SELF-ADAPTIVE EVOLUTION OF MODEL STRUCTURES AND PARAMETERS

A THESIS
SUBMITTED TO
THE UNIVERSITY OF ADELAIDE
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Jason John Bobbin
Monday, 25 March 2002

Certificate of Originality

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of a university or other institute of higher learning, except where due acknowledgement is made in the text.

I also hereby declare that this thesis is written in accordance with the University's Policy with respect to the Use of Project Reports and Higher Degree Theses. In particular, I consent to this thesis being made available for photocopying and loan.

28-3-2002.

Jason Bobbin

Science is like sex: sometimes something useful comes out, but that is not the reason we are doing it

Richard Feynman

Abstract

This thesis proposes a new method for the evolutionary design of models for applications in learning tasks. Evolving a model of the learning environment enables the products of the evolutionary process to be more easily understood. The evolutionary methods employed in this thesis directly encode the structures of interest and evolve the phenotype.

The models of interest are discrete structures. The evolution of discrete structures can be problematic. Novel mutation operators for the manipulation of structures need to be defined. Once defined, they have to be used to evolve useful models for the learning environment. This thesis proposes and evaluates a novel self-adaptive scheme for evolving discrete representations.

Most discrete model structures will require parameters to define the symbols with which they operate. This thesis proposes to segregate the tasks of model parameter optimisation from model structure optimisation. The resulting symbiosis of parameters and discrete structures is evolved by a consistent self adaptive scheme.

This thesis proposes a new self-adaptive, symbiotic model evolution framework, SASME. SASME is described generally and then applied to the specific task of evolving rule sets with explicit default hierarchies for learning problems. Experiments are conducted on the SASME framework to establish the efficiency of the self-adaptive mutation scheme when compared to *apriori* settings of the mutation rate. The self-adaptive scheme is found to perform optimally, removing the trial-and-error experimentation often required to get satisfactory performance from fixed mutation-rate evolutionary systems.

The SASME rule sets are evalutated on a number of difficult control tasks. In a one step procedure controllers are able to be evolved for the well known cart-pole problem under a variety of conditions. The much more difficult two-pole problem is solved by the inclusion of relational information in the rule premises. A final non-Markovian variant of the two-pole problem requiring a *recurrent* rule set is also solved by the SASME approach.

The SASME framework is used to produce rules which predict the level of algae in a lake from measured data. The rule sets produced in this data mining application are comprehensible and transparently show how the system is predicting the algae.

The more difficult problem of predicting the presence of algae species from water quality data is then undertaken and the resulting rule sets interpreted to discover what they infer about the ultimate causes of species transitions in the lake.

Acknowledgements

Firstly, I must gratefully acknowledge the intellectual, financial, and supporting role my supervisors, initially Xin Yao in Canberra, and presently Xin (in Birmingham) and Friedrich Recknagel in Adelaide have played in my PhD. Their supervision and advice has been first rate.

I thank the people who made possible the conferences which I attended during my studies, and more importantly my attendance of them, as it is through meeting and discussing research with other researchers that ideas develop. In particular, the workshops given by David Fogel, Hans–Paul Schwefel and Richard Sutton during my time in Canberra, and the 2nd international conference on the Applications of Machine Learning to Ecological Modelling held here in Adelaide all played an important role in the development of the ideas presented in this thesis. I would like to thank Xin, Bob McKay and Charles Newton in Canberra, and Fred in Adelaide for making these possible.

I would like to thank David Fogel for his support and his prompt reply to emailed requests for publications and other information, and indeed all of the people who answered my unsolicited email requests for information and papers.

I thank Hugh Possingham for his help and the table tennis, and Drew Tyre and all the people at PHLEM (Possingham Hyper-Laboratory in Ecological Modelling) for the stimulating discussions which occurred at the meetings (and the beer afterwards).

I would like to thank the people who volunteered and then actually read the thesis chapters I gave them (supervisors don't volunteer for this, it is their duty!); thanks Ron Smernik and Claire Stephens.

Discussing research with supervisors is mostly serious, with students mostly flippant, and this provides different insights; thanks Paul Darwen, Thomas Runarsson, Hugh Wilson and Mardi van der Wielan.

The postgraduate co-ordinators provide an invaluable service, thank-you Bob McKay and Miriam Ferguson in Canberra, and Rob Murray in Adelaide.

On the home-front, my house-mates have been understanding of my strange hours and behaviours; thanks Anna, Damian, Brett, Jason 2, Fiona, Virginia, Craig,

Michaela, Georgina, and Andy!

Finally, I must thank my family, and my dearest Claire, for their emotional support and practical assistance in this endeavor.

The work described in this thesis and the thesis itself were all produced almost exclusively with open-source software. This has included gcc, R, LaTeX, linux, and Swarm. The source code of the software developed over the course of this PhD will be released back to the community.

Publications arising from this thesis to 2001

[32]: Jason Bobbin and Xin Yao. Evolving rules for nonlinear control. *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation, Vienna, Austria*, 1999

[33]: Jason Bobbin and Xin Yao. Automatic discovery of comprehensible control rules by evolutionary algorithms. In Masoud Mohammadian, editor, *New Frontier in Computational Intelligence and its Applications*, volume 57 of *Frontiers in Artificial Intelligence and Applications*, pages 197–202. IOS Press, Amsterdam, 2000. Full version of conference paper [32]

[28]: Jason Bobbin and Friedrich Recknagel. Mining water quality time series for predictive rules of algal blooms by genetic algorithms. In *Proc. of 1999 Modelling and Simulation Society of Australia and New Zealand Conference (MODSIM'99)*, volume 3, pages 691–696, Hamilton, New Zealand, 6-9 December 1999. The Modelling and Simulation Society of Australia and New Zealand Inc

[31]: Jason Bobbin and Xin Yao. Automatic discovery of relational information in comprehensible control rules by evolutionary algorithms. In *Proc. of 1999 Australia-Japan Workshop, Canberra, Australia*, 23-26 November 1999

[29]: Jason Bobbin and Friedrich Recknagel. Inducing explanatory rules for the prediction of algal blooms by genetic algorithms. *Environment International*, 27(2–3):237–242, September 2001. Full version of conference paper [28]

[30]: Jason Bobbin and Friedrich Recknagel. Knowledge discovery for prediction and explanation of blue-green algal dynamics in lakes by evolutionary algorithms. *Ecological Modelling*, 146(1–3):253–262, December 2001

Contents

List of Tables

List of Figures

Preface

> The last thing one knows in constructing a work is what to put first.
>
> *Blaise Pascal*

The original proposal for this thesis was to compare traditional learning methods with the products of evolutionary learning systems. As I started to learn and read about the evolutionary computation universe ideas of a comparison diminished as I waded through the literature on evolutionary algorithms. They seemed like a very neat idea. Make the computer do the work. I go down the pub while the computer solves the problem. Surely this is the logical next step for Turing's child!

Alas, many nights were spent away from the pub as the damnable machine decided to not solve the problems that I ask it to. The final results are this thesis. Increasingly I became enamoured with trying to understand what the computer had learnt. In light of the no free lunch results it seemed even more important to value-add the products of any automated optimisation procedure. One way of value adding is to use the solution representation that I want to.

Initially I was interested in methods alone. Learning about how and why different methods of simulated evolution does (and doesn't) produce results was top of the list. Like most people in the field I have far more questions about these issues than anyone, including me, has been able to supply answers for. And again, this seemed to lead me towards evolving things I could understand. I wanted to evolve anything. Any solution whatever.

My initial toy problems in control theory led me to consider rule based systems. At the time, the only evolutionary rule based systems I was aware of were classifier systems, and my initial inclination (and advice) was not to use a *Michigan* approach. In one of the few cases during my PhD, I took the advice offered and am glad I did so. Perhaps I should have more often.

Theoretically I found the self-adaptive techniques of Fogel, Schwefel and Bäck to be most interesting. The tutorial by David B. Fogel I attended in 1997 was possibly the intellectual highlight of my ideas on methodology.

My interest in representations of solutions led to an interest in the applications I was using. It occurred to me that I can't say much about the representations I have

used to solve a problem if I don't understand the problem I am solving. There are large numbers of machine learning databases available for testing methods. But I wanted to test *learning*. I didn't really care about the mean square error or amount of mis-classification, I wanted to see what the evolutionary method had learnt about the problem. That is what automatic learning means to me. It doesn't have to tell me everything about the problem, but if it learns something interesting, I want to know what it has discovered. Otherwise my computer seems to sit there consuming power for hours before providing me with a number accurate to 5 decimal places on its expected mean square error when applied to some kind of problem. Of course, this is important, but it isn't what *I* was interested in.

This idea of concentrating on methods, but examining applications in depth, is what I have attempted to do in this thesis. I want this work to show that my method for evolving things can learn interesting things about interesting problems.

Chapter 1

Introduction

> Darwinian evolution works by the indirect and inefficient mechanism of natural selection.
>
> *Life's Grandeur* Stephen Jay Gould[1]

The field of evolutionary science began with the publication of Charles Darwin's "The Origin of Species" in November 1859. Darwin's work was influenced by earlier work, particularly Thomas Malthus's "An Essay on the Principle of Population", first published in 1798. Darwin's argument for the derivation of the origin of species is encapsulated in his summary of Chapter IV [53, p105]:

> If during the long course of ages and under varying conditions of life, organic beings vary at all in several parts of their organization, and I think this cannot be disputed; if there be, owing to the high geometric ratio of increase of each species, a severe struggle for life at some age, season, or year, and this certainly cannot be disputed; then, considering the infinite complexity of the relations of all organic beings to each other and to their conditions of existence, causing an infinite diversity in structure, constitution, and habits, to be advantageous to them, I think it would be a most extraordinary fact if no variation ever occurred useful to each beings own welfare, in the same manner as so many variations have occurred useful to man. But if variations useful to any organic being do occur, assuredly individuals thus characterized will have the best chance of being preserved in the struggle for life; and from the strong principle of inheritance they tend to produce offspring similarly characterized. This principle of preservation, I have called it, for the sake of brevity, Natural Selection; and it leads to the improvement of each creature in relation to its organic and inorganic conditions of life.

Darwin presents his argument in the origin of species as a series of observed facts that lead inescapably to the process he calls *natural selection.* The "Origin of Species" is a long collection of hard earned evidence about the observed facts referred to in Darwin's summary [59, page 49]. The choice of phrasing in this summary is striking because it describes an algorithmic understanding of natural selection. Darwin had no notion of what an algorithm is [162].

[1] [101, page 221]

1.1 The Algorithm of Natural Selection

Informally, an algorithm is a step-by-step procedure which allows an operation to be carried out without an application of intelligence [37]. Knuth [135, pages 5–6], writes that an algorithm must have five properties:

1 They must terminate after a finite number of steps.
2 They must be unambiguous.
3 They must accept input.
4 They must generate output.
5 They must be reproducible, in principle, by someone using paper and pencil.

Darwin's description of natural selection is a description of an algorithmic process, a process which is "mindless, purposeless and mechanical" [162]. The products of Darwin's algorithm of natural selection have the appearance of *design* without having been designed [59]:

> Give me Order, [Darwin] says, and time, and I will give you Design. Let me start with regularity—the mere purposeless, mindless, pointless regularity of physics—and I will show you a process that eventually will yield products that exhibit not just regularity but purposive design.

Dennet conjectures that it is the algorithmic nature of Darwin's explanation that gives it its power [59]. Dawkins likens Darwin's process of natural selection to a blind watchmaker [54, Page 5]:

> Natural selection, the blind, unconscious, automatic process which Darwin discovered, and which we now know is the explanation for the existence and apparent purposeful form of all life, has no purpose in mind. It has no mind and no mind's eye. It does not plan for the future. It has no vision, no foresight, no sight at all. If it can be said to play the role of a watchmaker, it is a *blind* watchmaker.

The modern significance of an algorithm is that it can be expressed as a computer program[2] and executed by a computer. According to Dennet [59, page 50], algorithms are:

Substrate neutral They can be implemented with any medium, and they will perform the same task.

Mindless An algorithm requires no thought to execute.

Consistent An algorithm will always do the same thing, wherever it is implemented on whatever it is implemented. The results are guaranteed.

What does Dennet mean by guaranteed results? Gould cites the *Burgess shale*, a collection of fossils from just after the Cambrian explosion[3] which contains many

[2] A computer program may be defined as the expression of a computational method in a computer language, where a computational method is a procedure having the characteristics of an algorithm (except possibly finiteness) [135, page 5][76, page 575]

[3] About 570 million years ago.

species which do not belong to any modern phylum, as evidence that were one to re-run evolutionary history, the same phylla would not be expected to become established. A re-run of the history of life would produce different results. Natural selection is a stochastic algorithm. The guaranteed results that it produces are products which are adapted to their environment.

It is a common misunderstanding to suppose that the algorithm proposed by Darwin is directed. That it leads to bigger, or smarter, or more complex individuals. It does not. There are numerous examples, the panda's thumb being a well known one [160, page 352][100, page 61][99]. Bears do not have an opposable thumb. Panda's are the bamboo eating descendants of meat eating bears. In its meat eating mode of existence the panda's thumb was co-opted into the paw structure required for a carnivorous mode of life.[4] The consumption of bamboo is greatly assisted by having the extra flexibility which comes with having an opposable digit. The bears thumb was already "committed" to the paw structure. The evolved solution was to adapt a bone in the panda's wrist, its radial sesamoid bone, into an effective but inelegant thumb. Thus the panda has 6 digits. The opposable digit that evolution supplied to the panda is not as useful as an opposable thumb. It is a suboptimal solution. Evolution has simply adapted what it had to work with to become better suited to the environment. If one were to design a panda, one would give it a thumb. What evolution does is not directed towards designing the perfect panda. It has adapted the extant forms to their ever changing environment.

The underlying mindlessness of an algorithm means that when executed on a computer, which is essentially an automated algorithm executing device, an algorithm does whatever it does automatically. One of the enduring goals of artificial intelligence research is the production of automated learning methods. These are methods which solve problems without the input of human intelligence. Simulating the natural selection algorithm is one way this goal may be accomplished.

1.2 Evolution for Automated Knowledge Acquisition

Using the algorithm of natural selection to adapt symbolic knowledge structures has been undertaken by the evolutionary computation community to solve a large number of different problems. The idea is to take Darwin's algorithm of natural selection and use it to adapt structures to learning problems. Although the evolution algorithm does not guarantee any form of optimality in the evolved solutions, it promises to be a flexible method capable of adapting a range of structures to different environments.

The view taken in this thesis is that the lack of any optimality assurances about the final solutions is *prima facie* reason to utilise representations which communicate

[4]The paw structure is common to all mammalian Carnivora.

discovered knowledge about the learning task.

The method of natural selection does not require any information about the fitness landscape that the solutions are being evolved on. The logical implementation of the algorithm does not consider in any way how the fitness of solutions are evaluated. Common-sense, and the "No Free Lunch" (NFL) theorem (see Appendix B), suggest that ignoring fitness landscape features will not lead to efficient search in general. However, it is frequently the case in machine learning that the reason for the problem being of interest is that little is known about it. Evolution can automatically adapt solutions to such problems.

Evolutionary computation can be used to evolve discrete structures for (often symbolic) learning problems, and to evolve numeric vectors for parameter optimisation. Evolutionary artificial neural networks (EANNs) are an example of a structure where both goals need to be performed; in general, the discrete architecture of the network needs to be optimised, and the weights of the connections need to be optimised. The optimisation of the architecture is a discrete problem, as individual nodes and connections are added or deleted. The optimisation of the weights is a continuous optimisation problem.

This thesis proposes a new method for the optimisation of discrete structures and continuous parameters. The method is applied to the evolution of discrete rule set structures, which operates on symbols from the environment, and a parameter vector which gives numeric meaning to the symbols referred to in the rule set.

1.3 Statement of Thesis

This thesis addresses the question: How can a representation be adapted to fit a learning environment? It is argued that a new method of symbiotic self-adaptation of a discrete symbolic structure and associated numeric components is an effective method for achieving this. The method is demonstrated by evolving comprehensible controllers in a dynamic control domain, and predictive models of ecosystem dynamics.

This thesis proposes the method of self-adaptive symbiotic model evolution, SASME. The proposed method enables evolutionary processes to successfully operate on a range of representations and evolve solutions to difficult problems with those representations. Simulated evolution is proposed as a method of adapting models to their learning environment. The method of self-adaptation is proposed to facilitate this process. It is demonstrated how self-adaptation can be exploited for the optimisation of parameters and the symbiotic optimisation of discrete model structures.

The motivation of the SASME method is to allow flexibility in the choice of model structures. The choice made in this thesis is to use rule set models which are designed to be comprehensible and evolvable. Comprehensible models address the fact that

evolution does not supply optimal solutions to problems. A comprehensible model allows for the models working to be elucidated and the method of relating input to output, which the evolution has discovered to be observed.

1.4 Contribution of Thesis

- This thesis describes a novel method for the adaptation of discrete representations in learning domains, SASME. The method proceeds by symbiotically evolving the parameters associated with a learning structure and the topology of that structure.
- The method is applied to the evolution of a novel rule set structure which allows the explicit formation of default hierarchies. The self-adaptive mechanism is compared to a variety of mutation strategies to justify the self-adaptation algorithm used.
- The method developed in this thesis is able to evolve comprehensible rule sets by means of evolutionary adaptation.
- The SASME algorithm is applicable to a wide variety of problems, as is evidenced by the diverse problems solved with it in this thesis. Rule sets are produced to give insights in two challenging learning tasks:

  Unsupervised learning of dynamic control strategies The SASME method with the rule set structures is used to evolve controllers for the well known cart-pole problem. The method is able to evolve robust controllers in a one-step procedure which automatically discovers optimal quantisation boundaries of the discretisation of the state space while evolving the rule set structures. The method is then applied to the more difficult two-pole problem which necessitates the addition of parameterised relations to the antecedent of the rule sets. Finally a very difficult non-Markovian variant of the two-pole problem is solved by using a novel recurrent rule structure. The results for this problem produced in this thesis are the only examples of machine learning, non-neural solutions to the two-pole problems. The produced rule sets perform as well as the neuro-control approaches, and are able to explicitly represent learnt knowledge about the problem.

  Data mining aquatic ecosystem data The SASME method is used to evolve rule set models for the prediction of ecosystem dynamics. Ecosystems contain noisy data with many interacting attributes affecting the dynamics. The ability of the method to elucidate learnt knowledge is demonstrated and the knowledge compared to domain knowledge for the learning task. The algorithm is a contribution to the emerging field of ecoinformatics. In the problem domain used in this thesis the SASME evolved rule sets perform competitively with the neural solutions but represent knowledge explicitly.

1.5 Outline of Dissertation

Chapter 2 reviews the method of evolutionary strategies and self-adaptive mutation for parameter optimisation. Evolutionary methods for learning problems, and the need for numeric and structural optimisation in representations for learning problems are discussed. Chapter 3 outlines the SASME algorithm. The method is described in general terms before being applied to a novel rule set structure. The self-adaptive mutation method is empirically validated by comparison with constant mutation rates in a series of experiments.

Chapter 4 introduces the well-known cart pole problem and briefly discusses some of the approaches used to solve it before evolving rule sets for controlling the system. The rule set method has a number of advantages compared to previous approaches, such as the ability to search for optimal discretisation boundaries instead of having these set *a priori*. The two-pole problem is then introduced and the requirement for the discovery of inter-attribute relationships in solving this problem discussed. The discrete rule model is then extended to allow a parameterized relationship to appear in the antecedent of rules in the rule set. The method is empirically verified and some discovered rule set models for controlling the two-pole problem presented. The problem is then made more difficult by removing velocity information. The two-pole problem without velocity is a non-Markovian control problem, and a further extension of the discrete representation is developed to solve this problem. It is empirically verified that the method of self-adaptive symbiotic model evolution can adapt this representation to solve the non-Markovian two-pole problem.

Chapter 5 discusses an aquatic ecosystem data mining problem. The SASME algorithm is used to adapt rule sets for the prediction of chlorophyll-a concentrations in a freshwater lake given some chemical and physical properties of the lake. The rule set models which are discovered are discussed and the premises with which the rule sets base their predictions are compared with domain knowledge. The more difficult question of species prediction is then attempted. Models are evolved for the prediction of the two dominant species types in the lake and the premises of the models are compared to elucidate on the observed changes in the lakes species compositions. The results highlight the benefits of elucidative representations in inductive learning problems.

Chapter 6 summarizes the thesis. Appendix A presents results on evolutionary strategies convergence and the origin of some constants used in the standard algorithm. Appendix B presents the No Free Lunch (NFL) theorem, and Appendix C describes the standard Michigan approach to learning classifier systems.

Chapter 2

Evolutionary Methods: Optimisation and Learning

> The purpose of computing is insight, not numbers.
>
> *Richard Wesley Hamming, 1915 - 1998*

Evolutionary algorithms are not the only example of simulating natural phenomena to solve computational problems. Simulated annealing is another well known example [134]. Evolutionary algorithms, and simulated annealing, belong to a class of algorithms referred to as *generate and test* methods [253]. A generate and test algorithm learns by proposing solutions to be tested in its environment and using the results of the testing to generate new solutions to be tested in the environment. Such methods are iterative. Table 2.1 shows the generic evolution algorithm.

The principal components in simulating evolution are the population, the method of modification, and the method of selection. An initial population of solutions are formed from which only some can survive. When generating new solutions, they must have some variation from the generating solution. If $\boldsymbol{x}[t]$ represents the population at time t, then the evolutionary method can be written as the difference equation:

$$\boldsymbol{x}[t+1] = s(v(\boldsymbol{x}[t])) \tag{2.1}$$

Table 2.1. *An evolutionary algorithm*

Pre: A fitness function $f \in \mathbb{R}$ must be defined which allows solution vectors to be evaluated

1. Generate a population of initial random parent solutions
2. Modify the solutions to form a child population
3. Select the next generation from the extant solutions by using the fitness function f to form the new parent population
4. Repeat from step 2 until stopping criteria is met

Table 2.2. *Friedman's evolutionary algorithm*

Observation–I A very large variety of organisms exist, and a very large number of each type are born
Observation–II Under a given environment, only a limited number of organisms survive and reproduce
Observation–III Random mutations often occur and one or more of an offsprings characteristics are different from those of its parent(s)
Observation–IV The survivors reproduce themselves and pass most of their characteristics, including the mutations to their offspring
Conclusion–I Assuming the above facts and a constant environment, a series of individuals is established which converges upon a type better fitted for survival

where $s(\cdot)$ is the selection operator and $v(\cdot)$ is the variation operator[1] [80, 75]. $v(\cdot)$ is the operator that implements step 2 in Table 2.1 and $s(\cdot)$ implements step 3. Given a problem representation and operators for that representation, the resulting evolutionary algorithm can be turned into a computer program and used to evolve solutions.

2.1 Evolution of Evolutionary Algorithms

The idea of using the algorithm of natural selection to simulate evolution for solving problems on computer has arisen several times independently. One of the earliest proposals is contained in a thesis submitted in 1956 by George Friedman [90]. Friedman noted that the principal of natural selection as proposed by Darwin can be expressed as a "Law of Natural Selection" in a number of steps consisting of 4 observations and a general conclusion as shown in Table 2.2.

Friedman's contribution was to propose that the mechanistic algorithm could be abstracted and used to evolve circuits for a variety of tasks. Friedman's experiments were never implemented in practice [76, Page 29], but the ideas he developed anticipated later work on evolving autonomous robotics, and are conceptually similar to the circuits evolved by Koza [141, 143, 144, 19, 145] and others [125]. Friedman's thesis does not appear to be influential in later work and has remained largely uncited.[2]

The well-known statistician George E. P. Box introduced the concept of mutation in

[1] In general the selection operator selects from the modified solutions and the parent solutions.

[2] It was recently reprinted in David B. Fogel, editor. *Evolutionary Computation : The Fossil Record.* IEEE, IEEE Press, 1998.

making changes to industrial processes as a means of optimising industrial output [38]. The method was termed "Evolutionary Operation" or EVOP and involved essentially a factorial design where each parameter was varied in a low and high setting. Box made the analogy with an evolutionary process explicit by comparing the modification of parameters with mutation and the analysis of production output and subsequent adoption of improved parameters with selection [38, page 83]. EVOP apparently attracted attention and was put into practice in several chemical plants in the US and is reported to be still in limited use [76, page 120].

R. M. Freidberg *et al.* published the first account of evolving a computer program [88, 89]. Although the word "evolution" does not appear in either paper,[3] it is widely accepted that the intention was to simulate evolution [76, page 145]. Freidberg used a binary string to represent computer programs, and the method contains some similarities with the ideas developed later by Holland [117] [76, page 145]. The two main similarities are the concept of implicit parallelism and schema analysis, and the problem of credit assignment. Friedberg also discusses contemporary topics such as the phenomena of "code bloat" in genetic programming and the issue of maintaining the link between parent and child phenotypes [88, page 3]:

> Form and intent, to be sure, are related quite discontinuously in the compact economical programs that programmers write, but a learning machine would probably develop much more inefficient programs in which many irrelevant instructions were scattered among the instructions that were essential to the intent. Among such programs, slight changes in form might well correspond to slight changes in intent, so that programs falling into the same classes tended to perform similar acts.

In this paragraph Freidberg introduces the idea that an evolved computer program could insert redundant instructions which would increase the length of the program without modifying the intent. The form of the evolved programs would be quite different to the ones designed by a programmer. Modern research in Genetic Programming has shown that evolved programs do, in fact, contain many irrelevant instructions to the intent. One possible reason for this is that the evolved programs add instructions in order to maintain parent-offspring fitness correlations in the face of otherwise disruptive mutations. Darwin referred to parent-offspring fitness correlations as the principal of inheritance.[4]

2.1.1 Evolutionary Programming

L J Fogel proposed to use the concept of simulated evolution on a population of algorithms to achieve a method of artificial intelligence [76, page 227]. Fogel proposed modeling artificial intelligence as predicting ones environment and performing

[3] Curiously, the word "evolution" did not appear in Darwin's first edition of the *The Origin of Species* [101, page 137]. Darwin never liked the word and first used it in *The Descent of Man.*

[4] Darwin developed his ideas on natural selection before the discovery of modern genetics, and refers to the passing of genes from parent to offspring as the principal of inheritance.

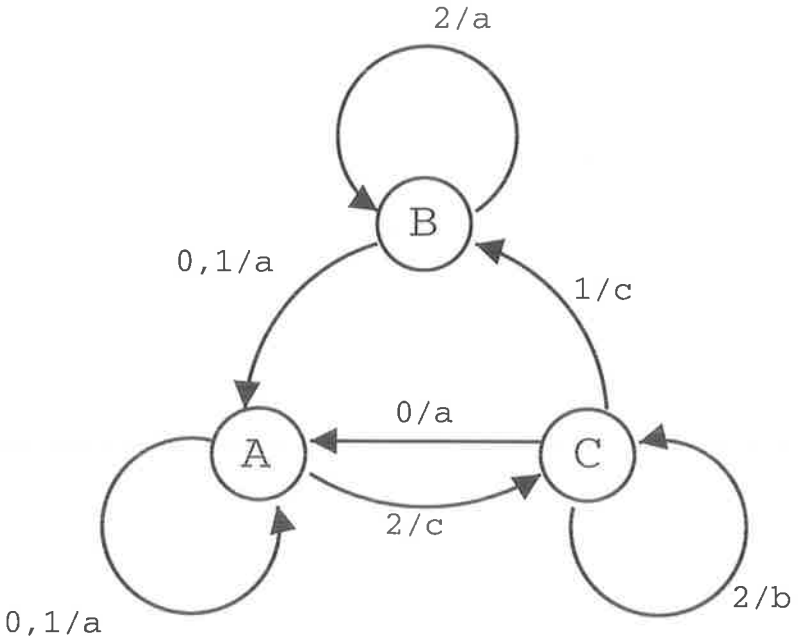


Figure 2.1. *A finite state machine*

Table 2.3. *Response of the finite state machine in Figure 2.1*

	State								
	A			B			C		
Input	0	1	2	0	1	2	0	1	2
Output	a	a	c	a	a	a	a	c	b
Next State	A	A	C	A	A	B	A	B	C

appropriate actions within that environment to achieve a desired goal [79, page 60]. The chosen representation of the environment was a string of symbols from a finite alphabet, and allowable actions were to be chosen from a finite set of symbols. The evolved program was a finite state machine, matching input symbols to output symbols depending on its current state, which was one of a finite number of possible states.

A finite state machine (FSM) is shown in Figure 2.1. The machine starts in state A, and takes an input symbol from a finite set of allowable input symbols, consisting in this case of the set $\{0, 1, 2\}$. After receiving an input symbol the machine delivers an output and changes its internal state. The FSM in Figure 2.1 produces an output from the set $\{a, b, c\}$ and switches to one of its internal states, i.e. $\{A, B, C\}$. Table 2.3 summarizes the behaviour of the FSM in Figure 2.1. The outputs of a FSM might be an action to perform within the environment, or might be a prediction of the next input symbol. In the latter case, any applicable error measure could be used, e.g. all-none, squared error, absolute error.

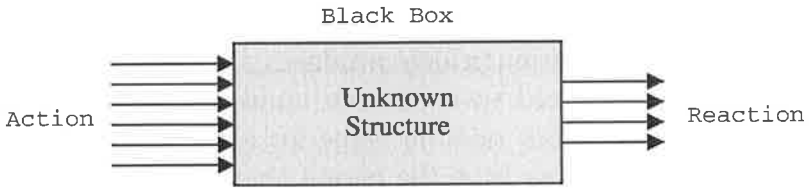


Figure 2.2. *A black box*

The number of possible configurations for a FSM given n inputs, a input symbols and b output symbols was calculated by Atmar to be [79, page 66],

$$N = (n^a b^a)^n$$

demonstrating that the search space of FSM's grows large very quickly. Fogel proposed to use the natural selection algorithm to adapt FSMs to a learning environment. The algorithm searched the massive space of possible FSMs to find well adapted machines for Fogel's chosen problems, such as the detection of prime numbers.

2.1.2 Evolutionary Experimentation

Evolutionary experimentation, like evolutionary operation, worked on physical objects with a number of parameters associated. Rechenberg likened the problems considered to a *black box* problem (Figure 2.2). A number of actions are available for variation which impact upon the reaction of the system in question. The function relating the actions to reactions is not known, and is referred to as a blackbox. In such a conceptual framework, Rechenberg suggests that an experimenter can ask 3 possible questions of the experimental object [184]:

1 What is the reaction to a given action?
2 Why does the reaction to a given action occur in the observed way?
3 How (by means of what action) does one get a desired reaction?

Rechenberg's proposal of experimental experimentation attempts to answer questions of type 3. An initial action vector $(a_1, a_2, \ldots, a_n)$ is chosen where the blackbox has n actions associated with it. In Rechenberg's conceptualization of a blackbox problem there are a number, say m, of reactions associated with an action vector (Figure 2.2), forming an optimisation problem in $\mathbb{R}^m$.

It is shown that for an inclined plane the evolutionary strategy performs more efficiently than steepest descent or what Rechenberg refers to as a Gauss-Seidel strategy; proceeding in one direction until an extreme maximum is reached and then proceeding in a direction perpendicular to the first, and so on.

In a classic experiment an evolution strategy is applied to a series of plane surfaces attached by hinges in a wind tunnel with the aim of minimizing drag. The amount of drag is modified by altering the angle of the planes. The evolution strategy

operates by altering the angle of all the hinges a random amount simultaneously. This perturbation is the mutation which produces the next shape to be tested. The hinges' positions are limited to a discrete number of angles and a binomial distribution is used to generate a random value for altering each of the angles. If the perturbed shape has less drag than the parent shape it is kept and the previous shape discarded. Otherwise the perturbed shape is discarded and the angles are set to the previous values. The shapes found by this procedure were superior to those found by gradient descend methods.

2.1.3 Genetic Algorithms

Holland developed algorithms known as genetic algorithms, which modelled the biological concepts of genes, mutation, crossover, inversion and selection along with a theoretical analysis of the application of genetic algorithms to optimisation [117]. Genetic algorithms use a binary string to represent solutions in a population of competing solutions. The idea of using a binary string representation can be traced to Bremerman in 1958 [76, page 311], when Bremerman proposed a formalized version of evolution operating on binary strings. The strings were manipulated by mutation, reproduction (sexual and asexual) and selection. Bremerman worked in the field for 35 years, and anticipated later work on optimal mutation rates for binary strings [41][76, page 311][10, page 206–7].

One of Holland's contribution is the schema theorem. The schema theorem is used to justify the binary representation that Holland uses, through the *principal of minimum alphabets*, and is seen as the central theorem for how GAs work [117][94, page 28–33][10, pages 123–126][79, page 117]. The theorem predicts that genetic algorithms work by optimally allocating trials between competing schema, or templates, and thereby maximizing the implicit parallelism of the method. This occurs when the cardinality of the representation is minimized, hence the choice of a binary representation.

More recently, schema analysis has been criticized on theoretical grounds. The idea of schema analysis "... is to discover a procedure for distributing an arbitrary number of trials ... so as to maximize the expected payoff" [117]. The method by which Holland asserts this is done is through analogy with the k-armed bandit problem. The problem with the derivation of the theorem stems from the expected loss of allocating trials in Holland's analysis not being conditioned on the previous trials, which it should correctly be [155]. Fogel states that the schema theorem is the solution to the wrong problem [79, page 116–117]. Macready and Wolpert show that a hill climbing Bayesian strategy based on previous trials performs better than Hollands strategy for allocating trials [155]. Others have reinterpreted the definition of schema to show that higher cardinality alphabets maximize implicit parallelism [7]. Fogel writes that [79, page 117]:

> In light of Macready and Wolpert (1998), there now appears to be no support for viewing the schema theorem as having fundamental importance. The

> theorem simply describes the expected number of each schemata at the next generation under proportionate selection when each complete solution is assigned a specified fitness value.

There is a large amount of critical literature on schema analysis and evolutionary performance [7, 2, 81, 78, 155, 79]. The view taken in this thesis is that the schema theorem is not a relevant tool for analysis of the structures and operators which will be used. Instead, other forms of analysis must be used.

2.1.4 Discussion

Evolution does not find optimal solutions. That is not the guaranteed results of the natural selection algorithm. Simulating evolution will frequently fail to produce a panda with thumbs. This is a strong motivation for using interesting representations in evolutionary methods. Whether a representation is interesting or not depends on the problem. One way representations can be of interest in learning problems is when the representation is able to convey information about the problem it has solved. If evolutionary learning produces a black box it will be hard to recognize that the panda it has made has no thumb.

This section has shown that there is no single, correct method, of representing solutions in a simulation of natural selection. Table 2.4 shows some of the representations used in the pioneering work on evolutionary simulation. The choice of representation depends on the type of problem being solved.

This thesis is concerned with learning problems. Learning tasks are those where a system has to learn from interacting with its environment. Objects which interact with their environment are referred to as models in this thesis. They model the inputs and outputs they observe in the environment.

Modern methods in evolutionary computation use a diverse range of operators and representations. The current algorithms can be grouped as genetic algorithms (GA) [117], evolutionary strategies (ES) [184, 208, 10], evolutionary programming (EP) [83, 70] and genetic programming (GP) [138, 139], although other terms have been used. All approaches adhere to the evolutionary computation algorithm in Table 2.1 and Equation 2.1, differing in representation of solutions, operators used and selection methods employed.

Until recently the development of the GA, ES and EP fields were entirely independent from one another. The GA and ES communities had their first contact in 1990 through their respective international conferences. The EP and ES community are reported to have had their first contact as recently as 1992, despite the fact that the two methodologies are very similar in nature [12, 77].

Much of the evolutionary computation literature contains algorithms which can not be comfortably classified as a GA, ES, GP *etc.* The terminology appears to be

Table 2.4. *A description of some of the different types of representations used with evolutionary algorithms*

Control circuits	Friedman proposed a method for the generation of autonomous robotic behaviour by utilizing the processes of variation and selection [90][76, page 29].
Computer programs	Friedberg [88, 89] attempted to create a method for the automated production of computer programs capable of solving problems. Freidberg used a binary representation for the computer programs. Later, Koza [138] would use LISP programs directly in the evolution of computer programs.
Parameters	Both Box [38] and later Rechenberg [184] mutate and select from sets of parameters i.e. vectors in $\mathbb{R}^n$. Both procedures look at optimising a process; a factory's output in the EVOP procedure [38], and the amount of drag in the initial work on evolutionary strategies [184].
Finite state machines	Fogel [83] pioneered the application of evolutionary methods to the automatic generation of finite state machines capable of solving "intelligent" applications in the 1960's [76, page 227].
Binary strings	Holland famously used binary strings to represent solutions in his genetic algorithms [117], although earlier work also attempted to solve problems using binary strings [88, 89, 84, 85].

blurred, and any distinction rather pointless. The algorithm developed in this thesis contains a free mixing of ideas and methodologies from several different canonical evolution algorithms.

2.2 Self-Adaptive evolutionary computation

With respect to operator probabilities in GAs, Mitchell writes [161, page 174]:

> ...it is not a choice between crossover or mutation but rather the balance among crossover, mutation, and selection that is all important. The correct balance also depends on details of the fitness function and the encoding. Furthermore, crossover and mutation vary in relative usefulness over the course of a run. Precisely how all this happens still needs to be elucidated.

An elegant solution to the problem of mutation strength setting is to have the algorithm decide the rate of mutation. This is called self-adaptation. Self-adaptive evolutionary methods operate by adjusting the rate of mutation in accordance to the performance of the solution's descendants.

Self-adaptive algorithms have been developed independently in the Evolution Strategies and Evolutionary Programming literature.

2.2.1 Evolution Strategies

ES are mostly used in numerical optimisation problems [11], where there is a function $\boldsymbol{f}$ such that:

$$\boldsymbol{f} : \mathcal{M} \subset \left\{ \begin{array}{c} \mathbb{B}^n \\ \mathbb{R}^n \\ \mathbb{Z}^n \end{array} \right\} \rightarrow \mathbb{R}^l \; ; \mathcal{M} \neq 0$$

$$\mathcal{M} = \left[\boldsymbol{x} \in \left\{ \begin{array}{c} \mathbb{B}^n \\ \mathbb{R}^n \\ \mathbb{Z}^n \end{array} \right\} \middle| \, g_j(\boldsymbol{x}) \geq \; (\leq) \; 0 \; \forall j \in \{1, 2, \ldots, m\} \right] \tag{2.2}$$

where there are n parameters $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$ and x_i is either a binary, integer (discrete) or real number. There are m constraints $g_j(\cdot)$ which must be satisfied, where m may be 0. There are l objective function $\boldsymbol{f} = \{f_1, f_2, \ldots, f_l\}$ and the problem is to find a non-dominated solution $\boldsymbol{x}^*$ such that:

$$f_k(\boldsymbol{x}^*) = \min\{f_k(\cdot)\}$$

where the minimization problem is considered without loss of generality since:

$$\max\{f_k(\cdot)\} = -\min\{-f_k(\cdot)\}$$

The problem is to find an $\boldsymbol{x}^*$ in $\mathcal{M}$ such that

$$\forall \boldsymbol{x} \in \mathcal{M} \left| \begin{array}{ll} f_k(\boldsymbol{x}^*) \leq f_k(\boldsymbol{x}) & \forall k \text{ and} \\ f_k(\boldsymbol{x}^*) < f_k(\boldsymbol{x}) & \text{for some } k \end{array} \right. \tag{2.3}$$

All of the problems considered in this thesis will be unconstrained (ie $m = 0$) and will have one objective function $l = 1$ (or multiple objectives rolled into one by assigning $F = f_1 + f_2 + \ldots + f_k$, where the non-dominated solution of the scalar function F is of interest, which will be the extremum $\min\{F\}$).

A defining feature of the ES approach is the use of self-adaptive strategy parameters. Where evolutionary methods usually have a set and unchanging mutation rate, the ES approach adjusts the mutation rate during the run. The self-adaptive mutation parameters are referred to as the strategy parameters or the step size. The strategy parameter(s) are usually denoted $\boldsymbol{\sigma}$. A solution, or individual, in an ES population is represented by the tuple $(\boldsymbol{x}, \boldsymbol{\sigma})$ where $\boldsymbol{x}$ is referred to as the object variable and $f(\boldsymbol{x})$ is the fitness evaluation of the individual.

Local and Global Optima

An important distinction should be made between local and global optima. A global minimum is the extreme value of some function for every point in the functions domain.

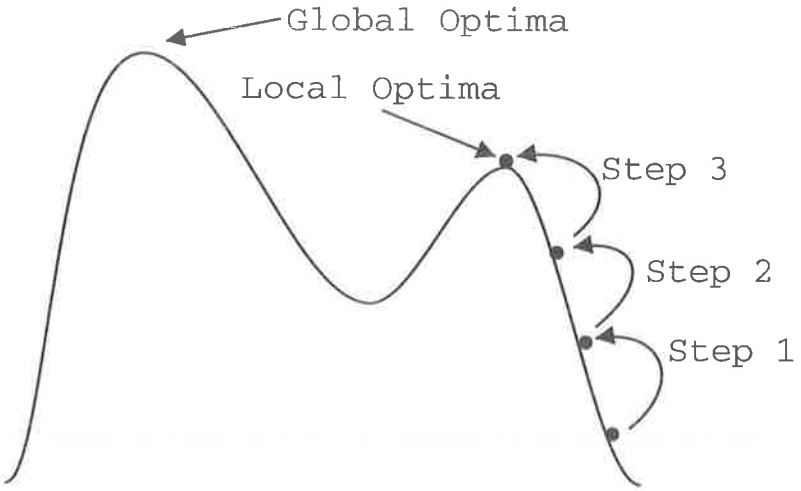


Figure 2.3. *Gradient ascent hill climbing. The curve represents the search space and the marks are the series of solutions generated. Successively better solutions are generated until a local maximum is found.*

Definition 2.1 (Global Minimum) *A solution* $\hat{\boldsymbol{x}}$ *is defined as a* global minimum *iff:*

$$f(\hat{\boldsymbol{x}}) \leq f(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in \mathcal{M}$$

The global optimum may not not be unique under this definition. A local minimum is an extrema whose immediate neighborhood contains no point with a smaller function value:

Definition 2.2 (Local Minimum) *A solution* $\hat{\boldsymbol{x}}$ *is defined as a* local minimum *iff:*

$$\exists\, \epsilon > 0 \quad \text{s.t.} \quad \forall \boldsymbol{x} \in \mathcal{M} : \parallel \boldsymbol{x} - \hat{\boldsymbol{x}} \parallel < \epsilon \Rightarrow f(\hat{\boldsymbol{x}}) \leq f(\boldsymbol{x})$$

The global minimum is also a local minimum. Typically one is searching for the global optimum. The question arises: when the global optimum is unknown, and one reaches a local optimum, how can it be determined whether it is the global optimum? There is no answer, other than to find a smaller function value, in which case the global optimum can be been ruled out. This situation is shown in Figure 2.3 for an iterative steepest ascent hill climbing algorithm.

Evolutionary algorithms are said to be global optimisers because they do not climb the nearest slope alone, but sample the search space for promising solutions. However, in common with natural evolution, simulated evolutionary methods do not discover global optima. They adapt the solutions in the population to their environment through a stochastic search. This may uncover the global optimum of a search space, and it may not. One should bear this in mind whenever the products of evolutionary search are being discussed.

(μ, λ) Evolution Strategies

The standard (μ, λ)-ES algorithm without rotation or recombination is presented below.

1 Set $g \leftarrow 1$ and generate an initial population $\mathcal{P}_g$ of λ individuals. Each individual i is a pair of real-valued vectors[5], $(\boldsymbol{x}^{(i)}, \boldsymbol{\sigma}^{(i)})$, $\forall\ i \in \{1, \ldots, \lambda\}$. The initial population of solutions $\boldsymbol{x}$ are chosen according to a uniform n-dimensional probability distribution over the solution space $\mathcal{M}$ of equation 2.2.

2 Evaluate the fitness $f(\boldsymbol{x})$ for each individual $(\boldsymbol{x}^{(i)}, \boldsymbol{\sigma}^{(i)})$, $\forall\ i \in \{1, \ldots, \lambda\}$. Sort the individuals in ascending order according to their fitness values and select the best μ parents out of the λ individuals for the next generation. The truncation level is typically set at $\mu/\lambda \approx 1/7$.

3 Each parent$(\boldsymbol{x}^{(i)}, \boldsymbol{\sigma}^{(i)})$, $\forall\ i \in \{1, \ldots, \mu\}$, creates λ/μ offspring on average, so that a total of λ offspring are generated: for $i = 1, \ldots, \mu$, $j = 1, \ldots, n$, and $h = 1, \ldots, \lambda$,

$$\hat{\sigma}_j^{(h)} = \sigma_j^{(i)} \exp(\tau' N(0,1) + \tau N_j(0,1))) \tag{2.4}$$

$$\hat{x}_j^{(h)} = x_j^{(i)} + N_j(0, \hat{\sigma}_j^{(h)}) \tag{2.5}$$

where $x_j^{(i)}$, $\hat{x}_j^{(h)}$, $\sigma_j^{(i)}$ and $\hat{\sigma}_j^{(h)}$ denote the j-th component of the vectors $\boldsymbol{x}^{(i)}$, $\hat{\boldsymbol{x}}^{(h)}$, $\boldsymbol{\sigma}^{(i)}$, $\hat{\boldsymbol{\sigma}}^{(h)}$, respectively. $N(0,1)$ denotes a normally distributed one-dimensional random number with zero mean and standard deviation one. The subscript j indicates that the random numbers are generated anew for each value of j. The *learning parameters* τ and τ' are set such that $\tau \propto 1/(\sqrt{2\sqrt{n}})$ and $\tau' \propto 1/(\sqrt{2n})$ where the constant of proportionality is usually chosen to be 1 [10, page 72].

4 Stop if the stopping criterion has been satisfied and return the best feasible individual found: otherwise, $g \leftarrow g + 1$ and take $(\hat{\boldsymbol{x}}^{(i)}, \hat{\boldsymbol{\sigma}}^{(i)})$, $\forall\ i \in \{1, \ldots, \lambda\}$ to the next generation by going to step 2.

