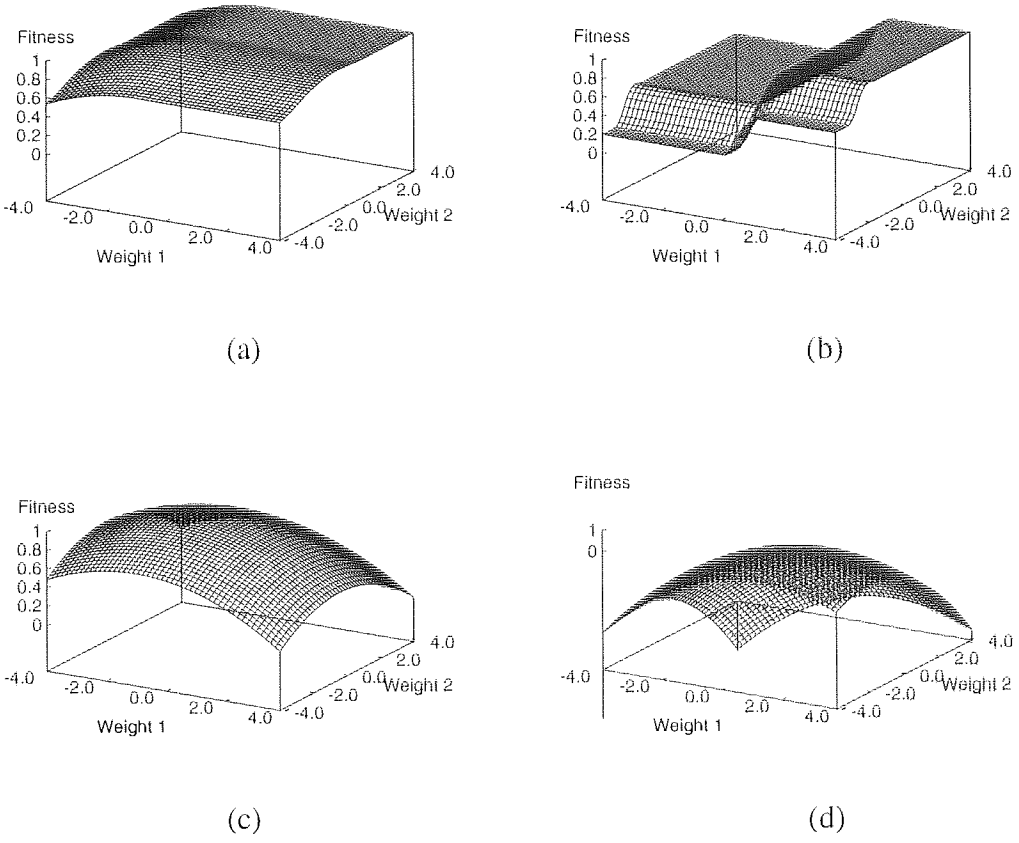



Figure 6.16: Fitness surfaces for the two output nodes of one local expert in the hand-crafted bumptree. (a) and (c) show results for the $\phi_1$ node with and without clipping respectively. The same is true of (b) and (d) for the $\phi_2$ node. For a full explanation, see text.

In order to test whether clipping in the node activation function was the cause of the GA's failure to find solutions, a number of experiments were performed in which clipping was disabled during the GA run. Figure 6.16(c) and (d) show fitness surfaces for the same expert nodes as in 6.16(a) and (b), but with clipping turned off. There are no longer any flat plains at all – each surface is a single smooth hill leading towards the optimum, particularly amenable to hill-climbing.

Unfortunately, disabling clipping during the GA run made no significant difference to the final performance of the bumptrees found by the GA. The only effect was to increase the overall error in the early stages of evolution, since removing clipping allows poorly chosen weights to produce greater errors. It appeared that fine-tuning the expert weights was not the problem. This was supported by further experiments in which various hill-climbing algorithms were applied to the expert weights of the best bumptrees found by the GA. No amount of tuning of the expert

weights of the GA-generated bumptrees produced any significant improvement in performance, and no solution was found whose performance approached that of the hand-crafted bumptree.

An examination of the decision boundaries of bumptrees produced by the GA reveals that the reason for the poor performance is simply that the GA is always becoming stuck in poor local optima. Figure 6.17 shows the way the input space is partitioned for one of the better solutions found by the GA. A single region covers more than 70% of the robot's workspace, and a second region covers the remaining area, all except for a tiny slice at the very bottom. Effectively, the entire inverse kinematics mapping is being approximated by just two linear models. The remaining 14 local experts in the leaf layer are never used.

The problem appears to be a general one, stemming from the fact that the population size of 400 is insignificant compared to the number of points in the 156-dimensional search space. Of the initial random population, the highest performers are always those bumptrees which happen to use only one or two experts to model the whole data set. Put simply, this is because there are many more ways of randomly generating poor sets of expert weights than there are of generating good ones. In the initial random population, those bumptrees which divide the input space between several experts are inevitably poor performers, because the chance of their having good sets of weights for all their experts is infinitesimal. Bumptrees which use only one or two experts to model the whole data set are favoured because a single 'lucky' set of expert weights will give a useful reduction in error (compared to other randomly generated bumptrees) over a large part of the training set.

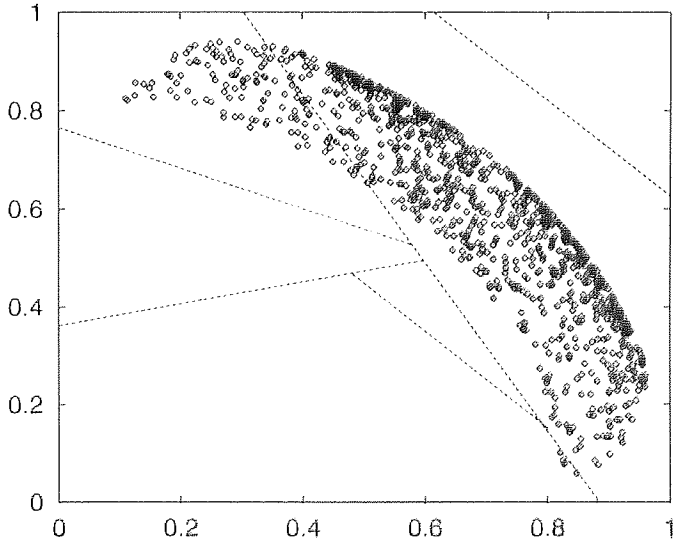



Figure 6.17: Partitioning of the input space by one of the better bumptrees found by the GA. The entire workspace is effectively divided into only two 'boxes', which explains the poor performance.

The robot arm problem reveals the limitations of the GA-bumptree approach. Evolution is inherently greedy, and blindly follows the path of least resistance. The situation is similar to that observed in the A-life experiments described in the previous chapter, where animats always evolved the sub-optimal behaviour of only ever turning one way. In the early generations the whole population converges to the only evolutionary path on which the GA can make headway. In the case of the robot arm, this consists of bumptrees which use only one or two local experts. The remainder of the search is just a case of fine-tuning bumptrees of the same basic design. There is no way back: recombination can only operate on what exists in the population, and mutations which introduce more local experts are inevitably lethal, since the weights of the new experts will not have been adapted at all.

### 6.3.3 Summary

Learning problems which involve multi-valued mappings are common in many real-world applications, and present particular difficulties for established neural network training algorithms. This is because most training algorithms (such as backpropagation) are based on minimising MSE and learn the conditional mean of conflicting target values, which may not itself be a valid target. The experiments described above illustrate that the GA's fitness function may easily be tailored to produce networks which map the single most likely mode of multi-modal training data, making the GA a feasible alternative training method for these kinds of problems.

## 6.4 Conclusions

The experiments described in this chapter demonstrate how the GA-bumptree can be applied to learning problems which have proved problematic for more established neural network training algorithms. Two classes of learning problems were investigated: problems of temporal credit assignment, typified by the pole balancing and car parking tasks, and problems of learning multi-valued mappings, namely a simple multi-modal 1-D mapping and the inverse kinematics of a two-joint robot arm.

The results in this chapter illustrate both the general feasibility of the approach and the limitations of the present GA-bumptree model. The models found by the GA for the 1-D mapping are somewhat sub-optimal (figure 6.10), and the GA reliably converges on a particularly poor local optimum in the case of the robot arm inverse kinematics problem.

The robot arm experiments were important because they showed that good bumptree models did exist for the problem, but that the GA was unable to find them.

This motivated fresh consideration of evolutionary optimisation and GA methodology in general, and led to the final piece of work undertaken during this project, a re-examination of the basic role of recombination in genetic search and the introduction of a new, generalised recombination operator. This work is described in the following chapter.

# Chapter 7

# Linkage mapped crossover

## 7.1 Introduction

The work described in this chapter was undertaken in the closing stages of the research project. This work was originally motivated by concerns about whether the subtree crossover described in chapter 3 is the most appropriate recombination operator for the GA-bumptree, or whether a better operator could be designed. Consideration of this issue raised fundamental questions about the design of recombination operators for *any* GA application, and prompted a more general examination of the role of crossover in GAs.

The following section defines genetic linkage, a central theme in this chapter, and section 7.3 gives a detailed examination of the relationship between genetic linkage and recombination in GAs. A new recombination operator, linkage mapped crossover (LMX), is introduced in section 7.4 and shown to be a generalisation of existing crossover operators. Section 7.5 proposes a method for extending this operator to produce ALMX – a generalised adaptive recombination operator which is able to infer genetic linkage relationships automatically during the course of a GA run. In section 7.6 the ALMX operator is tested on two binary problems based on the Royal Road functions of Mitchell, Forrest, *et al.* (1991), and the results are discussed.

## 7.2 Genetic linkage

Before going any further it is necessary to clarify precisely what is meant by the term *genetic linkage* within the context of this chapter. Genetic linkage exists when the effect of one gene depends on the values (i.e. the particular alleles) of other genes, a situation analogous to *epistasis* in biology. Any degree of linkage between two genes is possible, from very strong linkage, where the slightest change in one gene makes a profound difference to the effect of the other, to no linkage at all if the effect of each gene is completely independent of the other. Genes which are strongly linked form co-adaptive groups. Each group makes a contribution to fitness which is largely independent of any genes outside the group. The fitness of the group is determined by the interactions of all the genes within it – certain combinations of alleles may be particular good, others particularly bad, but no single gene within a co-adaptive group can be evaluated in isolation.

The principal claim on which the work in this chapter is based is that the correct role of crossover is only to exchange complete co-adaptive groups, since these are the only building blocks which may reasonably be treated independently. Some operator is certainly needed to search for good combinations of alleles within each group, but

crossover isn't it. Swapping some alleles within a co-adaptive group from one string to another is a meaningless operation, since the effect of those alleles within the context of the new string is likely to be unrelated to their effect in the original string. Mutation is a more appropriate operator for discovering particular combinations of alleles which *form* good building blocks. Once they exist in the population, crossover can perform its proper role of bringing the best building blocks together to form a complete solution.

The degree of linkage between genes in a GA is a direct reflection of the interdependencies which exist between the parameters to be optimised. As an example, consider applying a GA to the problem of optimising a number of parameters for a car design. There would be little or no linkage between the genes for headlamp shape, tyre width and seat shape, for instance, because these parameters seem to be independent – a comfortable seat is a comfortable seat, regardless of how wide the tyres are. On the other hand, the gene for tyre width is likely to be strongly linked to the genes for engine capacity and brake design – larger engines and/or better brakes both benefit from the greater traction offered by wider tyres.

If every parameter to be optimised is completely independent, then no linkage exists at all between the genes in the GA (unless more than one gene is used to code each parameter). In this case there is no benefit in using a GA in the first place – it is more appropriate to optimise the parameters one at a time. There is also no benefit in using a GA if the opposite extreme is true, and every parameter depends strongly on every other. In this case there are no independent building blocks which crossover can exchange – in fact this kind of problem is liable to be intractable for any optimisation algorithm except exhaustive search. The domain of the GA lies somewhere in between, in the case where some small groups of strongly linked parameters exist which crossover can assemble into larger, more weakly linked groups, and ultimately into a complete solution. If strong linkage effects exist on a larger scale than the size of the building blocks exchanged by crossover then the problem may appear to be deceptive, with apparently good building blocks combining to form poor solutions due to dependencies between them.

## 7.3 Genetic operators and linkage

Any crossover operator which tends to keep some groups of genes together while splitting other groups up is making certain assumptions about genetic linkage. The ability of the GA to find good solutions will depend largely on how justified these linkage assumptions are for the particular problem being solved. At this point it is useful to review a selection of existing genetic operators in order to make explicit the assumptions that each one makes regarding genetic linkage. Section 7.3.1 begins by

examining the most popular crossover operators, all of which are based on static assumptions about the linkage between genes. Section 7.3.2 goes on to describe two attempts to design adaptive operators, operators which aim to discover appropriate linkage relationships as the population evolves.

### 7.3.1 Non-adaptive operators

#### Uniform crossover

By treating each gene as an independent entity, Syswerda's (1989) uniform crossover represents the baseline operator as far as linkage is concerned. This operator is based on the assumption that no linkage at all exists between the different genes or, more precisely, that no two genes are linked more than any other. Because this is not generally the case (some degree of linkage exists in any problem worth applying a GA to), uniform crossover is often considered to be overly disruptive. Parameterised uniform crossover (Spears and De Jong 1991) allows the degree of disruption to be tuned by limiting the proportion of genes exchanged during each recombination, a parameter referred to a $P_0$. This approach accepts that some genes are likely to be linked, but makes no assumptions about which ones.

#### 1-point crossover

Holland's (1975) original 1-point crossover operator introduces a significant assumption about genetic linkage, the assumption that the closer genes are on the chromosome, the more tightly linked together they are[1]. This assumption, generally stated in terms of the building block hypothesis, is so familiar that it is easy to overlook some of its more subtle implications.

One problem with this view of genes linked together as a linear sequence is that there is no room in the picture for more than two genes to be equidistant, i.e. equally linked. As far as parameter optimisation goes, this is a completely arbitrary restriction – there is no reason to suppose that, in general, no more than two parameters will be equally interdependent. Consider designing a GA for a multi-parameter optimisation problem in which three of the parameters, A, B and C, are known to be highly interdependent. One-point crossover simply cannot capture this three-way linkage. The best that can be done is to code the parameters as three consecutive genes, but this immediately implies that A and B are twice as strongly linked as A and C (since crossover will tend to separate A and C twice as often as A and B).

---

[1] For the example of the car given earlier, this operator assumes that the genes for wheel size, engine capacity and brake design are coded close to each other on the chromosome.

Finally, 1-point crossover treats genes near the ends of the chromosome differently to those near the centre. For example, a gene near the centre of the chromosome is adjacent (and so equally linked) to two other genes, while a gene at the end is adjacent to only one.

### Multi-point crossover

Multi-point crossover operators make the same basic assumptions, and impose the same kinds of limits, as 1-point crossover. Linkage between two genes is assumed to be inversely related to their separation on the chromosome, as before. One important difference, however, is that some multi-point crossovers treat the chromosome as a ring, with the last gene joined to the first. This removes the discrepancies introduced by 1-point crossover for genes near the end of the chromosome. However, care must be taken when designing the GA to ensure that the assumption that strong linkage exists between genes at opposite ends of the chromosome is justified.