The order of equations 2.4 and 2.5 have been found to be important [14]. This is because updating a solution with an old mutation vector allows a good solution to be created with a mutation vector which did not create that solution and may be mistuned. By updating the mutation vector first, any good solutions which are formed are formed with the mutation vector that they carry into the next generation.

An often implemented extension in equation 2.4 is to implement a lower bound. The bound, σ_{lower}, is set so that

$$\hat{\sigma}_j^{(\cdot)} > \sigma_{\text{lower}} \qquad \forall j \in \{1, \ldots, n\}$$

[5]The superscripts mean that $(\boldsymbol{x}^{(i)}, \boldsymbol{\sigma}^{(i)})$ is the i-th solution of the population $\mathcal{P}$.

This extension has been implicated empirically in helping the algorithm to escape from local optima [152]. The lower bounds used by Liang et al. [152] are small but not near the internal computer representation of zero.

Schwefel originally suggested using lower bounds to ensure that the computer representation of the number is not zero, and that its effect on the phenotype is not zero [206][207, page 112]. That is,

$$\sigma_i \geq \epsilon_a \qquad \forall i \in \{1, \ldots, n\}$$

and

$$\sigma_i \geq \epsilon_b |x_i| \qquad \forall i \in \{1, \ldots, n\}$$

where

$$\left.\begin{array}{l} \epsilon_a > 0 \\ 1 + \epsilon_b > 1 \end{array}\right\} \quad \text{chosen according to computational accuracy}$$

Schwefel's suggestion for a lower bound is concerned only with computational accuracy. A computer will represent a number as a finite number of bits, and there is only a finite number of numbers which will be represented. The stepsize is updated with a multiplicative equation, and therefore its computer representation can become smaller than the smallest representable digit, that is, zero. Once zero, the stepsize can not become non-zero by Equation 2.4. Schwefel most likely proposed this strategy because he was working in the early 1970s and the computational equipment of the time would have used relatively few bits to represent a digit. Empirically the update equation does not appear to become zero on modern equipment[6] and hence computational accuracy is not a problem for the algorithms developed here.

The implications of a lower bound for self-adaptation are not clear theoretically, and no lower bounds are implemented in the results reported in this thesis.

Selection in Evolution Strategies

There are a number of selection methods used in ESs [210][10, page 78]. The elegant notation is due to Schwefel [10, page 78]. The most common selection methods are $(\mu + \lambda)$ methods and (μ, λ) methods. In both cases the population size is λ, and μ individuals are chosen as parents in the next generation. In $(\mu+\lambda)$ the μ parents for the next generation are chosen from the combined λ offspring and μ parents of the current generation. In this scenario, the μ parents are chosen from a pool of $\mu + \lambda$ solutions. In (μ, λ) selection the μ parents of the next generation are chosen from the λ offspring of the current generation. (μ, λ) selection is not elitist, which means that the best found solution can be lost to the evolutionary process. This cannot

[6] The effect could still happen. It just hasn't been observed by the author and so a lower bound is not implemented.

happen in $(\mu + \lambda)$ strategies, as the best found solution is guaranteed to remain in the population.

$(\mu + \lambda)$ selection was originally used with a $(1 + 1)$ algorithm. One offspring is generated from the parent solution; it is kept if it is better than the parent and is discarded otherwise [210]. The (μ, λ) method requires that $\lambda > \mu \geq 1$, and is normally implemented as *truncation selection*, where the best μ solutions are chosen and the rest discarded. This is the method used in the algorithm description on page 17.

The $(\mu+\lambda)$-selection method guarantees a monotonic sequence of fitness evaluations from successive generations. At first glance, this would appear to be a more effective selection method. However, Bäck notes several disadvantages of the $(\mu+\lambda)$-selection compared with (μ, λ)-selection [10, page 79]:

1 In the event of applying the algorithm to a problem with a moving objective function, the $(\mu + \lambda)$-ES will maintain outdated solutions in the population. The alternative is to re-evaluate the μ parents every generation, creating an overhead of μ extra evaluations compared to a (μ, λ)-selection method.
2 In the event of the solution reaching a small local optimum, it may be advantageous to have a method which does not keep that optimum. A $(\mu+\lambda)$-strategy will be forced to search a local optimum until a better optimum is found.
3 Probably most importantly, a $(\mu + \lambda)$ strategy inhibits self-adaptation. The offspring in (μ, λ)-selection are not chosen entirely based upon their fitness, but also upon their ability to produce good offspring in turn. This is because a solution has a lifetime of one generation, and so any lasting effects of a solution must come through its offspring. These offspring are generated by using the self-adaptive $\boldsymbol{\sigma}$ vector, which must be favourable for the generation of high quality solutions. This applies the necessary pressure on the $\boldsymbol{\sigma}$ vector to provide good overall system dynamics.

Point 1 should also be extended to the case where the fitness evaluation is noisy, or the solutions are evaluated in a stochastic environment.

Despite the risk of divergence, (μ, λ)-strategies are the preferred method for successful self-adaptive evolution strategies search. As an extension to the $(\mu \dagger \lambda)$-selection methods an extra parameter $\rho \in \mathbb{N}$ can be introduced to signify the maximum lifetime of a solution in the population [210]. A (μ, λ)-selection method corresponds to every solution having a lifetime of one generation, $\rho = 1$. The $(\mu + \lambda)$-selection strategy corresponds to every solution having a non-finite maximum lifetime, $\rho = \infty$. This allows the advantages and disadvantages of the selection methods to be scaled between one another. Throughout this thesis a (μ, λ) method will be used.

The selection pressure of the (μ, λ) selection can be varied by changing the ratio μ/λ. This is normally set at $1/7$ [14], based on maximizing the acceleration in convergence due to self-adaptation for the sphere model. The initial settings for the step size $\boldsymbol{\sigma}$ must be large enough to avoid quick descent to poor local optima near the initial positions of the solutions; if the step size is too small initially, the search

does not explore the search space sufficiently. Schwefel [208, Page 143] suggests setting the initial step size as $\sigma_i \approx \Delta x_i/\sqrt{n}$, where Δx_i is the estimated distance between the starting point and the optimum. Bäck [14, Page 80], however, suggests that in practice the initial deviations should be lower than this, and he uses a value of $\sigma_i = 3$ in all of his experiments. Obviously a constant value will not scale for all problems.

Empirical data suggests the only problem with an initial step size which is too large is that the evolution will take a few more generations to become stable and not random. So long as any bounds on variable sizes are handled carefully[7], the empirical studies in this thesis support the notion of using initial step values as large as is practical to ensure coverage of the space. Pathological fitness functions could be constructed which may not ever converge when the initial step size is too large.

Many extensions to the above notation and selection schemes are proposed in [210], including parameters to increase the lifespan of solutions and differentiate between different recombination methods, however the basic (μ, λ) method will be used in this thesis.

Recombination in Evolution Strategies

There are a number of recombination operations possible at Step 3 above. The most commonly used types of recombination $r(\cdot)$ of the population at generation g, $r(\mathcal{P}_g) = (\hat{\boldsymbol{x}}^{(\cdot)}, \hat{\boldsymbol{\sigma}}^{(\cdot)})$ in (μ, λ)-ES are as follows [11, 206, 207, 10]; each component of the objective variable in the new population is:

$$\hat{x}_i^{(\cdot)} = \begin{cases} x_i^{(a)} & \text{No Recombination} \\ x_i^{(a)} \text{ or } x_i^{(b)} & \text{discrete} \\ x_i^{(a)} + \frac{1}{2}(x_i^{(b)} - x_i^{(a)}) & \text{intermediate} \\ x_i^{(a)} \text{ or } x_i^{(b_i)} & \text{global, discrete} \\ x_i^{(a)} + \frac{1}{2}(x_i^{(b_i)} - x_i^{(a)}) & \text{global, intermediate} \\ x_i^{(a)} + \chi(x_i^{(b)} - x_i^{(a)}) & \text{generalised, intermediate} \\ x_i^{(a)} + \chi_i(x_i^{(b_i)} - x_i^{(a)}) & \text{generalised, global, intermediate} \end{cases} \tag{2.6}$$

Where $a, b, b_i \in \{1, \ldots, \mu\}$ $\forall i \in \{1, \ldots, n\}$ and χ, χ_i are uniformly randomly drawn from $[0, 1]$ (independent for each i). The subscript i indicates that the value is generated anew for each component of the vector. The discrete recombination mentioned above would correspond to uniform crossover in the GA. The "or" means an equally likely decision. The non-generalized forms of recombination can be extracted from the generalized forms by setting $\chi = \frac{1}{2}$ for non-global forms, and $\chi_i = \frac{1}{2}$ $\forall i \in \{1, \ldots, n\}$ in global forms.

[7] In practice, this requires the step size to be bounded above to prevent divergence. Once a good solution vector is found in conjunction with a small σ_i value, the solution will continue produce good offspring.

Any of the recombination schemes in Equation 2.6 could be used on the strategy vectors $\boldsymbol{\sigma}^{(\cdot)}$. Schwefel recommends using global intermediate recombination [208, page 148] on the strategy parameters $\boldsymbol{\sigma}$. Discrete recombination of the object variables $\boldsymbol{x}$ has been identified as the preferred method [10, page 75], and both of these recommendations will be used throughout this thesis.

Rotation angles

Early ES algorithms incorporated a single mutation stepsize σ, constant for each dimension of the object variable $\boldsymbol{x}$ [10, page 68]. Equation 2.4 then becomes:

$$\hat{\sigma}^{(h)} = \sigma^{(i)} \exp(\tau' N(0,1))$$

This thesis uses the usual extension of incorporating a separate step size for each component of the objective variable, ie, if $\boldsymbol{x} \in \mathbb{R}^n$ then $\boldsymbol{\sigma} \in \mathbb{R}^n$. Most of the theoretical results about ES use a single global step size σ, which guarantees spherical symmetry of the mutation distribution. This makes the mutations independent of the co-ordinate system. When using more than one mutation step size, the mutations will be dependent on the co-ordinate system, and hence rotating the fitness function will change the performance of the ES on that fitness function.

To prevent this co-ordinate dependence, another set of parameters can be associated with the solution to represent the rotation angle of the mutation vector, $\alpha_m \in [-\pi, \pi]$, $m \in \{1, \ldots, n_\alpha\}$. When there are n step sizes, there will be

$$n_\alpha = \frac{n(n-1)}{2} \tag{2.7}$$

angles required [13]. The angles allow for arbitrary linear correlations amongst the components of the object variable, thus enabling arbitrary orientation of the mutation ellipsoids (that is, areas of equal probability under mutation). The mutation vector is generated from the covariance matrix C, where

$$c_{ij} = \frac{(\sigma_i^2 - \sigma_j^2)\tan(2\alpha_{ij})}{2}$$

and the mapping from the vector interpretation of $\boldsymbol{\alpha}$ to the matrix interpretation is given by the index transformation $(i,j) \in \{1, \ldots, n-1\} \times \{i+1, \ldots, n\}$ to $\{1, \ldots, n \cdot (n-1)/2\}$ given by $(i,j) \mapsto \frac{1}{2}(2n-i)(i+1) - 2n + j$ [10, pages 69–70]. The mutation distribution is then $\boldsymbol{N}(\mathbf{0}, C)$. Bäck gives a generalized description of the rotation angles where there are $n_\sigma \in [1, \ldots, n]$ different mutation step sizes.

The rotation angles are updated by

$$\hat{\alpha}_j = \alpha_j + \beta \cdot N_j(0,1) \qquad \forall j \in \{1, \ldots, n \cdot (n-1)/2\}$$

where β is usually 0.0873 (five degrees in radians) and where α is circularly mapped onto the range $[-\pi, \pi]$ [13][10, page 72], ie

$$|\alpha_j| > \pi \quad \Rightarrow \quad \alpha_j = \alpha_j - 2\pi \operatorname{sign}(\alpha_j)$$

The incorporation of rotation angles adds extra parameters and complexity to the algorithms. For the investigations conducted in this thesis, the extra overhead is unwarranted. The reader is referred to Bäck for a discussion of rotation angles and their implementation [10].

Step Control

A number of modifications to updating the step control have been studied. One interesting method is the *fast evolution strategies* algorithm which changes the gaussian distribution in Equation 2.5 with a Cauchy one [252, 250, 251, 203, 48, 254, 25]. The Cauchy distribution is thought to provide a greater probability of escape from local optima through its "fatter tails".

The oldest method of updating the strategy vector in an evolution strategy is Rechenberg's 1/5 rule [10, page 67]. The rule is theoretically based to maintain the highest convergence velocity. It states that 1/5 of all mutations should be successful. If it is more, then increase the strategy vector, if less, decrease the vector. The 1/5 rule is discussed in the analysis of ES convergence results undertaken in Appendix A. The principal disadvantage of the 1/5 rule is that it cannot be used to scale the mutations of each component vector independent of one another.

In contemporary evolutionary strategies the step length is updated according to Equation 2.4 at each generation. If the update probability distribution is labelled $\overline{Z}$, then the step length is adjusted according to the scheme:

$$\hat{\sigma}_j^{(h)} = \sigma_j^{(i)} \overline{Z} \tag{2.8}$$

Schwefel [208, Page 143] and Bäck [14, Page 72] note several desirable characteristics which the above step control equation should have:

1 The equation should be multiplicative so that negative values are excluded.
2 The median $\overline{\xi}$ of the distribution $\prod_{i=1}^{\infty} \overline{Z_i}$ should be one so there is no deterministic bias in the update of σ over several generations. That is

$$\sigma = \sigma \prod_{i=1}^{\infty} \overline{Z_i}$$

3 The above characteristic requires that the probability of occurrence of a particular random value must be the same as that of it's reciprocal. This will mean that any overall trend in changes in the step length will be the product of selection alone.
4 Small changes in the step length σ should occur more often than large ones. (ie $\mathrm{E}(Z) \approx 1$).

Schwefel [208, Page 143] notes that the above conditions are satisfied by the log-normal distribution

$$\overline{Z} = e^{Y} \tag{2.9}$$

where Y is normally distributed. Rechenberg proposed a symmetrical two-point distribution for the update formula. When σ is updated by Rechenberg's distribution [24]

$$\sigma := \begin{cases} \sigma_0(1+\beta), & \text{if } \mathrm{u}(0,1] \leq 1/2 \\ \sigma_0/(1+\beta), & \text{if } \mathrm{u}(0,1] > 1/2 \end{cases} \tag{2.10}$$

the update also obeys most of the characteristics for the update formula suggested by Schwefel, where $\mathrm{u}(0,1]$ is a uniform sampling from the interval $(0,1]$, and $0 < \beta \lessapprox 3$.

Schwefel [208] also argues that the length of the step size vector for the offspring should be different to that of the parent, otherwise offspring would be generated mostly at a constant distance from the parent vector. The vector resulting from Equation 2.8 will be $(\sigma_1 \cdot \overline{Z}_1, \ldots, \sigma_n \cdot \overline{Z}_n)$ where $\overline{Z}_i$ denotes an independent sampling from the distribution $\overline{Z}$ for each i. The expected Euclidean distance of the new objective variable from the current variable after the mutation vector is updated is

$$r = \sqrt{(\sigma_1 \cdot \overline{Z}_1)^2 + \ldots + (\sigma_n \cdot \overline{Z}_n)^2}$$

If all the σ_i are similar in size then an application of the (weak) law of large numbers,

$$\lim_{n\to\infty} \mathrm{P}\left[\frac{\sum_{i=0}^{n} X_i}{n} = \mu\right] = 1$$

where each of the X_i are independent with finite variance and mean μ, and P denotes the probability measure [37], shows that the modulus of the step size of the updated vector will be the same as the original vector.[8] In practice, Schwefel suggests the effect is important at around $n \approx 30$ [208].

The effect is undesirable, since the algorithm would not be exploring different variances in updating the objective variable in Equation 2.5. Children would always be produced at the same expected distance from their parents. To solve this, the lognormal distribution in Equation 2.9 has the normal variable Y composed of the sum of two independent normal variables, one sampled anew for each component and one sampled only once for the individual. By sampling only once per individual, each individual is given a bias in the change in size of the n-ellipsoid that the mutations are generated in at each generation. There is still no overall bias in the change in mutation volume since the volume is equally likely to increase as decrease.

Bäck states that the parameters τ and τ' in equation 2.4 are robust [10, page 72].

[8] The method of *fast evolution strategies* uses a Cauchy distribution in the update which does not have a finite variance [250]. Therefore the law of large numbers effect would not hold, at least as stated. In this case independent sampling of the distribution at each vector component update may be all that is required.

Convergence Results for Evolution Strategies

Most convergence results for ES use the sphere model to calculate convergence velocities. There is a gap between theory and practice in ES implementations. Convergence results in the limit exist only when the mutation distribution remains covering. This says little about practical applications. Convergence speed results are of more interest from an implementation perspective. Results are derived for a variety of model fitness functions, and it is argued that these model functions approximate the likely local topology of the search space. This is debatable. The convergence speed analysis does, however, provide the basis for the 5 in Rechenberg's 1/5 rule and the optimal number of offspring to produce from λ parents. Convergence results are presented in Appendix A.

2.2.2 Evolutionary Programming

Evolutionary programming (EP) was developed as a method for the attainment of intelligent behaviour by computers. Intelligent behaviours are defined as *the capability of a system to adapt its behaviour to meet its goals in a range of environments* [72]. This goal was initially attempted by the use of finite state machines.

Like ES, EP is a method which simulates evolution from a phenotypic view point. Parent solutions are modified so that there is a continuous range of possible offspring behavious, and a strong behavioural link is maintained between parent and offspring [82]. The emphasis in EP is placed on finding useful mathematical transformations to modify solution p in phenotype space to a solution p' in phenotype space without consideration of any underlying natural genetic operators [73, 77].

Whilst there are no definitive classifications of EAs, there are some characteristics that most practitioners would agree are typical of what people refer to as EP approaches

1 Crossover is not used
2 Selection is stochastic
3 Many and varied solution representations are used
4 Self-adaptation is used

Crossover is typically not used as the algorithms utilize a phenotypic representation of solutions where crossover makes no sense. This is illustrated in the FSMs in Section 2.1.1, page 9, where it is hard to construct a crossover operation that is likely to produce useful offspring, since the representation of the solution to the problem is a phenotypic one.

The selection method employed in EP is typically a method known as *tournament selection*. In the tournament selection method, each individual x_k from the combined parent and offspring populations, $\mathcal{S}$, are compared to a subset of $q(> 1)$ individuals randomly chosen from $\mathcal{S}$. x_k is assigned a win score, $w \in \{0, \ldots, q\}$, according to

how many individuals in the subset it outperforms (has higher fitness than). All individuals in $\mathcal{S}$ are then ranked according to their win scores and the best are chosen to form the next generation [12]. The selection method is therefore elitist, and convergence results will hold while the mutation operator is covering.

EP has been used on a range of representation, including GP-like S-expressions [45, 46, 47], Neural Networks [74, 255], FSM [128] and real vectors [252].

Although the original EP work on FSM did not employ self-adaptation, EP most commonly does use self-adaptation, eg [79]. The original method of self-adaptation employed in EP was developed by D. Fogel in 1991 independently to that of Schwefel in 1981 [207]. The scheme was

$$\hat{x}_j^{(i)} = x_j^{(i)} + N_j(0, \sigma_j'^{(i)}) \tag{2.11}$$

$$\hat{\sigma}_j^{(i)} = \sigma_j^{(i)} + \zeta\sigma_j^{(i)} \cdot N_j(0,1) \tag{2.12}$$

in the notation of the corresponding ES equations (Equation 2.5 and Equation 2.4 on page 17). ζ is a scaling constant which is chosen so that the σ_j tends to remain positive [79, page 158]. Where the σ_j become negative they are reset to some small positive value $\epsilon > 0$. Empirically equations 2.11 and 2.12 were found to not perform as well as the ES update equations, Equations 2.4 and 2.5, and the lognormal update of Schwefel is widely used in the EP community now [79, pages 158–159].

2.2.3 Discussion

Self-adaptive evolutionary methods are strong optimisation procedures in a wide variety of parameter optimisation tasks. They have been shown to be able to adapt the mutation distribution effectively in a range of fitness domains including domains where the fitness evaluation is noisy. The key to the step size control used is that it is an unbiased update of the mutation probabilities, and will therefore only show behaviour like net increases or decreases when such behaviour is beneficial to the population. The mean mutation rate which is applied at each generation of a self-adaptive method is an emergent property of the problem being solved.

Self-adaptive methods emphasise the phenotypic links between parent and offspring. The methods are strongest where the selection scheme maintains the selection pressure based on an ongoing link between parent and child behaviours. This is most clearly done in non-elitist methods where the parent's objective value determines the number of offspring the parent will have and not the survival time of the parent in the population. Elitist methods allow parents with good objective values but no ability to produce children which are similarly characterised to persist in the population.

Empirically, self-adaptive methods appear to perform most effectively when the objective values of offspring are recombined by discrete recombination, and the strategy vectors recombined by intermediate recombination. Relatively few empirical studies

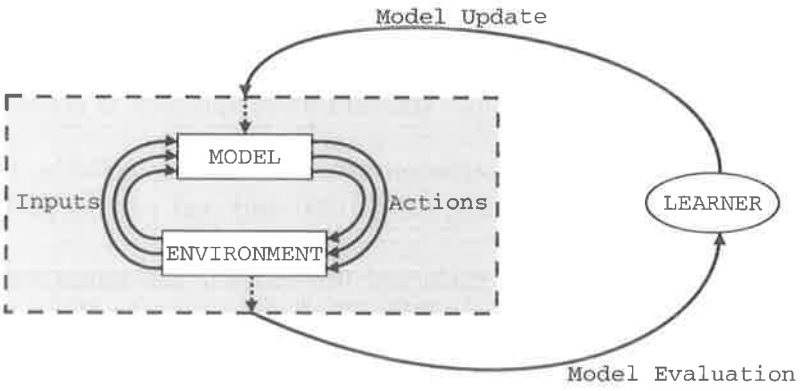


Figure 2.4. *The learning process.*

have been done to assess the performance of rotation angles. Theoretically, rotation angles have some interest, however they can introduce a considerable number of additional parameters (Equation 2.7). This extra overhead is not seen as warranted in the investigations conducted in this thesis and is unlikely to produce any improvements.

For the reasons outlined in this section, this thesis will use a self-adaptive evolutionary strategies method with (μ, λ)-selection, discrete recombination of the objective parameters, and intermediate recombination of the strategy vectors.

2.3 Evolutionary Learning

This thesis describes a self-adaptive evolutionary *learning* system. Evolutionary learning has a long history, dating from the pioneering work of Friedberg on evolving computer programs [90], Fogel's FSMs [83], and Rechenberg's drag reducing evolution strategy [184], among others [76].

Figure 2.4 shows a generic learning situation. The environment represents the problem, and the model is the proposed solution. The model and the environment interact for a certain time before supplying the learning algorithm with some form of feedback. The learner then proposes a new model which is similarly evaluated. In the case of evolutionary algorithms the learner might be a population of models or alternatively a population of parameters to be tried with a fixed model structure.

Learning problems can be divided into three broad categories depending on the information supplied to the learning algorithm [16]:

Supervised Learning Supervised learning problems supply the most information

to the learning algorithm. For each input pattern seen by the learning algorithm the correct output pattern is also supplied.

> **Example 2.1:** Most data mining problems are examples of supervised learning problems and prediction problems. An example is the famous iris data of Anderson and Fischer [3, 66]. Measurements of the sepal length and width and petal length and width from 50 examples of each of 3 different species of iris were collected, as shown in Figure 2.5. A supervised learning task from this data would be to predict the species of iris from the measurements provided. The input pattern would consist of 4 real quantities, the sepal length and width, and the petal length and width. The output would be one of three classes corresponding to the species. The problem is supervised since for each input pattern the model could be told the correct output.

Unsupervised Learning Unsupervised learning problems supply the least information to the learning algorithm. No output pattern is provided to the learning algorithm, which must instead discover its own relationships between input patterns. Evolutionary learning methods are seldom applied to unsupervised learning problems, however there is some scope for them to be.

> **Example 2.2:** Examples of an unsupervised learning algorithms include Kohonen neural network, [136], and many clustering methods [126]. An unsupervised learning problem could be formed from the iris data shown in Figure 2.5 by presenting the learning algorithm with the input measurements but not the species of the iris. The learner would divide the data according to similarity criteria to try and learn similarities in the 150 examples presented, without any reference to the actual species.

Reinforcement Learning Reinforcement learning (RL) problems [17] can be thought of as providing a level of information in between supervised and unsupervised. Typically there is some occasional performance information provided to the learning algorithm, but not for every input pattern seen. A typical RL problem consists of an agent connected to its environment via perception and action [130].

> **Example 2.3:** An example of a reinforcement learning task is the robot navigation problem, where a robot situated in a maze perceives its environment and receives rewards by moving to food or energy. The learning algorithm receives as input (the perception of the robot) and produces an output (a direction for the robot to move in). The algorithm has to learn to acquire the intermittent reward of food. There is no correct action shown for a given input, however, the learning algorithm receives some idea on whether its actions are correct from the value of food or energy that it acquires over time.

Evolutionary methods have been applied to all three types of learning problems. The methods used in this thesis will address RL problems and supervised learning problems.

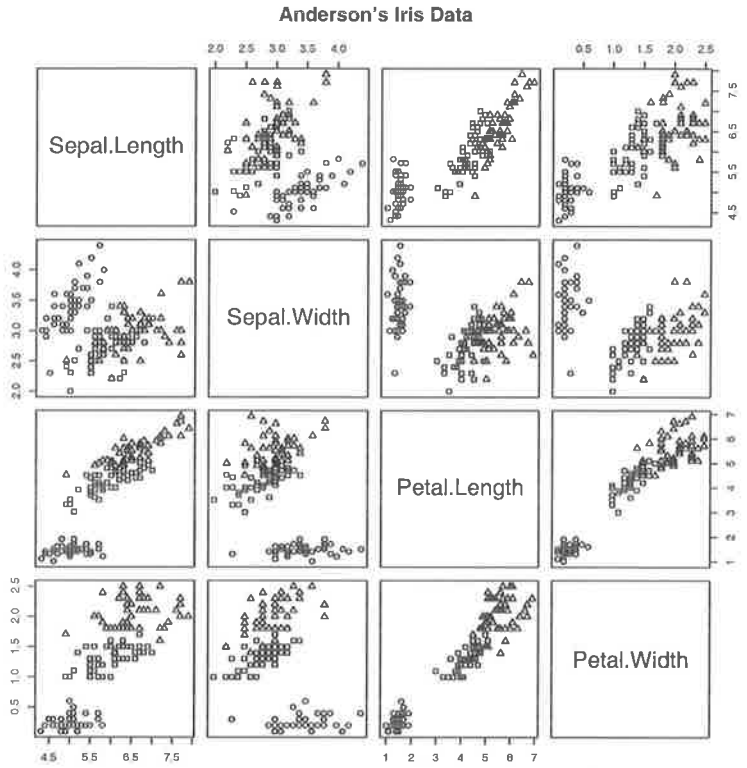


Figure 2.5. *Scatter plots of Anderson and Fischer's iris data. Each plot shows the distribution of the three different species,* Iris setosa, I. versicolor, *and* I. virginica, *as a pairwise function of the measured data (in centimeters) [3, 66, 126].*

There are a number of reasons one may consider evolving the solution to a learning problem as opposed to traditional approaches:

1 Evolution is a global search procedure. It may not be guaranteed to find the optimal solution, but empirically appears to find better optima than gradient based search, such as back-propagation for neural network weight optimisation.[9] There is some evidence that evolution's global search ability can give it an advantage over greedy approaches for rule generation, like CART and C4.5, especially when there are many interactions between attributes [86, 123].

2 Evolution may be preferred because it can explore more sophisticated representations. Rule sets, neural networks, equations and computer programs have all been evolved. The fact that evolution can adapt many and varied representations is exploited in this thesis to add modules to the representation under evolution as required to solve increasingly difficult problems.

3 Most traditional learning methods make assumptions about the search space, such as gradient information or the distribution of variables, which evolution does not.

4 Evolutionary algorithms are able to be implemented in a parallel manner. This

[9] A discussion of the relevant "No Free Lunch" results is conducted in Appendix B.

may make them more scalable than other methods in some domains.

There are also reasons why evolutionary methods may not be appropriate:

1 Evolutionary search is known to be slower than other methods for some applications [249].
2 The products of evolutionary search are unproven. Gradient descent methods guarantee a local optimum at least. Evolutionary methods do not. For this reason evolutionary search is often complemented by local search procedures.

Evolutionary methods have commonly been applied to discrete rule structures. The original structures used were Holland's learning classifier systems (LCS).

2.3.1 Learning Classifier Systems

LCSs are one of the founding ideas in the modern field of evolutionary learning. The LCS is a system designed to "infer environmental patterns from experience and associate 'appropriate' responses sequences with them" [115]. Goldberg states that a classifier system consists of [94, page 221]:

1 Rule and message system.
2 Apportionment of credit system.
3 Genetic Algorithm.

Rules are generally of the form [94]:

IF *condition* THEN *action*

There are two main approaches to evolutionary learning with LCS [56, 246][57, page 626–627]. In the *Michigan*[10] approach, the entire population of the EA forms the model in Figure 2.4. An individual in the EA population is a particular sub-part of the solution, most commonly a rule. Individual rules compete to remain in the population. This approach was developed by Holland [119, pages 171–181].

The second approach is the *Pittsburgh* approach. In this approach, each individual in the population is a complete model. This approach more closely corresponds with the ideas presented previously about what EAs are.

In the *Michigan* approach, the emphasis is for competition amongst individual rules. The population forms the complete rule set. The principal problem that occurs is one of *credit assignment.* What rules in the population are responsible for the populations performance?

[10]The names refer to the universities where the different approaches originated.

2.3.2 The Michigan Approach

By emphasizing the competition between individual rules, Michigan-style classifier systems attempt to learn complex concepts by discovering and combining simpler *build blocks* [121]. In this way it is thought that complex concepts can be learnt through the formation of *default hierarchies.*

Rule–II IF *lane is blocked* THEN *send alert message*

A default hierarchy is a group of coupled rules where subsequent rules in a hierarchy are able to correctly classify more specific conditions. They form exceptions to the initial rules. The hierarchies were proposed to form from the interactions in the system that determine the bid strength of the individual rules, and be sustained by the *bucket brigade algorithm.*

Rule–I is a very general rule which will make mistakes. Rule–II is more specific and will correct Rule–I. The functioning of Rule–II is dependent on later rules interpreting the alert message and avoiding the obstacle. Rule-II is more specific, and so has an advantage in the bidding competition. Since the second rule corrects the first, both rules are better off. Rule–I and Rule–II are said to form a *default hierarchy.* Figure 2.6 shows graphically the effects of a simple default hierarchy consisting of two rules, a general rule and an exception.

The idea of a default hierarchy is similar to the method of representing exceptions in traditional rule-based systems:

Rules in Holland's classifier system are context dependent, where the context is given by the other rules in the system. In Example 2.4, the correctness of Rule–I is dependent on the presence of Rule–II, and hence the algorithm's evaluation of the correctness of Rule–I is context dependent. Remove Rule–II and Rule–I may not survive. Holland called this linking of rules a kind of symbiosis. He terms the first rule the default rule and the second rule the "exception" rule, and notes that Rule–II may make mistakes which could be corrected by yet more specific exceptions. Appendix C describes Holland-style classifier systems and some of the extensions which have been made to them.

A more explicit representation of a kind of default hierarchy is furnished by a ripple down rule set, as shown in Figure 2.7 [49]. When an antecedent is true the exception rule(s) antecedent is tested before the rules consequence is applied. The exception rule is context dependent, its context being the previous rule. In this way entire rulesets can be produced, as shown in Figure 2.8.

Ripple down rules were developed for use with expert systems where they can help in the maintenance of such systems and the reuse of knowledge [49, 194, 195], although

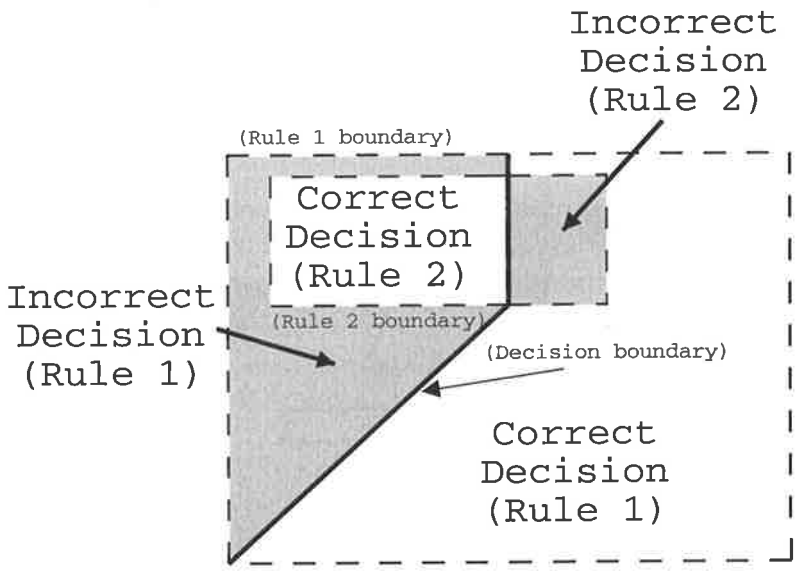


Figure 2.6. *An emergent default hierarchy. The grey area represents errors made by the two rules, the dashed lines are the boundaries of objects covered by the rules and the bold line is the decision boundary between objects whose correct action differs. Rule* 1 *is the general rule. Rule* 2 *increases rule* 1*'s payoff by preventing it from making mistakes, and rule* 2 *increases the overall system performance.*

IF condition A THEN action **1** EXCEPT IF condition B THEN action **2**
IF condition C THEN action **3**
IF condition D THEN action **4**

A	B	C	D	Action
1	1	◇	◇	**2**
1	0	1	◇	**3**
1	0	0	◇	**1**
0	◇	◇	1	**4**

Figure 2.7. *An example ripple down rule and associated action table. The* ◇ *symbol means that the condition can be either true or not true*

they have also been used in inductive systems [205]. Other proposals for adding exceptions include methods which can also negate the conditions of the initial rule [219]. It has been noted that ripple down rules (RDR) are more compact than flat rule lists [205]. That is, a flat decision list can be converted to a RDR with *at most* as many tests. They are said to be more comprehensible, as they are similar to the way that humans use knowledge. This was the motivation for their use as a representation in expert systems.

Holland's default hierarchy decides the order of rules based on a complex formulation of the bid strength and the specificity of the classifier, as discussed in Appendix C. Rule lists with exceptions decide on the priority of rules according to the topographic structure of the rule set. The latter makes it more obvious to an observer what rules

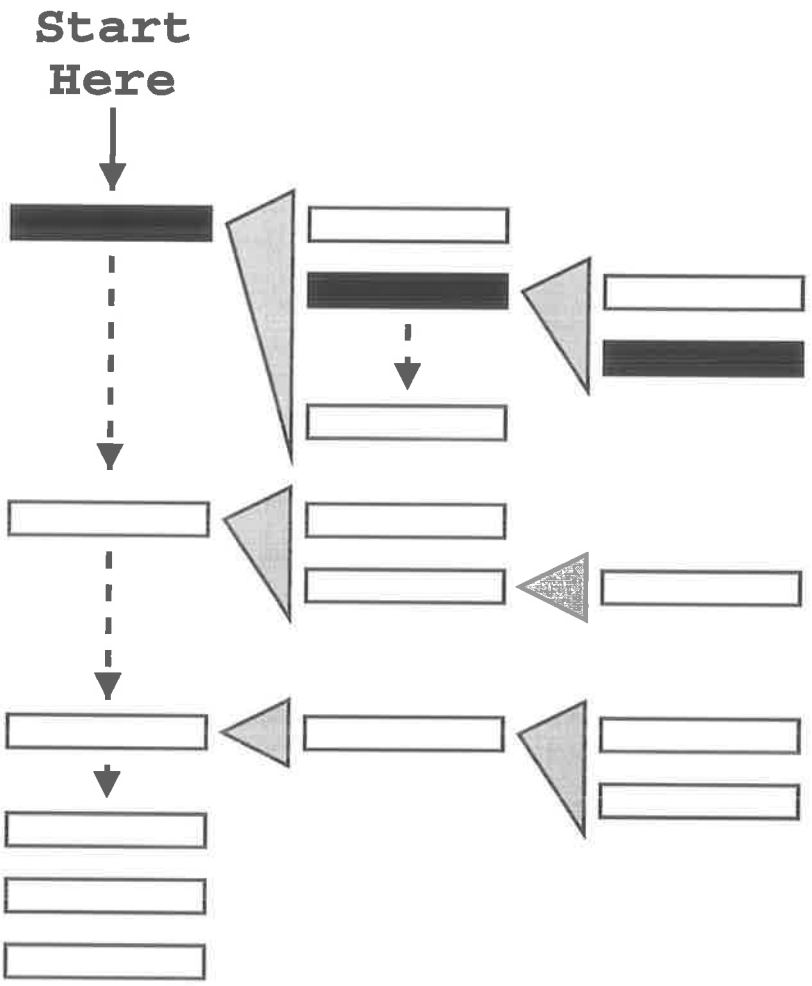


Figure 2.8. *A ripple down rule set. The boxes represent rules. If a rule is triggered (or antescendant is true) then the rules to its right are checked, if not then the rules beneath it are checked. The black boxes shows a hypothetical path through the ruleset. Rules to the right are called exceptions, and their validity is dependent on their context, ie. the rules to their left.*

are responsible for what consequences of the rule system.

2.3.3 The Pittsburgh Approach

The Pittsburgh approach to classifier systems represents entire classifier sets as individuals to be evolved by the evolutionary algorithm, as shown in Figure 2.9. This approach simplifies many of the evaluation and credit assignment issues associated with the Michigan model.

De Jong notes that when using a GA to adapt classifier sets there are two different possibilities for choosing the representation [58]. Either the GA can be modified to

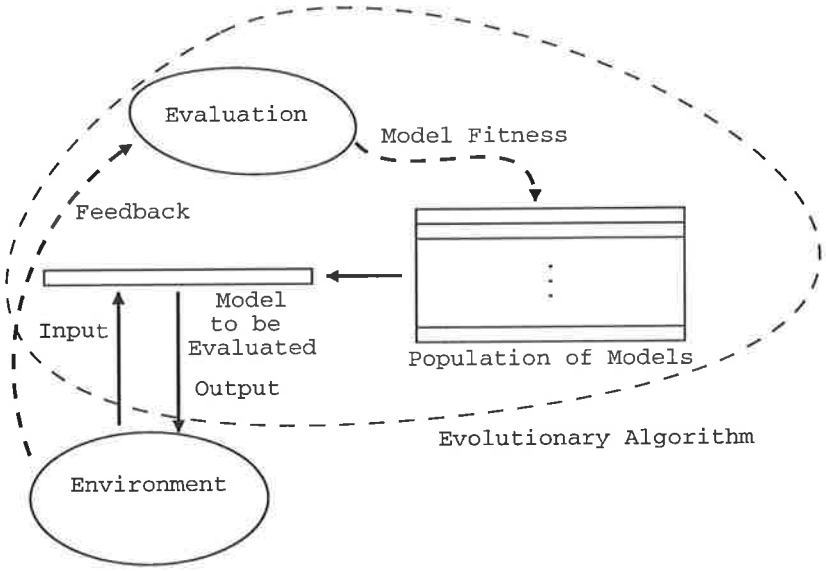


Figure 2.9. *The Pittsburgh approach to learning classifier systems. Each individual in the evolutionary algorithm population represents a complete rule set (or model) which can be tested. A model is selected and a number of input/output iterations are conducted with the environment. At the end of the testing the environment provides some feedback which is converted to the models fitness value. The fitness value is then used by the evolutionary algorithm to evolve better models.*

function with complex non-string objects, or a string can be created to represent the complex concept description language.

When using a string to represent the concept description language a method similar to the following is usually applied. A rule is a conjunction of elements and the elements are limited to conditions on each feature being in a particular value set. Assuming that the feature language consists of a set of features F_i, $i \in \{1, \ldots, N\}$ each taking v_i, $i \in \{1, \ldots, N\}$ different nominal values, where $n_j^{(i)}$ $j \in \{1, \ldots, v_i\}$ represents the j-th nominal value of attribute i, then we can assign a binary string to represent the disjunction of each of the nominal values of a particular feature, so

$$001101 \quad \text{represents} \quad n_3 \text{ or } n_4 \text{ or } n_6$$

A rule is then formed from the conjunction of the internal disjunction of each feature [58].

F_1	F_2	F_3	F_5
001101	011	101	0100

This forces the rule set to be represented as a fixed length binary string, and allows the symbolic rule information to be extracted from the *genotype*. However the method is most useful for inductive modelling of environments with nominal attribute representations.

The GABIL algorithm institutes this kind of search, and is reported to perform well when compared to a cluster of well known classification algorithms like C4.5 and AQ15 [129]. The method was improved by adding rule induction specific genetic operators from the classification algorithms. However, this would seem to somewhat negate the advantage of using a binary representation which was supposed to enable an unmodified GA to be used as the learning algorithm.

A different approach to using a binary string is taken by Grefenstette *et al.* in the development of SAMUEL [102, 105, 213, 104, 103]. SAMUEL uses a high level language to represent the condition and action part of the rules. An example of a rule instance for a continuous feature in SAMUEL is

$$(\text{SPEED } 100\ 250) \tag{2.13}$$

representing the situation $100 \leq \text{SPEED} \leq 250$. A nominal feature might be expressed like:

$$(\text{WEATHER IS [CLOUDY, WET]}) \tag{2.14}$$

whose attached action would be triggered when the weather was either cloudy or wet. The allowable values for continuous SAMUEL rule conditions are predetermined by dividing the known range of the variable by $N(\in \mathbb{N})$,where N is set by the experimenter and was limited to 255 in some of the early work on SAMUEL [102]. For example, the SPEED sensor might vary between 0 and 1000 and be discretised into $N = 20$ equal segments. The mutation operator modifies the values by uniform randomly choosing one of the N partition points as the new value.

Conditions such as (2.13) and (2.14) are combined in a conjunction and associated with a similar conjunction of actions. The conjunctions are allocated a strength, adjusted by a credit assignment algorithm and combined together to form a set of decision rules which Grefenstette refers to as a *tactical plan*. The tactical plan is an individual in the evolutionary algorithm. Crossover between tactical plans occurs at the level of individual rules, creating offspring with some rules from either parent. This is in contrast to typical GA crossover, which can break up individual rules at any level. Crossover was found to be useful in SAMUEL [105]. SAMUEL does not address issues relating to the number and size of partitions in the different continuous features, which has been shown to be important in discretised machine learning methods [153]. SAMUEL has been extended to operate in a co-evolutionary way where a number of independent populations of SAMUEL rule sets are evolved. The rules are evaluated in their local population by combining them with the best rules from the other population [180, 179]. The idea is to allow the algorithm to automatically find a suitable problem decomposition [52].

A similar approach to SAMUEL has been taken by De Falco *et al* [55]. Each attribute F_i was associated with two numbers k_1 and k_2, representing the low and high range

Internal vector:

3.25	1.05	22.1	23.5	44.4	55.3	1.22	2.34	33.2	100.2

Condition vector:

2	1	0	4	3	1

Figure 2.10. *An example of an internal and condition vector*

of the particular attribute. Four different antecedent were considered:

$$\begin{aligned} &\text{IF} \quad F_i \in [k_1, k_2] && (1)\\ &\text{IF} \quad F_i \leq k_1 && (2)\\ &\text{IF} \quad F_i \geq k_2 && (3)\\ &\text{IF} \quad F_i \leq k_1 \text{ OR } F_i \geq k_2 && (4) \end{aligned}$$

For each attribute i, a vector of parameters called the *internal* vector was constructed as a list of maximum and minimum values for each test on the attribute i. A condition vector is then formed with each component representing the choice of test to perform from the corresponding attribute using the conditions. The rule is then formed from the conjunction of the tests. The final value of the condition vector corresponds to the class to be assigned to the rule. The internal vector in Figure 2.10 would give the following rule:

IF $(F_1 \leq 3.25)$ AND $(F_2 \in [22.1, 23.5])$ AND $(F_4 \leq 1.22$ OR $F_4 \geq 2.34)$ AND $(F_5 \geq 100.2)$ THEN CLASS IS 1

The condition vector is evolved by an algorithm referred to as the breeder genetic algorithm (BGA) which is capable of directly dealing with real values [55].

Most evolutionary learning methods deal with propositional or zeroth order languages. There have been attempts to evolve relational, or first order descriptions [9, 108, 109, 110]. The DOGMA system uses predefined relations between attributes, for example, the relationship $on(X, Y, [yes, no])$ could be defined to be *yes* if the position of the attribute X is greater than the position of the attribute Y and *no* otherwise. Using the predefined definitions, DOGMA uses a language template Λ, to define how the relations can be conjunctively combined. Λ defines the hypothesis space that the GA operates on. Similarly to the rule systems described above, DOGMA uses bit strings to define the presence or absence of particular nominal values in the relationships defined by Λ [109].

In addition to the direct representation by the GA of rules, a number of hybrid approaches have been tried. GAs have been used to select features for use in standard machine learning induction techniques [221]. They have been hybridised with K-nearest neighbour techniques [132, 133]. GAs have been used to create rule sets in a divide and conquer method similar to traditional inductive learning approaches [154]. Cooperative coevolution has been applied to populations of evolving fuzzy set definition and rule sets [171, 179].

2.3.4 Other Discrete Representations

Rule sets are not the only discrete structures evolved by evolutionary algorithms. The evolution of discrete structures is undertaken in a variety of evolutionary learning applications.