As the number of crossover points is increased, the individual sections of chromosome crossed over become shorter, which corresponds to assuming that linkage decays more sharply with distance. In the limit, where a crossover point exists between every pair of genes, this operator effectively crosses over every other gene – a particularly unlikely linkage assumption.

### Inserting introns

Levenick's (1991) idea of inserting introns, sequences of non-coding genes, into the chromosome is one way of incorporating additional linkage assumptions into the standard GA. By inserting introns between groups of genes which are thought to be interdependent, standard one- or multi-point crossover operators can be biased against separating genes within these groups. This requires, of course, that those genes which are linked are known in advance, and assumes the usual relationship between separation and linkage within each group.

### Random respectful recombination

Radcliffe (1991) argues that any recombination operator should *respect*, *properly assort* and *strictly transmit* parental genes to form the child, and introduces the random respectful recombination operator ($R^3$) based on these principles[2]. These principles are best illustrated with simple examples: respect, for instance, can be stated at the principle that if both parents have webbed feet, their children must always have webbed feet. Similarly, strict transmission states that if one parent has webbed

[2]Although $R^3$ does not guarantee strict transmission (Radcliffe 1991).

feet and the other hooved feet, their children must only ever have feet which are either webbed or hooved (i.e. every gene in the child must have come from one or other parent). Proper assortment states that if one parent has webbed feet and gills, and the other hooved feet and lungs, then it must be possible to produce children with webbed feet and lungs, or hooves and gills. What is important to note here is that these are basic ground-rules for recombination, and do not introduce or rely on any assumptions about genetic linkage. If it is known that some genes are co-adaptive, appropriate linkage assumptions can be incorporated without violating these basic principles. In the above example, an operator could state that if a child inherits webbed feet from one parent, then it should be highly likely to inherit that parent's gills as well. $R^3$ makes no such assumptions about linkage at all, and actually reduces to uniform crossover for binary GAs (Radcliffe 1991).

### 7.3.2 Adaptive operators

#### Inversion

Holland (1975) was not unaware of the problems with the linkage assumptions on which his original 1-point crossover was based. The solution he proposed was the use of an additional genetic operator, inversion, which randomly changes the order of genes on a chromosome, and thus the implied linkage between them. Allowing genes to be re-ordered requires a change in the representation scheme, since it becomes impossible to identify a gene's function by its position on the chromosome. Holland proposed an order-independent representation where each locus on the chromosome is assigned an index which specifies which parameter it represents.

With this representation, each chromosome in the population may have its own particular order for the genes. Each chromosome therefore carries its own assumptions about which genes are linked (those which are close together) and which are not. When two chromosomes mate, the child's order is determined by the parent which happened to be selected first. The other parent's chromosome is temporarily re-ordered to match[3], and crossover occurs as usual.

The role of inversion is to generate new guesses about the nature of the genetic linkage for the GA to try out. The idea is that those orderings which most accurately capture the linkages between genes will tend to survive, and those which don't will die off. In this way the GA is able to discover the linkage relationships between genes, and adapt to preserve them[4].

---

[3]It has to be, or else some genes will be duplicated and some omitted.

[4]At first sight this approach may seem paradoxical: if the order of a chromosome has no effect on its fitness (which it cannot, by definition, or inversion would not work) then selection is blind to order, in which case why should good orderings be expected to prosper and poor ones to die? The answer is

### Punctuated crossover

Schaffer's (1987) punctuated crossover is a variation of multi-point crossover which attempts to discover appropriate patterns of linkage during the course of the GA run. Although punctuated crossover requires additional information to be coded on each chromosome, it does not require an order-independent coding. Unlike standard operators, punctuated crossover does not randomly distribute a fixed number of crossover points along the chromosome. Instead, each member of the population has appended to it a binary mask, the same length as the original chromosome, representing a particular set of crossover points. When two parents mate, the first action is to merge their masks in a logical OR operation to produce a combined set of crossover points. Thereafter, a child is assembled by starting at one end of the chromosome and taking genes from one parent until the first crossover point is reached, then from the other until the next crossover point, and so on. The child also inherits some crossover sites from each parent to form its own mask.

Because each chromosome carries its own crossover mask, i.e. its own set of linkage assumptions, selection operates on the masks as well as the original chromosomes. The masks are initialised randomly at the start of the run, and the idea is that the GA searches for good sets of crossover points in parallel with its search for good sets of genes. As with inversion, each time two parents mate, a new set of linkage assumptions is tried. In punctuated crossover, however, these assumptions are slightly different from the norm. For each consecutive pair of crossover points, the implication is that all the genes in between are equally linked. In addition, the usual assumption that genes which are at opposite ends of the chromosome are not linked does not hold. As an example, an even number of crossover points implies that the two genes at extreme ends of the chromosome are tightly linked, since if one is copied into the child, so must be the other, whereas an odd number of crossover points implies the opposite.

## 7.4 Linkage mapped crossover (LMX)

In the previous section it was shown how various commonly-used crossover operators make different (often strikingly different) assumptions about genetic linkage. Given that recombination is the GA's trademark operator, and supposedly the algorithm's principal search mechanism, it is perhaps surprising how little

---

that, while the number of offspring produced by a chromosome will be unaffected by the order of its genes, the fitness of the offspring will. If the order is poor, then crossover will split up co-adaptive sets of genes and the offspring will be of low fitness. These offspring will soon die off, taking their parents' inappropriate linkage assumptions with them to the grave.

attention is paid to the issue of genetic linkage in most GA applications reported in the literature. Adaptive operators such as inversion and punctuated crossover appear to have been almost universally ignored by GA practitioners. The typical approach seems to be to code the parameters to be optimised in an arbitrary order on the chromosome, try one or more of the standard crossover operators in turn, and hope for the best.

Linkage mapped crossover (LMX) is a generalised recombination operator in which the degree of linkage assumed to exist between each gene and every other is defined explicitly by a *linkage map*. For chromosomes of length $n$, the linkage map is a stochastic matrix of $n$ rows by $n$ columns. Each row represents the *linkage vector* for a single gene, a probability distribution representing the degree to which that gene is linked to each of the other genes on the chromosome.

When two chromosomes mate using the LMX operator, a fixed number of genes are exchanged. This number, $N$, is a parameter of the operator which must be chosen in advance, analogous to $P_0$ for parameterised uniform crossover. In LMX, the genes to be crossed over, $g_0,...,g_N$, are chosen according to the following algorithm:

- A random gene on the chromosome is picked. This is the first crossover site, $g_0$.

- The linkage vector associated with $g_0$ is copied into a temporary vector $\mathbf{v}$. Each element of $\mathbf{v}$ represents the degree of linkage which exists between $g_0$ and one of the other genes on the chromosome.

- The remaining genes, $g_1,...,g_N$, are sampled probabilistically according to the probability distribution represented by $\mathbf{v}$. Since the same gene cannot be picked twice, each time a gene is chosen the appropriate element of $\mathbf{v}$ is set to zero and the vector is re-normalised.

This process is illustrated in figure 7.1. 7.1(a) shows the first gene to be picked, chosen randomly, and the probability distribution provided by that gene's linkage vector. In this example, $g_0$'s linkage is assumed to decay with distance on either side of the gene. Although the gene's linkage to itself should, in theory, be maximal, this element of the probability distribution must be zero, since the same gene cannot be picked twice (all leading diagonal elements of the linkage map are zero for this reason). 7.1(b) and 7.1(c) show the effects of selecting the next two genes, and 7.1(d) shows the final set of genes which are chosen to be crossed over if $N$=8.

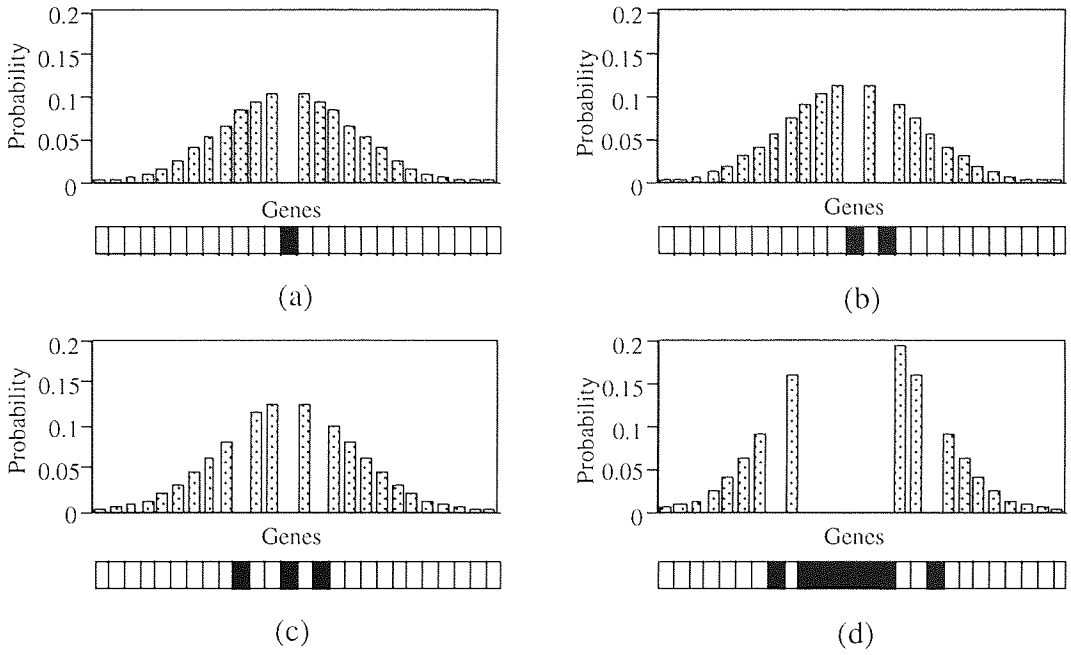



Figure 7.1: Selecting crossover sites for LMX. Figures (a) to (d) show progressive stages in the selection process. Each figure shows the crossover sites which have been chosen so far, and, above, the probability distribution according to which the next crossover site is sampled.

Because each gene has its own linkage vector, any known or supposed interdependencies between optimisation parameters can be incorporated into the LMX operator, simply by crafting the appropriate linkage map. This generality naturally extends to representing the linkage assumptions made by the standard genetic operators. Figure 7.2 shows linkage vectors which are equivalent to three standard crossover operators[5].

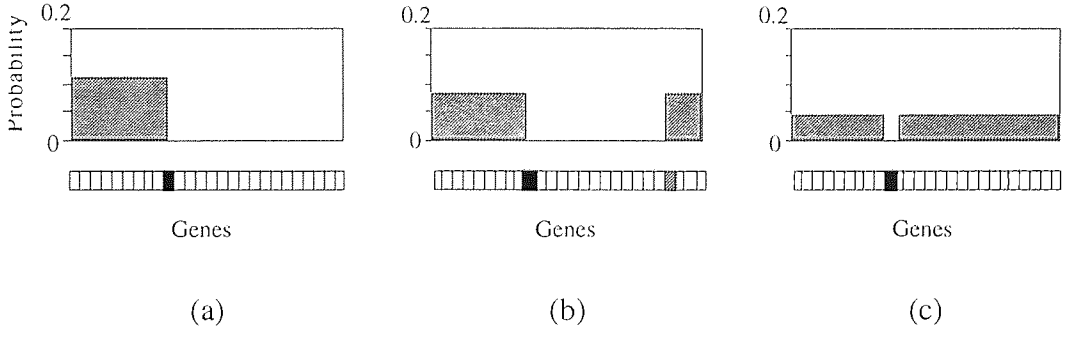



Figure 7.2: Three possible linkage vectors for the same gene, representing the effect of three standard crossover operators: (a) 1-point crossover, (b) 2-point crossover (in which the chromosome is treated as a ring), and (c) uniform crossover.

[5]Actually, certain implementation details would have to be changed for LMX to exactly emulate, say, 1-point crossover. An example is the fact that the number of genes exchanged in 1-point crossover is not constant, but depends on the starting point chosen. In principle, however, the generality of the linkage map approach should be clear.

There are three main advantages of LMX. Firstly, LMX places no arbitrary constraints on the kinds of linkage relationships which the GA can work with. This avoids problems such as the difficulty of coding three equally-linked parameters when using 1- or multi-point crossover, discussed in the previous section. Secondly, LMX drags the issue of genetic linkage firmly into the spotlight. Instead of thinking of crossover in terms of cut and splice operations on ordered lists of parameters, a viewpoint which leads to implicit and often subtle linkage assumptions, LMX requires the GA designer to consider linkage explicitly from the outset. Since LMX does not make any assumptions about linkage other than those which are represented explicitly in the linkage map, the GA designer is completely responsible for building in any *a-priori* knowledge about genetic linkage by hand-crafting the linkage map accordingly. Finally, LMX provides a framework for a new adaptive crossover operator, *ALMX*, in which linkage relationships, instead of being hand-coded, can be inferred automatically as the GA runs. This operator is introduced in the following section.

## 7.5 Adaptive linkage mapped crossover (ALMX)

In any non-trivial parametric optimisation problem it is impossible to exactly quantify the dependencies which exist between parameters, since to do so would require knowledge of the whole search space, i.e. it would require that the problem had already been solved. The best that can be done (and the approach typically adopted in GA applications) is to make a number of reasonable assumptions about which sets of parameters are likely to be interdependent and which are not, based on prior knowledge about the nature of the problem, and construct a recombination operator accordingly (or, more typically, chose the coding order accordingly and use standard 1-point crossover). This approach is dangerous for complex problems because making the wrong assumptions is likely to be worse than making no assumptions at all.

An ideal 'black box' optimiser should not rely on the user to hand-code any such *a priori* information. Instead, the algorithm should be able to make its own inferences about linkage based on feedback from the objective function. Holland's use of inversion and Schaffer's punctuated crossover are two attempts to allow the GA to do just this. ALMX is a third, and more general, approach.

ALMX makes no initial assumptions at all about genetic linkage. Each gene's linkage vector starts as a uniform probability distribution over every other gene, and so ALMX initially behaves exactly like uniform crossover. Each time a crossover occurs, the linkage map is adapted slightly as the algorithm tries to learn the correct linkage relationships. Because ALMX starts with no assumptions at all, it represents

a more general approach than either inversion or punctuated crossover, which both make certain initial assumptions. Whatever the order of the chromosome when using inversion, for example, there is always the assumption that a gene is twice as strongly linked to the next gene as it is to the next-but-one, and so on. An example with punctuated crossover is that all genes between two punctuation points are always assumed to be equally linked.

### 7.5.1 Adapting the linkage map

How is it possible to adapt the linkage map on-line to learn appropriate linkage relationships? Any learning process requires a reward signal to direct it, and, in the case of learning genetic linkage, this reward signal must come from the fitness function. This is because 'correct linkage' is a property of the particular optimisation problem at hand, and the fitness function is the GA's only interface to the problem.

The approach taken by Holland and Schaffer is a simple one: it is assumed that crossovers which are based on good linkage assumptions preserve building blocks and result in children of high fitness, and those which make poor linkage assumptions disrupt building blocks, producing children of low fitness. Since the assumptions on which each crossover is based are part of the chromosomes themselves in these systems, natural selection is responsible for weeding out poor sets of assumptions, allowing the GA to gradually discover appropriate linkage relationships.