Genetic Programming

Genetic programming addresses the problem of automatic programming, which is a form of evolutionary learning. *Automatic programming* aims to be a system that [16, Forward by John R. Koza]:

1 Produces an entity which runs on a computer.
2 Solves a broad variety of problems.
3 Requires a minimum of user-supplied problem-specific information.
4 In particular, doesn't require the size or shape of the final solution to be prespecified.
5 Implements, in some way, all the familiar and useful programming constructs (such as memory, iteration, parameterizable subroutines, hierarchically callable subroutines, data structures, and recursion).
6 Doesn't require the user to decompose the problem in advance, to identify subgoals, to handcraft operators, or to tailor the system anew for each problem.
7 Scales to ever-larger problems.
8 Is capable of producing results that are competitive with those produced by human programmers, mathematicians , and specialist designers or of producing results that publishable in their own right or commercially usable.
9 Is well-defined, is replicable, has no hidden steps, and requires no human intervention during the run.

It is the representation of solutions as computer programmes that is the most distinguishing feature of genetic programming.

Koza [139] proposed the evolution of LISP or S-Expression by genetic algorithms to address the problem of automatic programming.[11] Instead of representing a program as a binary string, a program is represented as an S-expression. An S-expression consists of a function followed by zero or more arguments:

Example 2.5: The following mathematical expressions could map to these S-expressions:

$$2 * 17 \mapsto (\text{MULTIPLY } 2\ 17)$$
$$1 - 4 * 3 \mapsto (\text{SUBTRACT } 1\ (\text{MULTIPLY } 4\ 3))$$

A function is referred to as a node. A function with no arguments, such as a numeric constant, is referred to as a terminal. The functions can also perform actions by

[11] As did several previous researcher. The key innovation of Koza was the recognition of the breadth of application of the method [16, page 101]

having terminal nodes such as (PUSH-CART-TO-RIGHT), or (PUSH-CART-WITH-FORCE *force-Newtons*). All functions return some value. When the functions are combined with looping, conditional, and memory array constructs, the S-expressions form a Turing complete programming language capable of representing any structured program [214]. A number of other Turing complete representations have been evolved by evolutionary methods, including classifier systems with internal memory and FSMs [16, pages 98–100].

The S-expressions are represented as tree structures which are interpreted *postorder*, creating a long linear S-expression (similar to the short expressions shown in Example 2.5). The tree structures are modified by subtree crossover, which swaps all nodes and terminals below a selected node (or terminal) with another subtree in another solution. The mutation in the tree structure is typically performed by selecting a random subtree and replacing it with a randomly generated subtree.

Applying crossover and mutation to programs written in an arbitrary programming language, such as C, is extremely unlikely to produce anything which would actually compile. The operators on the S-expressions are designed to greatly increase the chance of their application producing interesting computer programs. All functions in the S-expression are constrained to return the same type (usually a float) and to accept arguments of only this type. Koza refers to this property as closure [139], and it ensures that only valid programs can be generated.

The trees are usually induced from a finite set of primitive functions and terminal nodes [139, 4].

GP theory is based on the building block hypothesis, and a version of the schema theorem for GP [139, page 117–118]. However, the resultant building block hypothesis for GP lacks empirical support [168].

Genetic programming has been used in a wide range of machine learning applications [139], such as the evolution of circuits [19, 141, 143], the evolution of control laws [147], classifying protein sequences [142], ecosystem prediction [232, 231, 233, 189] and others [139, 140, 146].

Program trees have also been evolved by evolutionary programming methods [45, 46, 47], and S-expressions have been used in classifier systems [150].

Context Free Grammars

The rules of De Falco *et al.* [55] and DOGMA [109] demonstrate that the condition part of a rule can be made more sophisticated in order to give the rule set a larger range of behaviours. More generally, any allowable conditions of a rule could be considered as a *context free grammar* [16, page 271][60, page 193].

A context free grammar (CFG) is a language template which describes what sentences are legal in the language. The grammar describes how *terminal nodes* can be related by defining a set of *production rules* of allowable relationships.

Definition 2.3 (Terminal Node) *A* terminal node *of a context free grammar is a symbol for which there is no production rule.*

Definition 2.4 (Production Rule) *A* production rule *of a context free grammar is a rule of the form $X \rightarrow Y$ where X is a non-terminal and Y is a conjunction of terminals and non-terminals.*

A non-terminal is implicitly defined from the definitions as an element which has a production rule. Formally, a context free grammar can be considered as a four tuple (N, T, P, S), where N and T are disjoint sets of non-terminals and terminals respectively. P is a set of productions and S is a special non-terminal start symbol. The set of productions $\mathcal{X} \in P$ for the non-terminal $X \in N \cup S$ can be written as a mapping to the conjunction: $X \rightarrow a_1|a_2|a_3|\ldots|a_j$ for $a_i \in N \cup T$.

A simple if–then–else rule set can be constructed from the following grammar, eg [230]:

$$
\begin{array}{rcl}
N & = & \{R, \text{cond}, \text{result}, A, V\} \\
T & = & \{a_i,\ i \in \{1, \ldots, \text{Number of Attributes}\}, \mathcal{R}_j, <, >, \text{IF}, \text{OR}, \text{AND}\} \\
P & = & \left\{ \begin{array}{lcl}
S & \rightarrow & R \\
R & \rightarrow & \text{IF COND R RESULT} \mid \text{RESULT} \\
\text{COND} & \rightarrow & \text{AND COND COND} \mid \text{OR COND COND} \mid \\
 & & < A\ V \mid > A\ V \\
A & \rightarrow & a_i,\ i \in \{1, \ldots, \text{Number of Attributes}\} \\
V & \rightarrow & \mathcal{R}_j \\
\text{RESULT} & \rightarrow & \mathcal{R}_j
\end{array} \right\} \\
S &&
\end{array}
$$

The grammar describes how to derive a rule set which returns values $\mathcal{R}_j$ and makes comparisons between attributes and values $\mathcal{R}_j$, where j is the number of values, or parameters, associated with the rule set. In the example grammar the comparison and action values are of the same type, say reals, whereas a more complicated grammar could limit certain attribute comparisons and actions to different types of values. The rule set is constructed from the terminal nodes of the grammar.

The DOGMA system is constructed by adding certain types of relationships to the set of non-terminals and productions. The rules derived by De Falco *et al.* are constructed from a similar grammar to that offered. The consequent part of a rule is also represented in the grammar, and could, for example, be made a (grammatical) function of the attributes instead of real values $\mathcal{R}_j$[12]. Viewing the rule systems as a

[12]For example, by modifying a rule grammar to allow the consequence to be a linear combination of attributes, a regression tree could easily be represented.

context free grammar makes explicit their *declarative bias*[13], ie. the possible forms of solutions. Matching this bias to the problem is of central interest, since it allows the search space to be meaningfully reduced.

Grammars have been used for the creation of equations and executable structures in genetic programming [226, 227][16, page 270]. Grammatical genetic programming (CFG-GP) makes explicit the inductive bias of the representation, and has been used as a system for automatic bias generation in evolutionary learning systems [226, 227]. Rule sets are not usually created explicitly from grammars in applications, however it has been done and a GP algorithm used to induce the ruleset [230].

Self-Adaptive Finite State Machines

Evolutionary computation has been used to evolve other methods capable of learning from their environment. Finite state machines (introduced in Section 2.1.1 on page 9) are an example of a representation used with evolutionary computation with the goal of learning from the environment [83]. A FSM (Figure 2.1, page 10) is typically evolved through five different discrete mutation operators:

1 Add a state.
2 Delete a state.
3 Change the initial state.
4 Change an output symbol.
5 Change a next state transition.

The number of mutations per parent FSM is Poisson distributed with a rate of 3.0, and the mutation operator is uniformly chosen from the list of 5 operators. The maximum number of states was set to 25 and the minimum to 3. An extension to the methodology incorporated self-adaptation to attempt to improve the evolvability of the FSM [128]. It has been shown in a GP-like algorithm that freezing mutation on subtrees and treating them as modules can lead to a problem-specific co-evolved representation [6]. Fogel *et al* hypothesized that the extreme of mutation or no mutation of parts of discrete representations might be smoothed by using a self adaptive mutation rate [128]. Two different approaches were tried. The first is called selective self-adaptation where for each of the 5 mutation operators the component which was to be affected was chosen according to its self-adapted mutatability parameter instead of uniformly as it is in the traditional FSM. In this approach the mutability parameter decides the probability of the mutation operation affecting that part of the FSM by the relative value in the mutability parameter. In the second approach a mutability parameter was similarly defined, but the value was absolute, allowing

[13] A declarative bias is one in which the constraint imposed by the bias is transparent. It may be a language bias, a search bias, a selection bias or something else. A *language bias* is set before learning starts and determines what types of solutions can be represented. *search bias* is where the types of solutions most likely to be generated are affected, for example, the self-adaptive mutation vector is a search bias. *Selection bias* is the bias imposed be the fitness evaluation and selection scheme and determines what types of solutions are favoured, for example, including an information heuristic like the minimum description length (MDL) would affect the selection bias [229].

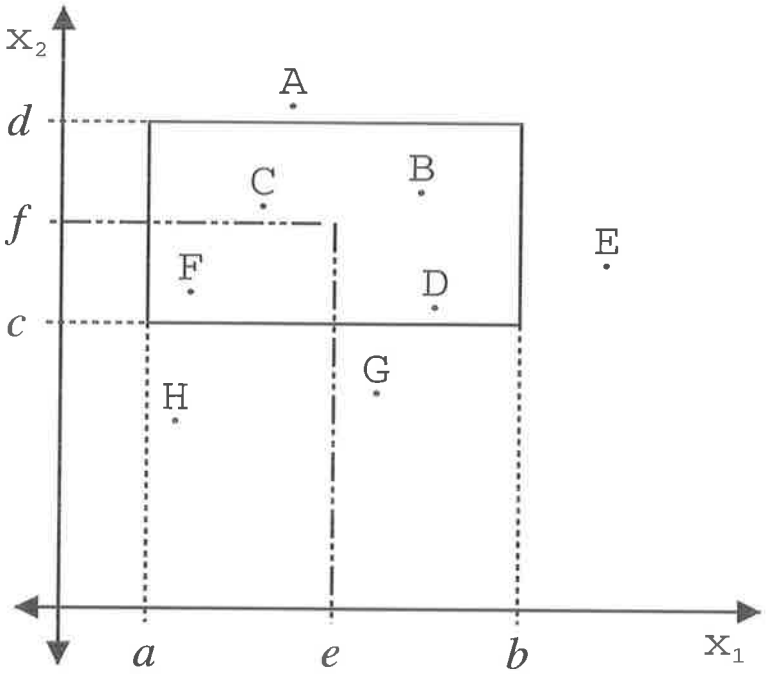


Figure 2.11. *A hyperbox cluster. The box is centered at the point (e, f), and contains the points F,D,B and C. The other points are not covered by the cluster.*

the probability of mutation of a component to be independent of all other components. The results from the self-adaptive approaches are statistically equivocal because of the experimental design, although there is definite empirical evidence of improvement using self-adaptive techniques on the simple test problems [128]. The problem identified by Fogel *et al* is that discrete representations are difficult to make self-adaptive.

Self-Adaptive evolution of Hyperbox Clusters

Hyperbox cluster for classifying spatial data was evolved using evolutionary programming to self-adaptively evolve hypercubes [92]. The experiments conducted considered only two dimensional data, where each hyperbox was represented as a 5 tuple

$$(x, y, \theta, w, h) \tag{2.15}$$

where (x, y) is the coordinates of the center of the box, θ is the anticlockwise rotation of the box, and w and h are the width and height respectively of the box. The situation for a box centered at (e, f) and with $\theta = 0$ is shown in Figure 2.11.The representation was variable length, with the number of hyperboxes modified according to a self adaptive addition or deletion probability σ_{add} and σ_{del}. The mutation vectors were updated by a formula similar to the standard ES formula (Equation 2.4, page 17). The value for n used in the usual τ and τ' update formula was modified to

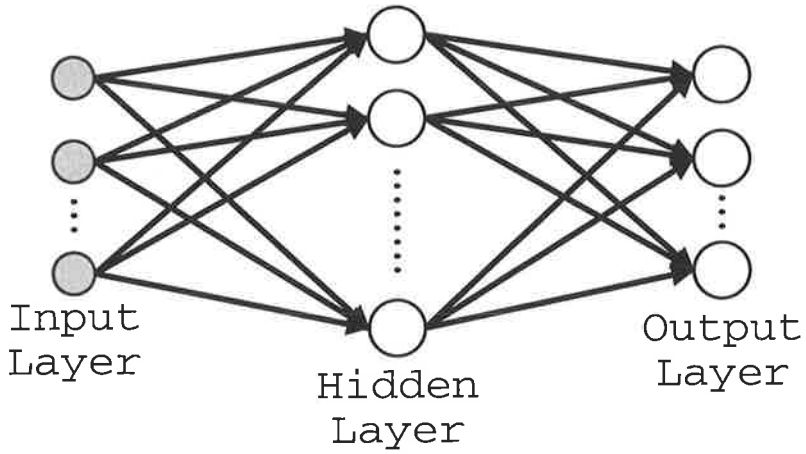


Figure 2.12. *A feed forward artificial neural network. Each arrow has a weight or strength associated with it, and each node has a function combining the nodes inputs to form the nodes output which is then modified by the weights as it is moved to the next layer*

account for the number of boxes and the number of parameters in each box, so that the 2-dimensional algorithm used $n = \text{NBox} \cdot 5$. Although the problem addressed was to look at clusters for spatial data, the similarities to rule-based representations in other learning problems are clear. The box in Figure 2.11 could be represented by the rule

$$(a < x_1 < b) \text{ AND } (c < x_2 < d)$$

which has the same form as the rules typically considered in CFS, for example, the conjunction of type (1) rules on page 35 used by De Falco *et al* [55]. Where the angle of the box is used the learning task becomes equivalent to a second order task where rules depend not only on feature values but on some relationship between features, even of a restricted form such as the representation used by Ghozeil *et al* [92].

Multiple Interacting Programs and Artificial Neural Networks

Angeline introduces a representation for expressing complex behaviours which he calls Multiple Interacting Programs (MIPS). MIPS are a combination of GP and artificial neural networks (ANNs)[4, 5]. Although it can be shown that ANNs are capable of representing any computable function given enough hidden units [124], Angeline notes that it is often the case that the number of nodes required for acceptable performance in feedforward neural networks trained with the back propagation (BP) algorithm can be prohibitively large to achieve acceptable generalization ability [5]. An ANN (Figure 2.12) can be described as a directed graph in which every node contains a transfer function of the form

$$y_i = f_i \left(\sum_{j=1}^{n} w_{ij} x_j - \theta_i \right) \tag{2.16}$$

where y_i is the output of node i, x_j is the jth input to the node and w_{ij} is the connection weight between the two nodes. θ_i is the threshold or bias of the node [249]. ANNs are usually trained using the BP gradient descent training method, or a similar training method. In the standard BP algorithm for an ANN with one output node, the difference between the actual output of the node, denoted A, and the correct output of the node (C) is calculated and multiplied by the derivative of the transfer function:

$$\delta = f'(I) \cdot (C - A) \tag{2.17}$$

where I is the input of the output node. The error, δ, is propagated backwards in a similar manner:

$$\delta_i = f'(I_i) \cdot W_i \cdot \delta \tag{2.18}$$

where δ_i is the error of node i, I_i is the input of node i and W_i is the weight between node i and the output node. The weights are updated according to:

$$w_{ij}(t+1) = w_{ij}(t) + \eta \cdot \delta_i \cdot x_j \tag{2.19}$$

and:

$$W_j(t+1) = W_j(t) + \eta \cdot \delta \cdot x_j \tag{2.20}$$

where the $w_{ij}(t)$ is the weight from node j in the input layer to node i at time (epoch) t, η is the learning rate, and x_j is the output of node j in the previous layer, as in Equation 2.16. In Equation 2.20, W_j is the weight from the output node to the jth hidden node. The extension to multiple output nodes and multiple hidden layers is trivial.

Angeline notes that the ubiquitous gradient descent methods used to optimise the weights of an ANN put some limitations on the ANN model [5]. In particular, the transfer function must be non-linear, non-decreasing and differentiable everywhere, as can be observed from equations (2.17), (2.18), (2.19) and (2.20). Instead of using Equation 2.16 at each node of the ANN, Angeline uses a GP evolved equation tree at each node. In this way, he conjectures that each node can evolve into an environmental niche that best suits the computational requirements of that node in delivering the desired output of the network. When the ANN topology underlying the MIPS net is not strictly feedforward, that is the output of a node can be fed back to itself and to other nodes, Figure 2.13, a recurrent MIPS net is created where the system of equations generated can refer back to themselves. In such a situation the initial values of the equations need to be specified. A recurrent ANN or MIPS net has a memory which makes it computationally similar to the CFS with an internal memory referred to in Section 2.3.1 on page 29.

Artificial neural networks are a discrete representation which have frequently been used to solve learning problems by evolution. There are many different evolutionary artificial neural network (EANN) algorithms, eg. Yao provides an extensive review [249].

An ANN is commonly applied to a n-tuple of data and required to predict a real value or class associated with that data. The learning task is to find the network

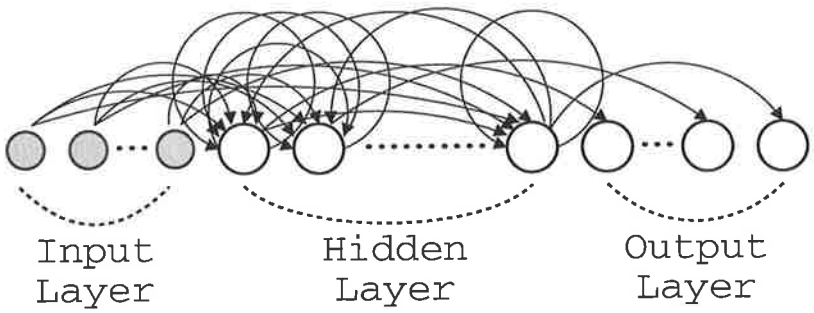


Figure 2.13. *A recurrent artificial neural network. Nodes can feed their output back to other nodes and to themselves*

which corresponds to the function

$$f(\boldsymbol{d}) = O \qquad \boldsymbol{d} \in \mathbb{R}^n,\ O \in \mathbb{R}$$

where $\boldsymbol{d}$ is the input pattern and O is the output. Nominal values are frequently mapped to $\mathbb{R}$. No assumptions about the function $f(\cdot)$ which is being approximated are made. If $f(\cdot)$ is continuous then it can be shown that there exists an ANN which can approximate it.

The principal problem for EANNs, in common with evolving all discrete representations, is maintaining the link between parent and offspring phenotype. In a classical BP-trained network the architecture of the network is constant and the weights are updated by small amounts at each epoch maintaining a functional similarity of offspring and parent. The BP and other gradient descent training methods are fast to find a local minimum in the weight space. It may be the case that different optimisation procedures may find better minimum in the weight space, that is, find a better set of weights that reduce the mean square error of the network when applied to the training data. A further complication for ANN training is that minimising the error on the training examples is not the goal of the method. An ANN is usually trained to *generalize* well. Generalisation requires that the network found will correctly predict unseen data patterns. However, finding the best approximation for the function $f(\cdot)$ in Equation 2.21 does not guarantee finding the best $f(\cdot)$ that generalises, and it follows that finding the best optimisation procedure on the NN weights will not guarantee the best results on independent test data.[14]

EAs have been used to optimise the networks weights. In a review on EANNs, Yao cites 88 papers where this has been done [249]. The algorithms employed usually concatenate the network weights as an evolvable string, a binary string or a real-valued string. An individual in the population is assigned a fitness corresponding to the network performance which occurs on the training set when the weights coded on the string are used in the network. The choice of operators to apply will depend

[14]To guarantee that the best trained network would result in the best generalization the topology of the network would need to be "correct".

on the method of representation. The evolutionary training of ANNs is in general slower than the BP algorithm and its variants, although counter examples to this exist [249]. The principal advantage lies in the freedom to choose non-differentiable transfer functions, problem specific error functions and the potential to find better optima in the training space through the avoidance of local minima which might otherwise capture gradient descent methods. Using an EA for weight modification in this way works well since it is easy to preserve the behavioural link between parent and offspring networks. The only requirement is that the genetic operators chosen are chosen to not be disruptive to the fitness linkage.

The largest problem in ANN modeling is how to decide on the topology of the network. EANNs are frequently applied to the problem of finding optimal topologies of ANN models. Yao cites 85 different papers where the topology of the network has been evolutionarily designed in his recent survey [249]. The central problem in evolving topologies for ANNs is that the representation is discrete. This makes it difficult to create an algorithm which can maintain some form of correlation between parent and child fitness. This problem is tackled in a number of ways [249].

One algorithm which addresses the issue of correlating parent-offspring behaviour is EPNet [255]. EPNet partially trains the network with a hybrid BP and simulated annealing (SA) algorithm. If the networks fitness does not improve it undergoes a structural mutation, starting with deletion of nodes/connections and then an addition. After a structural mutation the network is partially trained again to see if the fitness of the network can be improved compared to its parent. If not, the network undergoes a different structural mutation and further partial training. Once the network has a better fitness than its parent it replaces its parent in the population. The partial training is used to make the effect of the discrete mutations on the phenotype more continuous.

2.3.5 Discussion

A wide range of structures have been evolved to solve learning problems. The diversity of representations mirrors the diversity of learning problems, and nearly all representations which are appropriate for learning tasks have a discrete component.

The evolution of discrete structures is usually performed with discrete mutations applied with constant probabilities. The self-adaptive evolution of FSMs demonstrates that it is possible, although difficult, to adapt the rates of mutations in evolving discrete structures. There are two main advantages in being able to evolve these rates:

1 Self-adaptive rates are not tuned by the experimenter for different problems. Usually, there is nothing to set.
2 Self-adaptive mutation allows the parent to adapt the application rates of operators to produce children with correlated fitness. Maintaining the parent-child fitness correlation is important for evolutionary search to progress.

Many learning structures require the adaptation of both discrete and numerical values. In the case of rule sets, numerical values frequently correspond to partition values of continuous attributes, and the setting of these values is important in learning information in many domains with continuous attributes. However, most evolutionary rule learning methods concentrate on either the evolution of the discrete rule set, with pre-defined partition points, or like De Falco *et al.* use a very limited discrete rule structure and evolve the partition values. The optimisation of both types of structure is important.

EANN algorithms frequently perform some limited structure, or architecture, optimisation while optimising the parameters, or weights, associated with the structure. However, neural network structures are relatively simple compared to symbolic structures such as rule sets and programs (GP), and provide little insight about the information learnt. They are a blackbox model.[15]

2.4 Conclusion

There are two choices when evolving solutions to learning problems:

1 Modify the learning task to fit the representation used by the evolutionary process.
2 Modify the representation employed by the algorithm to fit the learning task.

Many applications demonstrate empirically that the second option is viable, allowing flexibility in the choice of structures. Choosing representations to apply evolutionary methods to, usually means choosing a discrete structure with some numerical components, especially in domains with continuous attributes.

Evolutionary methods are well established for the evolution of numerical components. Methods such as ES make few assumptions about the fitness landscape. Self-adaptive methods are able to adapt to changing landscapes, and are able to reduce the size of mutations as the numerical solution approaches the optima.

Discrete structures require specially defined operators for modification, for example, the evolution of FSM is performed with 5 discrete operators. Evolving the structures successfully means choosing rates of operator application which are appropriate in absolute and relative strength.

The ability to evolve the numeric components of the representation and simultaneously evolve the discrete structure of the representation would enable a greater level of flexibility in the choice of representation for learning problems. Representations

[15] That is to say, the neural network consists of an architecture and weights vector. The weights can be examined to produce rule sets, and sensitivity analysis can be done to elucidate some of the underlying behaviours of the induced model. However, they are not transparently comprehensible models.

for learning problems are important because performance in the learning environment is often not the only consideration. The knowledge learnt in attaining that performance is also important, and obtaining that knowledge is dependent upon the choice of representation used to solve the problem.

This thesis proposes a new self-adaptive method for the evolution of discrete structures and the simultaneous, symbiotic evolution, of the associated parameters.

Chapter 3

SASME: Self-Adaptive, Symbiotic Model Evolution

> I must Create a System, or be enslav'd by another Man's;
> I will not Reason and Compare: my business is to Create.
>
> *William Blake, Jerusalem*

This chapter proposes the method of self-adaptive, symbiotic model evolution, SASME. The SASME method segregates the task of parameter optimisation from the task of discrete model discovery. The parameters form one of the symbionts[1] and are optimised by a self-adaptive ES algorithm which is implemented in accordance with the literature analysis in section 2.2.3 on page 25. Concurrent with the parameter optimisation the other symbiont, the discrete structure, is optimised through the application of a set of discrete operators. The rate of application of discrete operators are a parameter of the system and are optimised by the parameter optimisation procedure. This makes them self-adaptive. Neither of the symbionts alone are a solution to the problem, and must be combined to form a solution.

The SASME framework is developed in general terms before being applied to the evolution of rule set structures. The rule set structure used explicitly represents exceptions in the tree structure. This allows a complete rule set model to be evolved in a Pittsburgh approach with an explicit representation of the default hierarchies. Default hierarchies allow for incremental behaviour changes to the ruleset as newly evolved exceptions to previously existing broader, more general rules, allow for more precise categorization of the learning task.

After the exposition of the SASME algorithm the method of self-adaptation is compared with optimal mutation settings in two constructed problems.

The SASME method differs from those discussed previously in two important ways:

[1] Symbiosis: "An intimate partnership between two organisms, in which the mutual advantages normally outweighs the disadvantages" [223]

Method The evolutionary learning algorithm developed here will combine the real-valued optimisation power of self-adaptive evolutionary algorithms with a novel self-adaptive strategy for the evolution of discrete structures.

Representation The method presented in this chapter will evolve entire solutions to problems as rule sets with exception lists allowing for the explicit representation of default hierarchies.

3.1 Algorithm Description

SASME consists of two parts. The *model structure* is represented directly with a discrete symbolic representation. The symbols referred to within the model are given quantitative definition through a set of associated *parameters*. The two are then symbiotically evolved. Table 3.3 on page 72 shows the list of symbols used for notation throughout this chapter.

3.1.1 Characterising the Learning Problem

The basic problem structure considered is one where the learner is able to sense a number N of attributes at each time step. The N-tuple vector of sensed attributes at time t is labelled

$$\boldsymbol{a}^{(t)} = (a_1^{(t)}, \ldots, a_N^{(t)})$$

where $t \in \mathbb{N}$. The superscript denotes an actual observation of the corresponding attribute, whereas the attribute itself can be referred to as a_i, $i \in \{1, \ldots, N\}$. For example, if the environment is piloting an aeroplane then the attributes sensed might be the air speed (1), height (2), pitch (3), yaw (4) and heading (5) of the aircraft respectively. The value of $a_1^{(t)}$ would be the numerical airspeed of the aircraft at time t in some units. When referring to the air speed variable, it would be labelled as attribute 1, or a_1.

All of the problems considered in later chapters have only real valued observations, $\boldsymbol{a}_i^{(t)} \in \mathbb{R}$, $\forall\ t$, $i \in \{1, \ldots, N\}$. However, in developing the algorithm it is allowed that the observations can come from unordered discrete sets as well, and handling such attributes is discussed.

If the set of all possible observations is labelled Ψ then $\boldsymbol{a}^{(t)} \in \Psi$, $\forall\ t$. Given a description of its environment $\boldsymbol{a}^{(t)}$, the model has to choose an action at time t, $\alpha^{(t)}$, to perform from some set of allowable actions[2] $\aleph$. A model π can be represented as a mapping from the set of possible observations to the set of possible actions

$$\pi : \Psi \mapsto \aleph \tag{3.1}$$

[2]The production of $\alpha^{(t)}$ will be from a finite number of functions, ω_i, which in general are themselves maps from environmental states to allowed actions. This is discussed later.

where neither Ψ nor $\aleph$ are necessarily enumerable. An application of the model at time t will give: $\pi(\boldsymbol{a}^{(t)}) = \alpha^{(t)}$.

The observed attributes at time t, $\boldsymbol{a}^{(t)}$, may not completely describe the state of the environment, $\boldsymbol{s}^{(t)}$. The observed values may be noisy, and may be incomplete. Where the set of state variables is labelled $\mathcal{S}$, the environment Υ will be the mapping,

$$\Upsilon : \mathcal{S} \times \aleph \mapsto \mathcal{S} \tag{3.2}$$

and the observation function

$$I : \mathcal{S} \mapsto \Psi \tag{3.3}$$

will map the actual state of the system to the observed state of the system, where the observed state of the system here refers to one of the possible observed states, accounting for the fact that the true state may not be entirely observable. A further complication arises when noise is added to the outcome of the above mapping, as will frequently be the case.

Note that the learning problems considered are not necessarily deterministic, that is the same state-action pair at time step t may not lead to the same state vector at $t + 1$. In general the problems can be considered to have the Markov property, that is the best action for any observed attribute vector will be independent of previous actions and attribute vectors. However, in the next chapter an interesting non-Markovian problem will be tackled.

The learning problem needs to also specify an initial state $\boldsymbol{s}^{(0)}$ which will imply a corresponding initial observation, $I(\boldsymbol{s}^{(0)}) = \boldsymbol{a}^{(0)}$.

A run of the model π will consist of some finite number $T > 0$ of interactions between the model and the environment. The set of inputs to the model and outputs from the model at time t form the run set of the model $\Re$, that is

$$\Re = (\boldsymbol{a}^{(t)}, \alpha^{(t)}) \qquad t \in \{1, \ldots, T\}$$

and the evaluation of the model is performed by some evaluation function

$$\Omega : \Psi \times \aleph \mapsto \mathbb{R} \tag{3.4}$$

in general. Without loss of generality $\Omega(\Re)$ can be considered a cost function, where the best model run will have the lowest $\Omega(\Re)$ value. The cost function allows the learning problem to be considered as an optimisation problem, where the problem is to find π such that $\min(\Omega(\Re))$ is achieved.

The problem faced by the evolutionary learner is to adapt the model π to the learning environment, where the cost function, $\Omega(\Re)$, evaluates the adaptations present in π.

The learning problems described cover a broad range of learning tasks such as sequential decision tasks [105, 164], reinforcement learning tasks [130, 165] and data mining tasks. The learning problem does not need to be supervised, and the correct action for each given state does not need to be known. All that is required is $\Omega(\cdot)$ so

that a model π can be evaluated after interacting in the environment. All problems are finite time problems where feedback is achieved after some set number, T, of iterations of π and the environment.

3.1.2 The Model Structure

Evolutionary algorithms have been used to evolve a variety of different discrete structures. Genetic programming uses genetic algorithms to evolve computer programs, most commonly as program trees [139, 140, 16], see Section 2.3.4 on page 36. Evolutionary programming has been used to evolve program trees similar to GP [4, 5], see Section 2.3.4 on page 41, and also to evolve finite state machines [83, 128, 79], see Section 2.1.1 on page 9 and Section 2.3.4 on page 39. Classifier systems use genetic algorithms to evolve a variety of discrete structures which represent rules, such as binary strings [115, 58], S-expression [150] and explicit rules [55], see Section 2.3.1 on pages 29–32. The DOGMA classifier system, [109] uses relational functions in rule premises, see Section 2.3.3 on page 35, and the SAMUEL system also represents rules explicitly in classifiers [102, 105, 213, 104, 103], see Section 2.3.3 on page 34.

The discrete structure and associated discrete operators form the model structure used in the SASME method. This thesis introduces a novel rule list with exceptions for that structure. However, the SASME algorithm could be employed on a variety of model structures, requiring only that operators like those introduced for the rule sets in this section are implemented for the structure. The optimisation of symbols within the model, and the rates of application of the operators are symbiotically evolved along with the chosen model structure.

The two requirements to evolve a structure are to have a covering set of mutation operations for that structure, and for the structure to be able to represent the solution to the problem. A covering set of mutation operators ensures that any solution will be reachable from a finite number of mutations. Although this is not strictly a requirement, it allows convergence in the limit to be guaranteed under certain circumstances (see Appendix A), and in the absence of domain knowledge it would seem an odd idea to allow the generation of a population which for all future generations could not reach some sections of the search space, when the solution to the problem is not known to be in the section of the solution space which is reachable. The second condition is obvious, for example, if the solution is not linear then satisfactory results may not be obtained if the discrete structure is limited to considering only linear equations.

A simple if then rule has the form

$$\text{IF } \textit{condition} \text{ THEN } \textit{action}$$

The *condition*, or antecedent, of the rule will be some form of comparison between attribute values or functions of attribute values and model parameters. For example,

$$\text{IF } \textit{air speed} > v_1 \text{ THEN perform action } a_1$$

where the value v_1 and the action a_1 are treated as symbols by the SASME algorithm. This allows linguistic constructions of rules to be made. The above rule could become

IF *air speed* is fast THEN reduce throttle

where the quantitative descriptions of *fast* and *reduce throttle* will be given numerical definitions by the parameter vector. This is similar to the way that fuzzy rule sets operate, where conditions and actions of rules are considered to be *fuzzy* quantities which are given quantitative values through the *defuzzyification* process using fuzzy set membership functions. Rule sets of this form are also similar to those used in SAMUEL and other forms of classifier systems.

Default hierarchies are a useful concept in rule sets which allow decision lists to be more succinctly represented, and provide a means for incrementally changing rule set behaviours. Incremental changes to phenotype behaviour is an important property of all evolvable systems. Default hierarchies are said to form in Michigan style CFSs due to the competitive bias in the rule bidding process in favour of more specific rules (see Appendix C). This creates competition to find specific corrections to existing behaviours in the evolving rule set. It is this competitive pressure which is responsible for the formation of the default hierarchies.

Default hierarchies are implicitly created in Michigan style classifier systems. In Pittsburgh style classifier systems default hierarchies are ignored and flat decision lists are evolved.

It was noted in Section 2.3.2, page 30 that the implicit default hierarchy formation in Michigan style classifier systems is similar in effect to the explicit representation of exceptions in ripple down rule sets. With this in mind, the rule set model used will be a Pittsburgh style classifier system, with rules represented as lists with exceptions, like those in Figure 2.7, page 31 and Figure 2.8, page 32. By using a Pittsburgh style CFS there is no need for complicated reinforcement distribution algorithms. Each individual in the population is a complete rule set and as such can have a fitness designation equal to the rule set's evaluation.

This gives a simple solution to one of the problems which has put serious limitations on the performance of CFS—the maintenance of the default hierarchy. As noted on page 166, the formation and maintenance of default hierarchies is frequently problematic [197, 246, 215] and this combined with the overall complexity of the Michigan style classifier system structure has limited its success in applications [240]. It has been noted that "classifier systems are a quagmire—a glorious, wondrous, and inviting quagmire, but a quagmire nonetheless" [95].

Rules with explicit exceptions represent similar information to what default hierarchies are thought to represent. There are limitations, however. Classifier systems use a variety of methods to weight rules matching a given environmental state in order to choose an action to be performed. This allows any rule which matches a state to compete in the bidding process. Exception lists are more hierarchical. Exception rules are fired only when the preceding rule is matched. When they match

a given state, but their parent rule in the tree does not, they are not able to bid to have their action performed. A rule list with exceptions mimics only one of the two classes of default hierarchies recognized by Holland [121]. They can represent default hierarchies distributed in space—rules with exceptions—but not default hierarchies distributed over time, or bridging hierarchies. The latter are difficult to construct in classifier systems (see Appendix C), and may be of most benefit in non-Markovian domains. Rules with exceptions have an advantage, however. They are more comprehensible, and better model the way experts structure knowledge [49, 194, 195].

Figure 3.1 shows an example of an incremental change made by adding exceptions to a rule set. The figure shows how more specific rules can correct inaccuracies in more general rules at different levels.

The simplest form of an evolved rule consists of a single comparison with the value of some attribute variable and the choice of an action.

$$\text{IF } \textbf{attribute} \in \textbf{values} \text{ THEN } \textbf{action} \tag{3.5}$$

The majority of individual rules within a ruleset will be of the type shown, although more complex rule structures will be considered in later chapters.

A rule set is a collection of rules. The rules will contain three parts available for evolutionary modification. These can be modified in the following way:

attribute The attribute which the rule tests can be modified to be a test on another attribute. This will also (normally) involve modifying the test values of the attribute, since different attributes will have different value ranges or different types for comparison.

values The value test can be modified to test another value set. This is a *symbolic* change. For example, if the test is represented as IF *temperature* IS **high**, it could be changed to IF *temperature* IS **medium**. Changing the numerical definitions of comparison symbols is explained in the next section. The change is essentially one of choosing from a list of possible value symbols a different value symbol. In the case of real attribute observations the value symbols will be real intervals, and changing symbols will correspond to the attribute value being in a different interval. If nominal values were to be evolved then the value set would consist of a collection of nominal values which are grouped together, and modifying the value set symbolically would mean substituting the set of nominal values for a different set of nominal values. Only problems with continuous values are considered in this thesis.

action The action comparison is modified in a similar manner to the value comparison. Instead of performing *action* 5 when the temperature is high, a mutation might lead to *action* 3 being chosen instead. The action symbol may represent a procedure which is performed to attain the output. For example, the action could specify the output to be some linear combination of a set of attribute

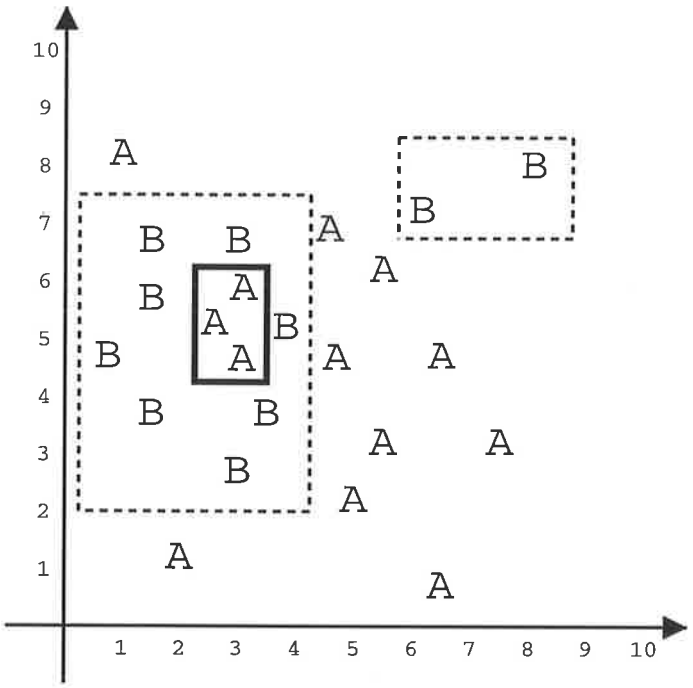


Figure 3.1. *Feature space description of ripple-down-rules. The graph shows the distribution of correct actions in a 2-feature space. The dashed lines show the areas of the feature space classified by the rule set. The bold rule corresponds to the bold region of the feature space, and makes an incremental improvement to the overall classification accuracy of the rule set by correcting the rule above it and labelling the area of A actions within the B action area correctly.*

values. When the rule is true, a real valued output would be produced from some linear combination of the current values of the attributes. Any parameters which need to be coevolved along with the symbolic actions are contained in the parameter vector, which is discussed later.

A rule can be made more general by allowing each attribute comparison to be a disjunction of conditions. For example, the following rule could be constructed: IF temperature IS (high OR very high).... Disjunctive attribute comparisons can be represented as a list of what value symbols are included in the condition. More than

one attribute can be conjunctively added to the rule antecedent in a similar manner. If we label the number of value symbols available for attribute i as $\mathcal{V}_i$ and the value symbols available for attribute i as $v_j^{(i)}$ where $j \in \{1, \ldots, \mathcal{V}_i\}$ and $i \in \{1, \ldots, N\}$, then an example of a rule would be:

$$\text{IF } (a_5 \in (v_1^{(5)} \text{ OR } v_4^{(5)})) \text{ AND } (a_9 \in (v_4^{(9)} \text{ OR } v_9^{(9)})) \text{ THEN action 3} \qquad (3.6)$$

which would be true when attribute 5 has the value $v_1^{(5)}$ or $v_4^{(5)}$ and attribute 9 has the value $v_4^{(9)}$ or $v_9^{(9)}$. When those conditions hold, the rule would output action 3 to be performed.

Rules of the form shown in (3.6) are quite general in structure. This presents an interesting problem. If the value symbols shown in the rule correspond to a discretisation of some real valued attribute[3] a_i, then the possibility of finding unlikely rule premises exists. For example, if the discretisation of the real attribute is into 5 divisions then the rule could be triggered when the attribute is in division 1 or division 4, perhaps corresponding linguistically to the attribute having a value of extremely small (1) or large (4), but not small (2), medium (3) or very large (5). It is possible that such a comparison is sensible, however more commonly it will not be. Even when it is sensible to compare attribute values in the union of real intervals which are not consecutive, it is likely that the rule set will be more comprehensible by having two distinct rules with the same consequence for the particular situation.

The algorithm could have imposed upon it some *a priori* bias in favour of testing particular groups of value symbols, for example, ensuring that continuous attributes are only compared on disjunctions of consecutive discretised symbols. Formally this could be handled by defining a grammar, as discussed in Section 2.3.4 on page 37, which would limit the antecedent conditions that the mutation operator could make and bias solutions towards more meaningful areas of the search space. For the problems considered in this thesis single attribute value comparisons only were required, and no internal disjunctions were used. For simplicity this will be assumed throughout the following chapters except where explicitly mentioned.

In a decision list single attribute value disjunctions are easily represented by two rules. If one of the mutation operations is to copy a rule with a "rule-condition mutation", then disjunctive rules will be effectively treated. Conjunctions between attribute comparisons can be created by allowing a null action. If a rule has a null action, then the rule's exception list is tested. If none of the exceptions are true, the rule is treated as false and parsing continues by checking its if-not list. The first true exception is treated as a true rule, having either it or one of its exceptions performed. In this way attribute conjunctions are effectively represented.

Simple rules, $\mathcal{R}_i$ are combined into a rule set, Ξ. The rule set, Ξ, is one part of the function, π, relating attribute observations to actions in Equation 3.1, page 48. An example of the evolved symbolic rule set, Ξ, and an example rule, $\mathcal{R}$, is shown in Figure 3.2. At this stage the representation is considered completely symbolically

[3]The argument here is concerned with any ordered attribute symbols

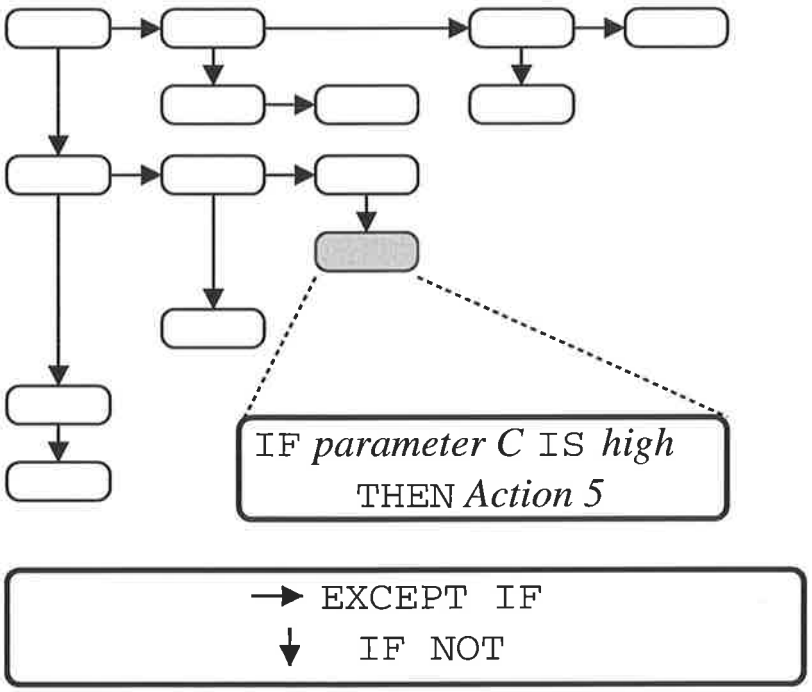


Figure 3.2. *The rule set representation. The rule set is represented as a decision list with exceptions as in Figure 2.8 on page 32. Each rule in the evolved rule set is a symbolic classification of an attribute value. The action of the rule is chosen from some list of possible actions. Rules to the right are tested only if the rule immediately to their left was tested and found true. Rules below are tested only if the rule immediately above them was tested and was false. Testing starts at the top left of the list.*

and is evolved according to an algorithm designed specifically for the evolution of discrete structures.

The simple rule structure shown in Figure 3.2 is of the form (3.5), shown on page 52. The observed value of attribute i is compared to a particular symbolic value set $v_j^{(i)}$, $j \in \{1, \ldots, \mathcal{V}_i\}$, where $\mathcal{V}_i$ is the number of value sets associated with attribute i, as defined earlier. The intersection of the value sets for attribute i,

$$\bigcap_{j=1}^{\mathcal{V}_i} v_i^j$$

is typically empty, although it need not be so. Where $a_i^{(\cdot)} \in \mathbb{R}$, the value sets are intervals of $\mathbb{R}$, eg.

$$(v_{\text{lower}}, v_{\text{upper}}].$$

Where the values of attribute $a_i^{(\cdot)} \in \Psi_i$ are nominal (that is, $|\Psi_i|$ is finite), the value sets will be collections of nominal values. $\mathcal{V}_i$ will be less than or equal to $|\Psi_i|$.

The set of allowable actions, $\aleph$, is not necessarily finite. The rule set Ξ will produce only a finite number of different action symbols, ω_i, $i \in \{1, \ldots, A\}$. The set of all

action symbols produced by Ξ is labelled $\mathcal{A}$. Each of the produced action symbols will be a mapping to an allowable action in the domain

$$\omega_i : \Psi \mapsto \aleph \tag{3.7}$$

such that $\omega_i(\boldsymbol{a}^{(t)}) = \alpha^{(t)}$. Most commonly the action function will be independent of the observed attribute, so that for a real-valued problem domain $\omega_i(\boldsymbol{a}^{(t)}) = k$, $k \in \mathbb{R}$, $\forall \boldsymbol{a}^{(t)} \in \Psi$. It can, however, be some function of the observed state such as a linear combination of observed values allowing the rule set to produce a discrete number of action symbols, yet still produce actions in the environment from a (real) value set, $\aleph$. In the case where the set of possible actions in the environment is itself finite then ω_i may be a labelling of those actions, so that $|\aleph| = A$

The set of mutation operators for modifying symbolic rulesets of the form shown in Figure 3.2 are as follows:

Add a rule A rule is added to the rule tree as follows.

1. A random rule, $\mathcal{R}$, is generated. The rule is of the form (3.5), shown on page 52. An **attribute**, a_i, $i \in \{1, \ldots, N\}$, is chosen uniformly randomly from the N attributes. Associated with the chosen attribute i are a number of discrete possible symbols or values, $\mathcal{V}_i$, from which a particular symbol is chosen $v_j^{(i)}$, $j \in \{1, \ldots, \mathcal{V}_i\}$, where j is chosen uniformly randomly. The rule action, ω_k, $k \in \{1, \ldots, A\}$, is similarly chosen uniformly randomly from the set of allowable actions, $\mathcal{A}$.
2. Each rule except the first in the rule set Ξ, Figure 3.2, is considered to have 2 possible insertion positions for the new rule $\mathcal{R}$. The first rule is considered to have 4 insertion points. The insertion points of each rule correspond to the exception or if-not rule associated with that rule. The initial rule is more complex since a new rule can be inserted before that rule, that is, the new rule becomes the initial rule and the previous initial rule becomes either the exception or if-not of the new rule. The part of the rule tree which is displaced by the addition of $\mathcal{R}$ is added with equal likelihood to either the exception list or if-not list of $\mathcal{R}$. Where there are n rules in the rule set there is $2 * n + 2$ random positions for the new rule, and the position is selected uniformly randomly. The possible positions for the new rule are shown in Figure 3.3.

Delete a rule A rule is deleted from the rule tree, Ξ, in a similar manner. Of the n rules in the rule set one is chosen uniformly randomly and deleted. Two possible actions for the deleted rule's exception and if-not lists are considered. Either the deleted rule can be replaced by the exception list and the if-not list can be appended to the exception rules if-not list, or the deleted rule can be replaced by the if-not rule and the exception list deleted. The first choice corresponds to the assumption that the deleted rule is always true, and so the exception list is always parsed. The second choice corresponds to the assumption that the deleted rule is always false, and so its exception list is never parsed. The situation is shown in Figure 3.4. The choice between the

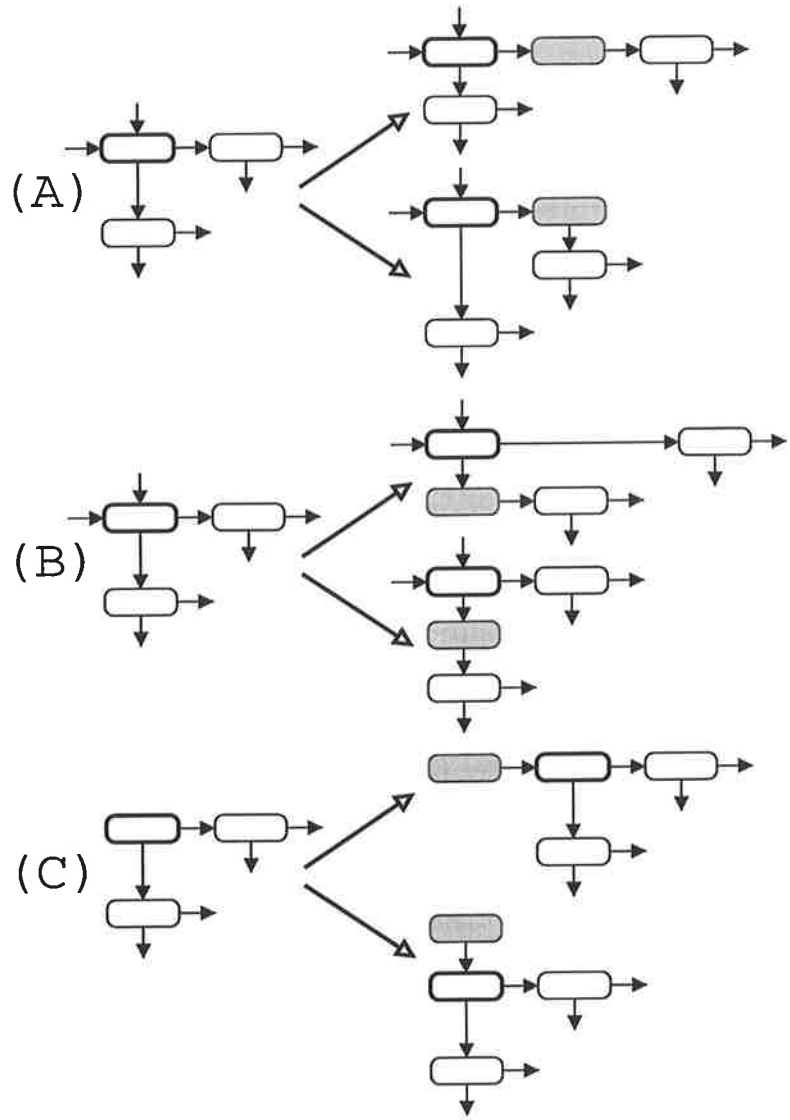


Figure 3.3. *The add rule mutation. The shaded rule is the newly generated random rule being added. The bold rule is the rule which has been randomly selected for the position of the new rule. (A) shows the two possible results from adding a rule as an exception to an earlier rule. (B) shows the effects of adding a rule as an if-not. (C) shows adding a new initial rule to the rule set.*

two outcomes is chosen randomly, and the deletion is not performed if it would remove all the rules in the rule-set.

Modify a rule A rule is chosen uniformly randomly and one of the following occurs:

1 Its action ω_k is changed to a new action $\omega_{k'}$ where $k' \in \{1, \ldots, A\}$ and $k \neq k'$.

2 The condition is modified by the same algorithm for creating a random rule, as in point (1) for adding a rule except that the original action coded

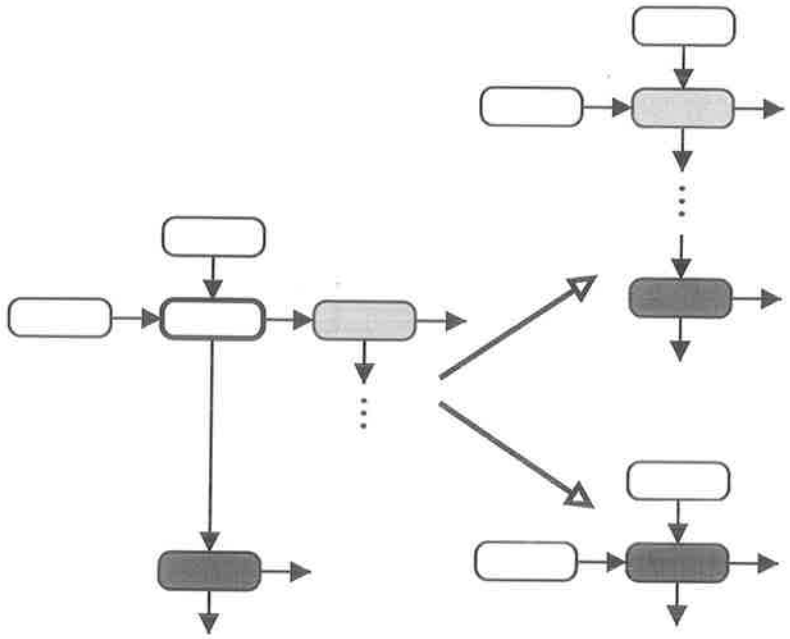


Figure 3.4. *The delete rule mutation. The bold rule is the rule to be deleted and the two shaded rules are the rules exception list (lightly shaded) and if-not list (darkly shaded). The unshaded rules are the possible positions of the parent rule to the rule to be delete, only one of which can actually exist. When the rule is deleted, there are 2 possibilities. (1) its exception list is moved to its position and its if-not list is added to the end of the exception lists if-not list. (2) the exception list is removed and the rules if-not list takes its place in the tree. (1) corresponds to an assumption that the rule is always true, and (2) corresponds to the assumption that the rule is never true.*

on the rule, ω_k, is kept and not modified.

Structural mutation A rule $\mathcal{R}_{\text{rand}}$ is randomly chosen and is either generalised or made more specific. Generalisation is performed on the rules exception by taking the rules exception, $\mathcal{R}_{\text{except}}$, and making the random rule, $\mathcal{R}_{\text{rand}}$, correspond to $\mathcal{R}_{\text{except}}$'s if-not. The exception rule $\mathcal{R}_{\text{except}}$ has it's if-not list moved to the random rules, $\mathcal{R}_{\text{rand}}$, exception list. The effect is to change the action which occurs when $\mathcal{R}_{\text{rand}}$ is false. If $\mathcal{R}_{\text{except}}$ is true then it or its exception list has its action performed regardless of the truth status or $\mathcal{R}_{\text{rand}}$. That is $\mathcal{R}_{\text{except}}$ is no longer considered under the context of $\mathcal{R}_{\text{rand}}$. The situation is shown in Figure 3.5. A rule is specified by the inverse operation. The randomly chosen rule $\mathcal{R}_{\text{rand}}$ is moved to the top of the exception list of its if-not list, $\mathcal{R}_{\text{if-not}}$. This situation is shown in Figure 3.5

Crossover Crossover works like a macro-mutation operator. A random ruleset Ξ_{rand} is selected from the population of rule sets and a single offspring is created from the crossover operation between the parent ruleset Ξ and the randomly copied rule set Ξ_{rand}. A rule set with exceptions has a special structure which the crossover operator takes advantage of. In particular, only rules on the

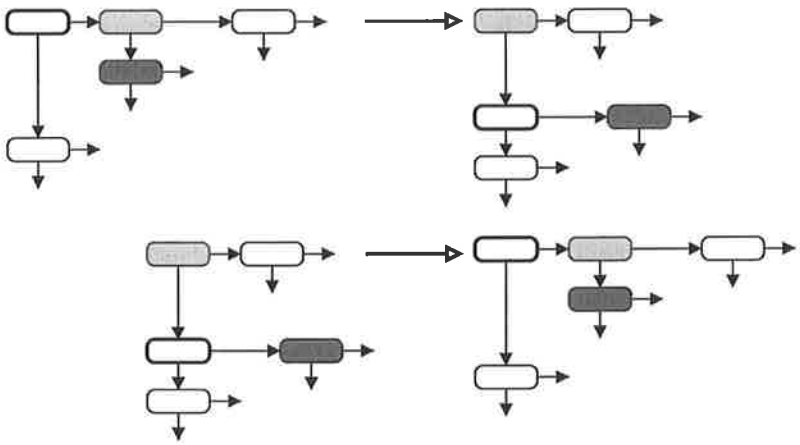


Figure 3.5. *A rule has its structure modified by one of two procedures. In the first the light grey rule is made more general by no longer being in the context of the bold rule. In the second the light grey rule is made less general by becoming an exception to the bold rule.*

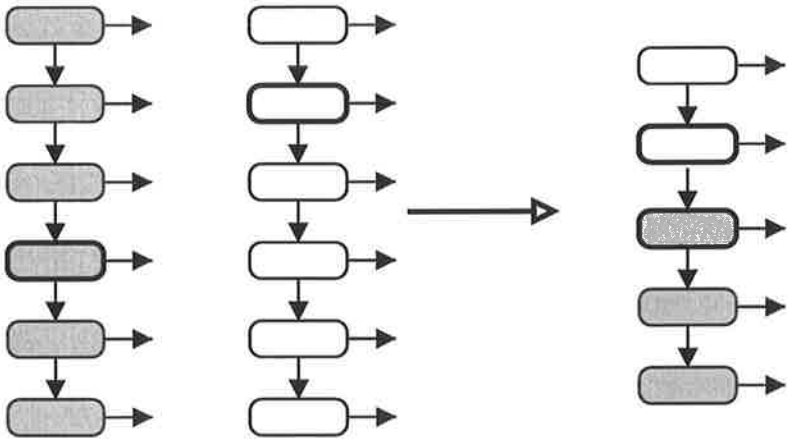


Figure 3.6. *The effects of the crossover mutation. A copy of a random rule in the population is made, the light grey rule set, and a crossover point selected at a random depth down the spine of that rule, the bold rule. A crossover point is chosen at a random depth down the spine of the parent rule, the bold rule in the white rule set. The two parts are then combined as shown to create the child rule resulting from the crossover mutation*

"spine" of the rule set, that is rules which can be obtained by following the if-not links from the root rule, are considered for crossover. The crossover procedure then consists of descending a random distance down the spine of the original tree and deleting the tree structure below the chosen position. The copied tree is then descended a random distance and the tree structure above the chosen point deleted. The two remaining structures are put together to form the child ruleset. This situation is shown in Figure 3.6.

A rule set is evolved by creating a random population of rule sets and modifying

those rule sets at each generation by applying with some probability the structural mutations listed. The rule sets are then evaluated in the problem domain, ranked, selected and the evolution continues. A natural question to ask at this point is: How to assign the probability of applying the discrete operators? Most frequently operators are applied with some low constant rate of mutation when evolving discrete structures. However, it seems likely that the probability of applying the different operators and also the relative probability of the operators will affect the evolvability of the representation. If the deletion operator is applied more frequently than the addition operator the mean number of rules in an offspring is going to be less than that in parents, giving a search bias which may or may not be warranted.

To solve this dilemma a self-adaptive approach is used. The probability of application of the different discrete operators is associated with each rule set. The evolution of these probabilities and the parameterisation of the symbols used by the rule set Ξ are discussed next.

3.1.3 The Parameters

The evolved model π consists of two parts: the symbolic, discrete rule set Ξ discussed in the previous section, and the parameter set.

A rule of the form (3.5), page 52, compares the value of an attribute to some set of possible values. If the attribute in question is i, then as before there are $\mathcal{V}_i$ different value sets, or symbols, labelled $v_j^{(i)}$, $j \in \{1, \ldots, \mathcal{V}_i\}$, where $v_j^{(\cdot)}$ is an interval $\forall\ j \in \{1, \ldots, \mathcal{V}_i\}$. Where the minimum value of attribute i is min_i and maximum value max_i, a set of partition points $p_j^{(i)}$, $j \in \{1, \ldots, (\mathcal{V}_i - 1)\}$ are created and the value sets are defined as:

$$\begin{array}{rcll} v_1^{(i)} & = & (-\infty, p_1^{(i)}] & \text{where } min_i < p_1^{(i)} < p_2^{(i)} \\ v_2^{(i)} & = & (p_1^{(i)}, p_2^{(i)}] & \text{where } p_1^{(i)} < p_2^{(i)} < p_3^{(i)} \\ & \vdots & & \\ v_j^{(i)} & = & (p_{(j-1)}^{(i)}, \infty) & \text{where } p_{(j-2)}^{(i)} < p_{(j-1)}^{(i)} < max_i \end{array} \tag{3.8}$$

The number of parameters associated with the value sets will be $\vartheta = \sum_{i=1}^{N} (\mathcal{V}_i - 1)$, or where each of the continuous attributes is divided into the same number of intervals $\mathcal{V}$, then there will be $\vartheta = N \cdot \mathcal{V}$ associated parameters.

The ordered tuple of partition points form a parameter vector

$$^{\text{attribute}}\boldsymbol{P} = (p_1^{(1)}, \ldots, p_{(\mathcal{V}_1-1)}^{(1)}, p_1^{(2)}, \ldots, \ldots, p_1^{(N)}, \ldots, p_{(\mathcal{V}_N-1)}^{(N)}) \tag{3.9}$$

which is associated with the rule set and gives meaning to the symbols manipulated by the rule set.

The action part of a rule corresponds to a particular symbol from the set of possible action symbols of the problem ω_k, $k \in \{1, \ldots, A\}$. As noted, the interpretation of

an action symbol depends on the problem at hand. The action symbol coded onto a rule can be used as an index for the corresponding action to be performed in the environment, $\omega_k(\cdot)$ Where there is a finite set of actions defined for the problem there will be no parameters associated with the action symbols.

> **Example 3.1:** An example might be a robot navigation task where the only actions are movement directions, N, E, S or W. In this case $\alpha_1 = \mathrm{N}$, $\alpha_2 = \mathrm{E}$, $\alpha_3 = \mathrm{S}$, $\alpha_4 = \mathrm{W}$ and $\aleph = \{\mathrm{N}, \mathrm{E}, \mathrm{S}, \mathrm{W}\}$. In such a situation there are no parameters associated with the models actions, and the action symbol $\omega_k(\cdot)$ maps all arguments to α_k, $k \in \{1, \ldots, 4\}$.

Alternatively the action symbol α_k may be interpreted as a model requiring κ_k parameters. If the actions in the environment are real then each action symbol might correspond to some particular value in the range of the possible action values. This value could be a constant value.

> **Example 3.2:** An example might be a greenhouse heater controller which can set a temperature between 0 and 100 degrees Celsius, and the problem is to maximise plant growth, or flowering, or water uptake under different conditions. In this case the set of allowable actions is a real interval $\aleph = (0, 100)$. . Even though the cardinality of $\aleph$ is not finite, there are only A possible actions symbols produced by the rule set Ξ. The action symbol ω_k will correspond to some temperature in the allowable range $\omega_k(\cdot) = \mathrm{temp}_k$ $(\in (0, 100))$. The actual temperature level, temp_k is a parameter of the learning system. The problem will then require A different parameters, and $\kappa_k = 1$, $k \in \{1, \ldots, A\}$

The action value could also be some function of the environment.

> **Example 3.3:** Example 3.2 can be extended to a case where $\kappa_k \neq 1$ for some k, by considering the temperature of the green house to be proportional to some attribute value a_i, say height of the plant. In this case the temperature would be set to $\mathrm{temp}_k = \beta_1 + \beta_2 \cdot a_i$ for appropriate values of β_1 and β_2. The action would then have 2 parameters associated with it, $\kappa_k = 2$. Temperature might only need to be set proportional to plant height depending on some other attribute, such as the season. In this example, not all of the action variables need to have the same number of parameters associated with them.

The total number of parameters associated with the actions will be $\kappa' = \sum_{k=1}^{A} \kappa_k$. The parameters associated with action ω_k are labelled

$$p_1^{(k)}, \ldots, p_{\kappa_k}^{(k)}$$

where each parameter $p_i^{(k)} \in \mathbb{R}$, for $k \in \{1, \ldots, A\}$ and $i \in \{1, \ldots, \kappa_k\}$. These parameters can be put into an ordered κ'-tuple vector as:

$$^{\text{action}}\boldsymbol{P} = (p_1^{(1)}, \ldots, p_{\kappa_1}^{(1)}, p_1^{(2)}, \ldots, \ldots, p_1^{(A)}, \ldots, p_{\kappa_A}^{(A)}) \tag{3.10}$$

In order to initialise the action vector $^{\text{action}}\boldsymbol{P}$ each action parameter must also have a maximum and minimum number defined for it.

The other set of parameters associated with the rule set are the probabilities of

performing discrete mutations. These are represented directly as mutation probabilities, $p_m \in (0, 1)$. The number of mutation variables required depends on the number of discrete mutation operators used in the symbolic evolution. Where there are ν mutation operators employed a parameter vector associated with the probability of applying mutation operators can be formed from the ordered ν-tuple,

$$^{\text{probability}}\boldsymbol{P} = (p_1, p_2, \ldots, p_\nu) \tag{3.11}$$

Combining the ordered parameter vectors in equations (3.9), (3.10) and (3.11) gives the complete parameter vector ${}^{x}\mathbf{P} \in \mathbb{R}^{(\vartheta+\kappa'+\nu)}$ associated with the discrete rule set. The exact size of the vector will depend on the particular problem and how it is represented, and how many discrete mutation operators are employed in the evolution. For example, there may be N real valued attributes, each divided into $\mathcal{V}$ discrete partitions ($\vartheta = N \cdot \mathcal{V}$), the action might also be a single real value and divided into $A(= \kappa')$ levels, or values, and the algorithm may use all 5 mutation operations mentioned in the previous section ($\nu = 5$). This would give a parameter vector composed of $\vartheta + \kappa' + \nu$ real components. Some of the components may have bounds associated with them. The probabilities are bounded below by 0.05 and above by 1.

The parameter vector is evolved by the evolution strategies method detailed in Section 2.2.1 on page 17. This requires a stepsize vector ${}^{\sigma}\mathbf{P}$, of length $\vartheta + \kappa' + \nu$ which is also associated with the parameter vector. There are two kinds of bounds that can be associated with the parameter vector, initial bounds and hard bounds. The initial bounds are used to create the initial values of the parameter vector. The initial values are uniformly distributed between the initial bounds, and the initial sigma value is set to 1/3 of the range of the initial bounds for each component, or to some specified initial value. The parameter vector values are maintained between the hard bounds. If a mutation would modify the value to be beyond a hard bound then the mutation is not performed and the original value is kept. The stepsize parameters are kept less than 1/3 of the hard bounds, or some smaller predefined number. Where a hard bound is not appropriate the values are set to $-\infty$ and ∞ and the corresponding parameter is unbounded. The probability values have a hard lower bound of 0.05 and a hard upper bound of 1. The lower bound on the probability of applying a discrete mutation ensures that all solutions in the structure space remain reachable to the algorithm. The initial value of the probability values is set to 0.5, and the initial step size is set to 1/3, its maximum value.

A rule set may or may not use a particular objective value in the rule set structure. For example, if the rule set only ever refers to speed when it is in its low range the value associated with defining speeds high range is not used in that rule set. In this case the objective value associated with defining the concept of high speed could converge to an arbitrary random number since there is likely to be an overall downward trend in the mutation step sizes of all variables owing to the once sampled normal distribution in the update formula, equation 2.4 on page 17. To avoid this effect the following condition is imposed on the parameter vector:

- An unused object variable value is reset uniformally randomly to an allowable value.

The condition ensures that the search space continues to be sampled to try and find a useful value when a value is not used. Unused values are not used in recombination to prevent recombination from adding too much noise to a value that is unused in most of the population but used in a particular rule set in the population.

In this way all components of the parameter vector are treated the same by the parameter evolution algorithm, which is essentially a typical ES instantiation. The object vector $\boldsymbol{x}$ is just ${}^{x}\mathbf{P}$, the $(\vartheta + \kappa' + \nu)$ real parameters associated with the rule set Ξ. The stepsize vector $\boldsymbol{\sigma}$ is ${}^{\sigma}\mathbf{P}$, the $(\vartheta + \kappa' + \nu)$ real parameters created as detailed above. The ES algorithm is searching for optimal values of the parameter vector ${}^{x}\mathbf{P}$ associated with the rule set Ξ.

The rule model π consists of the symbolic rule structure Ξ and the parameter vector[4] ${}^{x}\mathbf{P}$.

3.1.4 Evolving the Ruleset

The rule set, Ξ, and the parameter vector, ${}^{x}\mathbf{P}$, together determine the mapping, (3.1), that the evolved model makes between vectors supplied by the environment, (3.2), and actions. Part of the problem specification will be some function Ω which evaluates the output produced by the model Ξ, Equation 3.4.

An initial root rule is generated. A recursive procedure generates a random rule set with a maximum of *count* rules. The chain of if-not rules descended from the root rule are generated so that they cover the possible observable state space. Figure 3.7 shows the covering rules of the initial rule set shaded light grey. The maximum number of rules generated in the exception list is set to 5, and the probability of generating an exception and an if-not is set to 0.75. The performance of the algorithm is not particularly sensitive to how it the population is initialised. In this way the structure of the rule set, Ξ, can be initialised.

The problems considered in this thesis are all real valued attribute problem. The implementation described here requires an *a priori* decision about the number of discrete segments $\mathcal{V}_i$, $i \in \{1, \ldots, N\}$, to divide each of the N continuous attributes. A minimum, min_i, and maximum, max_i, value must also be specified for each attribute, $i \in \{1, \ldots, N\}$.

The (μ, λ) SASME algorithm for symbiotically evolving the parameter vector, ${}^{x}\mathbf{P}$, and the rule set, Ξ, can now be stated:

1 An initial population $\mathcal{P}_0 = g$, $g = 0$, of λ models π, is constructed. Each

[4]The probabilities of discrete mutation incorporated in ${}^{x}\mathbf{P}$ are not involved in the formation of the model π, only the $(\vartheta + \kappa')$ parameters which actually give meaning to the symbols referred to in Ξ.

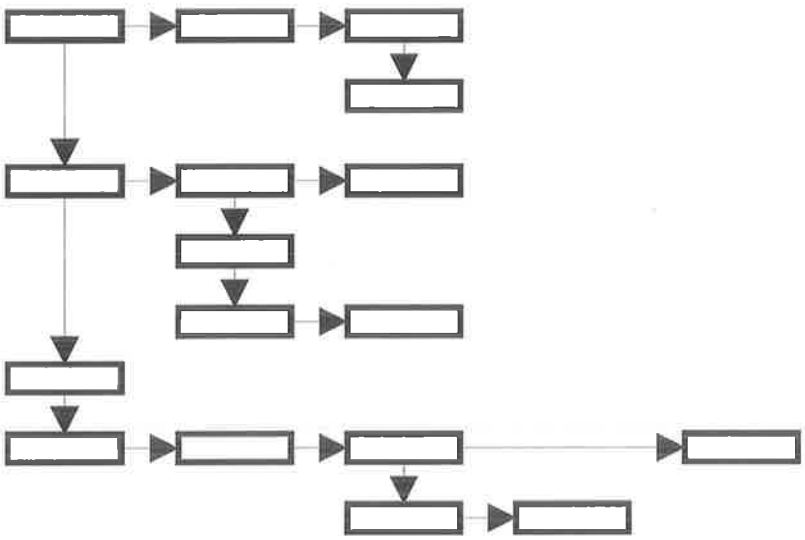


Figure 3.7. *The initial rule set. The shaded rules are generated to cover all possible input vectors, and their exception lists are set to some random rule set containing a specified maximum number of rules*

model π consists of three components, $(\Xi, {}^{x}\mathbf{P}, {}^{\sigma}\mathbf{P})$, a rule set Ξ, an objective parameter vector ${}^{x}\mathbf{P}$, and a mutation step size vector ${}^{\sigma}\mathbf{P}$. Each of these are initialised as follows:

Ruleset For each $j \in \{1, \ldots, \lambda\}$ a ruleset Ξ_j is created. The structure of the ruleset is created as described above. Each rule in the rule set is created as outlined in item 1 on page 56.

Parameter vector For each $j \in \{1, \ldots, \lambda\}$ a parameter vector ${}^{x}\mathbf{P}$ is created. Each component is a real value, and the length of the vector is $\vartheta+\kappa'+\nu$ as discussed in section 3.1.3, page 60. Each objective value has a maximum and minimum number associated with it, and it has its values initialised uniformly randomly between those values

Mutation vector For each $j \in \{1, \ldots, \lambda\}$ a parameter vector ${}^{\sigma}\mathbf{P}$ is created. The stepsize vector has the same length as the parameter vector, and is initialised to 1/3 of the value of $(max_i - min_i)$, $i \in \{1, \ldots, (\vartheta + \kappa' + \nu)\}$.

2 Each model π in the population $\mathcal{P}_g$ is evaluated on the problem by the fitness criteria. Rules which never have their premises satisfied are removed from the rule tree. Models are ranked according to their fitness evaluation and the best μ out of the λ models are chosen as the parents of the next generation.

3 Each of the μ parent model generates λ/μ models on average so that a total of λ offspring are created. Using the notation on page 56, and with τ and τ' set identically, the offspring are created as follows: for $i = 1, \ldots, \mu$, and $h = 1, \ldots, \lambda$,

3.1 Objective values which are not used in the evaluation of the rule set have their values randomly assigned as described on page 62.

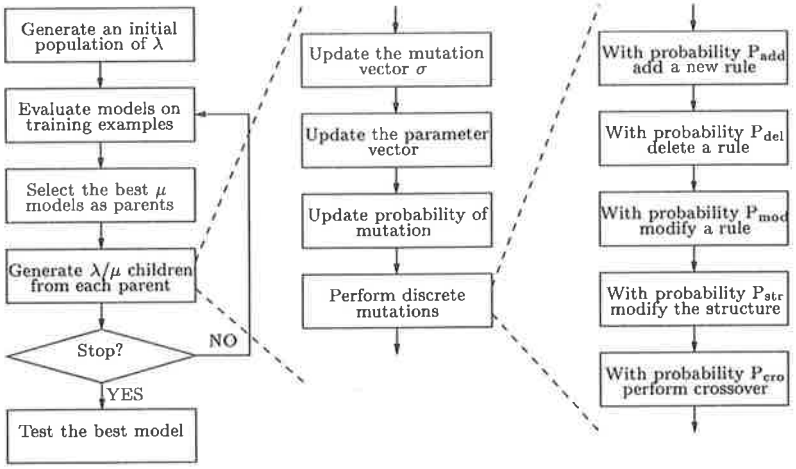


Figure 3.8. *The main components of the evolutionary process for model creation.*

3.2 The mutation and object vectors undergo recombination. The mutation vector is recombined by intermediate recombination and the object vector is recombined by discrete recombination.[5]

3.3 The mutation vector, ${}^{\sigma}\mathbf{P}$, is updated according to

$$ {}^{\sigma}\hat{\mathbf{P}}_j^{(h)} = {}^{\sigma}\mathbf{P}_j^{(i)} \exp(\tau' N(0,1) + \tau N_j(0,1))) \tag{3.12} $$

for $j = 1, \ldots, (\vartheta + \kappa' + \nu)$.

3.4 The objective vector is updated according to

$$ {}^{x}\hat{\mathbf{P}}_j^{(h)} = {}^{x}\mathbf{P}_j^{(i)} + N_j(0, {}^{\sigma}\hat{\mathbf{P}}_j^{(h)}) \tag{3.13} $$

for $j = 1, \ldots, (\vartheta + \kappa' + \nu)$.

3.5 The discrete model structure, $\Xi^{(i)}$, is mutated by applying the ν discrete mutation operators with probability ${}^{x}\mathbf{P}_j$, $j = (\vartheta+\kappa'+1), \ldots, (\vartheta+\kappa'+\nu)$. Some possible operators are discussed in section 3.1.2, page 56. This is repeated to obtain λ structures, $\hat{\Xi}^{(h)}$.

4 If the stopping criteria has been met, then terminate. Otherwise, set $g \leftarrow g + 1$, set the children $\hat{\pi}^{(i)} = (\hat{\Xi}^{(i)}, {}^{x}\hat{\mathbf{P}}^{(i)}, {}^{\sigma}\hat{\mathbf{P}}^{(i)})$, $\forall i \in \{1, \ldots, \lambda\}$, as the next generation and repeat from step 2.

Equations 3.12 and 3.13 are the same as the standard evolutionary strategies and evolutionary programming equations 2.4 and 2.5 on page 17. The algorithm is described graphically in Figure 3.8.

3.2 Evolving Rule Sets: An Example

To clarify operational aspects of the proposed method an example problem will now be illustrated. The data which the model is learning from is generated from the

[5] described in equation 2.6, page 20

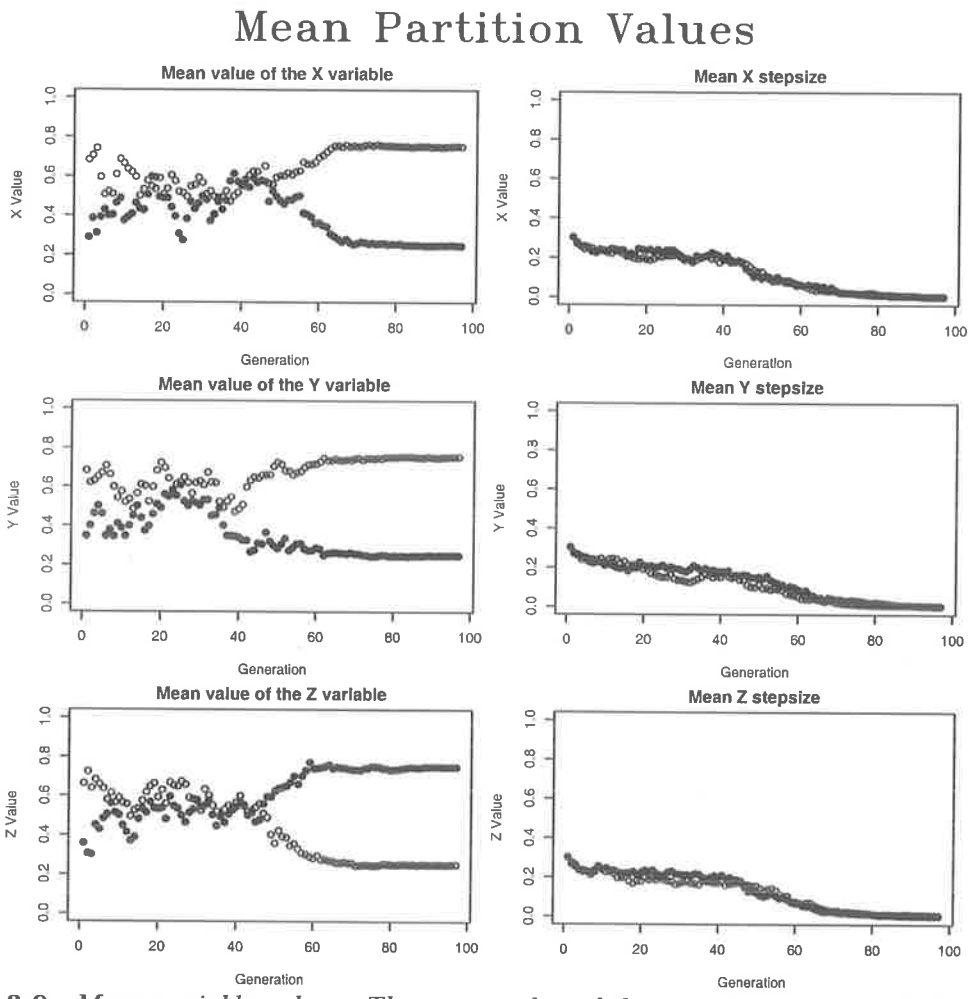


Figure 3.9. *Mean variable values. The mean value of the partition points and step sizes for each variable at each generation is calculated and plotted in the above graphs. The step sizes monotonically decrease in value as the variables approach the partition values of 0.25 and 0.75.*

following condition:

$$\text{IF } x \in (0.25, 0.75) \text{ AND } y \in (0.25, 0.75) \text{ AND } z \in (0.25, 0.75) \text{ THEN } c = 1 \\ \text{ELSE } c = 0 \tag{3.14}$$

A set of 1000 examples were generated by uniformly randomly generating a triplet (x, y, z), $x, y, z \in (0, 1)$, and applying the condition above to obtain learning examples (x, y, z, c). The algorithm was applied with each of the three variables divided into three ranges, $\{(-\infty, v_1], (v_1, v_2], (v_2, \infty)\}$, where the v_1 and v_2 were evolved for each variable and are referred to as the partition points for the variable. The error criteria used was the mean of the square of the difference between the model output and the correct output.

Figure 3.9 shows the evolution of the partition points for a particular run with a population of 100. Each point in the graphs is the mean value of the partition point in the population at that generation. Initially the partition points in the population

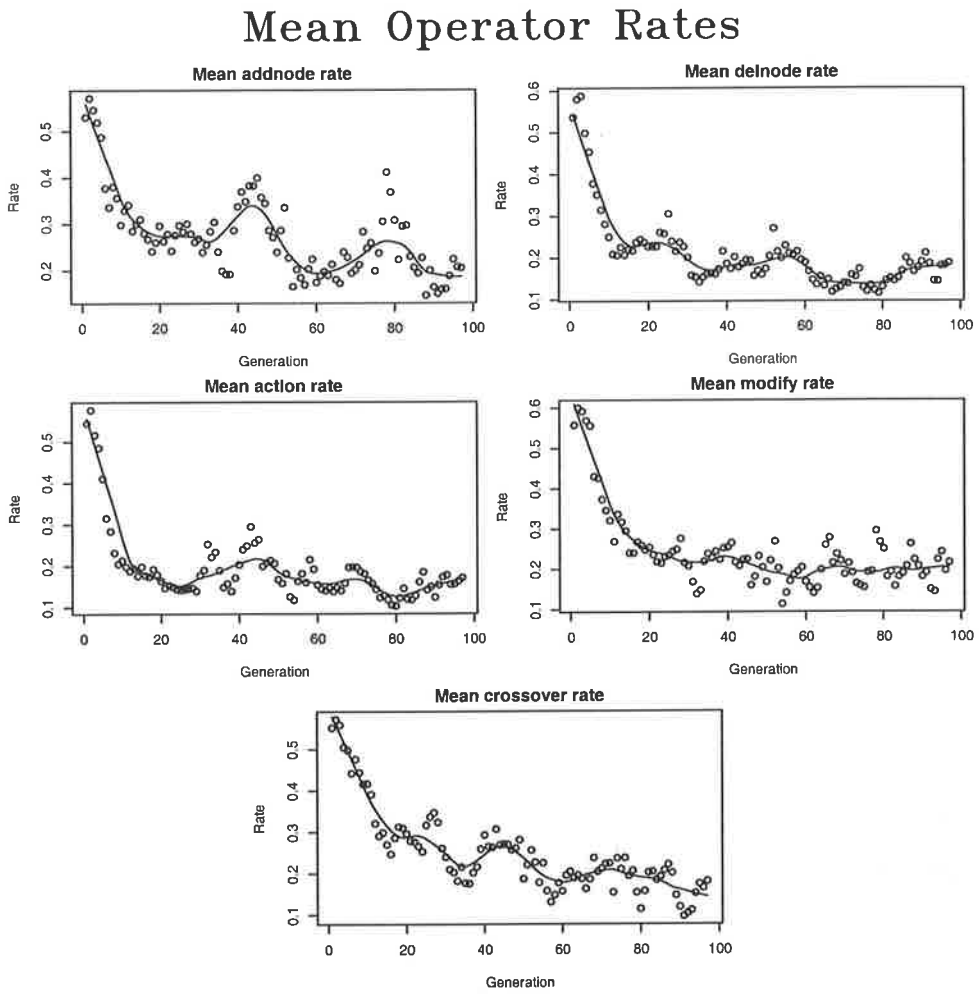


Figure 3.10. *Mean operator values. The self adaptive rate of applying a discrete mutation operator is averaged for each generation and graphed above. The graphs are a consequence of a decline in the usefulness of discrete operators as they become disruptive to the fit population of rule sets. The trend line is for illustration only.*

are random and the mean hovers around the middle of the range. As the rule set learns useful rules with certain partition values in the population these partition values are optimised to the decision planes bounding the region described in 3.14. The partition points converge to the correct value reducing the utility of exploring partition points at a large distance. This reduction causes selective pressure within the population for a decrease in the value of the deviations, corresponding to a decrease of the step size, ${}^{\sigma}\mathbf{P}$, in equation 3.13 (also, σ in equation 2.4 page 17).

The trajectories of the step sizes and the discrete operator probabilities are an emergent property of the algorithm; the method is not programmed to self-adapt by reducing the mutation step sizes, but rather the non-elitist (μ, λ)-selection method and unbiased step size updates (Equation 3.12) produce the behaviour. This behaviour is widely seen in optimisation problems, and here exhibited in a learning problem, showing that the self-adaptive strategy is operating correctly in a different

domain.[6]

The averaged probabilities of applying discrete mutations at each generation are shown in Figure 3.10. The operator probabilities decrease as the population evolves better rule structures and the application of mutations becomes disruptive. The relative and absolute rates of the different operators differ as the run proceeds, which supports the widely held belief in evolutionary computation that the utility of mutation and crossover operators change over the course of a run as discussed on page 14. The fitted trend curves in Figure 3.10 are only for illustration.

3.2.1 Evaluating the Self Adaptive Mutation Rates

To establish the performance of the self-adaptive mutation probabilities two sets of 100 runs were conducted for the self-adaptive algorithm and for 10 constant mutation values varying from 0.05 to 0.5. The first set of runs were performed with a population of 100 and the second set with a population of 200. The runs used the condition 3.14 and were halted at the generation where the solution was found or else stopped after 1000 generations had elapsed. Stopped runs were recorded as a failure and removed from the analysis, which biases the statistics in favour of the trials which had failures. Summary information about the number of generations to solve the problem were collected for all successful runs and are shown in table 3.1 for a population size of 100 and 3.2 for a population size of 200. The associated plots are shown in figure 3.11 and figure 3.12 respectively.

It is clear from the tables that no constant mutation rate is significantly better than the self adaptive rate for this particular problem. The optimal rate of mutation for a particular application of an evolutionary algorithm is most frequently not knowable. The pragmatic approach usually taken is to experiment with a number of different rates and choose the most promising. The results from the parameter tuning exercises are not normally published despite being part of the experimental set up.

Table 3.1 shows that a mutation rate of 0.3 is around the optimal constant values. The self-adaptive algorithm does at least as well and is able to find the solution in all runs. Self adaptation of the mutation rate is likely to have two principal advantages over a constant mutation rate:

1 The self adaptive mutation scheme can modify the rate of application of the discrete mutation operators independently of each other. This would allow, for example, the self adaptive scheme to apply the addition operator more frequently than the deletion operator. This could then create a bias towards larger rule sets at different stages of the evolutionary process.
2 The rate of mutation does not need to be constant throughout the run. For

[6]The domain is different because at the same time as the parameters are being optimised for the rule set, the rule set itself is being optimised to use and exploit the parameter settings to solve the problem.

Table 3.1. *Results with a population of 100. For each mutation rate 100 runs were performed with a population of 100 and the generation number at which the correct solution was found is used in the table below. Runs which did not find the solution in less than 1000 generations are recorded as failures. Figure 3.11 shows a box plot of these results.*

	Operator Probabilities				
Mutation Rate	Mean	Quartile 1	Median	Quartile 3	Failures
Self Adaptive	260.31	139	217	346	0
Rate=0.05	407.548	217	349	581	27
Rate=0.10	360.758	167	304	550	9
Rate=0.15	352.055	163	275	548	9
Rate=0.20	318.989	149	253	470	5
Rate=0.25	305.906	172	249	400	4
Rate=0.30	275.768	152	217	351	5
Rate=0.35	326.271	155	272	439	4
Rate=0.40	326.844	157	269	436	10
Rate=0.45	328.688	177	259	409	4
Rate=0.50	373.207	193	311	493	8

Table 3.2. *Results with a population of 200. For each mutation rate 100 runs were performed with a population of 200 and the generation number at which the correct solution was found is used in the table below. Runs which did not find the solution in less than 1000 generations are recorded as failures. Figure 3.12 shows a box plot of these results.*

	Operator Probabilities				
Mutation Rate	Mean	Quartile 1	Median	Quartile 3	Failures
Self Adaptive	183.38	99	147	235	0
Rate=0.05	295.67	128	243	417	12
Rate=0.10	244.042	109	192	341	5
Rate=0.15	190.04	89	148	256	1
Rate=0.20	198.374	104	157	241	1
Rate=0.25	182.333	102	146	229	1
Rate=0.30	210.388	120	174	257	2
Rate=0.35	206.394	119	175	249	1
Rate=0.40	217.606	131	164	263	1
Rate=0.45	221.541	131	176	288	2
Rate=0.50	250.061	149	201	328	1

example, as the rule set becomes more complete and accurate the number of children generated with new rules attached can be decreased, which may improve the number of offspring which survive from a given parent.

Figure 3.11 indicates that the optimal single mutation rate has an isolated performance minima at a rate of around 0.3. Values above or below this level decrease performance, and those further away decrease performance most. The relationship between performance and mutation rate could be much more complicated than this.

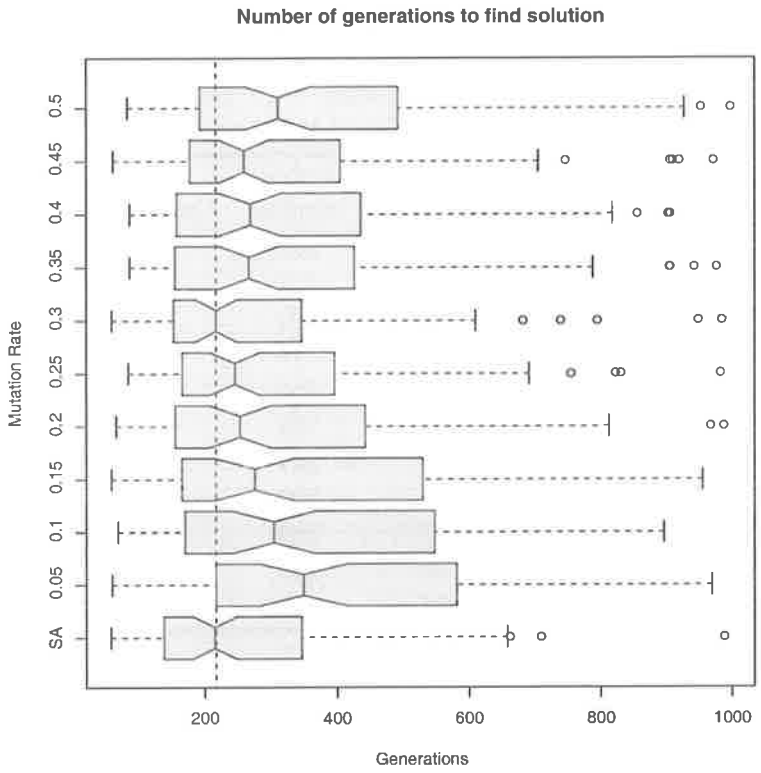


Figure 3.11. *Results for a population of 100. Plots of the relative success at different mutation rate. The graphs show the Self-Adaptive (SA) mutation rate algorithm in comparison to a number of fixed mutation rate values. The dashed line corresponds to the median of the SA algorithm.*

It is unprovable and in fact unlikely that the self adaptive scheme is the optimal mutation strategy. The self adaptive strategies provides a generationally dependent mutation rate, and so has the potential to perform better than a constant rate due to the increased degrees of freedom available. Whether it does so or not is not as important as the fact that it appears to perform at least as well as any constant strategy. This reduces the need for ad-hoc tuning of the mutation parameter.