One problem with Schaffer's punctuated crossover is that it introduces substantial amounts of noise into the reward mechanism. This is because each crossover is based on a combination of the crossover masks (i.e. linkage assumptions) from the two parents, so neither parent's crossover mask is ever evaluated independently. A parent which carries an excellent set of crossover points can still be completely disrupted if it mates with a chromosome with a poor set. Worse still, the resulting offspring do not inherit the crossover mask which was used to generate them – instead of the combined mask used in the crossover, each offspring inherits a new mask made of some crossover points from each parent. This means that a parent's good linkage mask won't necessarily be passed on, even though that parent may produce fit offspring because of it. It also means that relying on selection to cull poor offspring is not equivalent to removing the poor crossover masks which generated them from the population.

A more general problem, common to both inversion and punctuated crossover, is the whole notion that the production of good offspring is an indication that crossover has done a good job of exchanging building blocks. This idea that the fitness of an offspring directly reflects the quality of the linkage assumptions made during its conception is not necessarily correct. What actually happens when

crossover swaps a beneficial building block from one chromosome to another is that the fitness of the recipient increases, while the fitness of the donor decreases correspondingly. In fact, any case where crossover generates two children which are both fitter than their parents is an indication that crossover has 'accidentally' happened to assemble some new building blocks in the children which were not present in the parents. This can only happen if crossover has failed to respect building block boundaries, and has exchanged some genes within a building block. In this case, fit offspring have resulted from poor linkage assumptions; crossing the same set of genes again would always tend to disrupt the same building blocks.

Learning in ALMX is based on the following principle: the only case in which it is evident that one or more complete building blocks have been exchanged by crossover is when an increase in fitness in one chromosome is matched by a corresponding decrease in fitness in the other. To date, experiments with ALMX have used a particularly simple rule for adapting the linkage map, and the results of these experiments are described in the following sections. The complete ALMX algorithm for binary coded GAs is as follows:

- Initialise the linkage map. This is done once, before the first generation of the GA. Leading diagonal elements of the linkage map are set to zero, and every gene's linkage vector is set to a uniform probability distribution over the remaining genes. Formally, for chromosomes of length $n$, initialise each element of the linkage map:

$$l_{ij} = \begin{cases} 0 & \text{if } i = j \\ \dfrac{1}{n-1} & \text{otherwise} \end{cases} \qquad i = 1 \cdots n,\ j = 1 \cdots n \tag{7.1}$$

ALMX is, therefore, initially equivalent to uniform crossover and 'knows' nothing about linkage.

- Each time a crossover occurs between two parents, $P_1$ and $P_2$, choose the $N$ crossover sites, $g_0,\ldots,g_N$, as described in section 7.3. The first point is chosen randomly, and the remaining points are chosen probabilistically according to the first point's linkage vector. Swap the chosen genes between the parents to generate two children, $C_1$ and $C_2$. Formally:

$$C_{1j} = \begin{cases} P_{2j} & \text{if } j \in \{g_0, \cdots, g_N\} \\ P_{1j} & \text{otherwise} \end{cases} \qquad j = 1 \cdots n \tag{7.2}$$

and similarly for $C_2$.

- Evaluate $f(C_1)$ and $f(C_2)$, the fitness of the children, and update the linkage map accordingly. The simple learning rule used in the experiments so far can be summed up as follows: if one child shows an improvement in fitness compared to its closest parent, and the other is worse, it is assumed that the genes which were exchanged must have constituted a complete building block, and the linkage between the genes $g_0,\ldots,g_N$ is reinforced. In all other cases, the linkage between the crossover sites is weakened. For each pair of crossover sites, the corresponding element of the linkage map has added to it a small quantity:

$$\Delta l_{ij} = \delta\eta\rho l_{ij} \quad \forall i,j \in \{g_0,\cdots,g_N\} \tag{7.3}$$

where

$$\delta = \begin{cases} 1 & \text{if } \dfrac{f(C_1)-f(P_1)}{f(P_2)-f(C_2)} > 0 \\ -1 & \text{otherwise} \end{cases} \tag{7.4}$$

$$\rho = \begin{cases} 1 & \text{if } P_{1i} \neq P_{2i} \text{ or } P_{1j} \neq P_{2j} \\ 0 & \text{otherwise} \end{cases} \tag{7.5}$$

$\eta$ is the learning rate, a constant which has been set to 0.4 in all the experiments so far (a value chosen after some preliminary experiments). $\delta$ represents the learning rule, and $\rho$ is a term which ensures that the linkage between two genes is not updated if both have identical alleles in both parents. This is because swapping genes for which both parents have identical alleles has no effect, and so the crossover reveals nothing about the linkage between these genes.

- Re-normalise every row of the linkage map which has been changed.

- Choose one of the two children to keep and discard the other. In the experiments so far, this choice has been made randomly.

## 7.6 Experiments

The performance of ALMX was compared to that of two other crossover operators on two variants of a binary test function.

### 7.6.1 The Royal Road function R1

Royal Road functions are a class of artificial binary test problems, introduced by Mitchell *et al.* (Mitchell, Forrest, *et al.* 1991; Forrest and Mitchell 1993; Mitchell, Holland, *et al.* 1994) as a tool for studying GAs. Royal Road functions are so called because they are designed to lay out a simple path for a GA to follow towards a single global optimum. A simple 64-bit Royal Road function, R1, consists of the set of explicitly defined schemata S={$s_0$,...,$s_7$} shown in figure 7.3.

```
s0=11111111################################################################;  c0=8
s1=########11111111########################################################;  c1=8
s2=################11111111################################################;  c2=8
s3=########################11111111########################################;  c3=8
s4=################################11111111################################;  c4=8
s5=########################################11111111########################;  c5=8
s6=################################################11111111################;  c6=8
s7=########################################################11111111########;  c7=8
```

Figure 7.3: The 8 schemata which define R1.

Each schema $s_j$ has an associated value $c_j$, and the fitness of a string $x$ is defined as the sum of the values of each of the schemata of which it is an instance:

$$f(x) = \sum_{j=0}^{7} c_j \sigma_j(x) \qquad \sigma_j(x) = \begin{cases} 1 & \text{if } x \text{ is an instance of } s_j \\ 0 & \text{otherwise} \end{cases} \qquad (7.6)$$

For R1, the global optimum is the string 111...1, which has fitness 64. R1 was deliberately designed to be as simple as possible for a standard GA using 1-point crossover. It is completely non-deceptive: every building-block is part of the global optimum, and each makes a completely independent contribution to fitness. These building blocks (the schemata $s_0$,...,$s_7$) are designed to fit the linkage assumptions implicit in 1-point crossover – each schema is a short (i.e. low defining-length) contiguous linear sequence of linked genes, and no linkage exists between schemata.

Because the 'correct' linkage map is well defined, R1 is an ideal function on which to test an adaptive crossover such as ALMX. Figure 7.4 compares the performance of ALMX, 1-point crossover and parameterised uniform crossover on R1. In each case, a simple generational (no elitism) roulette-wheel GA was used, with a population of 400 64-bit binary strings. Because the intention was to study the

crossover operator, the crossover rate $P_c$ was set to 1.0 and mutation was turned off ($P_m$=0.0). For both ALMX and the uniform crossover, the number of crossover sites was set to 8 (equivalent to $P_0$=0.125). Each run was terminated after 50000 fitness evaluations, and the results below are averaged over 20 runs.

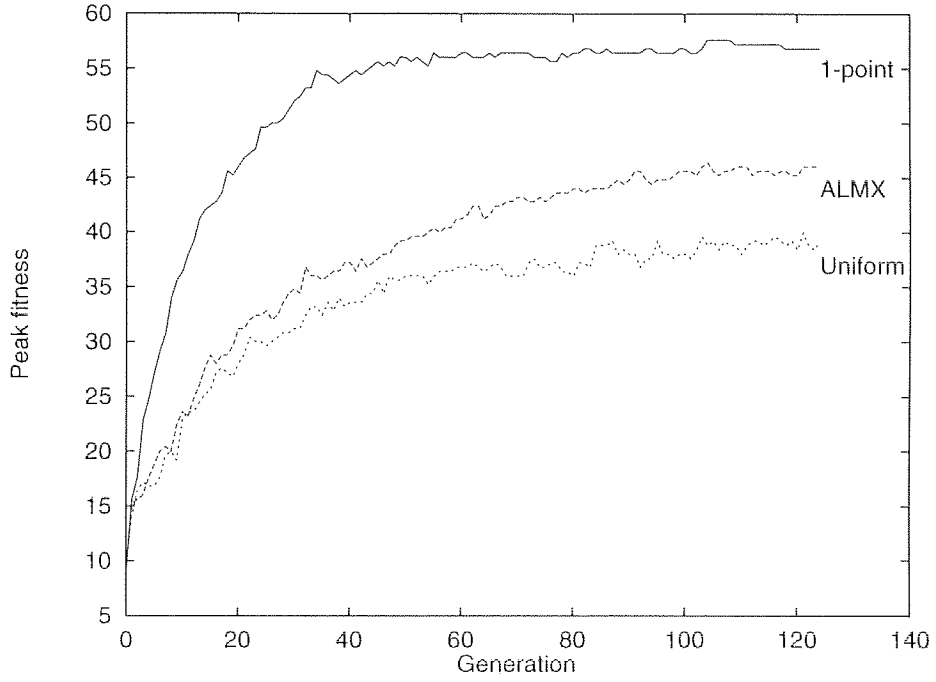
These results show that 1-point crossover is superior to both ALMX and parameterised uniform crossover on R1. This is only to be expected, since R1 is designed explicitly to suit the linkage assumptions made by 1-point crossover. Neither of the other two operators have the benefit of any such *a-priori* assumptions. What is encouraging is the fact that ALMX convincingly outperforms uniform crossover in the latter stages of the search, even though both operators begin with no assumptions about linkage at all. This indicates that ALMX has learned something useful about the patterns of genetic linkage in R1 during the course of the search.

### The crossover non-linearity ratio

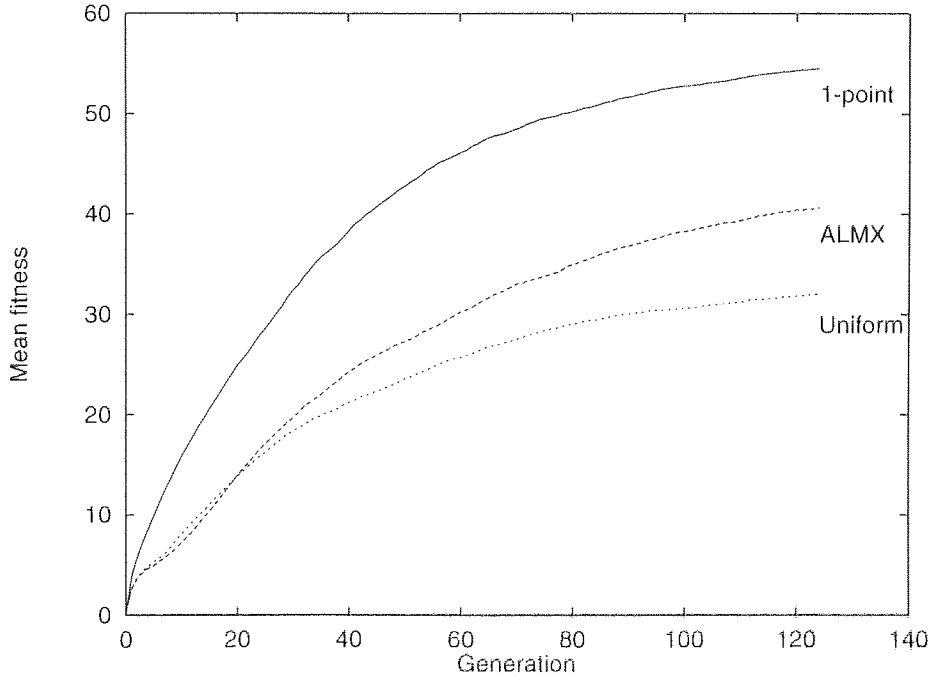
To compare the disruptive effects of different crossover operators, it is useful to examine their associated *crossover non-linearity ratios*. The crossover non-linearity ratio $\psi$ is a measure which was introduced by Mason (1993) to quantify the efficiency of crossover at recombining building blocks. Mason gives the following definition for $\psi$: let $f_P = \left[f(P_1) + f(P_2)\right]/2$ and $f_C = \left[f(C_1) + f(C_2)\right]/2$ be the mean fitnesses of the parents and children respectively, and let $\varepsilon_{PC} = f_P - f_C$, $\varepsilon_P = f(P_1) - f_P = f_P - f(P_2)$ and $\varepsilon_C = f(C_1) - f_C = f_C - f(C_2)$. Now:

$$\psi = \begin{cases} +\dfrac{\varepsilon_{PC}}{\varepsilon_P} & |\varepsilon_P| \geq |\varepsilon_C| \\ -\dfrac{\varepsilon_{PC}}{\varepsilon_C} & |\varepsilon_P| < |\varepsilon_C| \end{cases} \qquad (7.7)$$

To prevent $\psi$ from ever becoming infinite, $\psi$ is set to zero whenever $\varepsilon_{PC} = 0$. In general, a value of $\psi$ close to zero indicates that the crossover has exchanged a building block which makes an independent contribution to fitness, since the increase in fitness in one chromosome is matched by the decrease in fitness in the other. A large positive or negative value for $\psi$ shows that the fitness contribution made by the genes which have been exchanged is not independent, since the change in fitness of one chromosome is not balanced by that in the other. This implies that the crossover operator is disrupting, rather than preserving, building blocks.




(a)




(b)

Figure 7.4: Performance of ALMX, 1-point and parameterised uniform crossovers on Royal Road function R1. (a) shows the fitness of the best individual in the population, and (b) shows the population mean fitness. All results are averaged over 20 runs.

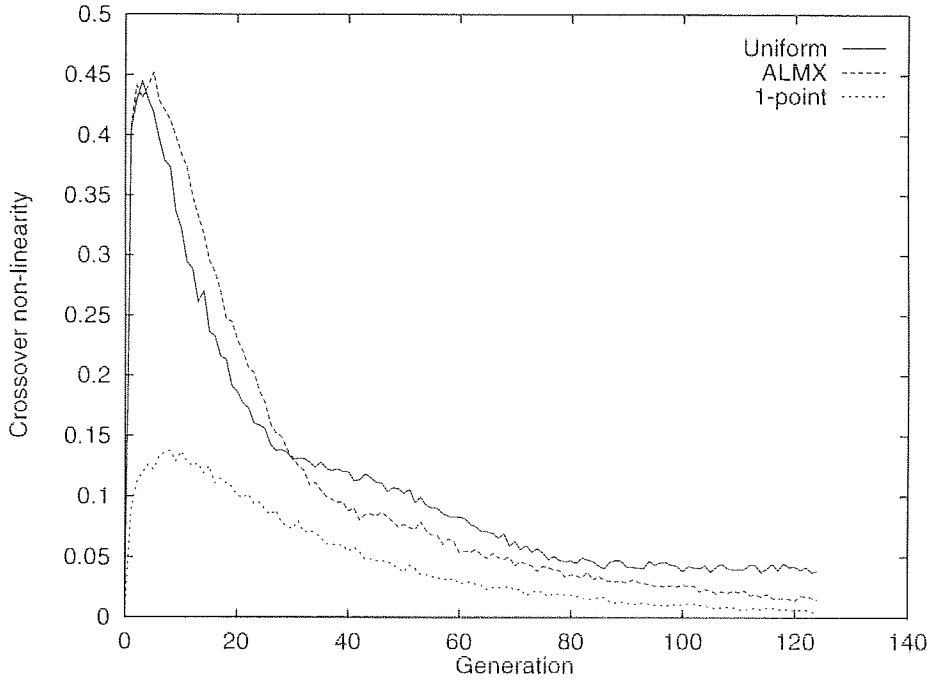



Figure 7.5: Change in average absolute crossover non-linearity per generation for parameterised uniform crossover, ALMX and 1-point crossover on R1.