Table 3.2 shows that the optimal constant mutation rate for a population of 200 is similar, although possibly lower. In general it will be the case that the optimal mutation rate will depend on the population size used. The self adaptive rate is again as good as any of the constant mutation rates, and is again the only strategy which was always able to find the solution in under 1000 generations.

A learning algorithm such as the one described in this chapter can be applied to a wide variety of different problems. It is unlikely that a single constant mutation rate will perform sufficiently for all problems the algorithm is applied to. The use of a self adaptive rate is motivated not so much by the desire to use the optimal rate for all problems, but rather to use an automatically tuned competitive rate for a

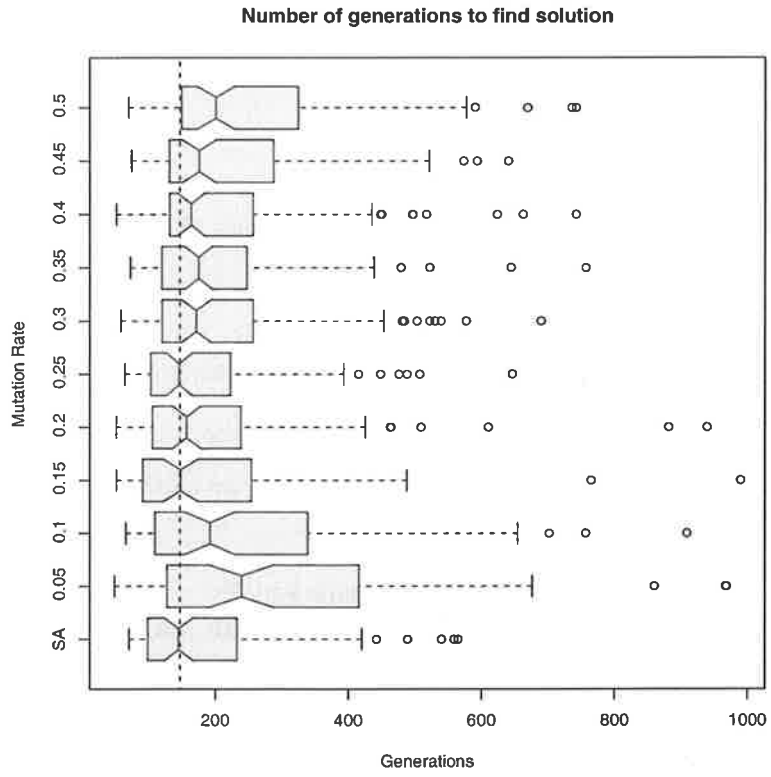


Figure 3.12. *Results for a population of 200. Plots of the relative success at different mutation rate. The graphs show the Self-Adaptive (SA) mutation rate algorithm in comparison to a number of fixed mutation rate values. The dashed line corresponds to the median of the SA algorithm.*

large number of problems. For the simple problem used here, 3.14, the self adaptive rate can be said to be at least as good as any of the experimentally discovered constant rates. For the non-trivial problems considered in this thesis there will not be sufficient resources to statistically analyses constant mutation rate strategies for comparisons with the self adaptive rate. It is clear, however, that the self adaptive strategy is sufficient for solving the problems presented in this thesis, and it is clear from this chapter that it is no worse than, and possibly better than, any constant rate for at least one problem, 3.14.

The problem considered in this section is a simple supervised learning task chosen to demonstrate some of the operating characteristics of the algorithm. This problem could be solved by an inductive learning method such as C4.5 [181, 182, 183] in a matter of seconds since the algorithm is presented with a set of examples and their correct prediction. In the next chapter the more difficult case of an unsupervised learning task is addressed.

Table 3.3. *Summary of symbols used throughout this chapter*

N	The number of attributes
π	The (evolved) model mapping observed environment states to allowable actions
$\aleph$	The set of allowable actions
Ψ	The set of observable states
$\boldsymbol{a}^{(t)}$	The N-tuple vector of sensed attribute values at time $t \in \mathbb{N}$
$\mathcal{S}$	The set of possible states
$\boldsymbol{s}^{(t)}$	The state at time $t \in \mathbb{N}$
Υ	The environment. A mapping of $\mathcal{S} \times \aleph$ to $\mathcal{S}$
I	The mapping of the actual state value $\boldsymbol{s}^{(t)}$ to the observed state value $\boldsymbol{a}^{(t)}$
$\Re$	The run set formed from the tuples $(\boldsymbol{a}^{(t)}, \alpha^{(t)})$, $t \in \{1, \ldots, T\}$
Ω	The evaluation function which is applied to the run set of the model, ie $\Omega(\Re)$ $(\in \mathbb{R})$
Ξ	A rule set
${}^{x}\mathbf{P}$	The parameters associated with a rule set, Ξ
${}^{\sigma}\mathbf{P}$	The mutation step size vector associated with the objective values ${}^{x}\mathbf{P}$
ϑ	The number of parameters associated with the value sets of the model
κ'	The number of parameters associated with the consequences of the model
ν	The number of discrete evolutionary operators defined for the model

3.3 Summary

This chapter has described the proposed SASME evolutionary framework. The general framework is designed to be a system capable of symbiotically evolving parameters and discrete rule structures in a self-adaptive manner. The discrete structures used throughout this thesis are a novel rule set structure which introduces an explicit default hierarchy representation to an evolved Pittsburgh-style learning system.

The SASME framework is designed to evolve representations which have considerable parameter components alongside discrete structures. Previous evolutionary methods have concentrated on evolving structures (like GP) or evolving parameters (like traditional ES/EP evolutionary systems).

The atomic parts of the evolved rule set are IF-THEN rules which are a high level, symbolic knowledge representation. This aids in the comprehensibility of the evolved knowledge [65].

There are several motivations behind the self-adaptation used in the SASME framework.

1 Self-adaptation of the parameter vector allows the parameters to adjust their

mutation rates according to changes which occur in the discrete structure which is symbiotically evolved with them.

2 Self-adaptation of the discrete structure allows the rate of change in the structure performed by different operators to be balanced by the algorithm. This balance occurs according to the rates of application of other operators, as well as the topology of the current structure. This allows the mutation rates of various operators to change relative to one another during the course of a run.

Finding the optimal mutation rate for tasks in general is not possible, however it is shown on a simple learning task that the proposed self-adaptation mechanism can effectively learn in a learning domain. It is also shown on a simple task where the optimal mutation rate is empirically established that the self-adaptation method performs as well as, and most likely better than, the optimal constant mutation strategy. Further, in two variants of the same problem the optimal constant rate appears to change while the self-adaptive method is still the best mutation strategy to employ. The best choice of mutation rate is a function of the structure of the current population and the search space. This is the motivation behind using a self-adaptive mutation rate.

The test case scenario presented demonstrates that for this particular learning problem the self-adaptive strategy presented is the best choice. Although it is only a single empirical study, it provides some confidence that the self-adaptive strategy will be effective more generally. However, under what conditions the self-adaptive strategy will perform better than the best constant rate is an open question. Most learning problem are sufficiently complex that only a handful of calibration experiments can be performed. If this had been done to find a fixed rate on the test problem used in this chapter then the likely performance of the algorithm would have been significantly worse than the self-adaptive strategy, since the self-adaptive strategy significantly outperformed nearly all constant strategies.

The principal innovation of the SASME framework is the explicit division of the parameter optimisation task from the discrete structure optimisation task. This allows a standard self-adaptive evolutionary strategies algorithm to optimise the parameters whilst another self-adaptive algorithm optimises its symbiont, the discrete structure.

The second innovation introduced in this chapter is the utilisation of rule lists with exceptions as the choice of representation for the discrete structure. These rule sets incorporate an explicit representation of a form of default hierarchy, allowing the hierarchies to be evolved by a Pittsburgh style evolutionary algorithm.

Chapter 4

Evolutionary Learning I: Unsupervised Learning of Dynamic Control Systems

> Living organisms are consummate problem solvers. They exhibit a versatility that puts the best computer programs to shame.
>
> John Holland [120]

This chapter introduces the cart-pole problems and applies the SASME algorithm to evolve rule sets to solve some of these problems.

4.1 Background

Control problems involve systems which are described by state variables, which have some variables, called control variables, over which choices are made, and which have some goal or desired state. A diagrammatic description of a control system is shown in Figure 4.1.

Control problems are usually unsupervised learning problems since the correct action for a given state is not known and instead must be discovered. The problems are teleological, and this usually allows a fitness function to be defined. Often there is a cost (or equivalently a reward) associated and the problem becomes to find the optimal control. Costs are frequently measured in units like time, energy, fuel used, money spent or similar.

There has been a lot of research on finding mathematical solutions to optimal control problems. One of the most common techniques employed is dynamic programming [18, 216]. Dynamic programming is a suite of algorithms which compute the optimal

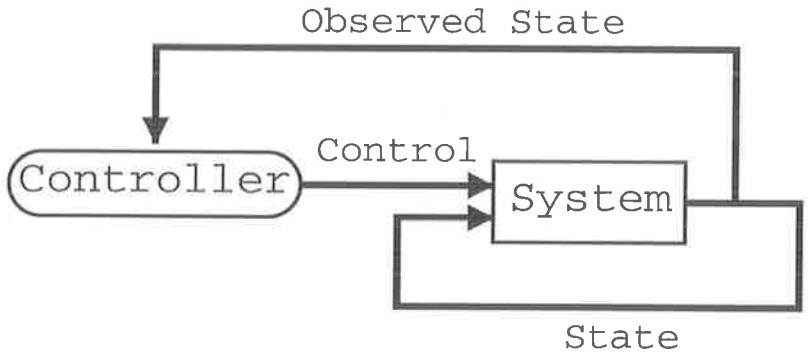


Figure 4.1. *A control problem. At any given time the state variables describe the system completely. The controller makes some observation of the state variables and can use this in determining what control signal to apply at a given time.*

control, or policy, of a multi-step problem for which Bellman's principle is applicable[1]. That is, problems whereby the optimal solution of an n-step problem is the optimal solution of the $n-1$-step problem where the optimal outcome of the first step is achieved [37]. This allows a recursive algorithm to be formulated which can find the optimal policy of the n-step problem. In particular, dynamic programming can be used to solve a Markov decision process, and is an important theoretical underpinning to a lot of reinforcement learning algorithms [130, 216]. Dynamic programming requires an accurate model of the system and for many problems is infeasible due to the excessive computing and storage requirements of the method. Much of the reinforcement learning literature is concerned with how to utilize the ideas used in dynamic programming in environments where an exact model is not known and how to make dynamic programming like algorithms which do not require large computational resources [216].

In this chapter a number of questions will be asked about the application of the self-adaptive rule-based induction system described in the previous chapter on a well known and frequently used control problem. The questions of interest here will be:

- How much information is required about the problem in order to find the solution?
- Can the system learn comprehensible control rules for the problem?
- Is the method able to automatically find an effective quantisation of the state space?

4.1.1 The Cart-Pole Problem

The test problem used in this chapter is the cart-pole balancing problem in Figure 4.2. The problem is also referred to as the broom-balancing problem and the

[1]Bellman's principle should not be confused with Lewis Carroll's "Bellman's principle" that what is said thrice is true.

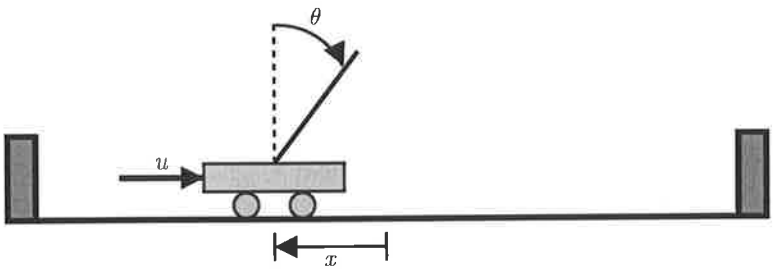


Figure 4.2. *The cart-pole system. The cart is situated on a limited length track and has mounted on it a hinged pole. The aim is to keep the deflection of the pole to within prescribed bounds while keeping the cart on the limited track. The control is a constant force applied horizontally to the cart. Usual performance measures are the sum of the differences between the system and a position with cart in the center of the track and pole vertical, and the amount of time for which the system has remained balanced.*

inverted pendulum problem. The system is a simple control scenario where the controller must balance a hinged pole on a cart. The cart is on a track of finite length and the pole must be maintained within a prescribed number of degree's from the vertical position. The controller has two available control options, to push the cart to the right or to the left, with some constant force. The problem can not be sustained in the equilibrium position of having the pole balanced and the cart stationary since some set non-zero force must always be applied to the cart.

The cart-pole problem is an example of a non-linear dynamic optimisation problem. Although the goals and the statement of the problem differ slightly, the system has been used extensively in the machine learning literature to test different control techniques [159, 17, 167, 236, 220, 139, 222, 153, 68, 151, 74, 131, 47, 32], and is one of the earliest applications of machine learning [159]. The problem is interesting because it is unstable, non-linear, and more than one state variable has to be controlled by a single control [222].

The state of the cart-pole system is described completely by 4 variables:

Cart Position The cart position in metres is labelled x, where x is the distance between the current center of the cart and the middle of the track.

Cart Velocity The cart velocity is labelled[2] $\dot{x}$ and is measured in metres per second.

Pole Angle The pole angle is labelled θ and is measured in radians with a value of 0 representing the pole in a vertical position.

Pole Velocity The angular velocity of the pole is labelled $\dot{\theta}$ and is measured in radians per second.

[2]The dot notation is used to denote the derivative with respect to time, ie $\dot{x} = \frac{dx}{dt}$ and $\ddot{x} = \frac{d^2x}{dt^2}$.

Table 4.1. *Constants used in Equations 4.1–4.4*

Symbol	Description	Value
u	The force applied to the cart	$[-10, 10]$ N
x	The position of the cart on the track	$[-1.0, 1.0]$ m
θ	The angle of the pole from vertical	$[-15, 15]$ degrees
M	The mass of the cart	1.0 kg
m	The mass of the pole	0.1 kg
g	Acceleration due to gravity	-9.8 m/s^2
l	Half the length of the pole	0.5 m
μ_c	Coefficient of friction of the cart on the track	0.0005
μ_p	Coefficient of friction of the poles hinge	0.000002

The control force is labelled u and is measured in Newtons. The equations of motion describing the cart-pole system are:

$$\ddot{x} = \frac{u - \mu_c \operatorname{sgn}(\dot{x}) + \tilde{F}}{M + \tilde{m}} \tag{4.1}$$

$$\ddot{\theta} = -\frac{3}{4l}\left(\ddot{x}\cos\theta + g\sin\theta + \frac{\mu_p\dot{\theta}}{ml}\right) \tag{4.2}$$

where $\tilde{F}$ is the effective force of the pole on the cart,

$$\tilde{F} = ml\dot{\theta}^2\sin\theta + \frac{3}{4}m\cos\theta\left(\frac{\mu_p\dot{\theta}}{ml} + g\sin\theta\right) \tag{4.3}$$

and $\tilde{m}$ is the effective mass of the pole,

$$\tilde{m} = m\left(1 - \frac{3}{4}\cos^2\theta\right) \tag{4.4}$$

and the parameters used in Equations 4.1–4.4 are shown in Table 4.1. The system is integrated numerically by using a fourth-order Runke Kutte method [39, page 112] with a time step of 0.02 seconds.

The cart-pole problem is intended as a test problem for the SASME algorithm. The problem itself is of no interest *per se* for this thesis. Rather it is representative of a type of non-linear dynamic control problem which has been extensively used in the literature. If one were interested in solving the problem there are at least two doctoral dissertations[3] which have been published on the control theoretic solutions to the problem, by Higdon in 1963, [111] and Schaefer in 1965, [204]. The intention here is not to investigate solutions to the cart-pole problem but rather to investigate the application of the SASME algorithm to evolving rules to control the cart-pole system.

In order to put the current approach in perspective, a brief overview of some of the other learning algorithms applied to the cart-pole problem is presented next.

[3] According to Wieland, [236]. The author has sighted neither reference.

4.1.2 Overview of Learning with the Cart-Pole problem

Previous attempts are extensively reviewed by Ling and Buchal, [153] and Varšek *et al.* [222]. Representations of the controller for the cart pole system have been implemented in a number of different ways.

Initial approaches were symbolic and were based on partitioning the state space in some way. One of the earliest partitioning approach was Michie and Chambers BOXES algorithm, [159]. In the BOXES approach to the cart-pole problem the state space is divided into 225 different boxes by partitioning the variables $(x, \dot{x}, \theta, \dot{\theta})$ into 5, 3, 5 and 3 ranges respectively. At any instance the system is in one of the 225 predefined boxes. The boxes algorithm associates an action with each box and learns by modifying actions which are associated with large errors after a number of iterations. The choice of the ranges in the state space is critical to the BOXES algorithm, and the learned strategies proved to not be *generic* in the sense that they failed to control the system from random initial states [153]. It was later shown that some of the boxes could be amalgamated making the controller more comprehensible. The algorithm could also be improved by BOXES learning being applied to a range of initial states and the different results collated by a voting algorithm for each box [202, 153]. The BOXES approach uses a representation similar to most of the reinforcement learning methods which have been applied [222]. In a number of these approaches on the cart-pole problem the individual boxes were combined or quantised by a panoply of different algorithms. Automatic quantisation of the partitioning of the state space is seen as important since for many problems a correct partitioning will not be known beforehand, and the results of the algorithms seem to be dependent on the partitions chosen [153].

Another representation applied to the cart-pole problem is to learn a numerical equation which approximates the control surface. This approach was taken by the classification and regression tree algorithm, CART, for the cart-pole problem [222, 153]. In the CART approach the state space is not pre-partitioned, but the results depended on some domain knowledge and heuristics, and the output was a very complicated numerical equation which did not generate any understanding or rules [153]. The approach was also unreliable [222].

A third representation used for solving the cart-pole problem are neuron-like approaches. A single *associative search element* (ASE) and a single *adaptive critic element* (ACE) were used by Barto *et al.* to solve the problem [17]. The approach used the same state space partitioning as the BOXES algorithm. The critic element is charged with predicting the outcome should the ASE change a particular action for a particular box, and is then used as reinforcement for the ASE as it learns by modifying its weights.

A number of evolutionary learning approaches have also been applied. A boxes-like approach was implemented by coding the boxes as a binary string corresponding to the action to be taken when the state space entered a particular box [167]. Varšek *et al.* use a more complex system to solve the problem [222]. The state space is

partitioned into a boxes like array, each variable being divided into 3 different ranges, giving $3^4 = 81$ bits corresponding to actions for the given state. The partition points for the chromosome were then also coded in binary and evolved. By assuming symmetry, the number of bits needed can be halved and a controller for the cart pole problem was evolved. The controller table was not comprehensible, and so a rule induction system was used to derive a rule set from the induced control table. The induced rule method was only supplied with action/box pairs for boxes which had been entered during the evolution of the rule sets, and the system was set to prune the rules drastically and to limit attribute comparisons to a single attribute-value pair for each rule. The rulesets derived were now comprehensible, but no longer reliable. The rule set was then optimised in a third stage by using a genetic algorithm to tune the numerical values used in defining the partition points used in the rule set. Again a binary genetic algorithm was used, this time with more bits to represent a number to allow for better precision. This was then the final controller for the cart pole system. The final controller was comprehensible and performed better than any of the previous controllers.

Another evolutionary approach to the cart pole problem is to evolve a neural network to balance the system. This has been done with a binary encoded weights array [236], with an evolutionary programming algorithm [74] and with the Symbiotic, Artificial Neuro-Evolution (SANE) method [97]. The resulting neural networks are able to control the cart-pole system, but provide little information about how they are doing so.

Genetic programming has also been used to evolve equations which can balance the cart pole system. Koza [139, pages 289–307] evolves a controller for the 3-state problem excluding the position of the cart. Evolutionary programming has also been applied to symbolic regression with the same function and terminal set and achieved similar results [47].

4.1.3 Implementation

Although the cart-pole problem has been widely used, the typical statement of the problem is not very difficult to solve. The most frequently used success criteria is the survival time of the system. The number of iterations used to evaluate the survival time has been limited to 10,000 iterations in many studies [159, 222, 153, 68], representing just 3.3 minutes of simulated time[4].

Value sets

The value sets that the evolved rule set Ξ utilizes are a partitioning of the real valued state space, $\boldsymbol{s}^{(t)} = (x, \dot{x}, \theta, \dot{\theta})$. Each of the four variables in the state space is divided

[4]The choice of 0.02s as the step size in the numerical integration appears to be universal.

Table 4.2. *Initial state variable ranges. Shows the limits of the initial position of the cart pole system used in trials.*

Symbol	Description	Range
x	The position of the cart on the track	$[-0.5, 0.5]$ m
θ	The angle of the pole from vertical	$[-2, 2]$ degrees
$\dot{x}$	The position of the cart on the track	$[-0.1, 0.1]$ m/s
$\dot{\theta}$	The angle of the pole from vertical	$[-1, 1]$ degrees/s

into three partitions,

$$\textbf{Low} \quad (-\infty, p_1] \tag{4.5}$$
$$\textbf{Medium} \quad (p_1, p_2] \tag{4.6}$$
$$\textbf{High} \quad (p_2, \infty) \tag{4.7}$$

where the real partition values are evolved as part of the object parameter ${}^x\mathbf{P}$. This makes $\vartheta = 8$ parameters in ${}^x\mathbf{P}$ associated with the evolution of the parameter set.

Action Parameters

Two different problems are addressed with respect to the set of allowable actions $\aleph$.

Constant Actions The allowable action set is restricted to a 10 Newton force applied to the left or the right of the cart. That is, $\aleph = \{-10, 10\}$. The number of actions, A, is 2, with $\omega_1 = -10$ and $\omega_2 = 10$. Consequently there are no evolved parameters associated with the consequence part of a rule, and so $\kappa_k = 0$, $k \in \{1, 2\}$, and $\kappa' = 0$. Experiments using this set of allowable actions have been used in the literature [159, 17, 167, 222, 153, 68].

Evolved Actions The allowable action set is restricted to a force in the range $[-15, 15]$ N to be applied to the cart. In this case $\aleph$ is the real interval $[-15, 15]$. The number of possible actions is restricted[5] to 2, $A = 2$. The number of evolved parameters associated with each allowed action, κ_k is 1, $k \in \{1, 2\}$, and the total number of evolved parameters associated with the evolved actions is $\kappa' = 2$. This means that each rule can have one of two consequences, which is to push the cart with one of the two evolved forces.

Initialisation of the system

When assessing the generality of the results it is usual to start the system from a range of initial positions. The initial values used in this chapter are shown in

[5]Experiments with more allowable actions were conducted but are not reported here. Increasing the number of allowable actions makes the problem easier to solve.

Table 4.2 and are equal to the widest values which occur in the literature. Some care has to be taken when evaluating individuals from random initial conditions. If each rule set in the population is evaluated from a different random initial state then some offspring may be eliminated because they were randomly applied to a very difficult initial state. Other offspring may survive only because they were started on an easier initial state. Generalisation can suffer when there is insufficient sampling of the initial positions. If a parent rule set can only control the system from a limited number of states, for example, only when the cart pole system is left of the centre of the track, and its offspring are tested from a single random initial state, then half of the offspring will end up being tested from the area of the search space that the parent has specialised in. If this area gives those offspring good fitness evaluations then they will persist in the population and the solutions will not be general.

To solve these issues the results reported in the next section use the following strategy to generate initial positions to evaluate solutions:

- The same initial conditions are used for all members of the population.
- Conditions are generated from the edges of the initial state variable regions described in Table 4.2. This strategy was found to effectively evaluate the success of controllers on any initial state generated within the initial limits.
- The controller is given the mean value of its performance over all initial conditions it is applied to.
- After the controller is evaluated on a particular initial condition it is always immediately after evaluated on the *reflection* of that condition. That is, if the condition is for the cart to be furthest to the right, going left as fast as possible with the pole to the right and moving right, then it is applied to the same state with left and right reversed.
- No two initial states in a set of states used for evaluation are identical.

Typically the population is evaluated from 2 different random points on the edge of the feasible region and their reflection. The fitness is assigned to the mean of the 4 evaluations.

Evaluation

A number of different feedback options are possible for the cart-pole system.

Time Only The least informative feedback is the survival time of the cart-pole system. This assumes no *apriori* knowledge about how to go about the problem. The main problem with using survival time alone is that the controller is not punished for moving the system away from the middle of the track. It is possible to generate controllers which do not take into account the position of the cart but which nevertheless are able to control the system for the required periods of times. This occurs because the pole is the most important state information for the system. The system cannot push the cart off the end of the track without first unbalancing the pole from nearly every state.

So the system often learns to balance the pole whilst moving the cart as little as possible without ever learning to keep the cart away from the ends of the tracks. There is nothing wrong with such a control strategy, since it solves the problem given to the system.

Time and Position The survival time reinforcement can be augmented by the mean distance from the centre of the track to make a reinforcement signal which punishes solutions which keep the cart away from the centre of the track. This is done by using the normalised distance of the cart from the centre of the track at each time step and the normalised survival time of the cart pole system. The fitness can then be calculated as follows:

$$\text{PosErr}_k = \sum_{i=1}^{S_k} \frac{|x_i|}{x_{\max}} \quad k = 1, \ldots, N \tag{4.8}$$

$$\overline{\text{PosErr}} = \frac{1}{N} \sum_{k=1}^{N} \frac{\text{PosErr}_k}{S_k} \tag{4.9}$$

$$\overline{S} = \frac{1}{N} \sum_{k=1}^{N} \frac{S_k}{S_{\max}} \tag{4.10}$$

$$f = \overline{S}(1 - \overline{\text{PosErr}}) \tag{4.11}$$

where PosErr_k is the sum of the normalised distance between the cart and the center of the track at each time step, i; x is the position of the cart and $x_{\max}$ is the maximum allowed position of the cart from the centre of the track; $\overline{\text{PosErr}}$ is the average normalised position error, averaged over the number of trials, N and normalised by the survival time[6] S_k, of the trial $k \in \{1, \ldots, N\}$; $S_{\max}$ is the maximum survival time; $\overline{S}$ is the mean normalised survival time, and f is the normalised fitness, which is to be maximized.

Full System State The position of the pole can also be used in the fitness calculation by replacing Equations 4.8 and 4.9 with the following:

$$\text{Err}_k = \frac{1}{2} \sum_{i=1}^{S_k} \left(\frac{|x_i|}{x_{\max}} + \frac{|\theta_i|}{\theta_{\max}} \right) \quad k = 1, \ldots, N \tag{4.12}$$

$$\overline{\text{Err}} = \frac{1}{N} \sum_{k=1}^{N} \frac{\text{Err}_k}{S_k} \tag{4.13}$$

where Err_k is the k-th trials normalised error, $\theta_{\max}$ is the maximum allowed angle of the pole, θ is the angle from vertical of the pole and $\overline{\text{Err}}$ is the normalised error. The other variables are identical to those described above, and Equations 4.12 and 4.13 combined with Equations 4.10 and 4.11 allow the fitness to be calculated.

[6] The number of simulated time periods that the cart pole system remained in a valid state.

Table 4.3. *Table of statistics from Figure 4.3 and 4.4 showing the generation at which the system had balanced the cart and pole system*

Fitness criteria	Evolve Action	Quartile 1	Median	Quartile 3
Complete state	No	11	18	32
Position and time	No	11	16	33
Time only	No	10	16	32
Complete state	Yes	11	22	39
Position and time	Yes	12	17	36
Time only	Yes	14	23	>39

4.2 Experimental Studies on the Cart-Pole Problem

Throughout this section a population of 100 rule sets are used to evolve controllers that can balance the cart pole system for a minimum of 5.5 hours of simulated time[7] from a set of random initial position with the parameters shown in Table 4.1.

4.2.1 Evolutionary performance

A set of 100 runs were conducted using each of the three fitness criteria discussed in the previous section and using a choice of two constant action values, -10 N or 10 N. The reported survival time is the mean survival time of the cart pole system when started from four different random initial conditions, as discussed on page 81.

The results shown in Figure 4.3 show the median survival time over the 100 runs of the best rule set in the population at each generation. The shaded area indicates the interquartile range of the best survival time. The results show that the choice of fitness function does not greatly affect the performance of the SASME algorithm. It also shows that a solution to the cart pole problem is most frequently evolved in less than 40 generations.[8]

The experiment was repeated with the evolutionary method able to evolve the values of the two applied control forces, as discussed on page 81. This makes the problem somewhat more difficult since there are an extra two parameters to evolve. Figure 4.4 shows the median evolutionary performance of the algorithm on this problem after 100 runs were performed on each of the fitness criteria.

Table 4.3 summarizes the results of Figure 4.3 and 4.4 and shows the distribution of the number of generations required to find a solution to the problem of balancing the cart pole system for 5.5 hours.

[7] 1 000 000 time steps of the model integration.

[8] These runs were terminated at 40 generations. By this time nearly all runs had found a controller which would balance the system for 5.5 hours.

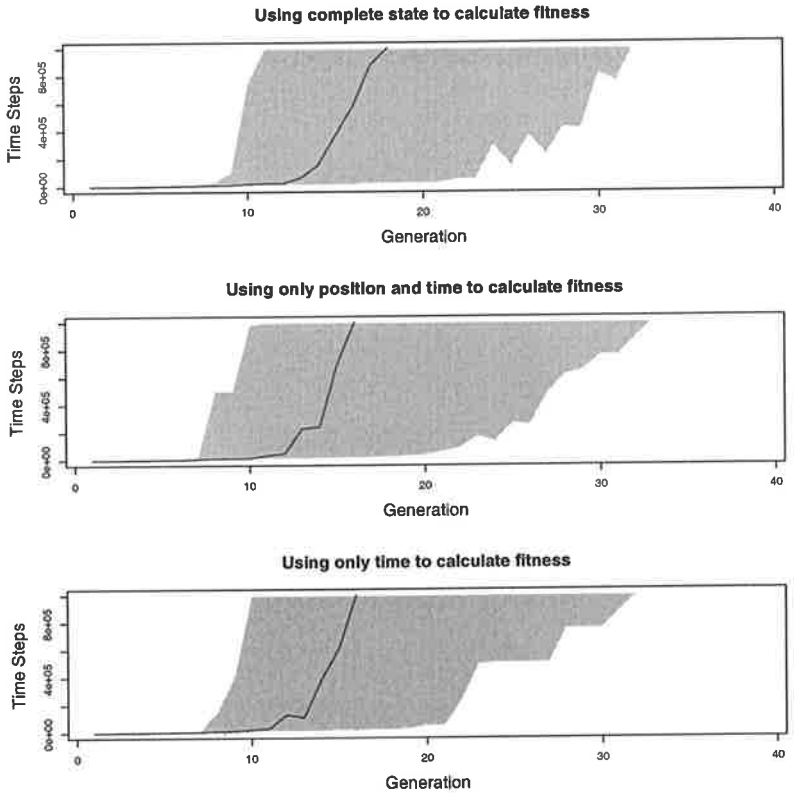


Figure 4.3. *Results of evolving controllers for the cart pole problem. The graphs show the median survival time of the best of generation over 100 runs. The shaded area indicates the interquartile range of the best survival time at each generation. Time is measured in 0.02 second increments, as used in the numerical integration. The three sets of runs correspond to different fitness functions: Using the normalised mean square error of the state of the complete system and the survival time, using only position and survival time and using only the survival time of the system.*

Discussion of Evolutionary Performance

From Table 4.3 it is clear that the SASME algorithm is able to reliably find a controller which can balance the cart pole system for 5.5 hours. Increasing the amount of feedback from the system to the learner does not improve the learning time in this case. There are two likely factors which cause this. The first is the rapid speed that the algorithm is able to solve the problem. This means that there is not much opportunity for improvement. The second is that the increased feedback is not likely to be as useful as the literature suggests in this case. Evolutionary learning uses the environmental feedback to update the model after the model has completed its interaction with the environment. There is no attempt to update the model *online*. Therefore, errors in the physical state of the cart pole system are averaged over all time steps and become swamped by difference in the survival time when comparing models. After the system is balanced the errors in the state can be reduced resulting in a cart pole system which remains close to the center of the track with the poles

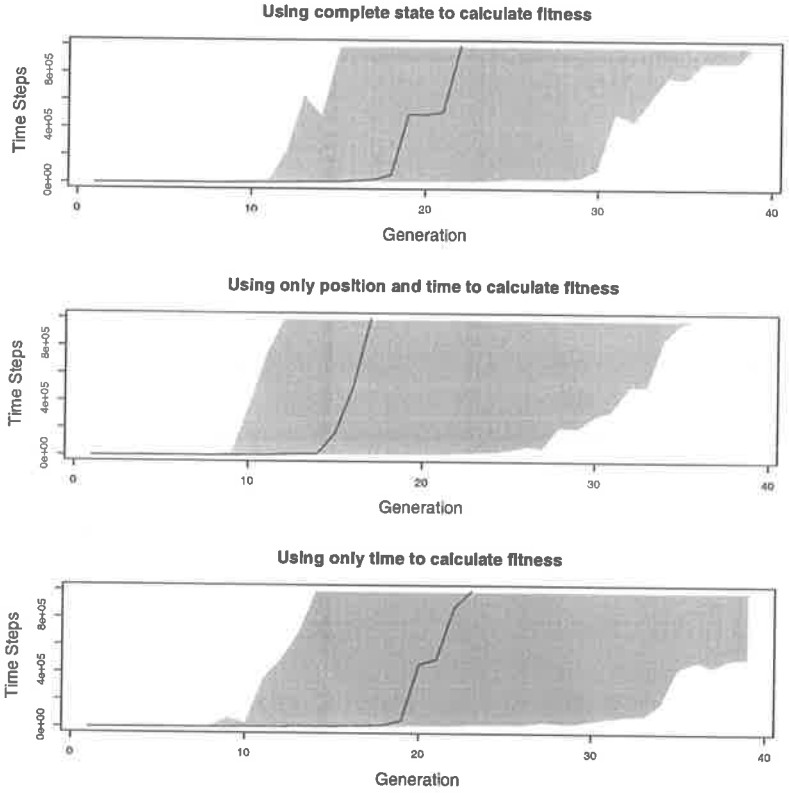


Figure 4.4. *Results of evolving controllers for the cart pole problem when the applied force is subject to evolution. Graphs show the median performance of the algorithm over 100 runs with the shaded areas showing the interquartile range of that performance. Compare with Figure 4.3 where the applied force is not evolved.*

nearer to upright.

When the problem given to the system is extended to finding the numerical values of the controller the system is still able to reliably find a controller which can balance the cart and pole from a range of initial conditions for 5.5 hours. Table 4.3 shows that the number of generations required to do so appears to increase by a small amount. The requirement to search for solutions using an evolved force magnitude increases the number of parameters which need to be evolved by two, increasing the complexity of the problem. It also reduces the amount of symmetry supplied to the learner, which usually discovers controllers which supply a different amount of force to the left as to the right of the cart.

In summary it can be said that the evolutionary system is able to reliably evolve controllers for the cart pole system for 5.5 hours of simulated time from a wide range of initial conditions with only sparse success–failure time feedback.

4.2.2 Trajectory of the Evolution

Section 3.2, page 65, demonstrates that the partition points of the evolutionary method converge to the artificial values used to generate the data set used in that section. In the case of the cart-pole problem used here there are no correct values of the partition points of the state variable attributes. Whether a partition point works depends on the particular control strategy that is being used by the rest of the rule set. The precision in the value of a partition point is also likely to be less important for the cart-pole problem. A rule which pushes the cart back to the left when it is further than 0.31m from the center of the track may work as well as a rule which pushes the cart left when it is further than 0.32m. Other values may require more precision because of the interaction between the evolved values and the rest of the rule set. The structure of the evolved rule sets will be considered in the next section.

A population of 100 rule set models is evolved for 40 generations using the survival time and with evolved actions. The fitness at each generation and the behaviour of the best model are shown in Figure 4.5. The evolved strategy is able to control the system for 5.5 hours of simulated time as shown in the bottom two graphs of Figure 4.5. The graph on the top right shows a combined close up view of the bottom two graphs over the first 30 seconds of simulated time. The graph shows the evolved control strategy oscillating the pole so that it is pointed towards the centre of the track (the deflection is to the right when the cart is to the left of the centre of the track).

The evolution of the partitioning of the state space occurs while the rule set is evolving. Figure 4.6 shows the evolution of the state space partitioning for the run used in Figure 4.5. Each point in the graphs represent the mean value of that partition point in the population at that generation. The graphs to the left show the mean stepsize of the corresponding partition value in the population at each generation. The values evolve to optimise the particular discrete rule set that they are co-evolving with. The step sizes reduce as the fitness of the child models become better when they mutate their parent values less.

The application rate of discrete mutation operators on the rule set is shown in Figure 4.7. The mutation rate graphs show that the relative and absolute values of mutation during a run vary significantly in this run. Overall the rate decreases as the run progresses, increasing the evolvability of the population as children closer in structure to their parents have a higher fitness.

Discussion of Evolutionary Trajectories

The evolution of the values associated with the rule sets for the cart pole problem are different to the evolutionary trajectories the values followed in the simple example problem of section 3.2 on page 65. In the earlier problem the values of the attributes

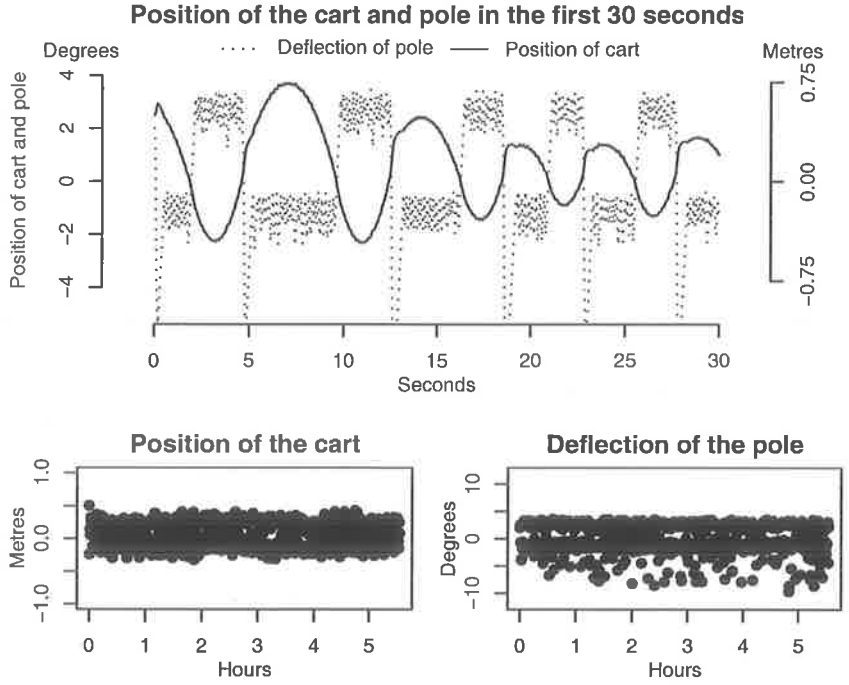


Figure 4.5. *Evolved action model graphs. The graphs show the survival time of the best found model at each generation and the behaviour of the final best found model over the first 30 seconds and over the entire 5.5 hours of simulated time.*

converged to the values used to generate the learning examples. There are no such corresponding values for the cart pole problem. A different run of the algorithm produces rule sets with different partition values. Different sets of partition values occur because the best set of values will depend on the rule set which is used to generate the control strategy. The self adaptive strategy shows some evidence of being able to modify its mutation step size in response to the environmental circumstances in which it finds itself. The step size does not monotonically decrease, which provides evidence that the evolving parameters are searching within an increasing radius at different stages in the search. This may occur when the discrete structure discovers a new rule utilizing a value which can then be optimised or tuned.

Figure 4.7 shows that the mutation probabilities vary in a similar way to Figure 3.10 on page 67. Again, the rates of mutation applied to the discrete structure at each generation vary in each run of the evolutionary algorithm according to the particular trajectory that a particular run takes through the search space. Figure 3.10 shows that in the last 3 generations the rate of application of the deletion operator acheived its largest values. This is most likely a stochastic effect, since the rate[9] becomes about 0.45, and a high rate will persist if enough high fitness individuals persist in

[9]Note that rate here refers to the probability of application of a particular mutation, not the actual rate of application. The usage conforms with the algorithm description given in the previous chapter.

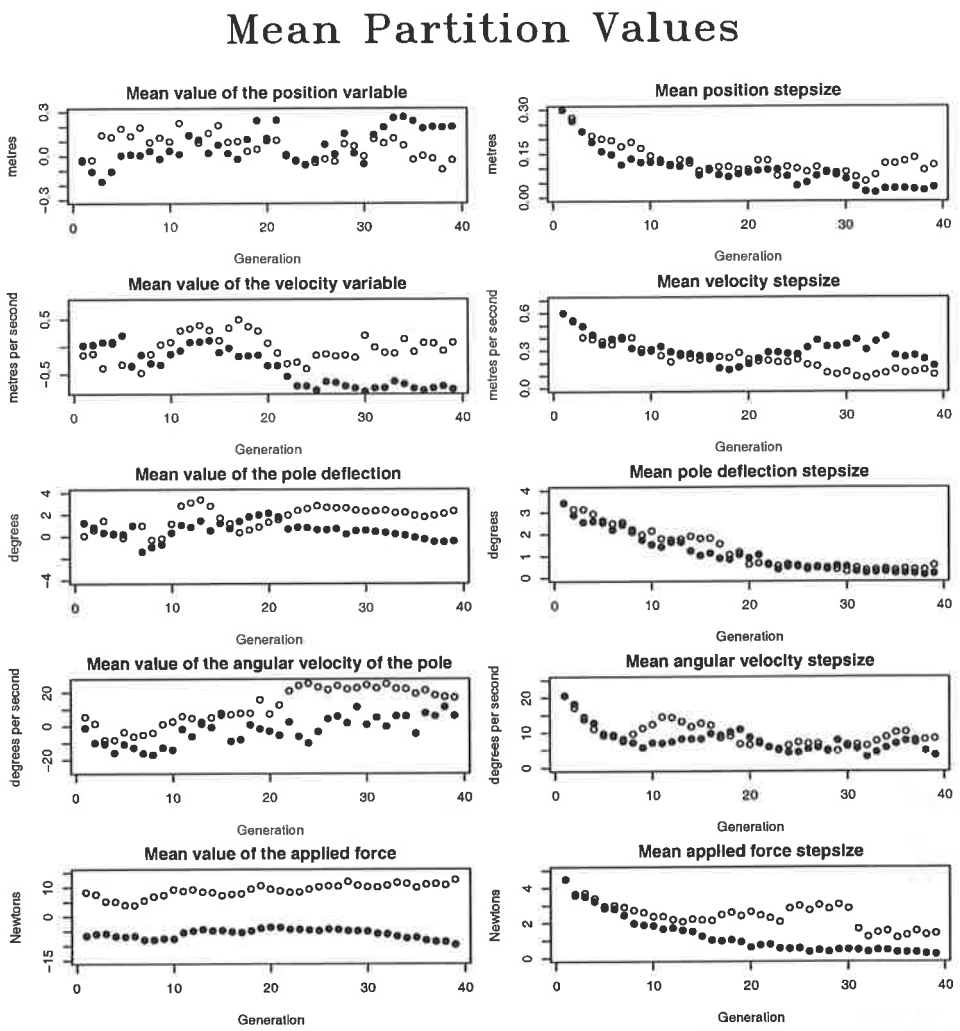


Figure 4.6. *Comparison of the evolution of the state variable partitions. Each point represents the mean values of that particular partition point in the population at that generation. The step sizes are the mean values of the standard deviations used to mutate parent models at that generation for that partition value.*

the population with that rate. The expectation is that over half of these individuals would not have experienced a deletion mutation. In this sense, the graphs show a snapshot of a particular finite population undergoing evolution; however, the trend remains for the rate of application of the mutation operators to decrease as the evolution proceeds.

Figure 4.5 shows that the symbiotic evolution of the parameters shown in Figure 4.6 and Figure 4.7 is able to evolve an effective controller of the cart pole system. The evolved model is able to control the system for 5.5 hours from a range of initial conditions, one of which is shown in Figure 4.5.

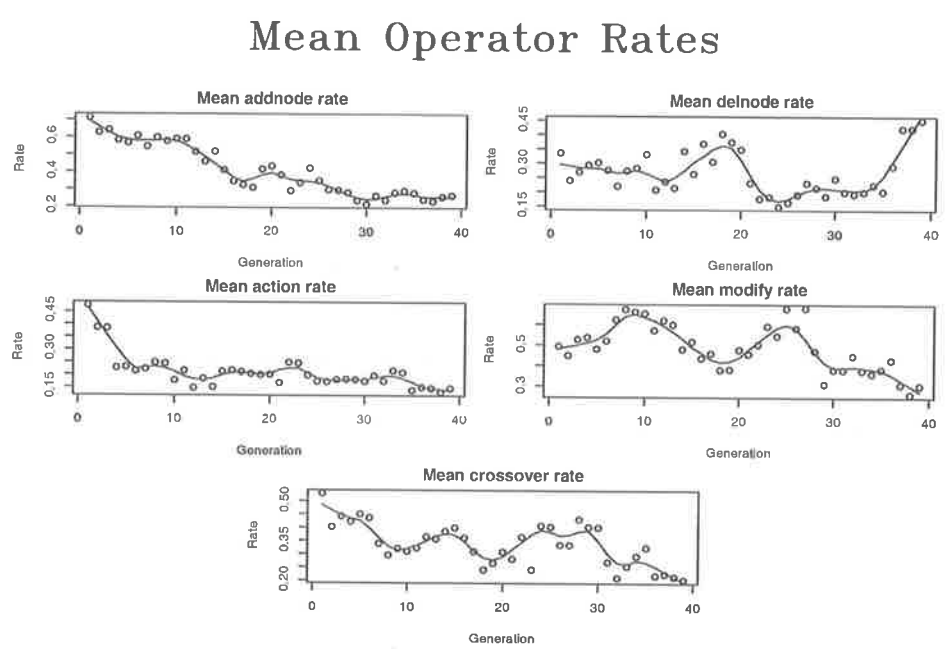


Figure 4.7. *Probabilities of discrete mutations. The mean value of the application rate of each mutation operator in the population at each generation is graphed.*

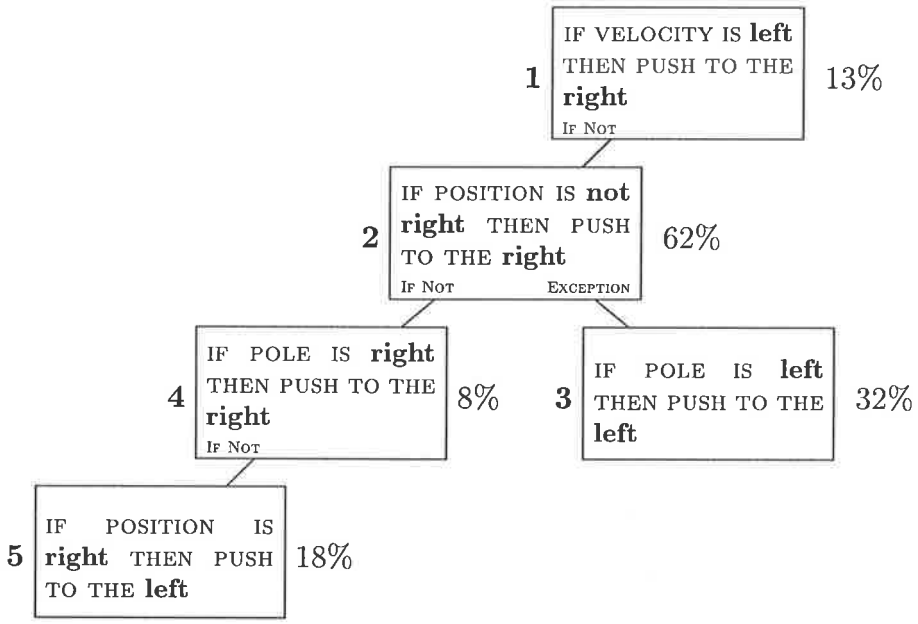
4.2.3 The Evolved Rule Structure

The SASME method symbiotically evolves the rule set along with the parameter vectors. Rule set models are chosen because they are transparent, which allows the *reasoning* that the controller uses to make decisions to be observed.

The controllers which are evolved for the cart pole problem are usually succinct, consisting of only a small number of rules. An example of an evolved rule set is shown in Figure 4.8. Each rule used in the controller has an expected condition and outcome. Rule **1** in Figure 4.8, for example, states that when the cart is moving quickly to the left push it to the right. Rule **2** states that if the cart is near the center or to the left of the track then it should be pushed to the right. Rule **3** corrects rule **2** by adding the exception that if the pole is leaning left then the cart should be pushed to the left anyway (pushing to the left would have the effect of balancing the pole). The pole falls quickly and so has the highest priority in the controller. The numerical values which correspond to the linguistic[10] labels are shown in the table below the rule set in Figure 4.8. The percentages to the right of the rules are the number of times that the rules antecedent is found to be true. Whether the rules consequence is used depends on whether the rule has any exceptions which have their antecedent found true.[11]

[10]The linguistic labels are arbitrarily defined by the author. No attempt at objectively assigning such labels has been made. For the SASME algorithm, the labels correspond to enumerated value sets.

[11]The sum of the usage of the rules along the main spine does not always add to 100% due to rounding errors in the reported percentages.



Description	Value
Position of cart is not to the right	$x < 0.3$m
Position of cart is to the right	$x > 0.2$m
Pole is to the left	$\theta < 0.004$rad $(0.2°)$
Pole is to the right	$\theta > 0.008$rad $(0.5°)$
Velocity of cart is to the left	$\dot{x} < -0.4$m/s

Figure 4.8. *An evolved rule set and associated parameters for the cart pole problem. The values of the numerical labels in the rule set are shown in the table. Rules joined to the right are tested if and only if the rule immediately to their left is true. Rules joined from above are tested if and only if the rules above are not true. The percentages refers to the percentage of cases that the rule is used in controlling the cart and pole for 5.5 hours from a particular random initial position.*

The table in Figure 4.8 shows that the two partition values evolved[12] for the position attribute are 0.3 m and 0.2 m. The first value set corresponds to a position of less than 0.3 m from the center of the track, which has been labelled as **not right** in this example. The second value set would correspond to a position on the track between 0.2 m and 0.3 m, and is not used in the rule set. The third value set is for a position further right than 0.2 m from the center of the track. Rule 5 uses this value set, although because of its position in the rule set this rule will only be tested for positions greater than 0.3 m.

The rule set shown in Figure 4.8 does not use the angular velocity, $\dot{\theta}$ nor does it use all of the partition points for the velocity and position attributes.

Unlike some other studies [222], the evolved partition points are not assumed to be

[12] All quoted values are rounded, the actual values produced and used in the simulation were of double precision.

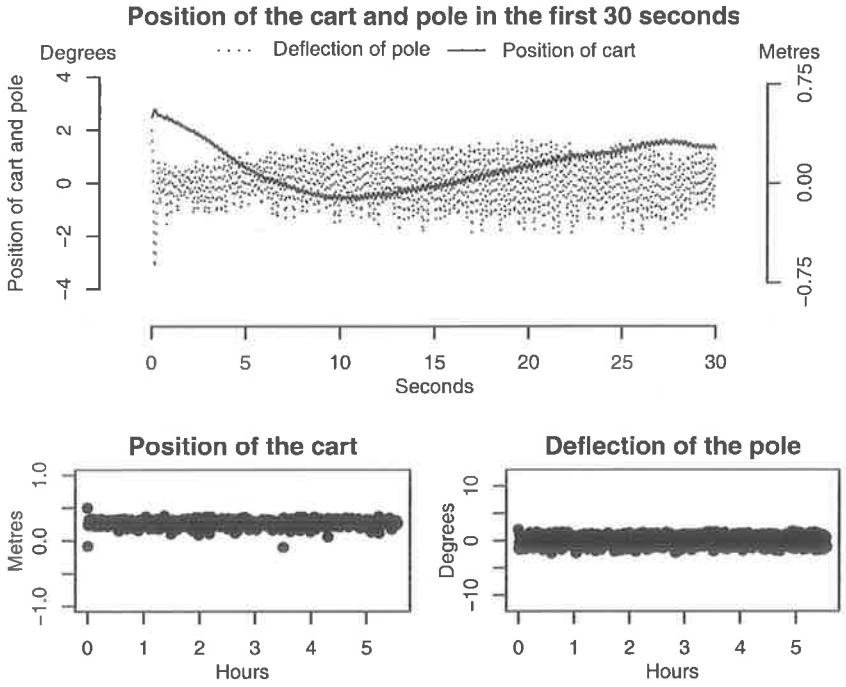
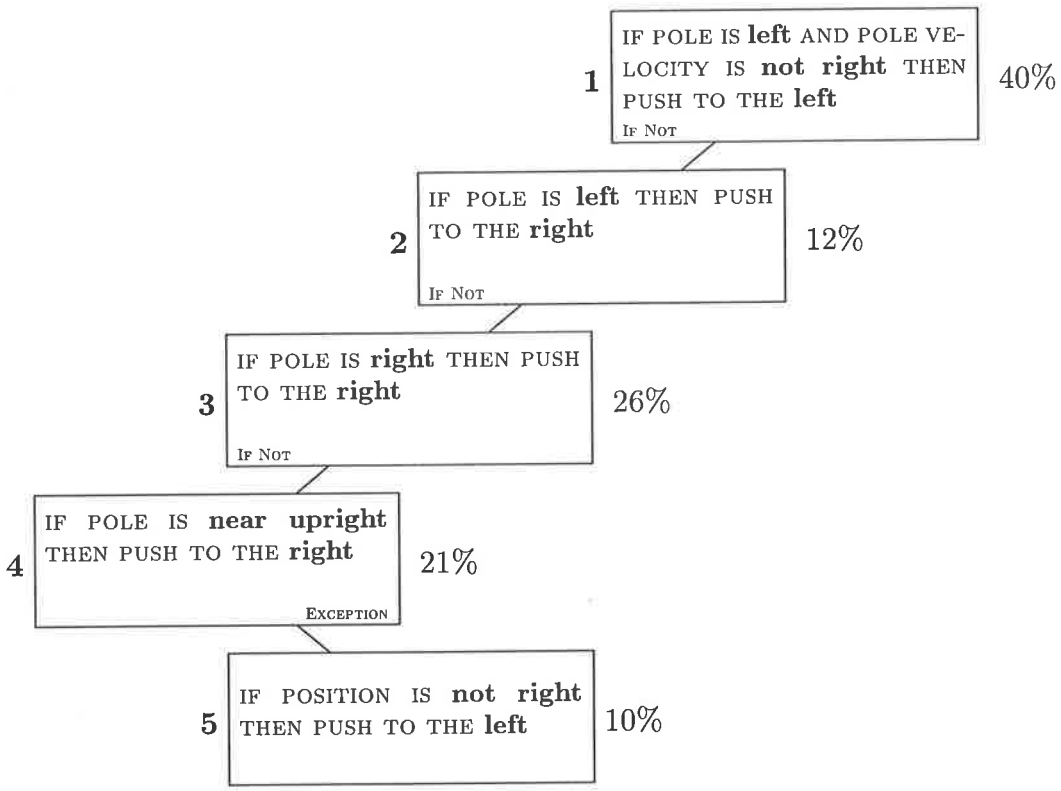


Figure 4.9. *Solution generated by controller in Figure 4.8, showing the behaviour of the controller applied to the system.*

symmetric. Because of this the rule set in Figure 4.8 does not maintain the cart near the centre of the track. The rule in Figure 4.8 was evolved with the survival time only as the fitness function. This fitness evaluation places no bias on the solution to maintain the cart-pole position near the centre of the track. Figure 4.9 shows the position of the cart and pole in the first 30 seconds of a simulation trial from a random initial state and over the 5.5 hours of simulated time using the controller shown in Figure 4.8. The plots of position and deflection against time over the 5.5 hours at the bottom of the figure show that the carts position is centred around 0.25m, and not around 0m.

Another rule set for the cart-pole problem is shown in Figure 4.10. This rule set uses a conjunction as its root rule. Rule **2** will result in the cart being pushed in a direction which would be expected to further unbalance the pole. However the rule is only parsed when Rule **1** is false, which requires that the pole will be moving quickly to the right (faster than 24°/s) even though it is currently left of upright. Rules **4** and **5** state that when the pole is near upright the cart should be moved to the left except when it is further right than 0.3m. This rule will have the affect of moving the pole to the right of 0.3m and unbalancing the pole to the left of upright. Presumably the preceding rules which deal with the unbalanced pole are biased towards moving the cart to the left of the position where they are initially used.



Description	Value
Position of cart is not to the right	$x < 0.3$m
Pole is to the left	$\theta < -0.003$rad (-0.2°)
Pole is near upright	$-.003$rad $(-0.2°) \leq \theta < 0.02$rad (1.1°)
Pole is to the right	$\theta > 0.02$rad (1.1°)
Angular velocity of pole is not to the right	$\dot{\theta} < 0.43$ rad/s (24 °/s)

Figure 4.10. *An evolved rule set for the cart pole problem. The values of the numerical labels are shown in the table. The output of this rule set is shown in Figure 4.11*

Figure 4.11 shows the long term trajectories of the cart and pole using this rule set. The cart stays near 0.3m from the centre of the track. Again, the fitness evaluation used considered only the survival time of the system. The behaviour of the system in the first 30 seconds in Figure 4.9 and 4.11 are remarkably different, showing that the evolved controllers for this problem utilize different strategies to control the system.

A more complicated case is where the algorithm has to also evolve the strength of force to apply along with the rule set. The model which produced Figure 4.5 was evolved with variable action strengths. The rule set which describes this controller is shown in Figure 4.12.

Figure 4.12 also shows the evolved forces which the rule set uses. In the previous rule sets the number of times a force to the right is applied is equal to the number of times a force to the left is applied, since the two forces are equal and the system cannot drift away from the centre of the track. In the case of different force magnitudes

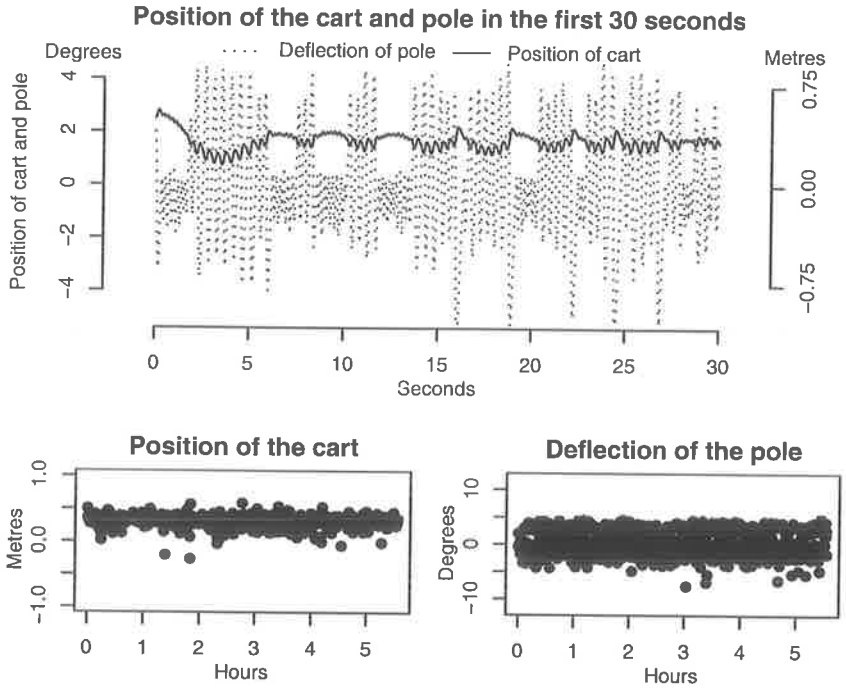


Figure 4.11. *Solution generated by controller in Figure 4.10, showing the behaviour of the controller applied to the system.*

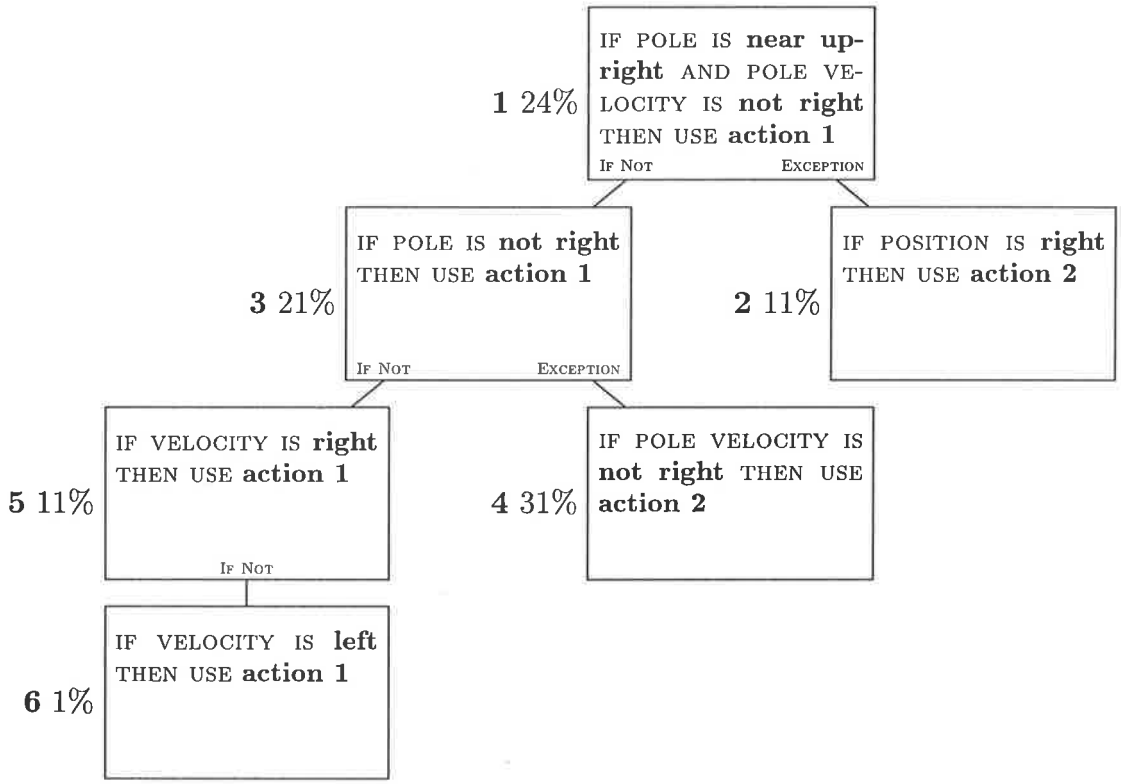
applied in different directions the application rate of the two actions in the controller shown in Figure 4.12 are within precision equal to the ratio of the applied forces,

$$\frac{\text{Value of action 1}}{\text{Value of action 2}} = \frac{\text{Rate of use of action 1}}{\text{Rate of use of action 2}} \approx 1.25$$

Discussion of Rule Set Models

The rule sets evolved by the system are a transparent control representation. By looking at the resultant structures some insights into how the model is controlling the system can be learnt. The models can be said to be comprehensible.

A different model is found with each new run of the evolutionary process. One reason for this is the unconstrained nature of the problem. Nevertheless some commonalities between models can be found. All of the models shown use the condition that when the pole is left the cart pole system should be pushed to the left. The process has evidently learnt some aspects of the pole balancing problem. Although these things may seem trivial to an onlooker—if the pole is about to fall over to the left it seems simple to learn that the cart must be pushed to the right to prevent this happening—the evolutionary process must learn these simple things automatically through trial and error, and then represent them so that they can be understood.



Description	Value
Position of cart is to the right	$x > 0.03$m
Pole is not right	$\theta < 0.05$rad (2.9°)
Pole is near upright	$-.01$rad $(-0.8°) \leq \theta < 0.05$rad (2.9°)
Velocity is left	$\dot{x} < -1$ m/s
Velocity is right	$\dot{x} > -1$ m/s
Angular velocity of pole is not to the right	$\dot{\theta} < 0.29$ rad/s (17 °/s)
Angular velocity of pole is to the right	$\dot{\theta} > 0.29$ rad/s (17 °/s)
Action 1	Apply a force to the right of 12.5N
Action 2	Apply a force to the left of 10.1N

Figure 4.12. *An evolved rule set for the cart pole problem. The values of the numerical labels are shown in the table below the ruleset. Rules to the right are tested iff the rule immediately to their left is true. Rules below are tested when the rules above are not true. The percentages refers to the percentage of cases that the rule is used in controlling the cart and pole for 5.5 hours from a particular random initial position.*

Balancing a pen upright on the palm of ones hand shows how quickly and intuitively the rules behind the balancing problem are learnt by humans. The machine learning has to discover for itself a symbolic representation of those human intuitions. In addition, it must also discover a set of thresholds, or values, which can be linked to the simple intuitions learnt to create a controller which can utilize a computer to produce the necessary output to solve the problem.

4.3 Summary of Results for the Cart-Pole Problem

Learning to control the cart pole system from a range of initial conditions for a long stretch of time can be achieved by the evolutionary system. Further, the rule sets produced demonstrate that some form of learnt knowledge can be extracted from the process. Unlike most other algorithms which have been employed for solving these kinds of dynamic control problems, the SASME evolutionary approach automatically finds the partition values of the discretisation of the state space, and does so while simultaneously evolving the rule sets which control the system.

The results of Figure 4.3 and 4.4, summarized in Table 4.3, show that the learning is robust to a number of different factors which modify the problem domain. At first it seems surprising that the more informative fitness functions do not lead to a reduction in the time required to learn a controller for the system. It appears the information supplied by using the complete state to evaluate solutions, is effectively swamped by the information contained in the length of time the system was balanced. The different fitness evaluations lead to a qualitative difference in the solutions, since solutions evolved with only survival time information are not effectively punished for maintaining the cart position away from the centre of the track so long as it does not hit the ends. Learning to maintain the pole in a near upright position is likely to occur regardless of the fitness evaluation since the velocity of the pole due to gravity can be more easily controlled with an upright pole.

This is evident in Figure 4.13 which shows the distribution of the normalised values of the two state variables, x and θ, for the two runs, one with complete state information, the other with only survival time information. The run using only the survival time learns to keep the pole near upright, however the cart is maintained at an arbitrary median distance from the centre of the track.

The rule set controller can only apply one of a discrete number of control forces. This prevents the system from performing fine control actions which could put the system in balance. It is also consistent with many real world control tasks.

Figures 4.3 and 4.4 also indicates that the evolution of the cart pole system can proceed when the controller is also required to evolve the values of the applied forces. This problem has more parameters than the original controller, but may in fact not be more difficult, as the more degrees of freedom the controller has the easier it may be to find a solution.

The cart pole problem is an interesting dynamic control situation. It is, however, not difficult to solve. In the next section a much harder variant will be described and solved.

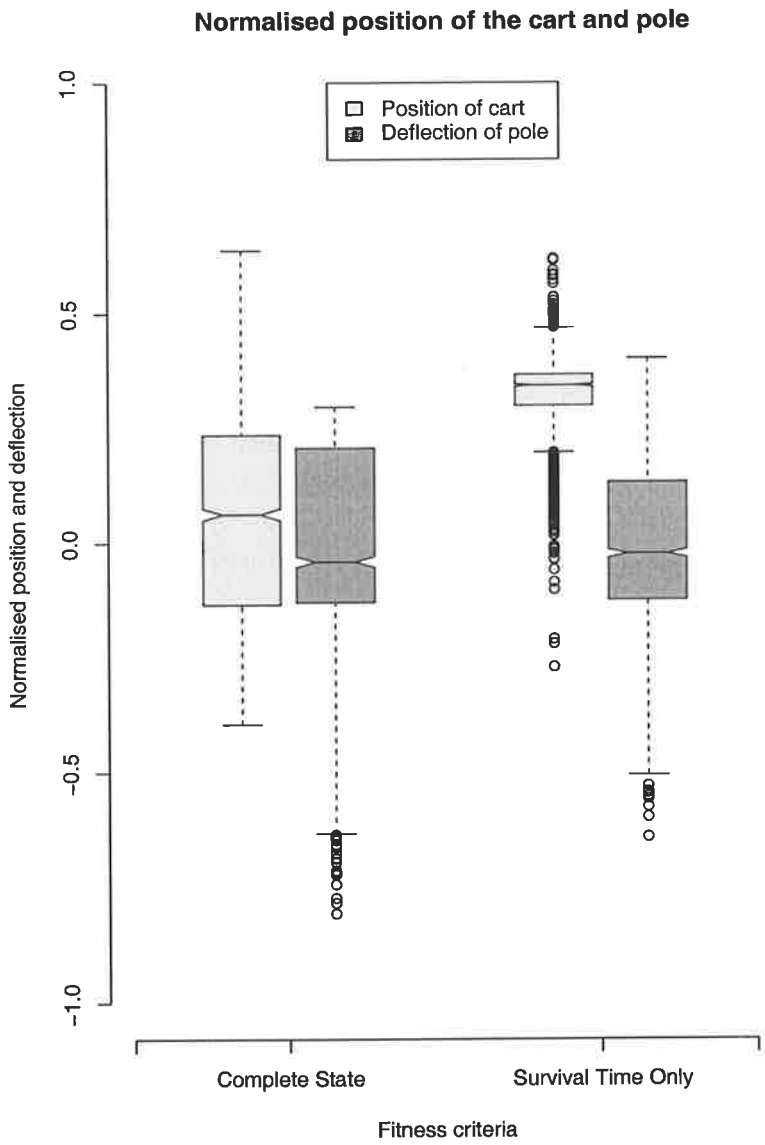


Figure 4.13. *A comparison of the cart position and pole deflection over the 5.5 hours of simulated time for two runs, one using the complete state of the system, the other using only the survival time of the system for fitness. The median position of the cart is away from the centre of the track when the survival time alone is used for evaluation of the solutions.*

4.4 The Two Pole Problem

The one pole system has been used as a benchmark problem for over 30 years, and many variations of the problem have been introduced to make the task more difficult. All variants remain relatively easy to solve, however. This is in part due to the system being able to be controlled by considering each of the state variables independent of the others. In fact, the system can be effectively controlled by

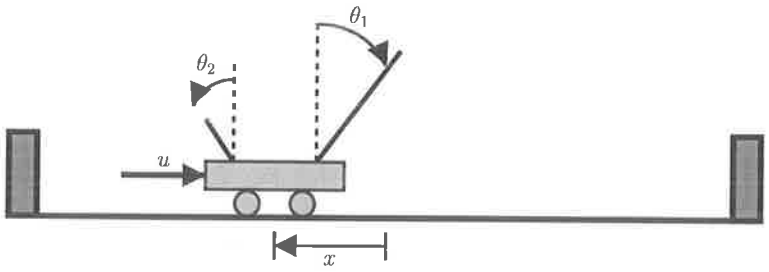


Figure 4.14. *The cart-two-pole system. The problem is similar in nature to the one-pole problem shown in Figure 4.2. The second pole is able to move independently of the first, and a failure state is attained when the cart moves beyond the ends of the track or when either pole declination is greater than largest allowed.*

applying a force equivalent to

$$F = F_{\max} \operatorname{sgn}(k_1 x + k_2 \dot{x} + k_3 \theta + k_4 \dot{\theta}) \tag{4.14}$$

where the constants k_i depend on the choice of constants in the equations of motion,[13] Equations 4.1–4.4, and $F_{\max}$ is the maximal allowed force. If one knows the form of Equation 4.14 then it is quite easy to evolve a controller which can solve the single pole problem.

When a rule set is used as the controller the only rule conditions which need to be considered to solve the system are single attribute comparisons. In particular, there is no need to consider relationships between the attributes. In this section a much more difficult and interesting variant of this form of control problem will be considered.

4.4.1 Description of the Problem Domain

The two pole problem is shown diagrammatically in Figure 4.14. The problem is the same as the one pole problem shown in Figure 4.2: to apply a force to the cart which keeps the poles from falling too far from vertical and maintains the position of the cart on the track.

The state of the cart-two-pole system is described completely by 6 variables. The position and velocity of the cart and pole are denoted the same as for the one pole problem described on page 77, except that the inclination of the poles from vertical is denoted by θ_i where $i \in \{1, 2\}$ corresponds to pole one and pole two of the system.

The control force is again labelled u and is measured in Newtons. The equations of

[13] For the constants used in the previous examples the system can be controlled for at least 5.5 hours by setting $k_1 = 2.22916$, $k_2 = -2.00793$, $k_3 = -11.0774$, $k_4 = -4.94273$ in Equation 4.14. This solution is not unique.

Table 4.4. *Constants used in Equations 4.15–4.18*

Symbol	Description	Value
u	The force applied to the cart	$[-15, 15]$ N
x	The position of the cart on the track	$[-2.4, 2.4]$ m
θ_i	The angle of pole i from vertical	$[-15, 15]$ degrees
M	The mass of the cart	1.0 kg
m_i	The mass of pole i	$(0.0, 0.1]$ kg, $m_i = l_i/5$ kg
g	Acceleration due to gravity	-9.8 m/s^2
l_i	Half length of pole i	$(0.0, 0.5]$ m, $l_1 = 0.5$ m
μ_c	Coefficient of friction of the cart on the track	0.0005
μ_{pi}	Coefficient of friction of pole i's hinge	0.000002

motion describing the cart-N-pole system are [236]:

$$\ddot{x} = \frac{u - \mu_c \operatorname{sgn}(\dot{x}) + \sum_{i=1}^{N} \tilde{F}_i}{M + \sum_{i=1}^{N} \tilde{m}_i} \tag{4.15}$$

$$\ddot{\theta}_i = -\frac{3}{4l_i}\left(\ddot{x}\cos\theta_i + g\sin\theta_i + \frac{\mu_{pi}\dot{\theta}_i}{m_i l_i}\right) \tag{4.16}$$

where $\tilde{F}_i$ is the effective force of the i-th pole on the cart,

$$\tilde{F}_i = m_i l_i \dot{\theta}_i^{\,2} \sin\theta_i + \frac{3}{4} m_i \cos\theta_i \left(\frac{\mu_{pi}\dot{\theta}_i}{m_i l_i} + g\sin\theta_i\right) \tag{4.17}$$

and $\tilde{m}_i$ is the effective mass of the i-th pole,

$$\tilde{m}_i = m_i\left(1 - \frac{3}{4}\cos^2\theta_i\right) \tag{4.18}$$

and the parameters used in Equations 4.15–4.18 are shown in Table 4.4. The system is integrated numerically by using a fourth-order Runke Kutte method [39, page 112] with a time step of 0.01 seconds.[14]

The multiple pole problem was introduced by Wieland in 1990 as a reinforcement learning problem [236]. It has recently been used as a difficult benchmark problem for several evolutionary neurocontrol studies [72, 228, 106, 96, 97, 98, 172]. The previous studies have compared different methods of evolving artificial neural networks and can be divided into 4 different approaches:

1 Direct Evolution of artificial neural networks [236].
2 Evolutionary programming evolution of artificial neural networks [72].

[14]The decrease in the value of the time step reflects the increased precision required for this problem and is consistent with practice in the literature eg. [236].

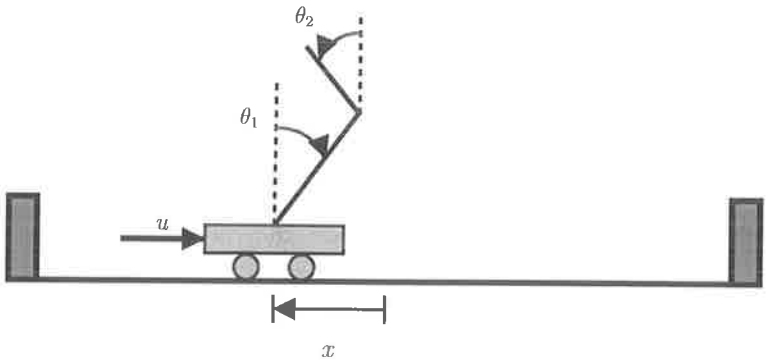


Figure 4.15. *The jointed pole problem consists of a single jointed pole. This problem is more similar in nature and difficulty to the classic one-pole problem of Figure 4.2 then the more difficult two-pole system of Figure 4.14*

3 Cellular encoding approaches to the evolution of artificial neural networks [228, 106].
4 Symbiotic, adaptive neuro-evolution (SANE) algorithm [96, 97, 98, 172].

There are no references to traditional reinforcement learning methods being applied to the two-pole problem, nor to other evolutionary methods.

Equations 4.15–4.18 look similar to the dynamics of the one-pole problem, Equations 4.1–4.4. The complication in the two-pole problem stems from the effects of the relative lengths of the poles on the dynamics of the system. In particular, Equation 4.16 shows that for a given cart acceleration, $\ddot{x}$, the angular acceleration of a pole is greater the closer the pole is to vertical and is inversely proportional to the length of the pole. This means that the shorter pole will accelerate faster than the larger one. When the larger pole is tilted before it can be brought upright, the smaller pole must be tilted to a larger angle. This requires the learning algorithm to learn to take the system further away from equilibrium in order to maintain the system in balance.

It is possible to derive the region of controllability for the two-pole problem [111, 236]. That is, the region of the state space within which the system can still be brought to equilibrium by the application of some allowable force. The important characteristic of this region is its response to changes in the relative lengths of the poles. The system is not able to be controlled when the poles are of equal length unless it starts with the two poles at an identical inclination.[15] As the relative lengths of the poles becomes closer the system's region of controllability quickly becomes narrower.

Another pole problem which is often used in the literature for machine learning is

[15] In which case the two poles will act identically in simulation and the system will be equivalent to the one-pole problem. Experiments on real systems will obviously not be able to exploit such a measure!

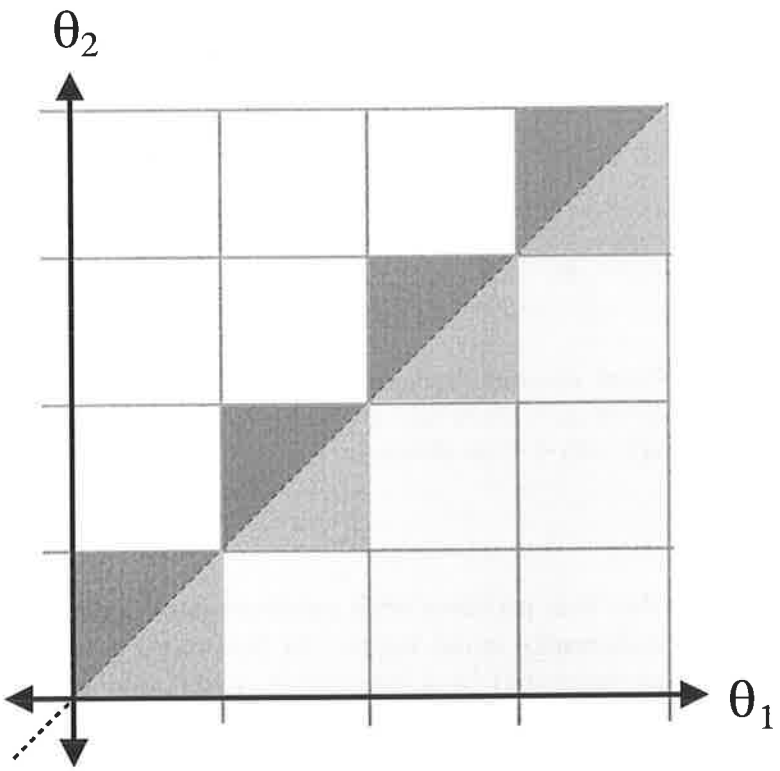


Figure 4.16. *A representation of the relation $\theta_1 < \theta_2$ by boxes. The light and medium grey area shows the values which satisfy the relation. The medium and dark grey areas show the consequences of under or respectively over estimation of the relation depending on whether the corresponding box is selected as belonging or not to the relation.*

the "jointed pole" problem,[16] shown in Figure 4.15, [236, 72, 62, 196]. This problem is also solvable only when the lengths, and therefore the natural frequencies, of the pole before and after the joint are sufficiently different. However this problem is much easier to solve than the two-pole problem because any control signal which moves the poles closer to vertical is always the correct control signal to apply [236]. This allows a strong selection scheme which operates on a small number of good controllers to quickly evolve a solution to the problem. Saravanan and Fogel were able to solve this problem with a population of 100 neural networks in 10 generations, whereas the two-pole problem required 800 generations to evolve a solution [72]. The jointed pole problem will not be used in this thesis.

From the above observations it is clear that to successfully control the two-pole problem it is necessary to be able to make decisions based on the relative angles of the two poles. This means that a relationship between two of the state variables must be discovered which can then be used to partition the state space into appropriate control options. The rule conditions used in the one-pole problem will be insufficient for this task, as they can only consider single attribute comparisons in dividing the

[16]The corresponding system of differential equations which describe this system are far less succinct to write down, and are provided in Wieland [236].

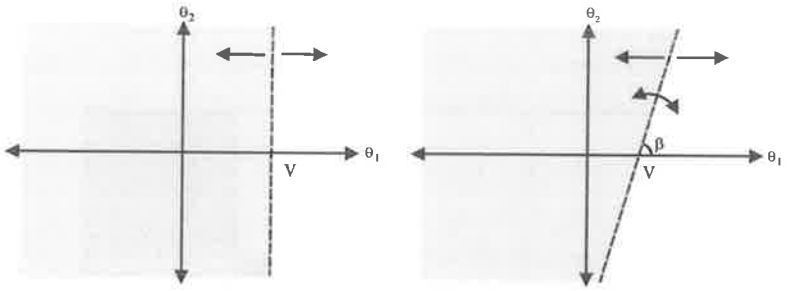


Figure 4.17. *A relational decision boundary. In the first diagram the single attribute comparison $\theta_1 < V$ is shown as a projection on the θ_1-θ_2 plane. In the second diagram the simple relation $\theta_1 + tan(\beta) \cdot \theta_2 < V$ is shown.*

control space.

It is conceivable to solve this problem with attribute-only learning by discretising the state space into sufficiently small boxes. In this case, a relationship between the angles could be approximated by a large number of single attribute comparison rules, as illustrated by the box approximation of the relation $\theta_1 < \theta_2$ in Figure 4.16. The boxes can be aggregated to form rules, however they can only form a rough approximation of the decision boundary which the relation describes. Even when the boundaries of the boxes are not regular the accuracy of the method will be decided by the number of boxes used to describe the decision boundary. The corresponding rule set or decision tree would be unwieldy and difficult to comprehend and to induce. It is also unlikely to be able to solve the problem as the domain is a very sensitive control problem.

For this reason it is necessary for the controller to be able to represent a relationship between some of the system attributes.

Representing Relationships

In addition to the state attributes, the rule set model is provided with the means of being able to represent a relationship between θ_1 and θ_2. Looking at the projection of the state space on the $\theta_1\theta_2$-plane the hypothesised relationship will allow for any linear decision boundary on that projection. The gradient of the decision boundary is explicitly used and evolved alongside the other parameters associated with the rule set.

The learnt relationship is coded as the parameterised relation:

$$\text{Relation}(\theta_1, \theta_2, \beta) = \theta_1 + \tan(\beta) \cdot \theta_2 \tag{4.19}$$

where β is an evolved constant such that $\beta \in (-\pi, \pi)$. The difference between the rule $\theta_1 < V$ and $\text{Relation}(\theta_1, \theta_2, \beta) < V$ is shown graphically in Figure 4.17 for some fixed V, β. The straight arrows in Figure 4.17 shows the effects of changing

the comparison constant V, and the curved arrows show the effects of changing the relationship parameter β. The parameterisation of the relation is to account for the fact that the relationship between the angles is not known.

In addition to the 6 state variables two new relations are added. The second relation is the reflection of the one described above with θ_1 and θ_2 exchanged and another relation parameter, or angle, α added to the evolved parameter vector. The relation is then treated identically to the other 6 state variables in the rule set. That is, for a given observed state the relationship is evaluated and the value used in the same way as the other state variables. The two relations add a total of 6 parameters to the parameter vector, corresponding to the 2 partitions of the value sets[17] which the relations will be compared against, and the angle that parameterizes the relation in (4.19).

Discussion of the Relational Representation

There are clear limits on the kinds of relationships the SASME algorithm can learn for a given problem. Specifically, it learns those relationships the user specifies for the current task. The algorithm does not create new relationships, nor does it build relationships by composing one with the other. Such functionality may be able to be incorporated, but the central theme of understanding the controllers which have been discovered means that the current approach is better suited to the current task.

The relationship used by the SASME algorithm to solve this task highlights the utility of using relational over propositional logic for such tasks. Relational representations have a larger hypothesis space, and can represent more objects. These objects may contain the solution to the problem at hand.

4.4.2 Task Descriptions

Two popular tasks from the literature are attempted with the relational rule set described above. Both tasks were attempted by means of a sequential learning task of increasing difficulty. The first involves balancing the system from an increasingly large range of initial conditions, and the second involves learning to balance the two pole system with poles of increasingly similar lengths.

Enlarging the Initial Positions

A number of initial condition strategies are pursued in the literature. The experiment attempted here use a similar initialisation strategy to that used by Polani and Miikkulainen [172, 173]. The cart-pole system is evaluated from 10 random

[17] Where the number of value sets for each allowed rule comparison is set to 3.

initial states and best of generation is tested from the edges of the initial region for a number of time steps. In the experiments reported here this evolution proceeds by starting with:

$$\begin{aligned} x &\in [-0.05, 0.05] \text{ m} \\ (\theta_1, \theta_2) &\in [-0.05, 0.05] \text{ rad} \end{aligned}$$

Once a solution is found which can balance the system for the required time the intervals are increased by 0.05m and 0.05° respectively until the system is able to balance the cart-pole problem from:

$$\begin{aligned} x &\in [-0.2, 0.2] \text{ m} \\ (\theta_1, \theta_2) &\in [-0.2, 0.2] \text{ rad} \end{aligned}$$

for 100,000 time steps. This task is meant to establish that the controller can find a solution to the two pole problem which is general enough to balance the system from a random initial position in the allowed range.

Changing pole lengths

As noted earlier, the more similar the pole lengths the more difficult the problem is to solve. The algorithm is required to balance the two-pole system initially with pole lengths[18] of 0.1 m and 1.0 m. The length of the second pole is then increased by 5% and the system has to be balanced again. The increments continue until the length of the short pole is just 10% less than that of the long pole - an extremely sensitive control problem [236, 72].

A note on Incremental Evolution

The difficulty of the two pole problem has meant that all of the learning approaches cited have used an incremental strategy for balancing the system with poles of near equal length. The incremental approach requires the learning algorithm to solve a series of increasingly difficult problems. Incremental evolution appears to be a good way of tackling complex problems where it is possible. If the first task is denoted t_1, and the next t_2 and so on, then the evolutionary algorithm solves the sequence of tasks:

$$t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow t_n$$

where t_n is the task which is of interest. For the initial condition problem mentioned above, t_i corresponds to the task of balancing the cart-pole system from a random initial condition in the range:

$$\begin{aligned} x &\in [(i \cdot 0.05), (i \cdot 0.05)] \text{ m} \\ (\theta_1, \theta_2) &\in [-i \cdot 0.05, i \cdot 0.05] \text{ rad} \end{aligned}$$

[18]Note that Table 4.4 shows the values of the half-pole lengths which are the values used in Equations 4.15–4.18.

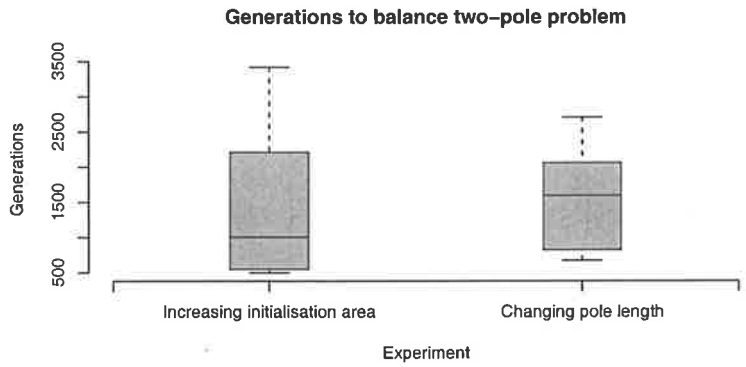


Figure 4.18. *Comparison of 10 different evolutionary results for solving the two-pole problem from a large initial area and with poles of similar length.*

where $i \in [1, 2, \ldots, 4]$. For the pole lengthening problem, the half-length of the second pole in task i will be

$$l_2 \;=\; 0.05 \cdot (1 + 0.05)^{(i-1)} \text{ m}$$

where $i \in [1, 2, \ldots, 47]$ and in task t_{47} the length is reduced to 0.45.

Self-adaptation of the evolutionary parameters should provide a natural way for the evolutionary approach used here to adapt its search performance to the changing nature of the problem being solved. The evaluation environment is simply changed from t_i to t_{i+1} when task t_i is solved, without any other modification to the algorithm.

Another algorithm used for evolving solutions to sequences of tasks is the δ-Coding extension of the GENITOR genetic algorithm [234, 157]. In this approach the solution to a task is used as the template to form a new population which is then charged with solving the next, incrementally more challenging task [96, 97, 98].

4.4.3 Results

The number of generations to solve the two pole problem was significantly more than that required for the easier one pole problem. A generation size of 100 rulesets was evolved for this problem, with typical runs requiring up to 4,000 generations to solve the problem. The evolved rulesets operated on 3 partitions of each of the continuous state space variables, as was done in the previous section. The rule consequences were limited to one of 4 real values, requiring 4 evolved parameters in the parameter vector.

Figure 4.18 shows the number of generations required to solve the two incremental problems described in the previous section. Each run had to balance the two-pole system for 500,000 time steps, or almost 83 minutes of real time. The graphs are constructed from 10 independent runs, and so are indicative of performance.

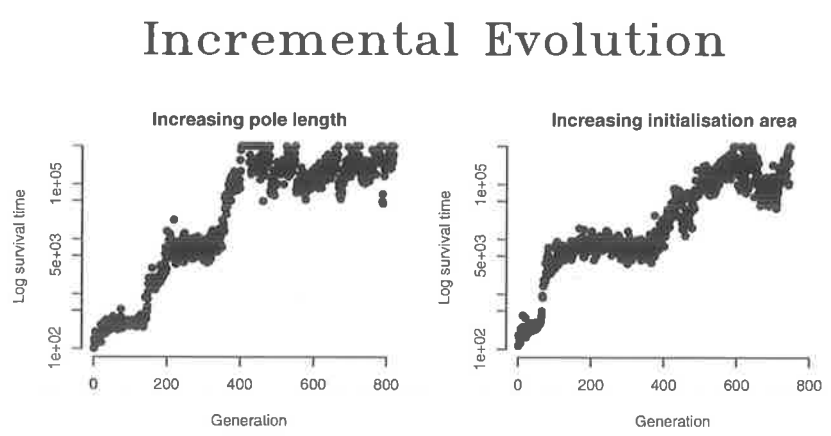


Figure 4.19. *Incremental evolution produced by increasing the pole length and the initialisation area of the two-pole problem. The controller had to balance the system for 500,000 time steps, or 83 minutes, before the problem was made more difficult.*

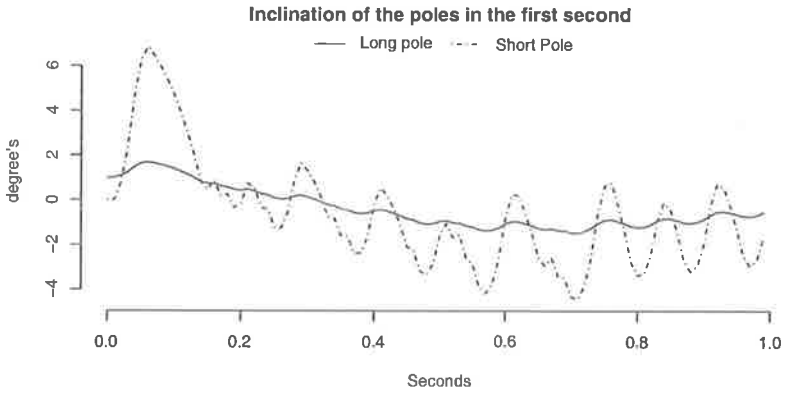


Figure 4.20. *Tracing the inclination of the first (long) and second (short) pole showing the controller performing the maneuvre of making the state space worse by swinging the second pole out past the first so that it can bring the system back in balance.*

A typical run of the incremental evolutionary tasks is shown in Figure 4.19 for each of the two problems. The graph shows that the evolutionary system is able to learn the initial easier task and use this as a springboard to learn to control the next more difficult task.