Figure 7.5 compares the change in degree of crossover non-linearity over time for each of the three crossover operators on R1. For each graph, each point represents the average value of $|\psi|$ over every crossover event in one generation. In each case, crossover non-linearity starts off very low, since most early crossover events will be between two parents of zero fitness, and produce two offspring of zero fitness. Crossover non-linearity quickly peaks as selection fills the population with instances of the various different schemata, and then gracefully decreases towards the end of the run as the GA converges and the population becomes more uniform. In the absence of mutation, the GA will tend towards a completely uniform population, in which case crossover non-linearity will be zero whatever the recombination operator, since children and parents will always be identical. As is expected, uniform crossover and ALMX are much more disruptive for R1 than 1-point crossover. However, as ALMX discovers some linkage relationships towards the end of the search, it begins to preserve some building blocks, and its level of crossover non-linearity drops relative to that of uniform crossover.

### Examining the linkage map

The fact that ALMX outperforms uniform crossover suggests that something useful is being discovered about the linkage relationships in R1. To establish exactly what has been learned during the course of a GA run, it is natural to examine the final state of the linkage map. In this case, looking at a single run is more instructive than

averaging over all 20 runs, so this is the approach taken here. Although the results below were generated by only one of the 20 runs of ALMX on R1, they are typical of all 20 runs.

Before looking at the linkage map it is worth examining figure 7.6, which shows how the relative frequencies of each of R1's 8 schemata changed during the course of this particular run. Although all 8 schemata were present in the initial population, schemata 1 and 2 failed to gain a foothold and eventually disappeared completely. The best string found during this run therefore had a fitness of 48. In the figure, the schemata are listed in the order in which they became fixed in the population as the GA converged. The first to spread through the whole population was schema 3, which finally became fixed (i.e. all 400 population members were an instance of it) in generation 43. The next to become fixed was schema 6, followed by schema 4. Schemata 5 and 0 had not quite reached fixation by the time the GA was terminated at 125 generations, at which time schema 7 was just beginning to spread through the population.

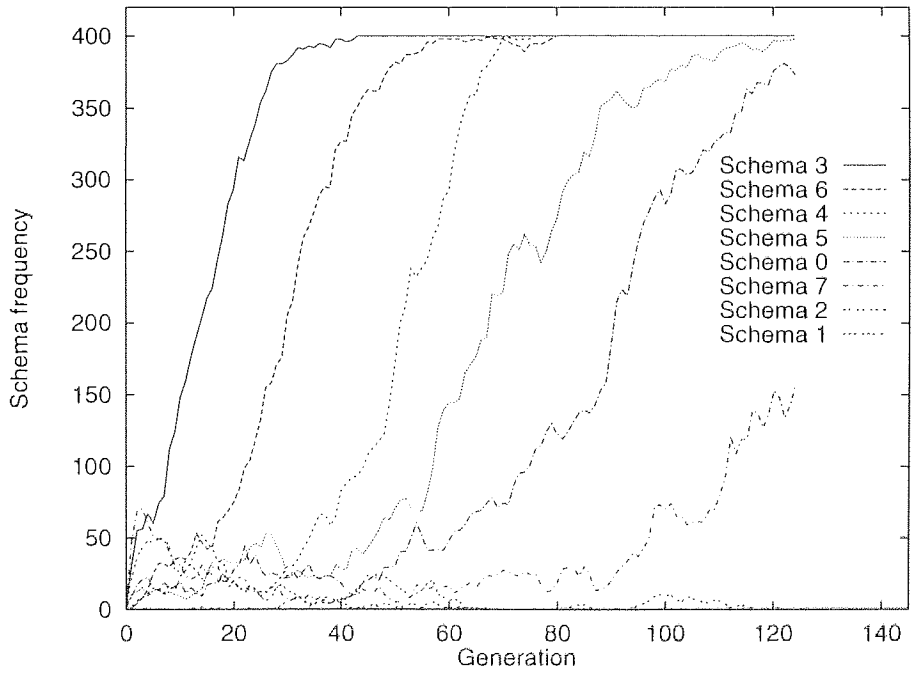



Figure 7.6: Change in relative schemata frequencies for one run of ALMX on R1. The vertical axis shows the number of members of the population which are instances of each schema. The key lists schemata in the order in which they spread through the population.

Figure 7.7 summarises the state of the linkage map at the end of this run. Each histogram is an average of the linkage vectors for the 8 genes which make up one of the R1 schemata. The figures are shown in the order in which the schemata became fixed in the population (see figure 7.6). The first figure shows the average linkage vector for genes 24 to 31, representing schema 3, the first schema to become fixed.

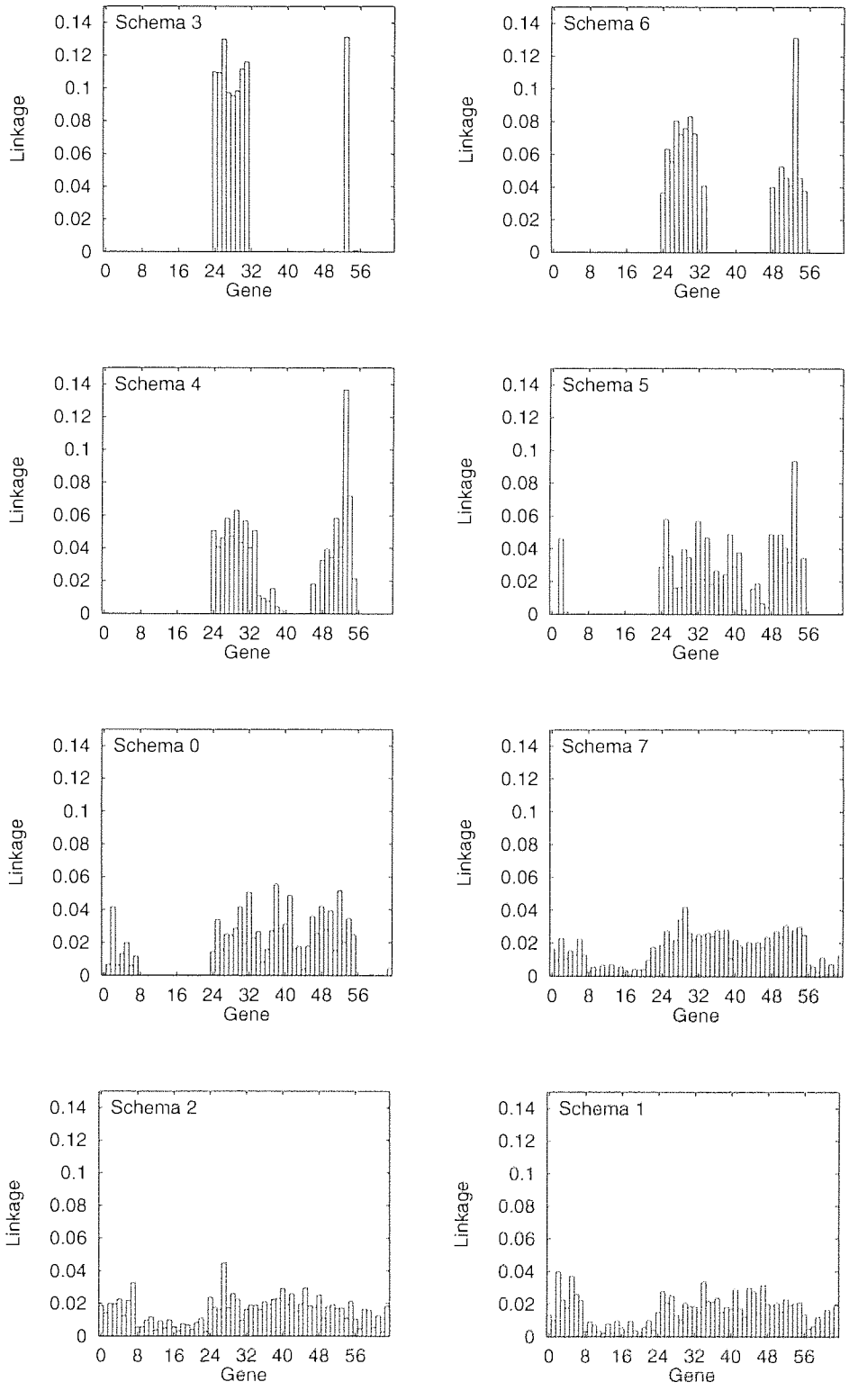



Figure 7.7: Summary of the linkage map at the end of the run. Each histogram is an average taken over 8 linkage vectors for the 8 genes which represent one schema in R1. Histograms are shown in the order in which the schemata became widespread in the population. See text for discussion.

As schema 3 has spread through the population, the ALMX algorithm has successfully discovered that these genes constitute a complete building block. The final linkage vector is near-perfect: if any gene from this schema is picked as the first crossover site, the chances are extremely high that the remaining genes will be picked from the same schema, and the whole building block will be crossed over. The only exception is the spurious linkage to gene 54, a gene which is not a member of schema 3. The reason for this is likely to be that this gene has completely converged early in the run so that its value has become fixed to 1 in the whole population. There is little pressure to reduce linkage to a fixed gene, since crossing over such a gene has no effect.

The second histogram in figure 7.7 shows the average linkage vector for genes 48 to 55, representing schema 6, the second schema to become widespread in the population. The ALMX algorithm has correctly discovered that these genes are strongly linked to each other, but has also linked them to the genes from schema 3. The reason for this can be found in the early stages of the search, when schema 3 is the only schema to be widely represented in the population. At this time, selection will ensure that most crossovers will involve at least one parent which is an instance of schema 3. If the other parent is not an instance of this schema, then any crossover which involves some, but not all, of the genes from schema 3 is likely to disrupt the schema in the first parent. However, so long as the second parent does not have zeros for every gene in the schema, there is a chance that the genes which are crossed over can complete the schema in the second parent. It is not necessary to cross over every gene in order to have the same effect as crossing the whole schema – only those genes which differ need to be swapped. The result of this is that many crossovers which happen to include just some of the genes from schema 3 will have the same effect as crossing the whole building block, irrespective of the remaining crossover sites. Each time this occurs, the ALMX algorithm will reinforce the linkage between all the crossover sites. The net result over many crossovers is a tendency for ALMX to link arbitrary genes to those in schema 3. When schema 6 begins to become widespread, the ALMX algorithm begins to reinforce the linkage between the genes in this new schema. These genes do not completely lose their linkage to the schema 3 genes, however, because the genes in schema 3 have converged in the whole population, so there is little pressure to reduce the linkage; one or two crossover sites within this schema make little difference.

The same pattern can be seen in the average linkage vector for genes 32 to 39, representing schema 4, and similarly for the genes in schemata 5, 0 and (to some extent) 7. In each case, although ALMX has reinforced the linkages between genes within the schema, there is also a tendency for genes to remain linked to genes in those schemata which were already prevalent when the new schema was discovered.

The average linkage vectors for the two schemata which eventually died out, schemata 2 and 1, show little linkage between the genes within these schemata, and fairly uniform linkage to all the genes in the other schemata.

### 7.6.2 The Rocky Road – interleaved R1

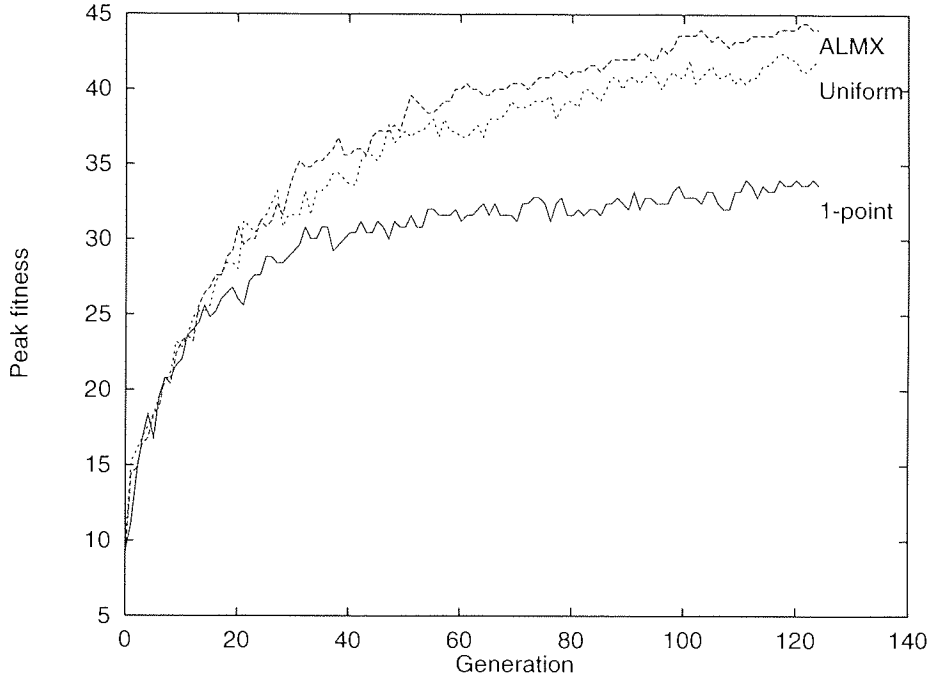
In a second set of experiments, the ALMX, parameterised uniform and 1-point crossover operators were tested on an 'interleaved' variation of R1 which will be termed $R^{INT}$. $R^{INT}$ is a non-deceptive function completely defined by 8 independent, order-8 schemata, in precisely the same manner as in R1. These schemata are shown in figure 7.8 below.

$s_0$=1#######1#######1#######1#######1#######1#######1#######1#######; $c_0$=8
$s_1$=#1#######1#######1#######1#######1#######1#######1#######1######; $c_1$=8
$s_2$=##1#######1#######1#######1#######1#######1#######1#######1#####; $c_2$=8
$s_3$=###1#######1#######1#######1#######1#######1#######1#######1####; $c_3$=8
$s_4$=####1#######1#######1#######1#######1#######1#######1#######1###; $c_4$=8
$s_5$=#####1#######1#######1#######1#######1#######1#######1#######1##; $c_5$=8
$s_6$=######1#######1#######1#######1#######1#######1#######1#######1#; $c_6$=8
$s_7$=#######1#######1#######1#######1#######1#######1#######1#######1; $c_7$=8

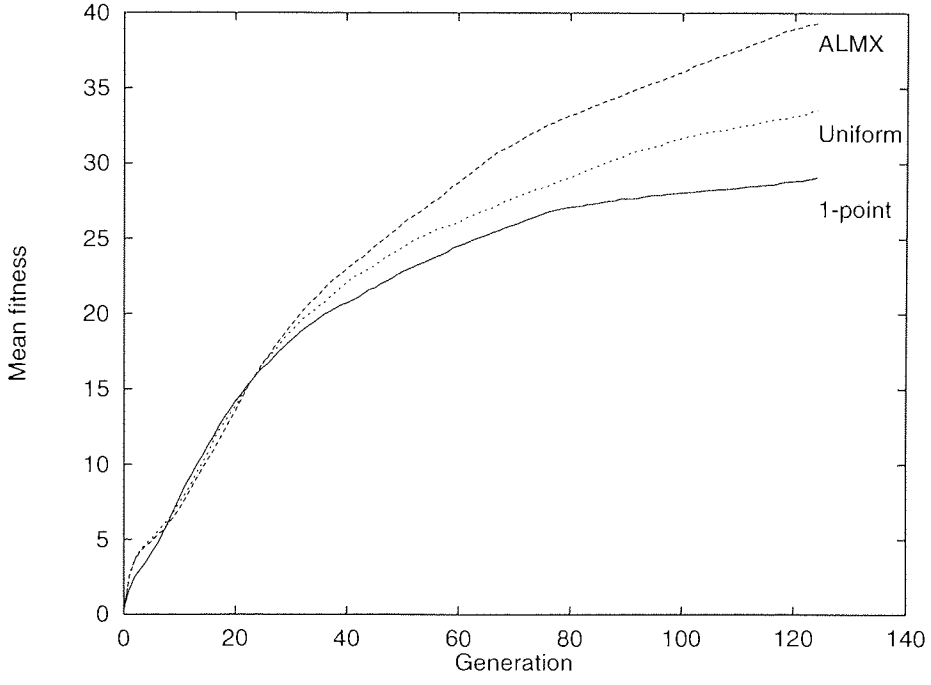
Figure 7.8: The 8 schemata which define $R^{INT}$.

Since both R1 and $R^{INT}$ are completely non-deceptive, and each has a single global maximum, both would normally be considered 'GA-easy'. In fact it is clear that, as parameter optimisation problems, R1 and $R^{INT}$ are completely equivalent. The only difference is the order in which the parameters are coded on the chromosome. Whereas the coding in R1 is designed specifically to reflect the linkage assumptions made by 1-point crossover, $R^{INT}$ is deliberately designed to confound 1-point (or multi-point) crossover. In $R^{INT}$, the linked genes are distributed along the whole length of the chromosome, making the average defining length of the schemata as great as possible. If R1 lays out a 'Royal Road' for the canonical GA, $R^{INT}$ represents an altogether rockier and more treacherous path towards the optimum.

Figure 7.9 shows the performance of each of the three crossover operators on $R^{INT}$. All parameters for the GA were the same as for the earlier experiments with R1, and each figure shows results averaged over 20 runs, as before. The poor performance of 1-point crossover on $R^{INT}$ makes an important point: making the wrong *a-priori* assumptions about genetic linkage is considerably worse than making no assumptions at all.




(a)




(b)

Figure 7.9: Performance of ALMX, 1-point and parameterised uniform crossovers on function $R^{INT}$. (a) shows the fitness of the best individual in the population, and (b) shows the population mean fitness. All results are averaged over 20 runs.

Parameterised uniform crossover and ALMX make no initial linkage assumptions, and so their performance is unaffected by the change in coding from R1 to $R^{INT}$. ALMX is the highest performer on $R^{INT}$ because its ability to learn something of the linkage relationships during the course of the run makes it considerably less disruptive than uniform crossover. This is particularly apparent in figure 7.9(b), which shows the population average fitness – ALMX is clearly producing far fewer 'lethal' low-fitness offspring during the closing stages of the search than either parameterised uniform or 1-point crossover.

Figure 7.10 shows the change in average crossover non-linearity over time. Here, the story is the same. Uniform crossover and ALMX are insensitive to the change in coding from R1 to $R^{INT}$, whereas 1-point crossover has gone from being much less disruptive than the other two operators to being initially more disruptive than either. As it infers some of the linkage relationships in $R^{INT}$, ALMX achieves the lowest final degree of crossover non-linearity.

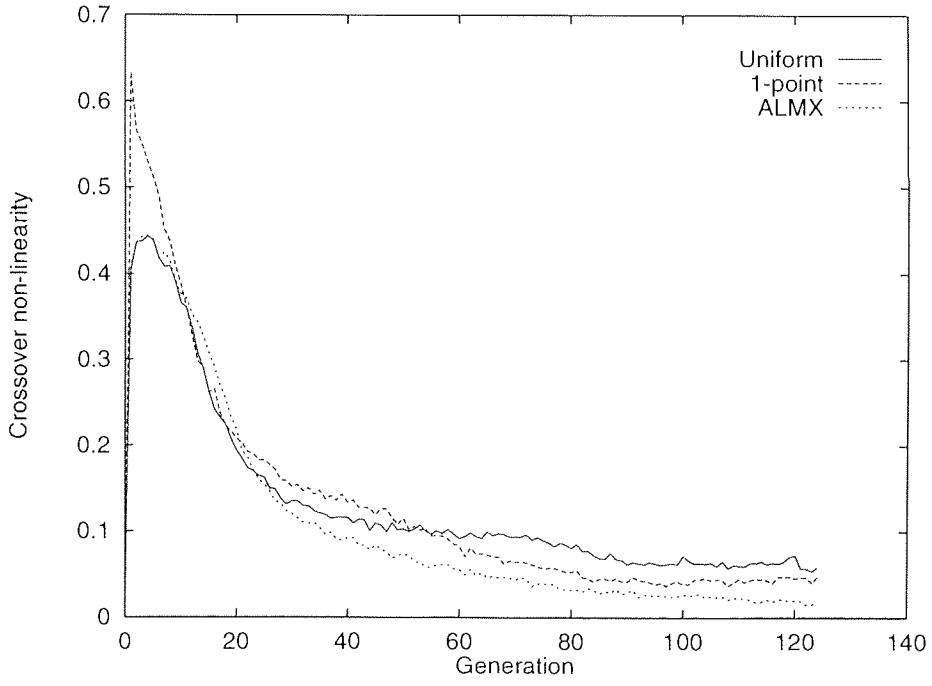



Figure 7.10: Change in average absolute crossover non-linearity per generation for parameterised uniform crossover, ALMX and 1-point crossover on $R^{INT}$.

## 7.7 Discussion and prospects

Recombination is the GA's trademark operator, and its usefulness depends critically on how exactly it respects appropriate genetic linkage relationships for the problem being optimised, that is, how likely it is that co-adaptive groups of tightly-linked

genes (representing highly interdependent parameters) will be kept together. Commonly used crossover operators have been shown to be inherently limited in the kinds of linkage relationships which they can respect.

The LMX operator was introduced as a generalised recombination operator which is able to respect arbitrary linkage relationships, including those associated with existing crossovers. The linkage map used by LMX is a way of making all linkage assumptions totally explicit – LMX does not introduce the kinds of subtle implicit linkage assumptions associated with a more traditional 'cut and splice' view of crossover.

It is possible to extend the LMX operator by incorporating a mechanism to adapt the linkage map after each recombination. In this way, the recombination operator need not rely on any prior knowledge, but is able to discover linkage relationships which are appropriate to the particular optimisation problem. The adaptive LMX operator (ALMX) described in this chapter uses a particularly simple learning rule to adapt the linkage map. Even so, it is able to correctly discover some important linkage relationships for the two test problems studied.

GAs have often been referred to as 'black box' or general purpose optimisers. The fact that most existing crossover operators make strong assumptions about which of the parameters being optimised are interdependent and which are not undermines this supposed generality. At best, the GA practitioner can try to choose the coding scheme which best fits the assumptions made by the crossover operator to any relationships known to exist between the parameters. Of course, this supposes that the degree to which the various parameters interact is known in advance, which is unlikely to be the case for anything other than trivial problems. If the GA designer gets it wrong, and the linkage assumptions don't match the real dependencies in the problem, the GA is likely to perform poorly, as illustrated by the dismal performance of the 1-point crossover GA on $R^{INT}$.

The ALMX operator offers a route to a truly general purpose optimiser, an algorithm which can discover the dependencies between parameters for itself, and tailor the search accordingly. This does not, however, rule out building in problem-specific knowledge if such knowledge is available. By initialising the linkage map accordingly, arbitrary linkage assumptions can be built into the operator as a starting point.

Time constraints have meant that the implementation of ALMX described here is extremely basic. The learning rule is crude, and improving this aspect of the operator is likely to be a fruitful area for further research. At present, the linkage reinforcement signal has only two states: either a fixed positive reinforcement, if one chromosome has improved as a result of the crossover and the other has got worse, or a fixed negative reinforcement in all other cases. The implementation of a

continuously varying reinforcement signal based on the crossover non-linearity ratio would be a promising topic for future work.

One issue which has not been addressed at all in this chapter is the problem of choosing an appropriate value for $N$, the number of genes exchanged in each crossover. If the value of $N$ is inappropriate, disruption will result. As a simple illustration, consider choosing a value of $N$=9 for one of the Royal Road functions described earlier. In these functions, each building block (schema) consists of 8 genes. Even if crossing over the first 8 genes exchanges a complete schema, crossing the 9th must disrupt another. Fixing $N$ to a particular value assumes that the co-adaptive groups which crossover is expected to assort each consist of exactly $N$ genes, an assumption which is valid for the Royal Road functions, but which is highly unlikely to be true for real problems. In a real problem, different building blocks will consist of different numbers of co-adaptive genes. If crossover is always to swap whole building blocks then the number of genes exchanged should depend on the starting point.

One solution to this problem would be to choose $N$ independently for each crossover, after the first crossover site has been chosen. $N$ could be based on the statistical properties of the linkage vector associated with the chosen starting point. Possibilities include crossing over all those genes whose linkage to the first gene is above the average, or crossing over genes until a certain proportion of the original linkage has been 'used up'.

In summary, the LMX operator in itself represents a generalised recombination operator which can model arbitrary linkage relationships, and which emphasises the need for genetic linkage assumptions to be made explicit during the design phase of a GA application. Its adaptive form, ALMX, offers a route towards a truly general-purpose weak optimisation method which needs no prior knowledge at all about the optimisation problem at hand, although such knowledge can easily be incorporated should it be available. Although the implementation of ALMX is in its infancy, the method shows promise on test problems.

# Chapter 8

# Conclusions

In the past few years there have been innumerable efforts to apply genetic algorithms to the optimisation of neural networks. Almost all of them have been plagued by two major stumbling blocks: the impractically large computing time involved and the problem of finding a non-redundant genetic representation for an essentially order-less structure like a neural network.

Central to this thesis has been the introduction of a novel GA-NN hybrid which overcomes (or at least avoids) these problems: the GA-bumptree. As well as having considerably lower redundancy than is typical in GA-NN applications, the hierarchical genetic coding scheme developed for the bumptree and is also shown to have the properties of closure, completeness, continuity and isomorphism with respect to the search space.

A principal aim of this project was to focus on applications where the computational overheads of GA search could be justified. As far as neural network training is concerned, pattern classification is an area where there is no obvious benefit to using a GA since strong methods (such as backpropagation or the bumptree's error minimisation rule) can be deployed to good effect and will outperform the GA's weaker search (as seen in section 3.4). For classification tasks, finding the best network topology is a more appropriate application for GAs, and results such as those in chapter 2 suggest that the GA's unconstrained search can discover novel and unexpected topologies even for well-studied problems. The results with the first GA-bumptree hybrid described in chapter 3 are also encouraging, with the bumptrees generated by the GA significantly outperforming those generated by a more constrained top-down constructive algorithm on all test problems.

The artificial life study reported in chapter 5 and the problems described in chapter 6 were investigated because they seemed to represent learning tasks where stronger training methods could not easily be applied. The A-life scenario was found to be unsuitable as a test problem for the GA-bumptree, partly because of its simplicity, but largely because the work in chapter 5 called into question the original authors' experimental method and their interpretation of results. The results reported in chapter 6 are more positive, and demonstrate how the GA-bumptree can be applied to two difficult kinds of learning task: temporal credit assignment problems, typified by the simulated pole balancing and car parking tasks, and learning multi-valued mappings such as the inverse kinematics of a robot arm.

Given that the computational overheads of genetic search can only be justified if more efficient methods are not available, however, it is worth mentioning some recent advances in the field of neural networks. Since this project began, TD methods have become widespread as the standard approach to learning temporal credit assignment problems, with many impressive results. A good illustration is Tesauro's (1993) TD-gammon system, a neural network backgammon player which learned

entirely through self-play and is now judged to be among the best players in the world. Similarly, the recent work of Bishop (1994) and Rohwer and van der Rest (1994) advances strong, principled methods for the problem of learning multi-valued mappings. Thus, while the experiments described in chapter 6 demonstrate the *feasibility* of applying the GA-bumptree to these kinds of learning problems, in the light of continuing progress in neural network learning algorithms it seems likely that the *practicality* of an evolutionary approach to neural network training will dwindle as stronger methods are refined.

One conclusion to emerge from the work described in chapter 6 is that the robot arm inverse kinematics problem reveals the limitations of the GA-bumptree. The results given in section 6.3.2 are important because they show that good solutions do exist in the search space, but that the GA simply isn't able to find them. Instead, the GA reliably converges on a simplified architecture which offers initial improvements in performance, but which is an evolutionary dead end.

As well as introducing and evaluating the GA-bumptree, this thesis has made some contributions to the field of GAs in general. The convergence profile introduced in chapter 4 is a tool which can reveal useful information about any real-valued GA, and the discussion in the latter part of the chapter is relevant to any GA which uses a geographic selection model. The observed relationship between the rate of convergence and the tendency away from gradual evolution towards punctuated equilibrium also has relevance beyond the context of the GA-bumptree, and may help to explain others' results. For example, in his invited talk on Tierra given at ECAL '93, Tom Ray observed that changing the instruction set used by the simulated replicators resulted in a change from gradual evolution to punctuated equilibrium, although he was unable to explain the change.

Finally, the re-examination of the role of recombination described in chapter 7 may prove to be a valuable contribution to the GA field. The concept of the linkage map offers a general framework for the rigorous design of application-specific crossover operators, while the ALMX operator is a step towards a truly universal self-adaptive recombination operator. Although the current ALMX algorithm is rather simple, the preliminary results certainly encourage further development of the method.

# References


Ackley, D. and M. Littman (1991). Interactions between learning and evolution. Artificial Life II. Eds. C. G. Langton, C. Taylor, J. D. Farmer and S. Rasmussen. Addison-Wesley. 487-509.

Ackley, D. H. (1987). *Stochastic iterated genetic hillclimbing.* Technical Report CMU-CS-87-107. Computer Science Department, Carnegie Mellon University.

Alander, J. T. (1994). *An Indexed Bibliography of Genetic Algorithms: Years 1957-1993.* Report Series No. 94-1. Department of Information Technology and Production Economics, University of Vaasa, Finland.

Ali Cherif, A. (1993). Collective behaviour for a micro-colony of robots. *Proceedings of the Third European Conference on Artificial Life (ECAL '93)*, Brussels, Belgium. 1-8.

Anderson, C. W. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control System Magazine*, April 1989:31-37.

Anderson, H. C. and A. C. Tsoi (1994). *A constructive algorithm for the training of a multilayer perceptron based on the genetic algorithm.* Technical Report, Department of Electrical Engineering, University of Queensland, Australia.

Antonisse, J. (1989). A new interpretation of schema coding that overturns the binary encoding constraint. *Proceedings of the Third International Conference on Genetic Algorithms.* Ed. J. D. Shaffer. Morgan Kaufmann. 86-91.