As was mentioned earlier, the controller has to learn to take the system away from equilibrium in order to maintain control of the system. When the initial conditions have the longer pole further from equilibrium than the shorter pole, the system has to learn to swing the shorter pole out to a greater inclination than the longer pole before starting to move the longer pole back towards upright. Figure 4.20 shows the rule set moving the shorter second pole off balance further than the longer and slower first pole before moving them both back towards equilibrium in the first second of the simulation.

Rule sets

Rule sets for the two pole problem are more complex then those evolved for the one pole problem. However, the rule sets are still able to convey information about how they are controlling the cart pole system.

The rule in Figure 4.21 is able to control the cart two-pole system for 500,000 time steps with a small pole length 10% that of the longer pole. The figure displays the symbolic value sets and action consequences on the rule nodes. The percentages to the right of the rule nodes are the percent of times the rule is triggered in balancing the system for 120,000 time steps. Rules to the left, or rules with only one parent, are if-not rules. Rules connected to the bottom right of a node are exception rules.

Rule **1** uses the evolutionarily tuned relationship defined in Equation 4.19. The angle of the evolved decision boundary is $-61°$, corresponding to a gradient of -1.8 in the projection of the state space on the $\theta_1\theta_2$-plane. The value set for the relation essentially defines the rule to be true when it is greater than 0. The consequence of the rule is to push the cart strongly to the right, which has the effect of leaning the poles to the left. All of the other rules in the rule set push the cart to the left with varying forces, having the effect of moving the pole to the right. When the long pole (θ_1) is right of upright rule **1** will move the system to the right until the short pole is 1.8 times as far to the left of vertical as the large pole is to the right. Only then will the short pole be brought back to equilibrium (unless Rule **2** becomes activated, which happens more often than not, when the system will be moved back into equilibrium in response to the short pole excessive velocity to the left). When the long is left of upright the other rules in the rule set will continue to move the system right until the small pole is 1.8 times as far to the right as the long pole is to the left. Then rule **1** will apply and move the short pole back into equilibrium.

4.4.4 A non-Markovian Variant of the Two-Pole Problem

A non-Markovian extension to the two pole problem was introduced and solved by Gruau et al. in 1995 by a cellular encoded neural network [106], and later by the SANE method of evolutionary neuro-control [98, 172]. The non-Markovian extension involves solving the problem with no velocity information.

Removing velocity information adds a hidden state to the problem, which the controller must be able to compute in order to control the system. The resulting system is no longer Markovian, since a correctly computed control force for a given state position will depend on velocity information which will in turn depend on the controllers previous actions. The state presented to the controller no longer contains sufficient information to solve the problem. The neural network approaches solve this problem by evolving a recurrent neural network. The recurrent links make it possible for the network to calculate the velocity in some form internally.

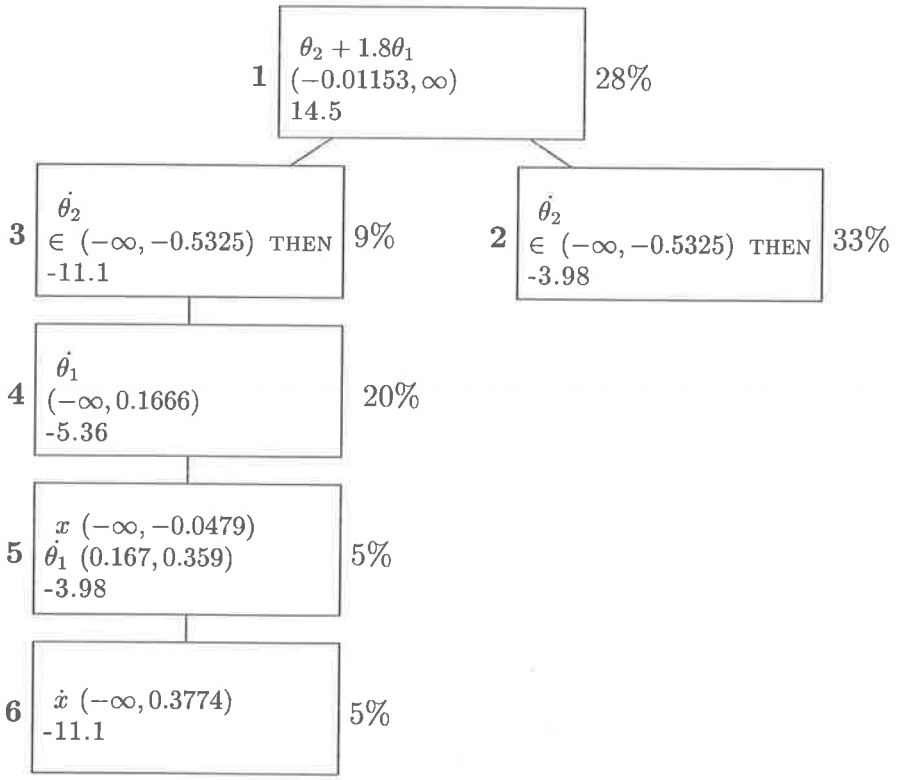


Figure 4.21. *A rule set for balancing the two-pole problem.*

A Recurrent Rule Set Model

The rule set approaches presented previously have only been able to make control decisions based on the current state of the system. The rule set model with relations can be further extended to allow for decisions based on previous actions. This is easily possible in the SASME framework by adding some new parameters which allow the controller to have a *memory* of previous actions.

To address the problem of controlling the system without velocity information a new state variable is introduced. The new state variable is the discounted sum of the controllers past actions. The discount rate is evolved. The new state is equal to:

$$\sum_{t=0}^{N} \gamma^t \cdot \text{Output}_t \tag{4.20}$$

where Output_t is the rule sets output at step t, N is the horizon and $\gamma \in (0, 1)$ is the evolved discount rate. A number of different possibilities are available for setting N. Setting $N \leftarrow \infty$ corresponds to the discounted model output since the trial began. Setting N to some finite number corresponds to considering the discounted sum of previous actions up until some finite horizon of past actions. The evolved discount

Table 4.5. *Comparison of enforced sub-population (ESP) SANE and cellular encoding on the two-pole problem with velocities [98]*

Method	Evaluations	Population Size
CE	840,000	16,384
ESP	169,466	1,000

rate γ can be interpreted as the discount rate, as an interest rate,[19] as a probability of something relevant happening in the past or as a mathematical trick to bound an infinite sum in the case of all previous actions being considered.[20]

Three new state variables were added to the model to replace the three velocity variables. This was not done to try and model the three velocity variables, rather to maintain a constant number of state variables in the model. Three arbitrary different values of N were chosen, $N = \infty$, $N = 50$, and $N = 10$. The three new state variables added three new parameters to model, γ_i, $i \in \{1, 2, 3\}$, all of whose values were kept positive and less than one, $\gamma_i \in (0, 1)$.

The non-Markovian variant of the two-pole problem is the hardest problem dealt with in this thesis. Gruau *et al.* used a population of 16,384 cellular encoded neural networks distributed across 64 processors on a super computer to solve the problem only once in 51 generations for 100,000 time steps. More recently a variant of the SANE approach to neuro-control has reduced the number of function evaluations by a factor of 5 [98], although generalisation suffered. Table 4.5 summarizes the function evaluations of the two methods.

Gruau *et al.* show that it is possible for the controller to control the system for 100,000 time steps without needing to calculate the velocities. It can do this because the system can be controlled by oscillating the poles backwards and forwards for 100,000 time steps. For the neural network study, Gruau *et al.* devise a lengthy fitness evaluation method which biases the networks towards producing controllers which move the system back to the centre of the track with the poles in an upright position. One reason they can do this is that the neural networks produce a continuous output which is capable of moving and keeping the system in a stable state. The rule sets used in this chapter are producing a piecewise constant controller which has only a small number of levels of force with which it can try and control the system. This prevents the controller from maintaining an equilibrium position since it must always apply a non-zero force to the cart and poles.

The principal difference between the continuous and discrete output controllers is the shape of the resulting control functions over time. In some domains the control

[19] Literally. The closer γ is to 1 the more *interesting* the model finds historical actions at any particular time frame in the past!

[20] Similar to how traditional reinforcement learning interprets the discount of future rewards in an infinite-horizon model [130].

Table 4.6. *Summary of results for evolving rule sets for the two-pole problem without velocity information for 120,000 time steps.*

Generations					Evaluations of
Min	Quartile 1	Median	Quartile 3	Max	median
4902	8343	11724	13678	30162	1,172,400

space must be discrete, which is the case for the problem which the SASME algorithm solves in this section. For other problems there may be no discrete controller output which will suitably control the system; this may be the case in a problem involving following some signal such as an audio signal. If the audio output is constrained to too few discrete values then the output may be very poor. For the cart-pole problem, the SASME algorithm is able to solve the problem with the extra constraint that it uses only a set number of discrete values for the controller output. It is the problem domain, and the problem statement, that determines whether a continuous or discrete controller is appropriate.

A number of different experiments were run to see if the rule sets could be evolved to balance this problem. All experiments were computationally expensive on the available hardware.

4.4.5 Results for the non-Markovian Two-Pole Problem

Due to the computational expense of running algorithms to solve this problem only a small number of runs are reported here. An evaluation of the performance of the algorithm in terms of efficiency, or generations required to find a solution, can not be performed. The significant point to be made in this section is that the SASME algorithm can evolve a rule set model to solve this difficult non-Markovian problem which previously has only been solved by neuro-control approaches.

Table 4.6 summarizes the results for 10 independent runs of the evolutionary system. All runs were executed from a single initial position and had to balance the cart-pole system for 120,000 time steps, or 20 minutes of simulated time, in common with previous studies [172, 173]. All runs were executed until they had found a solution.

Compared to the neuro-control methods the algorithm appears to have a higher computational requirement. Some things to note about the computational requirements:

1 The number of evaluations does not represent the computational time taken by the algorithm. This is because the computational time required depends on how long the cart-pole system remains valid in an evaluation. Figure 4.22 shows the survival time at each generation of the best in solution for one of the 10 runs. Note that the corresponding earlier figure (4.19) had the log of the survival time graphed. The algorithm seems to take many generations to tune the parameters into an appropriate range. This may also be the case for

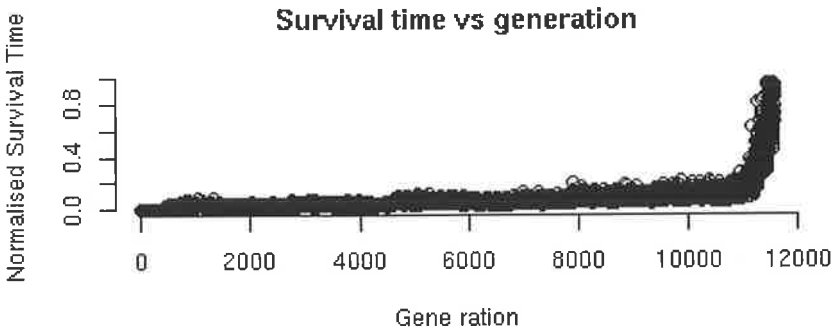


Figure 4.22. *This graph shows the normalised survival time of the best of generation solution during the evolution of a solution for the two-pole problem without velocity information*

neuro-control, and should be accounted for when comparing learning efficiency.

2 The above point raises another issue. The parameters used in the algorithm were not optimised to any great extent. In particular, the setting of the initial ranges of the evolvable parameters such as the value sets used in antecedent containing the discount rate were arbitrarily set to 1000, as this was thought to be much higher than required. The reason for the initial setting was to test whether the evolutionary procedure could learn to evolve a suitable discount rate for use in rules. The likely values that this rate might take were not known, and in fact depend on the evolved forces applied and the discount rate evolved. A large value maintains the generality of the algorithm.

3 Other specifically evolutionary parameters were not adjusted. In particular, if the execution time is dependent on the algorithm moving parameters into optimal ranges then the number of evaluations may benefit from an increase in the population size.

To establish whether the algorithm was learning to control the system effectively a longer run of 240,000 time steps, or 40 minutes, was conducted and a summary of the state space of the cart-pole system using the evolved controller is shown in Figure 4.23. The figure shows the range of the state variables over 50 second intervals for the 2,400 seconds of simulated time that the rule set controlled the system for. It appears that the controller has effectively learnt to compute and use the velocity information to control the cart-pole system. The behaviour of the velocity variables in Figure 4.23 appears to be under control and not oscillating. A later run was successfully evolved to balance the system for 500,000 time steps, or over an hour of simulated time, in just over 5,000 generations.

Evolved Rule sets

Figure 4.24 shows a rule set for controlling the non-Markov version of the cart pole problem. The evolved rule set uses the discounted sum of the previous outputs of

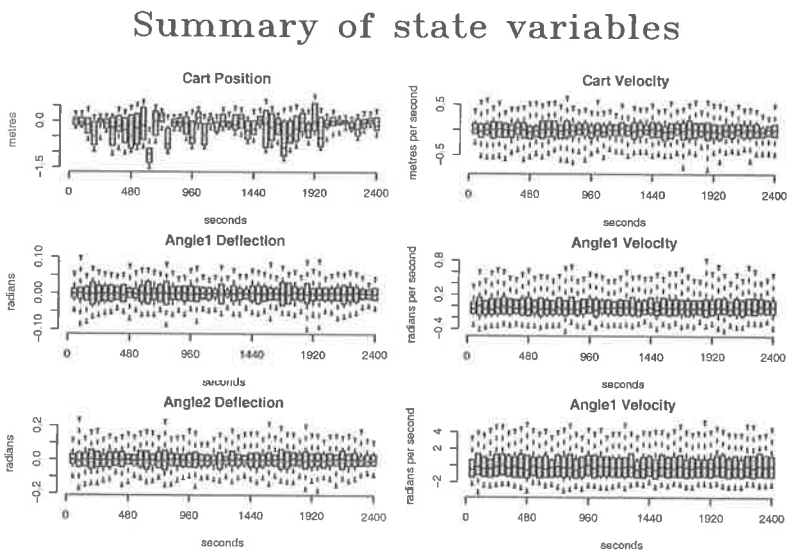


Figure 4.23. *Analysis of the controlled cart two-pole system. The state space summary shows no evidence of undamped oscillations which would arise from the controller not computing the velocities.*

the rule set as a surrogate for the velocity information which is no longer available to the rule sets. Rules **1** and **3** use the discounted sum of the previous 10 actions to decide on future actions. The discount rate used by the controller is 0.57. Rule **1** has the effect of moving the cart-pole system to the left when the discounted sum of previous actions is to the left. However, this rule must be true before the exception rule, Rule **2**, is tested. Rule **2** says that when the large pole is less than -0.6 times the inclination of the small pole the system should be moved to the right. The combined effect of the rules is to move the system to the right whenever the discounted sum of previous actions is left and when the large pole is not at a greater inclination to the left than the small pole. This combination is used 37% of the time. When the system has become very over-balanced then rule **1** will move the system to the left regardless of the sum of previous actions.

The discounted sum of previous actions is being used by the rule set as a surrogate for velocity information. The evolutionary system has evolved a *recurrent* rule set.

4.5 Discussion and conclusions

The cart-pole problems are an example of dynamic control problems where the controller must learn to control a number of state variables with only a single input and with sparse feedback. The single pole problem is a well known test problem which has been used extensively to demonstrate a wide variety of automated learning techniques.

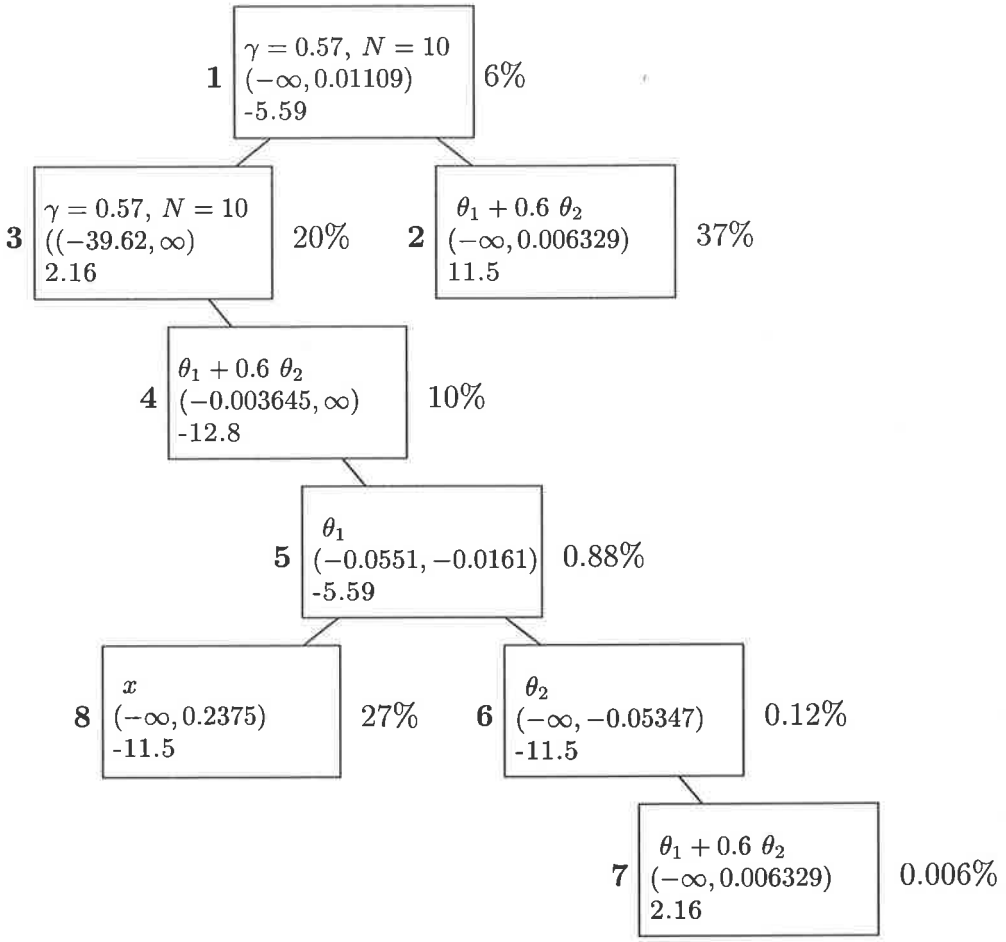


Figure 4.24. *A rule set for balancing the two-pole problem with no velocity information.*

The SASME framework evolves rule sets which operate and produce symbols to control this system. The symbiotic evolution of the parameters give meaning to the symbols used in the rule set. This allows the efficient self-adaptive ES parameter optimisation procedure to optimise parameter values while the self-adaptive discrete mutations build the rule set model. The resulting system produces discrete rule models which can be understood entirely in terms of the symbols they operate on. These structures can be examined for the knowledge they contain. The model which is learnt is not a black box.

The two-pole problem is a much more difficult variant of the single pole case, which can be made even more difficult by excluding velocity information. To solve the two-pole problem the discrete model structure has to be extended to consider simple relationships between the inclinations of the two poles. A relationship with a single parameter is added to the allowed predicates of the rule system and the evolutionary process evolves rule sets which utilize this relationship to solve the two-pole problem under a variety of conditions.

When the problem is made incrementally harder by lengthening the short pole or extending the range of initial conditions the self-adaptive evolutionary process is able to adapt the existing solutions to solve the more difficult problems. Self-adaptation has been shown to effectively be able to raise and lower the self-adapted mutation rate [14], and this ability is exploited in the so-called incremental evolutionary problems posed.

To the authors knowledge, no-one has produced a controller which was not a neural network to control any variant of the two-pole problem.

The value of non-neural network controllers lies in their explicit structure. A controller which controls a system using a representation which is chosen to convey information on some facet of the problem can be instructive and has the potential to provide new knowledge about the problem domain. An explicit control structure can be understood in part, and so the behaviour of the controller can be understood to some extent.

The two-pole problem without velocity information represents a very challenging problem for this system. The neural network controllers use recurrent links to enable the non-Markovian variant to be controlled. The discounted sum of previous actions is equivalent to a recurrent link in a network. The recurrent rule set methodology which results from this addition is able to solve the non-Markovian problem. To do this, it learns to use the recurrent information as a proxy for the velocity information which is no longer available for the problem.

Chapter 5

Evolutionary Learning II: Elucidation of Ecosystem Processes

> Everything should be made as simple as possible, but not simpler.
>
> Albert Einstein

Evolutionary methods have recently been applied to a range of supervised knowledge acquisition tasks [129, 93, 67, 64, 109, 110, 171, 154, 224, 166, 65]. Freitas provides a survey of data mining with evolutionary methods [86], and Kovacs *et al.* provide an extensive (478 papers) bibliography of classifier system papers [137]. There are several reasons for using an evolutionary approach for data-mining:

1. Traditional deterministic rule induction methods, such as ID3 [181], search for decision trees with essentially a greedy hill climbing algorithm guided locally by information theoretic measures [64, 166, 65, 86]. Evolutionary rule induction systems take a global view of the rule-set generation problem.
2. There is little flexibility in the traditional rule induction systems, and obtaining different rules with nearby accuracy is difficult [64]. Evolutionary approaches often provide a different hypothesis on each run from the same data, which is natural for many learning problems where there is redundancy and interactions between variables.
3. The representation that the evolutionary search uses is easily modified to suit either domain knowledge or questions about the data. The fitness function is one mechanism whereby evolutionary search can be easily directed towards finding more interesting results to many problems.

There are probably two main reasons to perform an inductive learning task. The first is to correctly classify new situations. If an inductive system is applied to learning stock market information, the principal motivation is likely to be to use the system to correctly forecast future market movements. The other main reason is elucidation. When inductive learning tasks are applied to epidemiological data the most common purpose is to understand what is causing the phenomena the data is measured from.

Naturally, the two are not mutually exclusive. In particular, a blackbox model which successfully provides predictions of future stockmarket performance is unlikely to be trusted when it goes against our expectations. The system will be more trusted if the basis of its predictions are better understood.

This is one of the motivations for using a rule set as the representation for knowledge. IF-THEN rules are a high-level, symbolic knowledge representation which aids knowledge discovery [65].

This chapter addresses the issue of whether the SASME model can learn comprehensible rules in noisy real world data. The ecological data used in this chapter is typical of many real world domains[1] where measured data is inaccurate, incomplete, noisy, contains many interactions, and contains unevenly distributed outcome variables which require obscure relationships to characterize correctly. Another domain where this is important and where genetic learning has been applied is in discovering rules in clinical research databases [123].

It is widely stated that by performing a global search, evolutionary methods are more effective learners of rule based models than the usual greedy local search methods [64, 166, 65, 86]. The veracity of this statement is determined by empirical comparisons. This chapter does not undertake to perform an exhaustive comparison between the SASME induced rules and induced rules from other algorithms. However, a case study with CART will be performed to establish that the system is a competitive inductive learning algorithm.

One issue which will not be addressed by this work is that of selecting appropriate error measures for ecological time series models. There is much to criticize in the usual Root Mean Square Error (RMSE) measurement, principally its unit-specific nature [8]. Alternative unit-free measures exist which perform essentially the same task as the RMSE, [8]. However, these measures are appropriate for the reliable comparison of a method which is applied to several different time series, their utility is low for the application considered here where the comparisons are always with the same time series. Of more interest to ecological studies are measures which allow some distance between actual and predicted components of two series to be more fairly evaluated. Such measures would solve the problem of a model predicting a high point in the time series one time step early being unduly punished by the RMSE measurement. The literature on time series comparisons measures is vast, and includes some practicable algorithms for performing comparisons such as those described, [34]. However, all other studies on the datasets considered in this chapter have used the RMSE error measurement, [230, 238]. In Section 5.1.4 a modified version of the RMSE measure is used, demonstrating that other error measures are easily incorporated in the evolution of a SASME model.

The next section describes the data mining task used in this chapter and the problems associated with it. Two distinct problems are then addressed: the prediction of chlorophyll-a concentration and the prediction of the dominant algal species.

[1]Such as those found in the UCI data repository, [26].

Table 5.1. *A short glossary of some ecological terms used in this chapter [223]*

Term	Definition
Algae	Prokaryotic and eukaryotic photosynthetic organisms with chlorophyll-a and other photosynthetic pigments releasing O_2. Plant body unicellular, colonial, filamentous, siphoneous or parenchymatous, never with roots, stems or leaves. Not a natural group, but the word is useful in many contexts.
Blue-green algae	Cyanophyceae, Cyanobacteria. Prokaryotic organisms with chlorophyll-a and phycobilins. Unicellular, colonial, or filamentous. Occur in fresh- and salt-water, in soils and as nitrogen-fixing symbionts. Often of interest in man-made aquatic environments due to toxins produced by some species, eg. *Microcystis* spp in Myponga reservoir, South Australia.
Chlorophyll-a	Green pigment involved in photosynthesis. Chlorophyll-a is the primary photosynthetic pigment in all those organisms that release oxygen ie. all plants and all algae including the blue-green algae.
Colonial	The vegetative form of many species of algae in which the sister cells are connected in a group to function as a unit.
Eutrophic	State of a nutrient rich lake in which the hypolimnion (cold, lower layer) becomes depleted of oxygen during the summer by the decay of organic matter falling from the epilimnion (warm, upper layer). A eutrophic lake is usually shallow, with much primary productivity.
Filamentous	Generally a chain (unbranched or branched) of cells joined end on end.
Microcystis spp	Colonial species of blue-green algae.
Oscillatoria spp	Filamentous species of blue-green algae.
Phormidium spp	Filamentous species of blue-green algae.
Stratification	Thermal layering of lake water bodies
Succession	The sequence of communities which replace one another in a given area, until a relatively stable community (ie the climax) is reached, which is in equilibrium with local conditions.

5.1 Mining Data from an Aquatic Ecosystem

Table 5.1 is a short glossary of aquatic ecology terminology used in this chapter.

The structure of the problem considered in the previous chapter was such that the model had to interact with the environment in order to receive the next input vector, Figure 4.1, page 76. The model's output affected the trajectory of the system. In the current chapter, the problem being considered is a supervised learning problem where there is no interaction between the model and its environment, as shown in Figure 5.1. In this case the next input will not be different regardless of the models output.

This problem is often referred to as *data mining* [86]. The goal is to discover regularities or patterns in the information supplied to the model.

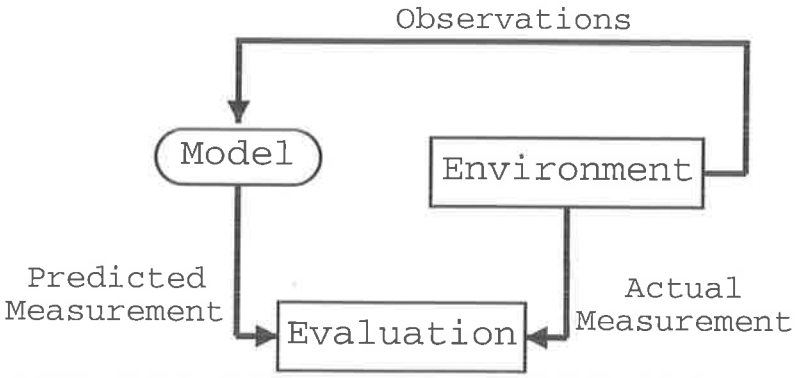


Figure 5.1. *A block diagram showing some of the components involved in supervised learning tasks.*

Research in modelling ecosystem dynamics can be divided into *deductive* and *inductive* approaches.

Deductive approaches use traditional mathematical techniques, like ordinary differential equations, and a detailed understanding of ecosystem processes to produce a model capable of simulating future ecosystem behaviour. Examples of deterministic aquatic ecosystem models which address lake eutrophication modelling include AQUAMOD, MSCLEANER, SALMO and SALMOSED [187, 186, 192]. The accuracy of such models depends upon the correctness and completeness of the description of ecosystem processes upon which they are based. The first three models mentioned model the nutrient cycles and food web interactions of algae and zooplankton in lakes in order to predict algal and zooplankton abundance. SALMOSED, by comparison, extends the nutrient cycles to the sediment allowing it to simulate impacts of external and internal nutrient loadings to algal and zooplankton abundance. Deductive models have significant shortcomings when the knowledge upon which the model is based is incomplete or incorrect, as comparisons of SALMOSED and the other models demonstrate [192].

Another issue with deductive models is the problems associated with data availability, accuracy and completeness. It is well known from normal ordinary differential equations that modifying initial conditions or parameters a small amount can sometimes result in completely different dynamic behaviour (eg. [39, page 435]). It has been suggested that deductive modelling of eutrophication in aquatic ecosystems is best suited to long term, strategic analysis of lakes [225]. By contrast, inductive modelling appears more appropriate for short term predictions.

Machine learning based inductive methods, including SASME and ANNs, do not require information about the ecological processes being modelled. Instead, they require data which is exemplary of those processes. From the data, patterns and relationships are learnt and used to make future predictions about the ecosystem [156]. Assumptions about the distribution of input and output attributes are not usually made. Where there are insufficient data available from an ecosystem, methods based

on machine learning may fail.

There are growing databases of ecological information from a wide variety of sources which cover a staggering range of natural phenomena. Freshwater aquatic ecosystem data are being collected around the world as the importance of monitoring and being able to better understand the changes that occur in freshwater systems like lakes and rivers becomes more apparent. The freshwater data used in this study were obtained from Lake Kasumigaura in Japan.

5.1.1 Model Validation and Representation

Inductive models learn by generalising the knowledge they acquire about the ecosystem from the information they are shown. The model that is induced from the available data learns to associate environmental states that have equivalent consequences but which appear differently. The complexity of the raw data is aggregated by the model. A good model is a compressed form of the data. The compression is not lossless, however.

There are many ways that knowledge can be represented in an induced model, including neural networks, rule sets, fuzzy sets, learnt equations and regression trees. Most learning applications in ecology have concentrated on acquiring accurate models of the systems as determined by independent test data. However, a predictive model should provide not only accurate classification but also insight and understanding into the predictive structure of the data [40].

Model accuracy usually means how well the model represents the real world. Whilst this is frequently measured by the predictive accuracy of the model according to some criteria, an accurate model should also make those predictions according to the way the world really interacts. This is assessed through the models performance. A model which accurately predicts the outcome in every situation must have learnt something about the way the world works. The knowledge can also be represented by the model explicitly.

This chapter considers how ecosystem data can be used to induce models which are both predictive and descriptive.

Evaluating models solely on their predictive error ignores any information the model can provide about the relationships it has learnt and how they relate to the causal explanations of ecosystem behaviour. For example, the application of machine learning methods such as neural networks for ecosystem prediction has focussed on minimising the root mean square error of predictions rather than understanding the underlying processes by which the model decides its predictions. This chapter tries to balance both issues by using the evolutionary rule induction method to provide accurate predictive models which have some descriptive power.

Representing knowledge in a transparent manner is of great importance to inductive

modeling. An inductively gained model is a hypothesis derived from the data, and certainly not a proven law for underlying mechanisms. The quality of the inductively gained model can be assessed from the performance on testing. However, in a complex system like an ecosystem it is practically impossible to assure that the testing data are representative. This is due to the infinite number of interactions between variables which may determine the problem, yet are unknown, and the noise and reliability of the data. A model derived from measured data can only generalize to the extent that the data provided accurately represent the universe of possible measurements. So while testing is important, examining the explicit learnt knowledge contained in an inductive model can provide better insight into how accurate the model's representation of that ecosystem is. Improved understanding of inductive ecosystem models may also increase our confidence in the model outputs and gradually turn it into a "grey box".

Evolutionary methods are general optimisation methods which can be applied to a range of representations which are of interest for knowledge discovery. Previous work has used genetic programming to evolve equations relating input and output data for aquatic ecosystems [189, 230], and neural networks. Neural networks learn an equation relating input and output, but the equation is contained in the weights of the networks and cannot easily be examined. The genetic programming induced equations are an example of explicit knowledge representation. The prototypes for the equations can be modified to search for equations of a particular form, and so knowledge of interest can be discovered. The explicit symbolic rule sets evolved by the SASME algorithm introduced in this thesis are another example of an explicit knowledge representation. The type of knowledge representation which is most appropriate to a given problem will depend on the problem and how the information in that problem is best understood. A variety of different representations would appear to be a sound strategy with no a-priori bias.

A number of non evolutionary methods are also regularly applied to ecosystem data, including methods for the production of equations and neural networks [237], rules and more traditional statistical models. Breman's classification and regression tree (CART) method produces decision trees on a continuous output variable (a regression tree[2])[40]. The representation of knowledge employed by CART is quite different to the default hierarchy rule sets used by the evolutionary method presented here. In the previous chapter it was demonstrated on a difficult problem that the evolutionary rule sets are able to learn rules with relationships in the predicate. The data available here does not support the learning of such rules.

5.1.2 Lake Kasumigaura

Lake Kasumigaura is situated 60 kms north east of Tokyo and is Japan's second largest lake. The lake is shallow, with a maximum depth of 7 meters and a mean

[2]A regression tree is a piecewise constant or piecewise linear estimate of a regression function

Table 5.2. *Physical characteristics of Lake Kasumigaura, Japan [190]*

Trophic State	hypertrophic
Morphometry:	
- Maximum depth	7 m
- Mean depth	4 m
- Surface area	220 km^2
- Volume	900 million m^3
Range of water temperature	2.1–32.0 °C
Mean water retention time	200 days

depth of 4 meters. Because of the lakes shallowness and strong mixing of the lake water by winds, persistent *stratification* of the water body does not occur. The lake has been monitored for more than 20 years at different sampling sites providing a long record of water quality data. The lake is hyper-eutrophic and very productive, mostly as a result of anthropogenic changes to the lake environment. The lake was an estuary before the construction of the Hitachigawa watergate near the lake outlet. The lake is now mainly used for recreation and as a sanctuary for water birds, although some aquaculture takes place in the lake itself. The catchment of the lake is used by agriculture and is subject to urbanisation. Table 5.2 lists the physical characteristics of the lake.

The lake experiences recurrent algal blooms, especially in the summer months, due to its nutrient richness and shallowness. It is the sudden growth and abundance of the algae that is of interest in modeling. It is well known that algae abundances respond positively to high nutrients, light and water temperatures. The modeling question is twofold. Firstly, can the water quality data available be used to model the abundance of algae in the lake, and secondly, what relationships exist between the available water quality data and the changing algae abundance?

Lake Kasumigaura under went a transformation in the algal community during the late 1980's [217, 218, 91]. From the mid 1970s until the late 1980s the lake experienced dense summer blooms of *Microcystis* spp, a colonial blue-green algae. Over that time changing land use practices in the lake's catchment and increased urbanisation of the towns around the lake shore, among other factors, led to a change in the nutrient loadings of the lakes. It has been hypothesised that the anthropogenic evolution of the lake has led to a higher ratio of nitrogen to phosphorus, which in turn has favoured the occurrence of the filamentous blue green algae, *Phormidium* spp and *Oscillatoria* spp [91]. The relationship between available nutrients and dependent species in the aquatic ecosystem is an active research area [217, 218, 91, 20, 30]

The summer blooms of *Microcystis* spp are the highest density blooms in the database. Since the late 80s there has been a decrease in the observed magnitudes of the summer chlorophyll-a levels coincident with a sudden decrease of *Microcystis* spp cells in the recurrent summer blooms. Chlorophyll-a measurements are often used as

a proxy for total algal abundance, although the relationship between chlorophyll-a measurements and algal cell counts is inexact due to the differing amounts of chlorophyll-a found in different algae species and cells sizes, and in algae under different conditions.

The problems addressed in this chapter are concerned with learning associations between the abundance of algae in the lake and the measured physical and chemical properties of the lake water. Algae abundance is based on measured cell counts of the different species. An outline of the measurement methodologies is presented in [218].

It is not reasonable to expect a high degree of predictive accuracy in this kind of problem domain. Inaccuracy in model outputs has several different sources [113]:

1 Uncertainty in the measurement of the dependent variable.
2 Uncertainty in the measurement of the driving, or model variables.
3 Structuring of the model.

Uncertainties in measured quantities are inherent in the nature of aquatic ecosystem modeling. Possibly the largest source of these uncertainties stems from using point measurements to approximate aquatic ecosystem states and behaviours. For Lake Kasumigaura the point source of data used in this chapter is central to the upstream Takahamairi Basin of the lake, and likely to be representative of many of the basin's physical properties, such as water temperature [169]. However the measurement of algal cells and physical properties like transparency are more problematic. Wind, currents, weather and the time of day all affect the concentration of algae at a particular point. The measurements were all taken around midday, however the other factors are not, and cannot realistically, be controlled for [63].

A further uncertainty in the measured physical parameters stems from the use of a single depth for taking water samples and measurements. Although the lake is shallow, it is possible that there are differing conditions at different depths in the lake which are not correlated and which do affect algal growth rates.

The uncertainties due to model structure reflects the fact that the model only has a limited amount of (noisy) information available to learn relationships from. Not all of the environmental parameters that could determine algal productivity are present in the measured data. The choice of data with which to base the modeling has been made with recourse to theory, and represents the most prominent physical features which are thought to contribute to algal productivity.[3] This source of error is somewhat mitigated by the likelihood that it will be swamped by measurement errors of the known driving variables.

Another aspect of model structure which contributes to prediction errors is the bias of the models learnt relationships. In general, ecological problems are highly non-linear with many competing processes and interactions. When applying a classic

[3]The problem of selecting what features of a system to measure is ubiquitous in inductive modeling, and is demonstrative of the philosophical problems of separating observation and theory.

Table 5.3. *The distribution of the water quality data used for modeling*

Measured Data	Abbreviation	Quartile 1	Median	Quartile 3	Units
Chlorophyll-a	Chl-a	44.740	70.293	93.608	mg/L
Nitrate	NO3	104	640	1076	μg/L
Ortho Phosphate	PO4	3	5	16	μg/L
ph	ph	8.44	8.98	9.40	
Nitrate:Phosphate	6.8	68	302		
Transparency[5]	Transp	70	90	120	cm
Water Temperature	Temp	9.8	18.5	24.6	°C

neural network model to this problem, an architecture with which to learn the desired relationships must be first specified. This immediately puts some bounds on the likely complexity of the relationship which will be learnt (both below and above). If the model is not able to represent the relationships between the attributes which affect the prediction then there will be more errors.

5.1.3 Data Handling

Table 5.3 shows the distribution of the water quality data used in the modeling. Most of the variables which are applicable for modeling were sampled at periods ranging from fortnightly to monthly. In most studies which have used data from Lake Kasumigaura the data has been linearly interpolated to give daily values [188, 190, 191, 237, 232, 230]. This approach has been used, unless otherwise noted, for the training and testing in this chapter. Interpolation adds no information that is not already present in the data set, however it makes the model outputs more comprehensible due to the time-series nature of the data. It should be noted that the raw data displays little auto-correlation in most of the measured variables, and it is likely that interpolation introduces an amount of noise to the modeling task. It is, however, the nature of ecosystem data to be noisy.

The learning task the model faces is to associate environmental conditions with measured values of algal abundance. As noted, the lake has transitioned from being *Microcystis* spp dominated to being *Phormidium* spp and *Oscillatoria* spp dominated during the period over which the data were measured. For this reason a typical *Microcystis* spp year, 1986, and a typical *Phormidium* spp and *Oscillatoria* spp dominated year, 1993, are used for testing the model. This means that the error rates achieved on the testing data will not be indicative of the error rates associated with the modeling task of predicting values from the lake data in general. To calculate those error rates some form of cross-validation should be used [238]. In this chapter, the testing set is chosen to evaluate the explanatory power of the model in two interesting years which are kept hidden from the model learning. The question being asked is: can the model learnt from the lake data correctly predict the measured values in two different years which are representative of the successional

change in algal species which has occurred in Kasumigaura?

5.1.4 Model Evaluation

A root mean square (RMS) error formula is used to evaluate the models produced by the evolutionary process. The RMS error is far from perfect for ecological studies as it fails to capture many of the characteristics about the problems which are of interest to ecologists. Nevertheless, it has been used in numerous studies and will be used here, albeit with a modification.

The first problem addressed in this chapter is the prediction of chlorophyll-a levels from the measured data. The problem property of interest in the prediction of chlorophyll-a levels is when the maximums will occur. The chosen test years are not randomly selected, but have been chosen to assess the models ability to predict chlorophyll-a in a typical *Microcystis* spp and a typical *Oscillatoria* spp and *Phormidium* spp dominated year. Both years were highly productive. To force the model to concentrate on the maximums, the RMS error is modified to punish underestimation of chlorophyll-a level. The aim is to produce a model which successfully predicts the chlorophyll-a peaks. The fitness function used is

$$f = \begin{cases} \sqrt{\frac{(x_{\text{pred}} - x_{\text{actual}})^2}{\text{Examples}}} & \text{If} x_{\text{pred}} > x_{\text{actual}} \\ \sqrt{\frac{4 \cdot (x_{\text{pred}} - x_{\text{actual}})^2}{\text{Examples}}} & \text{If} x_{\text{pred}} < x_{\text{actual}} \end{cases}$$

where Examples is the number of input vectors in the input set. The RMS error values reported are the usual RMS error calculation, the above function is only used internally for fitness evaluation.

The number of rules in a model can be restricted. The smaller the rule set the more comprehensible it is likely to be and the less prone to over-fitting [86]. In the initial experiments the rule set is limited to only 10 rules. This limit combined with the biased fitness evaluation ensures that the rule set is focussed on producing rules which describe the peak algal abundance. In Section 5.3, experiments without bounds on the number of rules and with the RMS error as the fitness evaluation will be described.

The ability to choose the fitness function and other conditions to suit the questions of interest in a modeling task is in fact a strength of the evolutionary method. Evolution is a general purpose optimisation procedure that can work effectively with a variety of fitness functions, representations and arbitrary conditions. Yet it can still optimise to find a good solutions to these problems.

Table 5.4. *Table of input parameters for the lake model*

Input Set	Lake Data
1	Water Temperature, Secchi Depth, PO_4, NO_3, NO_3 : PO_4 ratio, Dissolved Oxygen, ph, Solar Radiation
2	Water Temperature, Secchi Depth, PO_4 and NO_3 concentration
3	Water Temperature, Secchi Depth, NO_3 : PO_4 ratio
4	Water Temperature, Secchi Depth, PO_4 and NO_3 concentration, NO_3 : PO_4 ratio
5	Water Temperature, Solar Radiation, PO_4 and NO_3 concentration
6	Water Temperature, Secchi Depth, Solar Radiation, PO_4 and NO_3 concentration
7	Water Temperature, Secchi Depth, PO_4, NO_3, ph

Table 5.5. *Summary of results for different input sets*

	Root Mean Square Error				
Input Set	Lowest	Quartile 1	Median	Quartile 3	Highest
1	31.4	36	38.8	41.9	48.3
2	31.9	34	35.3	37.5	45.2
3	38.7	44.6	45.8	46.6	51.4
4	31.5	34	35.3	37.2	45
5	33.9	36.9	39.1	40.7	45.9
6	30.9	34.1	35.7	37.7	44.3
7	30.7	36.2	38.5	40.7	49.1

5.2 Predicting Chlorophyll-a Levels

The first problem considered is the prediction of chlorophyll-a levels from the measured physical and chemical properties of the lake. Chlorophyll-a is always present in the lake and is mostly due to the abundance of algal cells in the lake. The level of chlorophyll-a changes in accordance to the waxing and waning of different algae species in the lake throughout the years.

The evolutionary algorithm is used to extract patterns in the measured chemical and physical characteristics of the lake and the level of chlorophyll-a. A number of different runs on different input sets were conducted. Each input set consisted of a set of different measured lake characteristics. The different input sets are summarized in Table 5.4.

The evolutionary algorithm made 120 independent runs with each of the input sets in Table 5.4 with a population size of 200 and for 200 generations. The results on the different input sets are summarized in Table 5.5 and displayed graphically in Figure 5.2. The results on input sets 2, 4 and 6 are essentially the same. Input sets

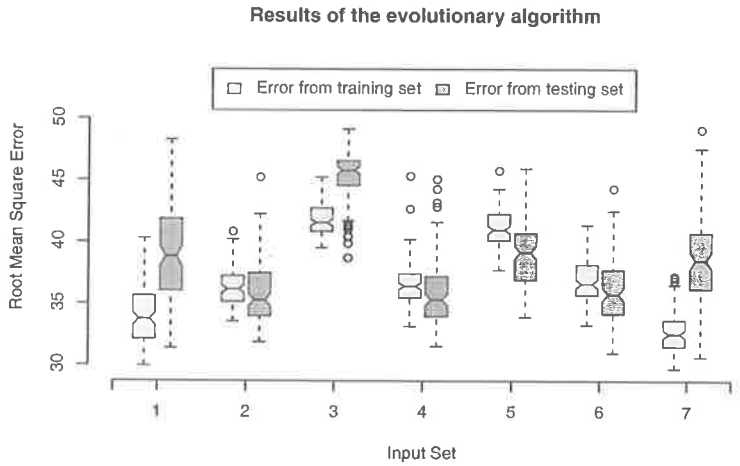


Figure 5.2. *Comparison of the root mean square error of the best rule sets found when applied to the training and testing sets for the different input sets shown in Table 5.4.*

1 and 7 show the largest variance, and both sets contain the pH. Input sets 1, 2, 4, 6 and 7 all have equivalent lowest bounds on the testing set. All of these runs contain the secchi depth, or water transparency, and the phosphate and nitrate levels in the lake. Input set 3 is lacking the nutrient levels and Input Set 5 is lacking the secchi depth.