Bäck, T. (1994). *Evolution Strategies: A Thorough Introduction.* Tutorial presented at the First IEEE International Conference on Evolutionary Computation, Orlando, Florida.

Bäck, T., F. Hoffmeister and H.-P. Schwefel (1991). A survey of evolution strategies. *Proceedings of the Fourth International Conference on Genetic Algorithms.* Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann. 2-9.

Bäck, T. and H.-P. Schwefel (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation* **1**(1): 1-23.

Bagchi, S., S. Uckun, Y. Miyabe and K. Kawamura (1991). Exploring problem-specific recombination operators for job shop scheduling. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann.

Bagley, J. D. (1967). *The behaviour of adaptive systems which employ genetic and correlation algorithms*. PhD Thesis. University of Michigan.

Baker, J. E. (1985). Adaptive selection methods for Genetic Algorithms. *Proceedings of an International Conference on Genetic Algorithms*. Ed. J. J. Grefenstette. Lawrence Earlbaum.

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. *Proceedings of the Second International Conference on Genetic Algorithms*. Ed. J. J. Grefenstette. Lawrence Earlbaum. 14-21.

Baldwin, J. M. (1896). A new factor in evolution. *American Naturalist* **30**: 441-451.

Barto, A. G., R. S. Sutton and C. W. Anderson (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics* **SMC-13**(5): 834-846.

Baum, E. B. and D. Haussler (1990). What size net gives valid generalization? *Neural Computation* **1**: 151-160.

Beer, R. D. and H. J. Chiel (1990). Neural Implementation of Motivated Behaviour: Feeding in an Artificial Insect. *Advances in Neural Information Processing Systems 2*. Ed. D. S. Touretzky. Morgan Kaufmann. 44-51.

Beer, R. D. and H. J. Chiel (1991). The Neural Basis of Behavioural Choice in an Artificial Insect. *Proceedings of the First International Conference on Simulation of Adaptive Behaviour*. Eds. J-A. Meyer, S. Wilson. MIT press.

Belew, R. K. (1989). When both individuals and populations search: adding simple learning to the Genetic Algorithm. *Proceedings of the Third International Conference on Genetic Algorithms*. Ed. J. D. Shaffer. Morgan Kaufmann.

Belew, R. K., J. McInerney and N. N. Schraudolph (1991). Evolving networks: using the genetic algorithm with connectionist learning. *Artificial Life II*. Eds. C. G. Langton, S. Taylor *et al.* Addison-Wesley. 511-547.

Bengio, Y. and S. Bengio (1990). *Learning a synaptic learning rule*. Technical Report 751, Département d'Informatique et de Recherche Opérationelle, Université de Montréal.

Berenji, H. R. and P. Khedkar (1992). Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Transactions on Neural Networks* **3**(5):724-740.

Bethke, A. D. (1981). *Genetic Algorithms as Function Optimizers*. PhD Thesis. University of Michigan.

Bishop, C. M. (1994). *Mixture Density Networks*. Technical Report NCRG/4288, Neural Computing Research Group, Aston University, UK.

Booker, L. (1987). Improving Search in Genetic Algorithms. *Genetic Algorithms and Simulated Annealing*. Ed. L. Davis. London, Pitman. Chapter 5.

Bornholdt, S. and D. Graudenz (1991). General asymmetric neural networks and structure design by genetic algorithms. *Neural Networks* **5**: 327.

Bos, M. and H. T. Weber (1991). Comparison of the training of neural networks for quantitative x-ray fluorescence spectrometry by a genetic algorithm and backward error propagation. *Analytica Chimica Acta* **247**: 97-105.

Bostock, R. T. J. (1994). PhD Thesis (in preparation). Aston University, Birmingham, UK.

Bramlette, M. B. and E. E. Bouchard (1991). Genetic algorithms in parametric design of aircraft. *Handbook of Genetic Algorithms*. Ed. L. Davis. Van Nostrand Reinhold. 109-123.

Bramlette, M. F. (1991). Initialisation, Mutation and Selection Methods in Genetic Algorithms for Function Optimisation. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann. 100-107.

Bramlette, M. F. and R. Cusic (1989). A comparative evaluation of search methods applied to parametric design of aircraft. *Proceedings of the Third International Conference on Genetic Algorithms*. Ed. J. D. Shaffer. Morgan Kaufmann.

Bridges, C. L. and D. E. Goldberg (1987). An analysis of reproduction and crossover in a binary-coded Genetic Algorithm. *Proceedings of the Second International Conference on Genetic Algorithms*. Ed. J. J. Grefenstette. Lawrence Earlbaum. 9-13.

Brindle, A. (1981). *Genetic Algorithms for function optimization.* PhD Thesis. University of Alberta.

Brindle, J. S. and S. J. Cox (1991). Recnorm: simultaneous normalisation and classification applied to speech recognition. *Advances in Neural Information Processing Systems 3*. Eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky. Morgan Kaufmann, San Mateo, CA. 234-240.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* **RA-2**(1): 253-262.

Broomhead, D. S. and D. Lowe (1988). Multi-variable functional interpolation and adaptive networks. *Complex Systems* **2**: 321-355

Brunn, D. E. (1993). Simple rules governing leg placement by the stick insect during walking. *Proceedings of the Third European Conference on Artificial Life (ECAL '93)*, Brussels, Belgium. 137-144.

Camazine, S. (1993). Collective Intelligence in Insect Colonies by means of Self-Organization. *Proceedings of the Third European Conference on Artificial Life (ECAL '93)*, Brussels, Belgium. 158-173

Cartwright, H. M. and S. P. Harris (1993). The Application of the Genetic Algorithm to Two-Dimensional Strings: The Source Apportionment Problem. *Proceedings of the Fifth International Conference on Genetic Algorithms.* Ed. S. Forrest. Morgan Kaufmann.

Caruana, R. A. and J. D. Shaffer (1988). Representation and hidden bias: Gray vs Binary coding for genetic algorithms. *Proceedings of the Fifth International Conference on Machine Learning.* Morgan Kaufmann, Los Altos, CA. 153-161.

Caudell, T. P. and C. P. Dolan (1989). Parametric connectivity: training of constrained networks using Genetic Algorithms. *Proceedings of the Third International Conference on Genetic Algorithms.* Ed. J. D. Shaffer. Hughes Research Labs, Morgan Kaufmann.

Cavicchio, D. J. (1970). *Adaptive Search using Simulated Evolution.* PhD Thesis. University of Michigan.

Cecconi, F. and D. Parisi (1990). *Evolving organisms that can reach for objects.* Technical Report. Institute of Psychology, C. N. R., Rome.

Chalmers, D. J. (1990). The evolution of learning: an experiment in genetic connectionism. *Proceedings of the 1990 Connectionist Models Summer School..* Eds. Touretzky, D. S., J. L. Elman *et al.* Morgan Kaufman.

Chang, E. I. and R. P. Lippmann (1991). Using genetic algorithms to improve pattern classification performance. *Advances in Neural Information Processing Systems 3.* Eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky. Morgan Kaufmann. 797-803.

Cheok, K. C. and N. K. Loh (1987). A ball-balancing demonstration of optimal and disturbance-accomodating control. *IEEE Control Systems Magazine*, February 1987:54-57.

Cliff, D., I. Harvey and P. Husbands (1992). *Incremental evolution of neural network architectures for adaptive behaviour.* Technical Report CSRP 256, School of Cognitive and Computing Sciences, The University of Sussex, Brighton, UK.

Cliff, D., P. Husbands and I. Harvey (1992). Evolving visually guided robots. *Proceedings of the Second International Conference on Simulation of Adaptive Behaviour*, MIT Press Bradford Books, Cambridge, MA.

Cliff, D., P. Husbands and I. Harvey (1993). Analysis of evolved sensory-motor controllers. *Proceedings of the Second European Conference on Artificial Life (ECAL '93)*, Université Libre de Brussels, Brussels.

Collins, R. J. and D. R. Jefferson (1991). AntFarm: Towards Simulated Evolution. *Artificial Life II*. Eds. C. G. Langton, C. Taylor, J. D. Farmer and S. Rasmussen. Addison-Wesley.

Collins, R. J. and D. R. Jefferson (1991). An artificial neural network representation for artificial organisms. *Parallel Problem Solving from Nature*, Springer-Verlag.

Collins, R. J. and D. R. Jefferson (1991). The Evolution of Sexual Selection and Female Choice. *Proceedings of the First European Conference on Artificial Life*, MIT Press.

Collins, R. J. and D. R. Jefferson (1991). Selection in Massively Parallel Genetic Algorithms. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann.

Darwin, C. (1859). *On the Origin of Species*. John Murray, London.

Dasdan, A. and K. Oflazer (1993). *Genetic synthesis of unsupervised learning algorithms*. Dept. Computer Engineering and Information Science Technical Report. Bilkent University, Turkey.

Dasgupta, D. and D. R. McGregor (1992a). Designing application-specific neural networks using the structured genetic algorithm. *Proceedings of COGANN-92*. IEEE Computer Society Press.

Dasgupta, D. and D. R. McGregor (1992b). Nonstationary Function Optimisation using the Structured Genetic Algorithm. *Proceedings of Parallel Problem Solving From Nature 2*

Davidor, Y. (1991). A Naturally Occurring Niche & Species Phenomenon: The Model and First Results. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann.

Davis, L. (1985). Job shop scheduling with genetic algorithms. *Proceedings of an International Conference on Genetic Algorithms*. Ed. J. J. Grefenstette. Springer-Verlag, Berlin.

Davis, L. (1989). Adapting operator probabilities in Genetic Algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*. Ed. J. D. Shaffer. Morgan Kaufmann.

Davis, L. (1991). *Handbook of genetic algorithms*. Van Nostrand Reinhold, New York.

Dawkins, R. (1986). *The Blind Watchmaker*. W.W. Norton, New York.

Dayhoff, J. E. (1990). *Neural network architectures: an introduction*. Van Nostrand Reinhold, New York.

de Garis, H. (1990). Genetic Programming: Modular neural evolution for Darwin machines. *Proceedings of IJCNN*, Washington.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Ph.D. Thesis, University of Michigan, Dissertation Abstracts International **36**(10) 5140B

De Jong, K. A. (1994). Genetic Algorithms: A 25 Year Perspective. *Computational Intelligence Imitating Life*. Ed. J. M. Zurada, R. J. Marks and C. J. Robinson. IEEE Press, New York. 125-134.

De Schutter, G. and E. Nuyts (1993). Birds use self-organised social behaviours to regulate their dispersal over wide areas: evidence from gull roosts. *Proceedings of the Third European Conference on Artificial Life (ECAL '93)*, Brussels.

Deb, K. and D. E. Goldberg (1989). An investigation of niche and species formation in genetic function optimization. *Proceedings of the Third International Conference on Genetic Algorithms*. Ed. J. D. Shaffer. Morgan Kaufmann.

Deneubourg, J. L., S. Goss, N. Franks, A. Sendova-Franks, C. Detrain and L. Chretien (1991). The dynamics of collective sorting: robot-like ants and ant-like robots. *Proceedings of the First International Conference on Simulation of Adaptive Behaviour*, MIT press.

Doolittle, W. F. and A. Stoltzfus (1993). Genes-in-pieces revisited. *Nature* **361**: 403.

Dorigo, M. and U. Schnepf (1991). Organisation of robot behaviour through genetic learning processes. *Proceedings of the Fifth International Conference on Advanced Robotics*, Pisa, Italy.

Dorigo, M. and U. Schnepf (1992). Genetic-based machine learning and behaviour based robotics: a new synthesis. *IEEE Transactions on Systems, Man and Cybernetics* **22**(6).

Dorigo, M. and E. Sirtori (1991). Alecsys: a parallel laboratory for learning classifier systems. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann.

Dorit, R. L., L. Schoenback and W. Gilbert (1990). How big is the universe of exons? *Science* **250**

Dumont, J. P. C. and R. M. Robertson (1986). Neuronal circuits: an evolutionary perspective. *Science* **233**: 849-853.

Eldredge, N. and S. J. Gould (1972). Punctuated equilibria: an alternative to phyletic gradualism. *Models in Paleobiology*. Ed. T. J. M. Schope. Freeman Cooper, San Fransisco. 82-115.

Eshelman, L. J. and J. D. Schaffer (1991). Preventing premature convergence in genetic algorithms by preventing incest. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann.

Fahlman, S. E. and C. Lebiere (1990). The cascade-correlation learning architecture. *Advances in Neural Information Processing Systems 2*. Ed. Touretzky, D. Morgan Kaufmann. 524-532.

Fisher, R. A. (1958). *The Genetical Theory of Natural Selection.* (2nd Edition). Dover, New York.

Fogarty, T. C. (1989). Varying the probability of mutation in the genetic algorithm. *Proceedings of the Third International Conference on Genetic Algorithms.* Ed. J. D. Shaffer. Morgan Kaufmann.

Fogel, D. B. (1993). On the philosophical differences between evolutionary algorithms and genetic algorithms. *Proceedings of the 2nd Annual Conference on Evolutionary Programming*, EP Society, La Jolla, CA. 23-29.

Fogel, D. B., L. J. Fogel and V. W. Porto (1990a). Evolutionary programming for training neural networks. *Proceedings of the 1990 International Joint Conference on Neural Networks.* 601-605.

Fogel, D. B., L.J. Fogel and V. W. Porto (1990b). Evolving neural networks. *Biological Cybernetics*, **63**:487-493.

Fogel, L. J. (1994). Evolutionary programming in perspective: The top-down view. *Computational Intelligence Imitating Life.* Eds. J. M. Zurada, R. J. Marks and C. J. Robinson. IEEE Press, New York. 135-146.

Forrest, S. and M. Mitchell (1993). Relative building-block fitness and the building-block hypothesis. *Foundations of Genetic Algorithms 3.* Morgan Kaufmann, San Mateo, CA.

Frean, M. (1990). The upstart algorithm: a method for constructing and training feedforward neural networks. *Neural Computation* **2**: 198-209.

Funahashi, K.-I. (1989). On the approximate realisation of continuous mappings by neural networks. *Neural Networks* **2**: 183-192.

Gentric, P. and H. C. A. M. Withagen (1993). Constructive methods for a new classifier based on a radial-basis-function neural network accelerated by a tree. Technical Report, Laboratoires d'Electronique Philips, Limeil-Brevannes, France.

Ghahramani, Z. (1993). Solving inverse problems using an EM approach to density estimation. *Proceedings of the 1993 Connectionist Models Summer School.* Eds. M. C. Mozer, P. Smolensky *et al.* Earlbaum Associates. 316-323.

Gilbert, W. (1978). Why genes in pieces? *Nature* **271**: 501.

Goldberg, D. E. (1987a). Genetic Algorithms with sharing for multimodal function optimisation. *Proceedings of the Second International Conference on Genetic Algorithms.* Ed. J. J. Grefenstette. Lawrence Earlbaum.

Goldberg, D. E. (1987b). Simple genetic algorithms and the minimal deceptive problem. *Genetic Algorithms and Simulated Annealing.* Ed. L. Davis. Pitman, London. 74-88.

Goldberg, D. E. (1989a). Genetic Algorithms and Walsh functions: Part 1, A gentle introduction. *Complex Systems* **3**: 129-152.

Goldberg, D. E. (1989b). Genetic Algorithms and Walsh functions: Part 2, Deception and its analysis. *Complex Systems* **3**: 153-171.

Goldberg, D. E. (1989c). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Mass.

Goldberg, D. E. (1991). Real-coded genetic algorithms, virtual alphabets and blocking. *Complex Systems* **5**: 139-167.

Goldberg, D. E. (1991). *The theory of virtual alphabets.* Technical Report, University of Illinois.

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

Goldberg, D. E., K. Deb and B. Korb (1991). Don't worry, be messy. *Proceedings of the Fourth International Conference on Genetic Algorithms.* Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann. 24-30.

Goldberg, D. E., B. Korb and K. Deb (1990). Messy genetic algorithms: motivation, analysis and first results. *Complex Systems* **3**: 493-530.

Goldberg, D. E. and R. Lingle (1985). Alleles, loci, and the travelling salesman problem. *Proceedings of an International Conference on Genetic Algorithms*. Ed. J. J. Grefenstette. Springer-Verlag, Berlin. 154-159.

Goldberg, D. E. and R. E. Smith (1987). Nonstationary function optimization using genetic algorithms with dominance and diploidy. *Proceedings of the Second International Conference on Genetic Algorithms*. Ed. J. J. Grefenstette. Lawrence Earlbaum. 59-68.

Golub, G. H. and C. F. Van Loan (1989). *Matrix Computations (2nd Edition)*. The John Hopkins Press Ltd., London.

Gorges-Schleuter, M. (1989). ASPARAGOS, an asynchronous parallel genetic optimization strategy. *Proceedings of the Third International Conference on Genetic Algorithms*. Ed. J. D. Shaffer. Morgan Kaufmann.

Goss, S., J. L. Deneubourg, R. Beckers and J.-L. Henrotte (1993). Recipes for collective movement. *Proceedings of the Third European Conference on Artificial Life (ECAL '93)*, Brussels.

Gould, S. J. (1989). Through a lens, darkly. *Natural History* **9(89)**: 16-24.

Gould, S. J. and N. Eldredge (1993). Punctuated equilibrium comes of age. *Nature* **366**: 223-227.

Grefenstette, J. J. (1987). Incorporating problem specific knowledge into genetic algorithms. *Genetic Algorithms and Simulated Annealing*. Ed. Davis, L. Pitman, London.

Grefenstette, J. J. and J. E. Baker (1989). How genetic algorithms work: a critical look at implicit parallelism. *Proceedings of the Third International Conference on Genetic Algorithms*. Ed. J. D. Shaffer. Morgan Kaufmann. 20-27.

Grefenstette, J. J., R. Gopal, B. Rosmaita and D. Van Gucht (1985). Genetic algorithms for the travelling salesman problem. *Proceedings of an International Conference on Genetic Algorithms*. Ed. J. J. Grefenstette. Springer-Verlag, Berlin. 160-168.

Grossberg, S. (1988). *Neural Networks and Natural Intelligence.* MIT Press.

Gruau, F. C. (1992a). *Cellular encoding of genetic neural networks.* Technical Report, Ecole Normale Supérieure de Lyon, France.

Gruau, F. C. (1992b). Genetic synthesis of Boolean neural networks with a cell rewriting developmental process. *Proceedings of COGANN-92.* IEEE Computer Society Press.

Gruau, F. C. (1994). *Neural network synthesis using cellular encoding and the genetic algorithm.* Ph.D. Thesis, l'Ecole Normale Supérieure de Lyon, France.

Hancock, P. J. B. (1990). GANNET: Design of a neural net for face recognition by genetic algorithm. *Proceedings of the IEEE workshop on Genetic Algorithms, Neural Networks and Simulated Annealing applied to problems in signal and image processing*, Stirling, UK.

Hancock, P. J. B. (1992). *Coding strategies for genetic algorithms and neural nets.* Ph.D. Thesis, University of Stirling, Scotland.

Hancock, P. J. B. and L. S. Smith (1990). GANNET: Genetic design of a neural net for face recognition. *Parallel Problem Solving from Nature,* Lecture notes in Computer Science 496, Springer Verlag.

Hanson, S. J. (1990). Meiosis networks. *Advances in Neural Information Processing Systems 2.* Ed. D. S. Touretzky. Morgan Kaufmann. 533-541.

Hanson, S. J. and L. Y. Pratt (1989). Comparing biases for minimal network construction with back-propagation. *Advances in neural information processing systems.* Ed. D. S. Touretzky. Morgan Kaufmann. 177-185.

Harp, S. A. and T. Samad (1991). Genetic synthesis of neural network architecture. *Handbook of Genetic Algorithms.* Ed. L. Davis. Van Nostrand Reinhold. 202-221.

Harp, S. A., T. Samad and A. Guha (1989). *The Genetic synthesis of neural networks.* Technical report CSDD-89-14852-2, Honeywell.

Harp, S. A., T. Samad and A. Guha (1989). Towards the genetic synthesis of neural networks. *Proceedings of the Third International Conference on Genetic Algorithms*. Ed. J. D. Shaffer. Morgan Kaufmann.

Harp, S. A., T. Samad and A. Guha (1990). Designing application-specific neural networks using the Genetic Algorithm. *Advances in Neural Information Processing Systems 2*. Ed. D. S. Touretzky. Morgan Kaufmann.

Harvey, I. (1993). The puzzle of the persistent question marks: a case study of genetic drift. *Proceedings of the Fifth International Conference on Genetic Algorithms*. Ed. S. Forrest. Morgan Kaufmann. 15-22.

Harvey, I., P. Husbands and D. Cliff (1992). Issues in evolutionary robotics. *Second International Conference on Simulation of Adaptive Behaviour*. Eds. J.-A. Meyer, H. Roitblat and S. Wilson. MIT Press.

Hendriks, W., J. Leunissen, E. Nevo *et al.* (1987). The lens protein α-Crystallin of the blind mole rat, *Spalax Ehrenbergi*: evolutionary change and functional constraints. *Proceedings of the National Academy of Sciences* **84**: 5320-5324.

Hess, F. (1993). Electronic sound creatures. *Proceedings of the Third European Conference on Artificial Life (ECAL '93)*, Brussels.

Hinton, G. E. and S. J. Nowlan (1987). How learning can guide evolution. *Complex Systems* **1**: 495-502.

Hintz, K. J. (1989). Procedure learning using a variable-dimension solution space. *Proceedings of the Third International Conference on Genetic Algorithms*. Ed. J. D. Shaffer. Morgan Kaufmann. 237-242.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor.

Hollstien, R. B. (1971). *Artificial genetic adaptation in computer control systems*. Ph.D. Thesis. Dept of Computer and Communication Sciences, University of Michigan.

Hoptroff, R. G., T. J. Hall and R. E. Burge (1990). Experiments with a neural controller. *Proceedings of the 1990 International Joint Conference on Neural Networks*. IEEE Press, New York. 735-740.

Hush, D. R. and B. G. Horne (1993). Progress in supervised neural networks: what's new since Lippmann? *IEEE signal processing magazine*. January 1993:8-39.

Janikow, C. Z. and Z. Michalewicz (1991). An experimental comparison of binary and floating point representations in genetic algorithms. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann. 31-36.

Jervis, T. T. and F. Fallside (1992). *Pole balancing on a real rig using a reinforcement controller*. Cambridge University Engineering Department Technical Report CUED/F-INFENG/TR 115.

Jones, A. J. (1992). *A schemata theorem for trees*. Technical Report (preprint). Dept. Computing, Imperial College London.

Jones, A. J. (1993). Genetic algorithms and their application to the design of neural networks. *Neural Computing and Applications* **1**: 32-45.

Juliff, K. (1992). *Using a multi chromosome genetic algorithm to pack a truck*. Royal Melbourne Institute of Technology Technical Report CS TR 92-2.

Karr, C. L. (1991). Design of an adaptive fuzzy logic controller using a genetic algorithm. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann.

Keesing, R. and D. G. Stork (1991). Evolution and learning in neural networks: the number and distribution of learning trials affect the rate of evolution. *Advances in Neural Information Processing Systems 3*. Eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky. Morgan Kaufmann. 804-810.

Kirkpatrick, M. (1982). Sexual Selection and the evolution of female choice. *Evolution* **36**(1): 1-12.

Kirkpatrick, M. and M. J. Ryan (1991). The evolution of mating preferences and the paradox of the lek. *Nature* **350**: 33-38.

Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems* **4**: 461-476.

Kitano, H. (1990). Empirical studies on the speed of convergence of neural network training using genetic algorithms. *Proceedings of the Eighth National Conference on AI (AAAI-90).* MIT Press.

Kohonen, T. (1988). *Self-organisation and associative memory.* Springer-Verlag, New York.

Koza, J. R. (1992). *Genetic Programming: On the programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, Mass.

Koza, J. R. and J. P. Rice (1991). Genetic generation of both the weights and architecture for a neural network. *Proceedings of the 1991 International Joint Conference on Neural Networks.* Seattle. 397-404.

Langton, C. G., Ed. (1989). *Artificial Life.* Addison-Wesley, Reading, MA.

Lawrence, E. (1989). *A guide to modern biology.* Longman Scientific and Technical, UK.

LeCun, Y. (1989). Generalisation and network design strategies. *Connectionism in Perspective.* Ed. R. Pfeifer, Z. Schreter, F. Fogelman-Soulie and L. Steels. Elsevier Science (North-Holland).

LeCun, Y., J. S. Denker and S. A. Solla (1990). Optimal Brain Damage. *Advances in Neural Information Processing Systems 2.* Ed. D. S. Touretzky. Morgan Kaufmann.

Levenick, J. R. (1991). Inserting introns improves genetic algorithm success rate: taking a cue from biology. *Proceedings of the Fourth International Conference on Genetic Algorithms.* Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann. 123-127.

Liepins, G., M. Hilliard, M. Palmer and M. Morrow (1987). Greedy genetics. *Proceedings of the Second International Conference on Genetic Algorithms.* Ed. J. J. Grefenstette. Lawrence Earlbaum. 90-99.

Lippmann, R. P. (1987). An introduction to computing with neural nets. *IEEE acoustics, speech and signal processing magazine* **4**(2): 4-22.

Lucasius, C. B. and K. G. (1989). Application of genetic algorithms in chemometrics. *Proceedings of the Third International Conference on Genetic Algorithms.* Ed. J. D. Shaffer. Morgan Kaufmann. 170-176.

Mahfoud, S. W. (1992). *Crowding and preselection revisited.* University of Illinois IlliGAL Report 92004.

Maley, C. C. (1993). The effects of dispersal on the evolution of artificial parasites. *Proceedings of the Third European Conference on Artificial Life (ECAL '93)*, Brussels.

Manderick, B. and P. Spiessens (1989). Fine-grained parallel genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms.* Ed. J. D. Shaffer. Morgan Kaufmann. 428-433.

Mandischer, M. (1993). Representation and evolution of neural networks. *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms.* Eds. R. F. Albrecht, C. R. Reeves and N. C. Steele. Springer, Wien and New York. 643-649.

Mansour, N. and G. C. Fox (1991). A hybrid genetic algorithm for task allocation in multicomputers. *Proceedings of the Fourth International Conference on Genetic Algorithms.* Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann. 466-473.

Marshall, S. J. and R. F. Harrison (1991). Optimisation and training of feedforward neural networks by genetic algorithms. *Proceedings of the Second International Conference on Artificial Neural Networks.* 39-43.

Mason, A. J. (1993). *Crossover non-linearity ratios and the genetic algorithm: escaping the blinkers of schema processing and intrinsic parallelism.* School of Engineering Technical Report 535b, University of Auckland, New Zealand.

Mataric, M. J. and M. J. Marjanovic (1993). Synthesising complex behaviours by composing simple primitives. *Proceedings of the Third European Conference on Artificial Life (ECAL '93)*, Brussels.

Mauldin, M. L. (1984). Maintaining diversity in genetic search. *Proceedings of the 1984 National Conference on Artificial Intelligence*. 247-250.

Maynard Smith, J. (1987). When learning guides evolution. *Nature* **329**: 761-762.

Menczer, F. and D. Parisi (1990). *'Sexual' reproduction in neural networks.* Technical Report PCIA-90-06. C.N.R. Rome.

Michie, D. and R. A. Chambers (1968). BOXES: an experiment in adaptive control. *Machine Intelligence*. Ed. E. Dale and D. Michie. Oliver and Boyd, Edinburgh. 137-152.

Miller, G. F., P. M. Todd and S. U. Hegde (1989). Designing neural networks using Genetic Algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*. Ed. J. D. Shaffer. Morgan Kaufmann.

Minsky, M. and S. Papert (1969). *Perceptrons: an introduction to computational geometry.* MIT Press, Cambridge.

Mitchell, M., S. Forrest and J. Holland (1991). The Royal Road for genetic algorithms: fitness landscapes and GA performance. *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, MIT Press, Cambridge, MA.

Mitchell, M., J. H. Holland and S. Forrest (1994). When will a genetic algorithm outperform hill climbing? *Advances in Neural Information Processing Systems 6*. Eds. J. D. Cowan, G. Tesauro and J. Alspector. Morgan Kaufmann.

Montana, D. J. and L. Davis (1989). Training feedforward neural networks using genetic algorithms. *Proceedings of the Eleventh IJCAI*. 762-767.

Moody, J. (1989). *Fast learning in multi-resolution hierarchies.* Yale Computer Science Technical Report.

Moody, J. and C. Darken (1988). Learning with localised receptive fields. *Proceedings of the Connectionist models summer school.* Morgan Kaufmann.

Moody, J. and C. Darken (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation* **1**: 281-294.

Moody, J. and V. Tresp (1994). A trivial but fast reinforcement controller. *Neural Computation* **6** (to appear).

Mozer, M. C. and P. Smolensky (1989). Skeletonisation: a technique for trimming the fat from a network via relevance assessment. *Proceedings of the 1989 IEEE conference on Neural Information Processing Systems.* 107-115.

Mühlenbein, H. (1989). Parallel genetic algorithms, population genetics and combinatorial optimisation. *Proceedings of the Third International Conference on Genetic Algorithms.* Ed. J. D. Shaffer. Morgan Kaufmann.

Mühlenbein, H. and J. Kindermann (1989). The dynamics of evolution and learning - towards genetic neural networks. *Connectionism in perspective.* Eds. R. Pfeifer, Z. Schreter, F. Fogelman-Soulié and L. Steels. Elsevier Science (North-Holland). 173-197.

Ng, K. and R. P. Lippmann (1991). A comparative study of the practical characteristics of neural networks and conventional pattern classifiers. *Advances in Neural Information Processing Systems 3.* Eds. R. Lippmann, J. Moody and D. Touretzky. Morgan Kaufmann.

Nguyen, D. and B. Widrow (1990). The truck backer-upper: an example of self-learning in neural networks. *Neural Networks for Control.* Eds. W. T. Miller, R. S. Sutton and P. J. Werbos. MIT Press. 287-299.

Nolfi, S. and D. Parisi (1993). Auto-teaching: networks that develop their own teaching input. *Proceedings of the Second European Conference on Artificial Life (ECAL-93)*, Brussels.

Nowlan, S. J. and G. E. Hinton (1991). Evaluation of adaptive mixtures of competing experts. *Advances in Neural Information Processing Systems 3.* Eds. R. Lippmann, J. Moody and D. Touretzky. Morgan Kaufmann.

Nowlan, S. J. and G. E. Hinton (1992a). Adaptive soft weight tying using Gaussian mixtures. *Advances in Neural Information Processing Systems 4.* Eds. J. E. Moody, S. J. Hanson and R. P. Lippmann. Morgan Kaufmann. 993-1000.

Nowlan, S. J. and G. E. Hinton (1992b). Simplifying neural networks by soft weight sharing. *Neural Computation* **4**(4): 473-493.

Odetayo, M. O. and D. R. McGregor (1989). Genetic Algorithm for inducing control rules for a dynamic system. *Proceedings of the Third International Conference on Genetic Algorithms.* Ed. J. D. Schaffer. Morgan Kaufmann. 177-182.

Oliver, I. M., D. J. Smith and J. R. C. Holland (1987). A study of permutation crossover operators on the travelling salesman problem. *Proceedings of the Second International Conference on Genetic Algorithms.* Ed. J. J. Grefenstette. Lawrence Earlbaum.

Omohundro, S. M. (1991). Bumptrees for efficient function, constraint, and classification learning. *Advances in Neural Information Processing Systems 3.* Eds. R. P. Lippmann, J. E. Moody and D. S. Touretzky. Morgan Kaufmann.

Paredis, J. (1991). The evolution of behaviour: some experiments. *Proceedings of the First International Conference on Simulation of Adaptive Behaviour.* Eds. J.-A. Meyer and S. W. Wilson. MIT press.

Paredis, J. (1994). Steps towards co-evolutionary classification neural networks. *Proceedings of Artificial Life IV.* Eds. R. Brooks and P. Maes. MIT Press/Bradford Books.

Parisi, D., S. Nolfi and F. Cecconi (1991). Learning, behaviour and evolution. *Proceedings of the first European Conference on Artificial Life.*

Pearce, M., R. Arkin and A. Ram (1992). The learning of reactive control parameters through genetic algorithms. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.* 130-137.

Penfold, H. B., U. Kohlmorgen and H. Schmeck (1993). *Deriving application-specific neural nets using a massively parallel genetic algorithm.* Dept. of Electrical and Computer Engineering Technical Report, University of Newcastle, N. S. W., Australia.

Powell, M. J. D. (1985). *Radial basis functions for multivariate interpolation: a review.* Dept. of App. Maths and Theor. Physics Technical Report DAMTP 1985/NA 12, Cambridge University, UK.

Press, W. H., S. A. Teukolsky, W. T. Vetterling and B. R. Flannery (1992). *Numerical Recipes in C.* (Second Edition).Cambridge University Press.

Quenette, P. Y. (1993). The collective vigilance as an example of self-organisation: a precise study of the wild boar (*Sus scrofa*). *Proceedings of the Third European Conference on Artificial Life (ECAL '93)*, Brussels.

Radcliffe, N. (1990). *Genetic neural networks on MIMD computers.* Ph.D. Thesis, University of Edinburgh.

Radcliffe, N. (1991a). *Genetic set recombination and its application to neural network topology optimisation.* University of Edinburgh Technical Report EPCC-TR-91-21.

Radcliffe, N. J. (1991b). Equivalence class analysis of genetic algorithms. *Complex Systems* **5**: 183-205.

Radcliffe, N. J. (1991c). Forma analysis and random respectful recombination. *Proceedings of the Fourth International Conference on Genetic Algorithms.* Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann. 222-229.

Radcliffe, N. J. (1992). Non-linear genetic representations. *Parallel Problem Solving From Nature 2*, Elsevier Science Publishers, B.V.

Radcliffe, N. J. (1993). Genetic set recombination and its application to neural network topology optimisation. *Neural Computing and Applications* **1**(1): 67-90.

Radcliffe, N. J. (1994). The algebra of genetic algorithms. *Annals of Maths and Artificial Intelligence* (to appear).

Radcliffe, N. J. and F. A. W. George (1993). A study in set recombination. *Proceedings of the Fifth International Conference on Genetic Algorithms*. Ed. S. Forrest. Morgan Kaufmann.

Ray, T. S. (1991a). Is it alive or is it GA? *Proceedings of the Fourth International Conference on Genetic Algorithms*. Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann. 527-534.

Ray, T. S. (1991b). An approach to the synthesis of life. *Artificial Life II*. Eds. C. G. Langton, C. Taylor, J. D. Farmer and S. Rasmussen. Addison-Wesley.

Ray, T. S. (1991c). *Population dynamics of digital organisms*. Artificial Life II Video Proceedings, Addison-Wesley.

Rechenberg, I. (1994). Evolution strategy. *Computational intelligence imitating life*. Eds. J. M. Zurada, R. J. Marks and C. J. Robinson. IEEE Press. 147-159.

Renals, S. and R. Rohwer (1989). Phoneme classification experiments using radial basis functions. *Proceedings of the 1989 International Joint Conference on Neural Networks*, Washington, IEEE TAB Neural Network Committee. 461-467.

Reynolds, C. W. (1994). *Evolution of corridor following behaviour in a noisy world*. Technical Report, Electronic Arts, San Mateo. Submitted to Simulation of Adaptive Behaviour 1994.

Riolo, R. L. (1991). Lookahead planning and latent learning in a classifier system. *Proceedings of the First International Conference on Simulation of Adaptive Behaviour*. Eds. J-A. Meyer and S. Wilson. MIT press.

Robbins, G. E., M. D. Plumbley, J. C. Hughes *et al.* (1993). Generation and adaptation of neural networks by evolutionary techniques (GANNET). *Neural Computing and Applications* **1**: 23-31.

Rohwer, R. and J. C. van der Rest (1994). *Minimum description length, regularisation and multi-modal data.* Technical Report NCRG/4322, Neural Computing Research Group, Aston University, UK.

Rohwer, R., M. Wynne-Jones and F. Wysotski (1994). Neural Networks. *Machine Learning, Neural and Statistical Classification.* Eds. D. Michie, D. J. Spiegelhalter and C. C. Taylor. Prentice-Hall. 84-106.

Ronald, E. and M. Schoenauer (1994). Genetic lander: an experiment in accurate neuro-genetic control. *Proceedings of Parallel Problem Solving from Nature 1994* (to appear).

Rosenblatt, F. (1962). *Principles of neurodynamics.* Washington, Spartan Books.

Rudnick, M. (1990). *A bibliography of the intersection of genetic search and artificial neural networks.* Technical Report CS/E 90-001, Oregon Graduate Institute.

Rumelhart, D. E., G. E. Hinton and R. J. Williams (1986). Learning internal representations by error propagation. *Parallel distributed processing.* Eds. D. E. Rumelhart and J. L. McClelland. MIT Press. 318-362.

Rumelhart, D. E. and D. Zipser (1986). Feature discovery by competitive learning. *Parallel Distributed Processing* Eds. D. E. Rumelhart and J. L. McClelland. MIT Press.

Saravanan, N. and D. B. Fogel (1994). Evolving neurocontrollers using evolutionary programming. *Proceedings of the First IEEE Conference on Evolutionary Computation.* IEEE Press.

Saunders, G. M., J. F. Kolen, P. J. Angeline and J. B. Pollack (1992). *Additive modular learning in preemptrons.* Technical Report, Laboratory for Artificial Intelligence Research, Ohio State Univ.

Savage, J. M. (1977). *Evolution.* (3rd edition). Holt, Rinehart and Winston, New York.

Schaffer, J. D. (1987). An adaptive crossover distribution mechanism for genetic algorithms. *Proceedings of the Second International Conference on Genetic Algorithms.* Ed. J. J. Grefenstette. Lawrence Earlbaum.

Schaffer, J. D., R. A. Caruana, L. J. Eshelman and R. Das (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. *Proceedings of the Third International Conference on Genetic Algorithms.* Ed. J. D. Shaffer. Morgan Kaufmann.

Schaffer, J. D. (1991). On crossover as an evolutionarily viable strategy. *Proceedings of the Fourth International Conference on Genetic Algorithms.* Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann. 61-68.

Schoenauer, M. and E. Ronald (1994). Neuro-genetic truck backer-upper controller. *Proceedings of the First IEEE Conference on Evolutionary Computation.* IEEE Press. 720-723.

Schoenauer, M., E. Ronald and S. Damour (1993). Evolving networks for control. *Proceedings of Neuronimes '93*, Paris.

Schwefel, H.-P. (1994). On the evolution of evolutionary computation. *Computational Intelligence Imitating Life.* Eds. J. M. Zurada, R. J. Marks and C. J. Robinson. IEEE Press. 116-124.

Schwefel, H.-P. and R. Männer, Eds. (1991). *Parallel Problem Solving from Nature.* Lecture notes in Computer Science 496. Springer Verlag.

Shibata, T. and T. Fukada (1993). Coordinative balancing in evolutionary multi-agent-robot system using genetic algorithm. *Proceedings of the Third European Conference on Artificial Life (ECAL '93).* Brussels.

Smith, T. R., G. A. Pitney and D. Greenwood (1987). Calibration of neural networks using genetic algorithms, with application to optimal path planning. *Proceedings of the Space Operations Automation and Research First Annual Workshop.* SA conference publication.

Spears, W. M. (1994). *Simple subpopulation schemes.* Technical Report, Navy Center for Applied Research in Artificial Intelligence, US.

Spears, W. M. and K. A. DeJong (1991). On the virtues of parameterized uniform crossover. *Proceedings of the Fourth International Conference on Genetic Algorithms.* Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann. 230-236.

Stadnyk, I. (1987). Schema recombination in a pattern recognition problem. *Proceedings of the Second International Conference on Genetic Algorithms.* Ed. J. J. Grefenstette. Lawrence Earlbaum.

Stork, D. G., B. Jackson and S. Walker (1991). Non-optimality via pre-adaptation in simple neural systems. *Artificial Life II.* Eds. C. G. Langton, C. Taylor, J. D. Farmer and S. Rasmussen. Addison-Wesley. 409-429.

Suh Jung, Y. and D. Van Gucht (1987). *I*ncorporating heuristic information into genetic search. *Proceedings of the Second International Conference on Genetic Algorithms.* Ed. J. J. Grefenstette. Lawrence Earlbaum. 100-108.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning* **3**:9-44.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms.* Ed. J. D. Shaffer. Morgan Kaufmann.

Syswerda, G. (1991). Schedule optimisation using genetic algorithms. *Handbook of genetic algorithms.* Ed. L. Davis. Van Nostrand Reinhold, New York. 332-349.

Tanese, R. (1989). Distributed genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms.* Ed. J. D. Shaffer. Morgan Kaufmann.

Tinbergen, N. (1966). *The study of instincts.* Oxford University Press.

Todd, P. M. and G. F. Miller (1991). Exploring adaptive agency II: simulating the evolution of associative learning. *Proceedings of the First International Conference on Simulation of Adaptive Behaviour*. Eds. J.-A. Meyer and S. W. Wilson. MIT Press.

Torreele, J. (1991). Temporal processing with recurrent networks: an evolutionary approach. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann.

Twardowski, K. (1993). Credit assignment for pole balancing with learning classifier systems. *Proceedings of the Fifth International Conference on Genetic Algorithms*. Ed. S. Forrest. Morgan Kaufmann. 238-245.

Vico, F. J. and F. Sandoval (1991). Use of genetic algorithms in neural networks definition. *Proceedings of IWANN91*. Lecture notes in Computer Science 540, Springer Verlag. 196-203.

Vico, F. J. and F. Sandoval (1992). Neural networks definition algorithm. *Microprocessing and Microprogramming* **34**: 251-554.

Voigt, H., J. Born and I. Santibanez-Koref (1993). *Evolutionary structuring of artificial neural networks.* Technical University of Berlin Technical Report.

Vose, M. (1991). Generalising the notion of schema in genetic algorithms. *Artificial Intelligence* **50**: 385-396.

Walker, S. (1987). *Animal Learning. An introduction.* Routledge and Kegan Paul, New York.

Weigend, A. S., D. E. Rumelhart and B. A. Huberman (1990). Back-propagation, weight elimination and time series prediction. *Proceedings of the 1990 Connectionist Models Summer School.* Morgan Kaufmann. 65-80.

Weiss, G. (1990). *Combining neural and evolutionary learning: aspects and approaches.* Technical University of Munich Technical Report TUM FKI-132-90.

Whitehouse, H. L. K. (1973). *Towards an Understanding of the Mechanisms of Heredity.* (3rd edition). Edward Arnold, London.

Whitley, D. (1989). The GENITOR algorithm and selection pressure: why rank-based allocation of trials is best. *Proceedings of the Third International Conference on Genetic Algorithms*. Ed. J. D. Shaffer. Morgan Kaufmann.

Whitley, D., S. Dominic and R. Das (1991). Genetic reinforcement learning with multilayer neural networks. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Eds. R. K. Belew and L. B. Booker. Morgan Kaufmann. 562-569.

Whitley, D. and T. Hanson (1989). Optimizing neural networks using faster, more accurate genetic search. *Proceedings of the Third International Conference on Genetic Algorithms*. Ed. J. D. Shaffer. Morgan Kaufmann.

Whitley, D. and J. Kauth (1988). GENITOR: a different genetic algorithm. *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, Denver Colorado. 118-130.

Whitley, D., T. Starkweather and C. Bogart (1990). Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing* **14**(3): 347-361.

Whitley, D., T. Starkweather and D. Fuquay (1989). Scheduling problems and travelling salesmen: the genetic edge recombination operator. *Proceedings of the Third International Conference on Genetic Algorithms*. Ed. J. D. Shaffer. Morgan Kaufmann. 133-140.

Whitley, D., T. Starkweather and D. Shaner (1991). The Travelling salesman problem and sequence scheduling: quality solutions using genetic edge recombination. *Handbook of Genetic Algorithms*. Ed. L. Davis. Van Nostrand Reinhold. 350-372.

Whitley, D. and T. Starkweather (1990). Genitor II: A distributed genetic algorithm. *Journal of Experimental and Theoretical Artificial Intelligence* (in press).

Widrow, B. and M. Hoff (1960). Adaptive switching circuits. *1960 WESCON Convention Record*. Institute of Radio Engineers.

Widrow, B., N. K. Gupta and S. Maitra (1973). Punish/reward: learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man and Cybernetics* **SMC-3**(5):455-465.

Widrow, B. and S. D. Stearns (1985). *Adaptive Signal Processing*. Prentice Hall.

Wong, F. and P. Y. Tan (1992). Neural networks and genetic algorithm for economic forecasting. *AI in economics and business administration* .

Wood, D. (1991). A Von Neumann approach to a genotype expression in a neural animat. *Proceedings of the First International Conference on Simulation of Adaptive Behaviour*. Eds. J.-A. Meyer and S. Wilson. MIT press.

Wright, A. H. (1991). Genetic algorithms for real parameter optimization. *Foundations of Genetic Algorithms*. Ed. G. J. E. Rawlins. Morgan Kaufmann. 205-218.

Wynne-Jones, M. (1991). Constructive algorithms and pruning: improving the multi layer perceptron. *Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics*, Dublin.

Wynne-Jones, M. (1993). Node splitting: a constructive algorithm for feed-forward neural networks. *Neural Computing and Applications* **1**(1): 17-22.

Yao, X. (1992). *A review of evolutionary neural networks.* Technical Report, Commonwealth Scientific and Industrial Research Organisation, Victora, Australia.

Zadeh, L. A. (1973). Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man and Cybernetics* **SMC-3**.