It seems likely based on both theory and results that nutrient levels in the form of phosphate and nitrate measurements as well as transparency measurements are important inputs for the prediction of chlorophyll-a levels. It can be noted that secchi depth, which is a measure of water clarity, should be well correlated with the current chlorophyll-a concentration since chlorophyll-a will tend to make the water less transparent. There will be other events which will also make the water less transparent, such as an increase in the amount of suspended sediment. Nutrient levels are clearly going to be indicative of chlorophyll-a, since algae require nutrients survive and multiply. However, the relationship is not straight forward. A high chlorophyll-a level is indicative of a high algal abundance, and algae consume the available nutrients leading to a low nutrient abundance! On the other hand, low nutrient abundance inhibits the growth of algae.

Table 5.5 shows that Input Set 5 produces results which are better than Input Set 3 (the test set error of the 50% of runs between the first and third quartile of Input Set 5 are all better than the 50% of runs between the first and third quartile of Input Set 3). The implication is that available nutrient measurements in the form of phosphate and nitrate are more important to modeling than the transparency, at least for the models produced by applying the SASME algorithm to evolve rule lists with exceptions.

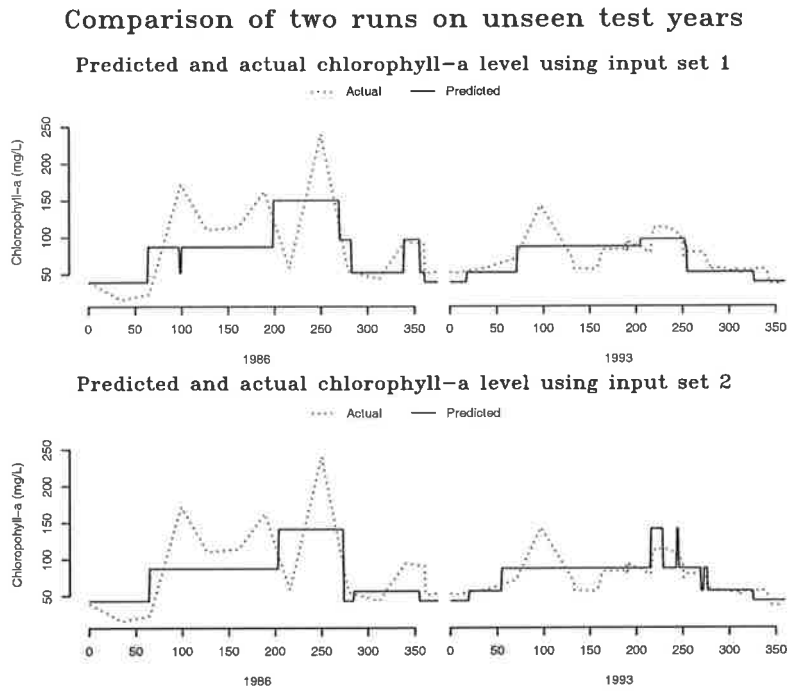


Figure 5.3. *Comparison of rule set output on the unseen test data. The top graph shows the model which uses Input Set 1 in Table 5.4, and is shown in Figure 5.6. The lower graph shows the model using Input Set 2 shown in Figure 5.5.*

5.2.1 Comparing Models

What causes the differences in the observed behaviour in the models? In this section two different models from two different rule sets are compared to answer this question. The first rule set is generated from Input Set 1, which consists of all of the available input data, and the second rule set is from Input Set 2, which appears to be one of the most promising input sets used in Table 5.5. A comparison of the performance of two randomly chosen rule sets from the two input sets is shown in Figures 5.3 and 5.4 for testing and training respectively.

Using Input Set 2

For Input Set 2 consisting of Temperature, Secchi Depth, Phosphate and Nitrate concentrations, a typical run of the genetic algorithm discovered a model with a RMS error of 37.80 when applied to the testing set, and a training set RMS error of 28.08. The rule for this model is shown in Figure 5.5.

Figure 5.5 shows that the model firstly uses the secchi depth to classify low algal abundance. This rule is mostly true over the low algal winter months. The model uses nitrate and phosphate concentrations to indicate algal consumption of available

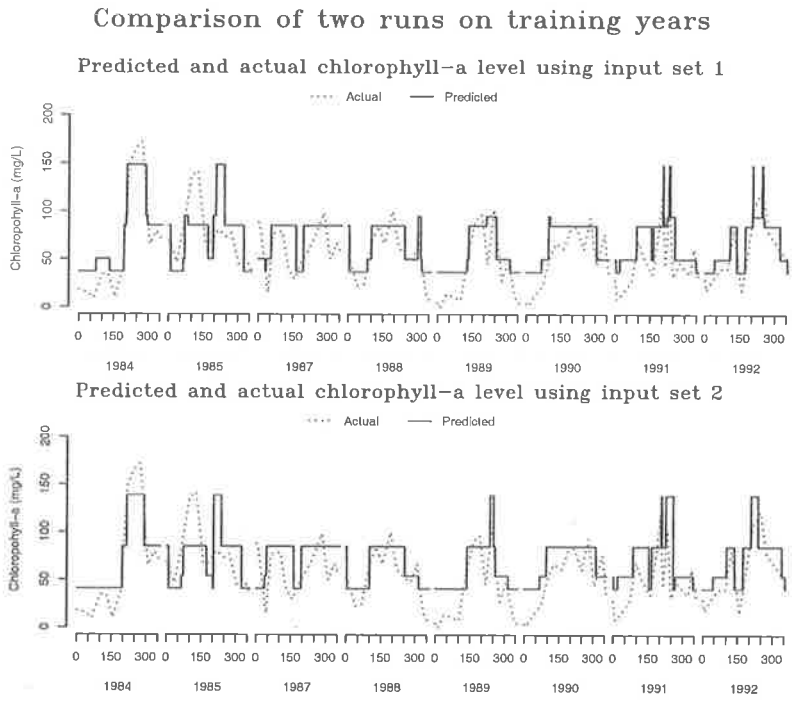
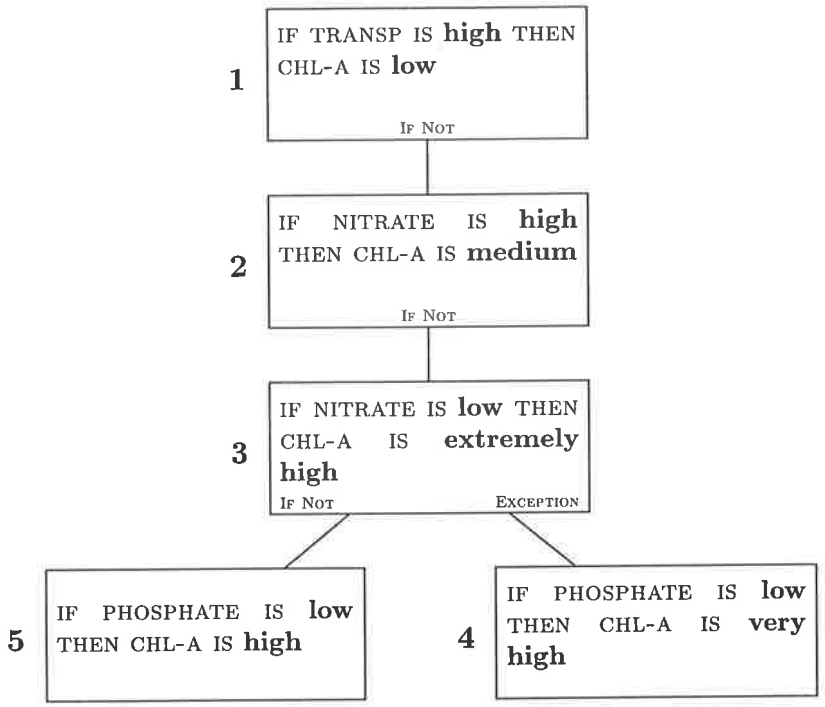


Figure 5.4. *Comparison of rule set output on the training data. The top graph shows the model which uses Input Set 1 in Table 5.4, and is shown in Figure 5.6. The lower graph shows the model using Input Set 2 shown in Figure 5.5.*

nutrients. It is interesting that the level of nitrate is used to indicate the most severe blooms. For Kasumigaura, the model discovered that large algal abundance is well correlated with low nitrate levels over the summer months. This may be explained by the fact that blue-green algae, which are dominating in summer, are able to fix nitrogen from the atmosphere for photosynthesis and do not depend on dissolved nitrogen in the water. If phosphate is also low then the algal cell count is very high (97 mg/l). This indicates that the largest blooms (*extremely high* chlorophyll-a in the order of 140 mg/l) occur when algae have consumed most of the available free nitrogen (nitrate levels are low), but phosphate levels are not near to exhaustion.

This model does not, however, succeed in predicting all of the peaks in algal abundance. The high algal abundance in early 1986 and also in the training set are not predicted by this model. The median prediction of 120 independent runs using Input Set 2 shows that the median model also misses these algal abundance peaks. This suggests that either the driving forces for these peak abundances are not present in the input set, or that the concepts required to predict the algal abundance at this time cannot be represented or discovered by the learning algorithm. It can be expected that there is a large amount of noise in the lake measurements, and this noise may be the explanation.



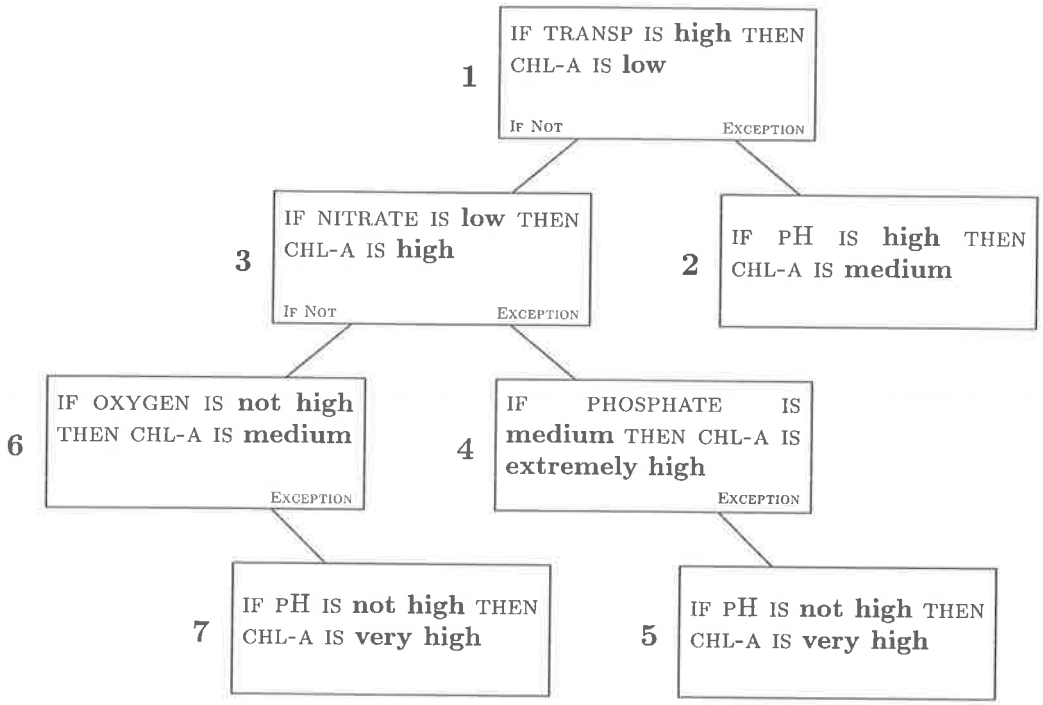
Description	Value
Transparency is high	Transp > 110cm
Nitrate is high	NO3 > 577μg/l
Nitrate is low	NO3 < 577μg/l
Phosphate is low	PO4 < 34μg/l
Chlorophyll-a is low	Chl-a = 34mg/l
Chlorophyll-a is medium	Chl-a = 55mg/l
Chlorophyll-a is high	Chl-a = 67mg/l
Chlorophyll-a is very high	Chl-a = 97mg/l
Chlorophyll-a is extremely high	Chl-a = 140mg/l

Figure 5.5. *An evolved rule set for predicting algae*

Using all Input Set 1

Input set 1 contains all of the available input parameters. The variance of the RMS error for this data set is higher, which is possibly due to the higher degree of freedom available from the larger number of input parameters when producing models with this dataset. A random rule set from Input Set 1 is shown in Figure 5.6.

The different model structures produce qualitatively different predictions. In particular, the timing of high algal abundance are picked up by different model structures differently. The model detailed in Figure 5.6 produced the prediction on the unseen testing years of 1986 and 1993 shown in Figures 5.3 This model had a very good root mean square error of 35.76.



Description	Value
Transparency is high	Transp > 93cm
pH is high	pH > 9.2
pH is not high	pH < 9.2
Nitrate is low	NO3 < 560µg/l
Phosphate is medium	25 < PO4 <= 168µg/l
Oxygen is not high	DO < 18mg/l
Chlorophyll-a is low	Chl-a = 44mg/l
Chlorophyll-a is medium	Chl-a = 60mg/l
Chlorophyll-a is high	Chl-a = 101mg/l
Chlorophyll-a is very high	Chl-a = 112mg/l
Chlorophyll-a is extremely high	Chl-a = 174mg/l

Figure 5.6. *An evolved rule set for predicting algae*

Even though the two models produce similar RMS errors they produce different predictions. One example of a different prediction is the algal bloom around day 350 of 1986 in Figure 5.3. This small bloom is correctly predicted by the model produced from Input Set 1, but not by the model produced from Input Set 2. Why?

The model in Figure 5.6 uses Rule **5** to categorize this peak. That is, it was a time of low nitrate and medium phosphate levels and the pH level was less than 9.2. Had the lake been in a more alkaline state then the prediction would have been for an extremely high level of chlorophyll-a (174mg/l). This rule tells us that the peak chlorophyll-a level occurred at a time when the nutrient nitrate levels were low and phosphate levels medium, and the pH was less than 9.16. It could be concluded from

Table 5.6. *Summary of results for Input Set 1 with normal fitness evaluation*

Lowest	Quartile 1	Median	Quartile 3	Highest
39.18	40.48	42.21	42.92	44.61

the model that higher chlorophyll-a at approximately 174 mg/l would be expected at pH levels above 9.16. Nevertheless, the validity of this conclusion is unconfirmed. It is also unclear of what cause and effect relationship is occurring. Does the high chlorophyll-a level increase the measured pH? What is clear, however, is how the model is making its predictions. And that is the point.

5.2.2 Comparison with CART

The classification and regression tree method of decision tree generation is a greedy recursive partitioning method which can be used to predict real outcome variables [40]. This makes it a good candidate for the generation of solutions for comparison with the evolved rulesets. This comparison is conducted with the CART implementation supplied with the R statistical package [127]. When applied to the Kasumigaura data using Input Set 1 from Table 5.4, CART produced the decision tree shown in Figure 5.7, and produced the training and testing output shown in Figure 5.8. The value of n in a parent node is the number of training examples which have not been classified by that node, and so are eligible for splitting into the child nodes. The number in the nodes in Figure 5.7 is the expected value of the outcome variable for the examples that reach that node. That is, it is the weighted average of the outcomes of the leaf nodes below it.

CART produced an RMS error of 45.47 on the testing set. This is not directly comparable with those in Table 5.5 since the evolutionary algorithm was trained with a fitness function which explicitly punished under-prediction more than over-prediction, and the algorithm was limited to 10 rules. To compare, the evolutionary algorithm was applied to the same data set without a rule limit and with the usual RMS error criteria for fitness. The number of levels of prediction for the rule set was increased to 9, and the number of boxes for dividing variables was increased to 4. The results of 10 runs with a population of 200 for 200 generations are shown in Table 5.6. An example of the predicted output of a SASME evolved rule set is shown in Figure 5.9.

The evolved rule set which produced the unlabelled output is shown in Figure 5.10. The rule set in the figure has the value ranges of the attribute labels included on the rules. The numbers to the right are the number of training examples which satisfied the rules premise. This is different to the number of instances that the rule has its consequence applied, since subsequent exception rules will have their consequence performed when their premise is found to be true. The rule set structure is consistent with the previous rule set figures. That is, any rule connected to the left of its parent

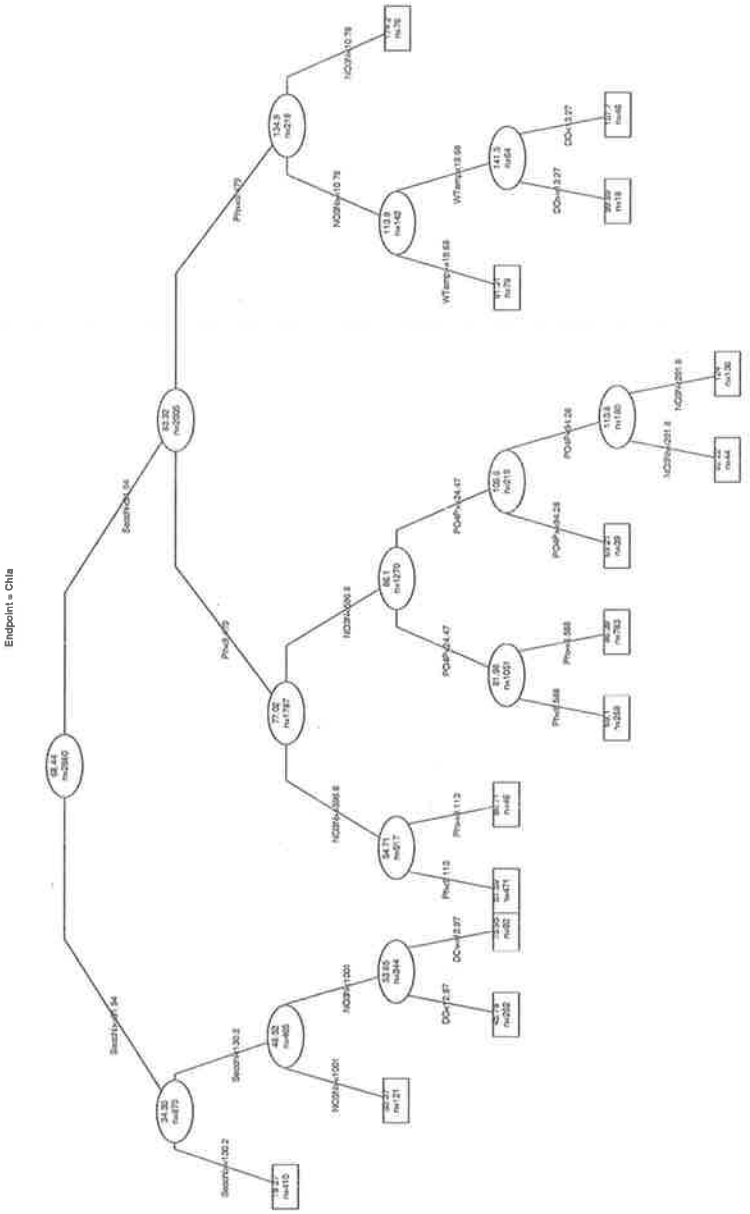


Figure 5.7. *Cart rule set produced from Input Set 1. The output of the rule set is shown in Figure 5.8.*

or the sole child of a parent rule is tested when the parent rule is found to be not true. Children connected to the right of the parent rule are tested only when the parent rule is found true.

Table 5.6 shows that the SASME algorithm outperforms CART in each of the 10 runs performed, that is, the worst test root mean square error of the evolved rules is better than that produced by CART. The differences between the two algorithms

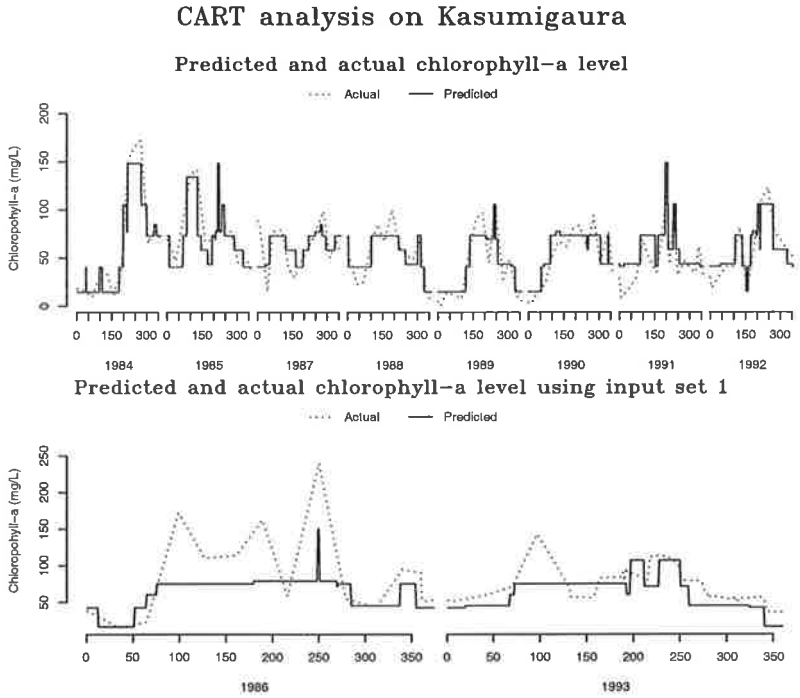


Figure 5.8. *Output of the CART model applied to Lake Kasumigaura.*

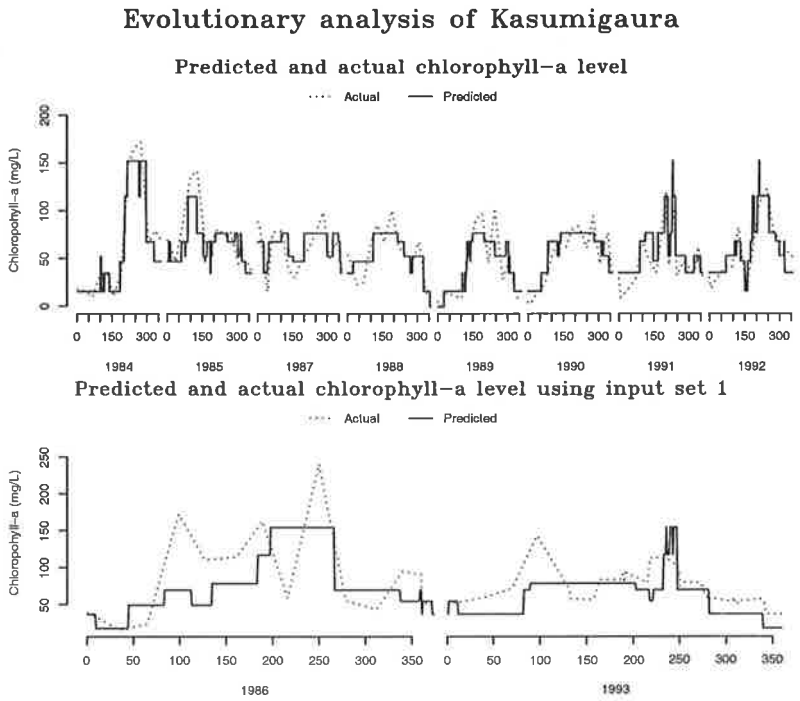


Figure 5.9. *Output from the evolutionary model trained using the root mean square error. The model is shown in Figure 5.10*

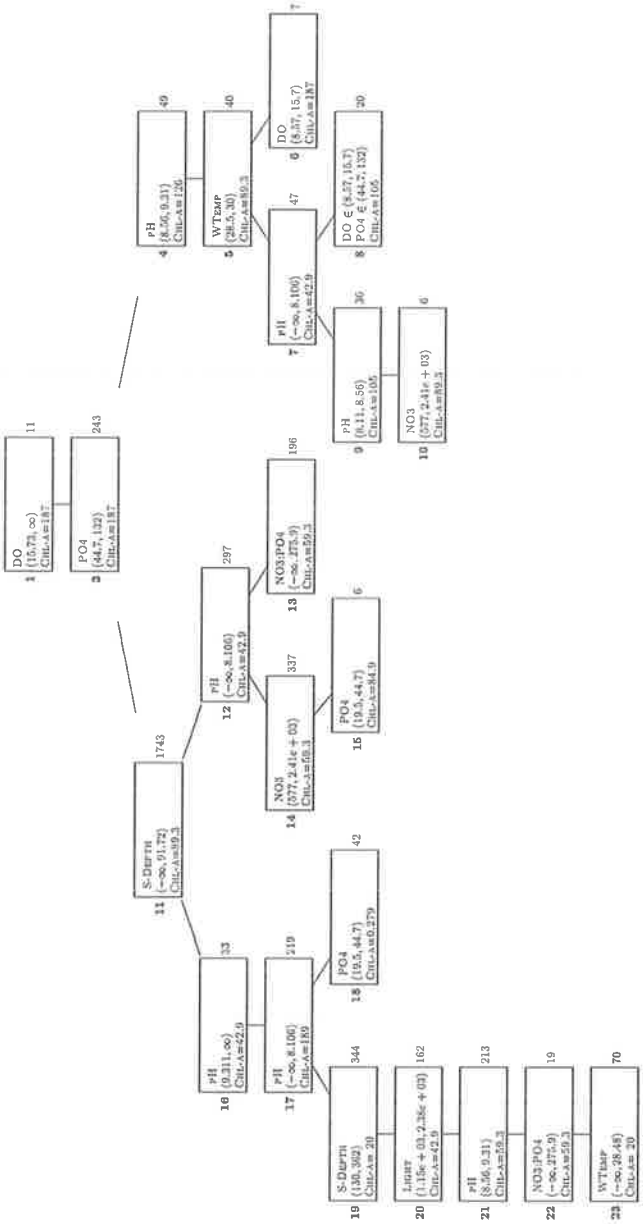


Figure 5.10. *Rule set for any length*

is not large, and it shows that the evolutionary approach is able to find solutions as good as those found by a greedy partitioning algorithm. The output graphs in Figures 5.8 and 5.9 appear equivocal with regard to prediction quality, which is indicative of the problems of using root mean square error in ecological time series data.

5.2.3 Discussion of Chlorophyll-a Prediction

The self-adaptive evolutionary rule induction method is able to symbiotically optimise symbolic rule structures and associated parameter vectors to discover rule set models for predicting chlorophyll-a in Lake Kasumigaura. The models are comprehensible and questions about why particular predictions have been made can be answered by tracing the decision process that the rule set model undertakes. Further, there are no assumptions on the structure of the model itself, which enables arbitrary criteria to be placed upon the model production to suit the questions of interest. This was demonstrated by limiting the number of rules that a model could contain and by modifying the fitness function to emphasise the problem characteristic of interest. The first condition was imposed to maximise comprehensibility of the evolved rule sets, and the second to force the rule sets to accurately predict the peak algal abundances.

In situ time series hold unique information about ecosystem processes and behaviour. Inductive modeling techniques can be used to explore this information. Machine learning techniques offer a new quality of inductive modeling by extracting not only seasonal and annual patterns, but related connectivity between key variables as well.

The rules produced for chlorophyll-a prediction are descriptive and comprehensible and are likely to contain some information about actual ecosystem function. An issue with inductive models in general is unraveling correlations between inputs measured and unmeasured, and the cause and effect of the relationships. A classic example is the observation that in New York over summer, drownings and lemonade sales both increase. Deducing that lemonade sales are somehow implicated in drownings is the mistake that inductive modeling could make on such a data set.

It is possible that some of the prediction results in this section have made that mistake. The strength of the transparent representation is that the basis of predictions is explicitly apparent. If a neural network connects lemonade to drownings and is asked to make a prediction in a summer where the lemonade factory is on strike, it will incorrectly conclude that there will be a decrease in the number of drownings. A rule set, by contrast, will make the same error, but the basis of the prediction is apparent and the cause of the error easily observable. The other possibility is that of finding a previously unknown relationship in the data. Maybe lemonade drinking really does increase a persons chance of experiencing flotation difficulties in water!

Ultimately, the induced hypotheses need to be tested in the field. The inductive model produces hypotheses which are able to be understood, and therefore can be tested. It performs as well as neural network methods and so is likely to have learnt similar relationships to the neural network models for the prediction of chlorophyll-a [189, 238].

Table 5.7. *Summary of cell counts for colonial (* Microcystis *spp) and filamentous (* Oscillatoria *spp and* Phormidium *spp) algae.*

Type	Lowest	Quartile 1	Median	Quartile 3	Highest	Units
Colonial	0.0	0.0	362.5	18575.0	644117.0	Cells/l
Filamentous	0.0	0.0	4778.0	36997.5	502320.0	Cells/l

5.3 Species Prediction

The anthropogenically eutrophied waters of Lake Kasumigaura favour a range of blue green algal species. The populations of these species has waxed and waned throughout the monitoring period from 1984 to 1993. As noted in Section 5.1.2, since the early 1980s the dominant species composition of the lake has changed from *Microcystis* spp to *Oscillatoria* spp and *Phormidium* spp. It is of interest both theoretically and also from a management perspective to understand why this change has occurred.

Learning rules to associate algal species with the physical and chemical properties of the lake is a more difficult problem than learning the association with chlorophyll-a levels. Algal species are specialized for certain conditions and occur only temporarily in the lake, which means that numbers of measurements indicate no species present. Learning becomes impossible when there are too few non-zero cases for classification. *Microcystis* spp is frequently present in the data, and raw data can be used. *Microcystis* spp is an example of a colonial algal species, which formed large algae cell colonies in Lake Kasumigaura up until the late 80s. The other species of interest, *Oscillatoria* spp and *Phormidium* spp are less common in the lake. However, they are both examples of filamentous algae, and they both appear to be favoured by similar conditions in the lake. For the purposes of this study, the two cell counts are combined to give a filamentous algae cell count. The ecological legitimacy of this could be questioned, however aquatic ecosystem experts concur that it is acceptable for the use it has in this section.[6]

A summary of the two output variables is shown in Table 5.7.

Two separate rule set models are constructed in this section. One model attempts to predict the presence of *Microcystis* spp in the lake given the current physical and chemical parameters in the lake. The other model attempts to do the same for *Oscillatoria* spp and *Phormidium* spp. The explanations that the models use to predict the algae species can then be compared to see what the model predicts is the reason for the disappearance of *Microcystis* spp from the lake in recent times, and the dominance of the filamentous algae.

Two different experiments are attempted with this problem.

[6] *Friedrich Recknagel*, personal communication, 2000

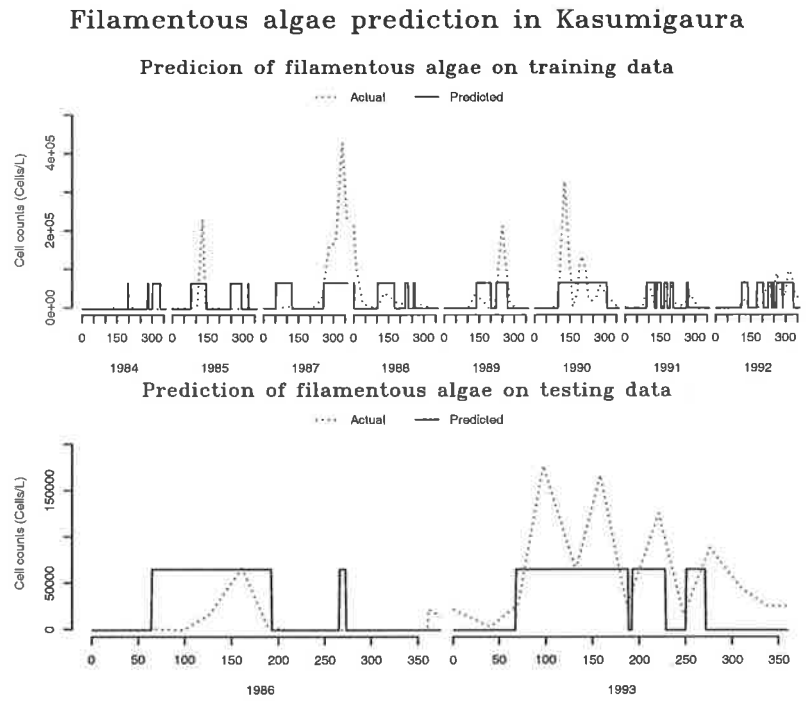


Figure 5.11. *Results of the prediction of* Oscillatoria *spp and* Phormidium *spp counts in both the training and testing data.*

5.3.1 Experiment 1

The first experiment uses the data from Input Set 1 in Table 5.4 on page 125. The problem the evolutionary algorithm faces is to cluster the species cell counts into two clusters according to the measured physical and chemical properties of the lake. The values of the cell counts around which the clusters form is to be decided by the algorithm. Table 5.7 shows that more than 25% of the time the measured cell counts are 0. An optimal cluster distribution on this data will therefore include one cluster corresponding to a negligible cell count. A second cluster will then be associated with the presence of some number of algal cells.

The filamentous algae *Oscillatoria* spp and *Phormidium* spp were the most difficult to predict. The output of the model is shown in Figure 5.11. The rule set produced is shown in Figure 5.12.

The results for predicting *Microcystis* spp are shown in Figure 5.13. The rule set which produced these results is shown in Figure 5.14.

The interesting parts of the rule sets are those rules which predict the presence of the algae species. For the filamentous algae the rule set in Figure 5.12 uses rules **4**, **5** and **7** to predict the presence of filamentous algae. All of the positive algae prediction rules occur as exceptions to the rule dealing with low Secchi depths, **2**.

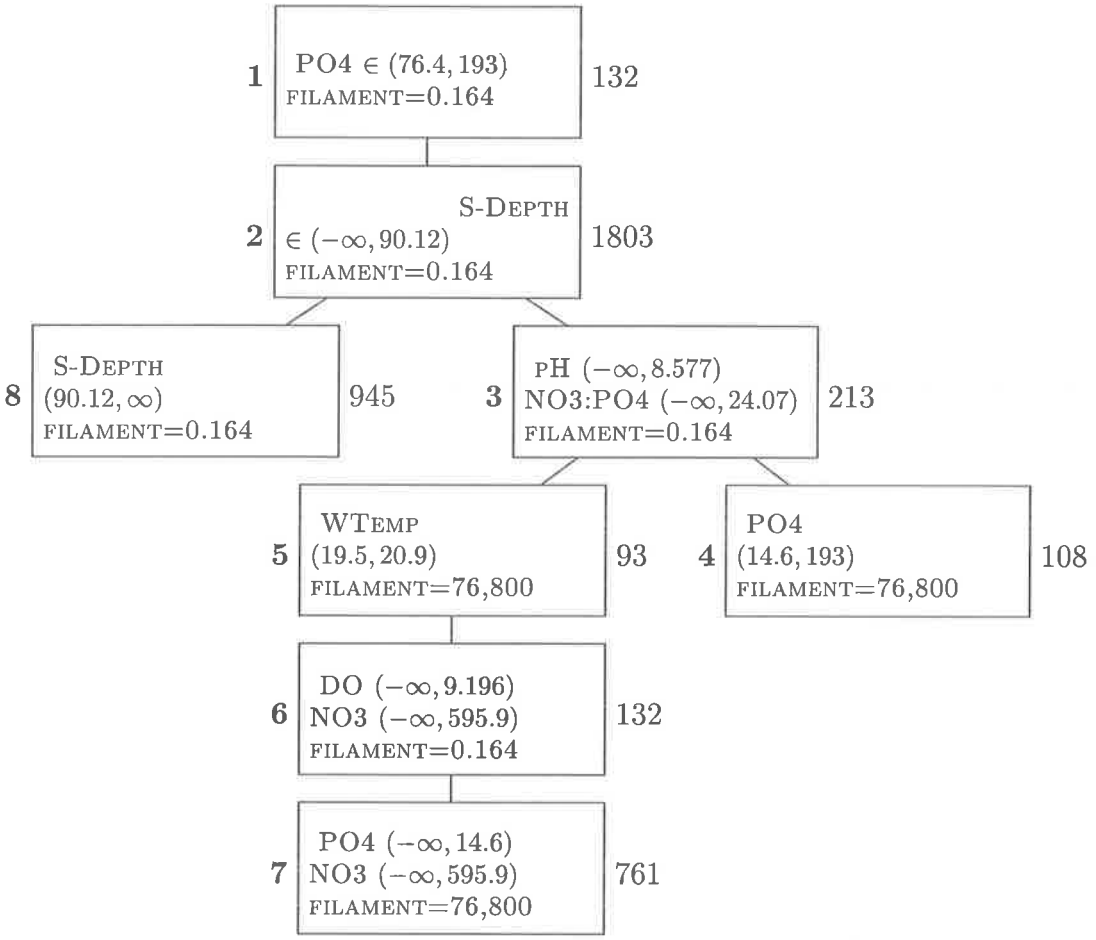


Figure 5.12. *An evolved rule set for predicting filamentous blue green algae, as represented by the combined cell counts of* Oscillatoria *spp and* Phormidium *spp.*

Rule **7** predicts for filamentous algae most often in the training set, and does so on the condition of phosphate being low, and nitrate not being high.[7] Rule **5** predicts for the algae when the pH is low, the phosphate level is relatively high, and the nitrate to phosphate ratio level is low. The final positive rule, **5**, predicts based on a narrow temperature band, and does so in only 3% of training examples. This rule may be overfit to the data. Rule **9** in particular is consistent with filamentous algae being able to grow during periods of low phosphate.

The rules for *Microcystis* spp shown in Figure 5.14 are simpler to understand. The positive predictions are made when phosphorus is not low, and either pH is extremely high, rules **3** and **2**, or else the water temperature is very high **5**.

[7]Qualitative description of attribute ranges are based on the distribution of attribute values in Table 5.3

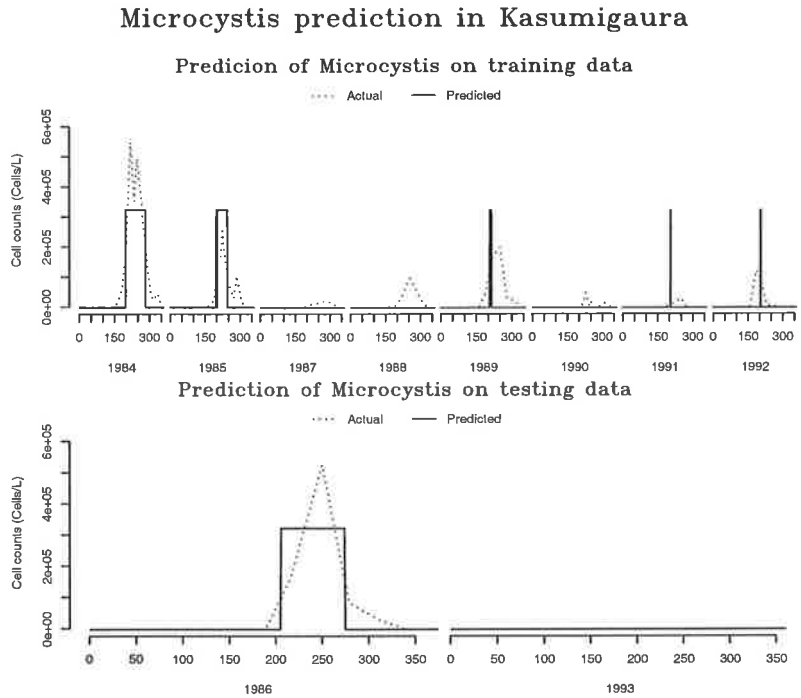


Figure 5.13. *Results of the prediction of* Microcystis *spp counts in both the training and testing data.*

5.3.2 Experiment 2

In this experiment the problem of species prediction is also addressed, but a different data base is used and the rules are forced to learn preclassified situations. In the previous experiment the levels of the clusters that the algorithm learnt were subject to evolution. In the current experiment these are set at 0 and 50,000 cells.

Measured Data

The experiments reported in this section use 1986 and 1993 as the testing years as before. However, the model is trained on a raw data set which contains only 105 records. Of those records, 91 are used for training and the remaining 14 are from the two testing years.[8] Because of the small number of training records a different strategy is employed to try and prevent overfitting of the data. The current experiments use a random sample of the training examples in each generation to evaluate the population. From the 91 patterns in the training set, each generation

[8]This particular data set ends in August 1993. The difference in time frame is due to a difference in the availability of some of the input data used in these experiments. This data is only available until August, and not as commonly available throughout the sample period. Result graphs are from linearly interpolated testing sets to give a sense of the time at which the different measurements are taken.

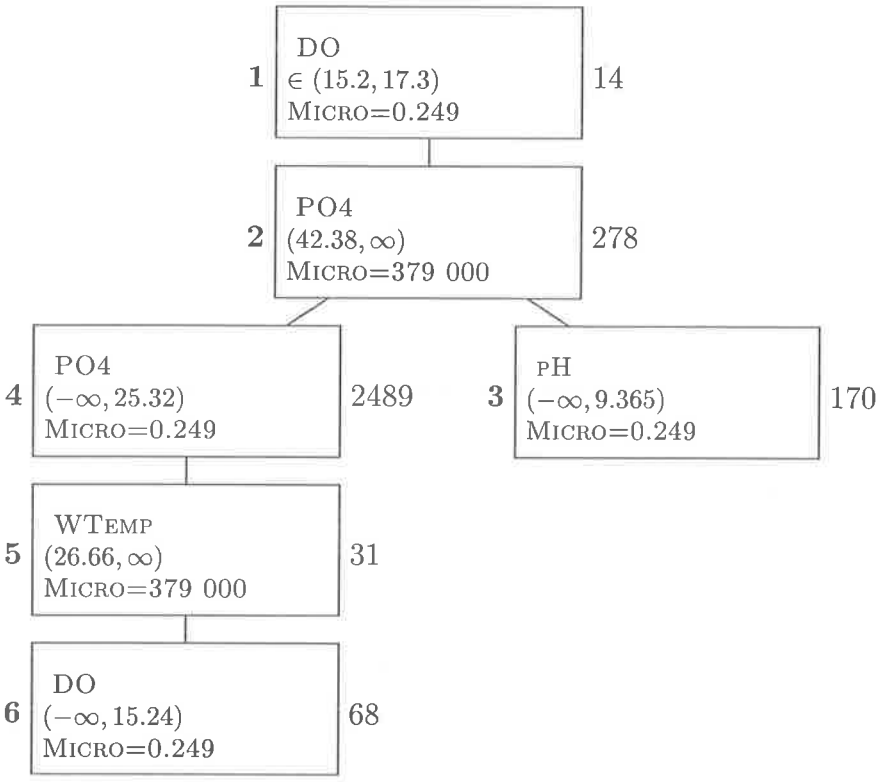


Figure 5.14. *An evolved rule set for predicting* Microcystis *spp, an example of colonial blue green algae.*

a generational training set is constructed consisting of 91 examples made up by sampling with replacement from the training set. This places 63%[9] of the training examples in the generational training set, and presents every generation with a different set of training patterns, helping to maintain the populations ability to generalise.

Table 5.8 shows the chemical and physical lake parameters used for this experiment. The parameters were chosen according to availability and perceived predictive potential based on algal growth requirements. The algorithm should be able to discover the most useful lake parameters and discard less useful parameters. It is known for many inductive techniques that redundant or unimportant inputs can make the learning task harder, and so choosing a minimal input set is likely to achieve the best results.

Two evolved lake classifiers were created. One classifier was trained on the task of classifying when *Microcystis* spp cell counts were measured to be higher than 50,000 cells given the data in Table 5.8. The second classifier considered the combined cell counts of *Oscillatoria* spp and *Phormidium* spp for the same task.

[9]The probability of not selecting a training example N times is $(1 - \frac{1}{N})^N$, giving $\approx 36.6\%$ for $N = 91$. The limit as $N \to \infty$ is $\frac{1}{e} \approx 36.7\%$

Table 5.8. *A summary of the input parameters used in Experiment 2*

Measured Data	Abbreviation	1st Quartile	Median	3rd Quartile	Units
Nitrate	NO3	104	640	1076	μg/L
Ortho Phosphate	PO4	3	5	16	μg/L
ph	ph	8.44	8.98	9.40	
Water Temperature	Temp	9.8	18.5	24.6	°C
Transparency	Transp	70	90	120	cm
Diss. Total Phosphorus	DTP	16	20	34	μg/L
Ammonia	NH4	21	63	162	μg/L
Nitrite	NO2	9	18	30	μg/L

5.3.3 Experimental Results and Discussion

All experiments were conducted with a population of 200 models for 100 generations. The combined cell counts of *Oscillatoria* spp and *Phormidium* spp were used in one experiment and *Microcystis* spp in the other. The aim is to elucidate possible environmental difference in the preferences of the two blue-green algal types. *Microcystis* spp were again the easiest to predict. For both datasets a cutoff level of 50,000 cells was chosen. Although the levels of the prediction are not subject to evolution, the fitness function the model is trained with the RMS error of the models predictions and not the misclassification rate. This means that the model is punished more for missing high algae peaks than for missing low ones.

A true positive prediction (TPP) is when the model predicts that the algae are present, and the measured data shows that there are more than 50,000 cells present. Similarly for true negative predictions (TNPs).

The sensitivity of the model is defined as:

$$\text{Sensitivity} = \frac{\text{True Positive Predictions}}{\text{False Negative Predictions} + \text{True Positive Predictions}} \tag{5.1}$$

the specificity is similarly defined:

$$\text{Specificity} = \frac{\text{True Negative Predictions}}{\text{False Positive Predictions} + \text{True Negative Predictions}} \tag{5.2}$$

Sensitivity is the ratio of true predicted positive results to actual measured positive results, and similarly for specificity. They measure the error rates of the model on positive and negative outcomes respectively. A third measure which is useful is the positive predictive power of the model. This is defined as the ratio of true positive results to predicted positive results, and represents a measure of the confidence one can have in a positive prediction of algal presence from the model. The error rate is the ratio of true predictions to examples presented, whether positive or negative. 100 independent trials were conducted for each of the datasets. The error rate, sensitivity, specificity and positive predictive power were calculated for each run and the distribution of the error measures over the 100 runs are graphed.

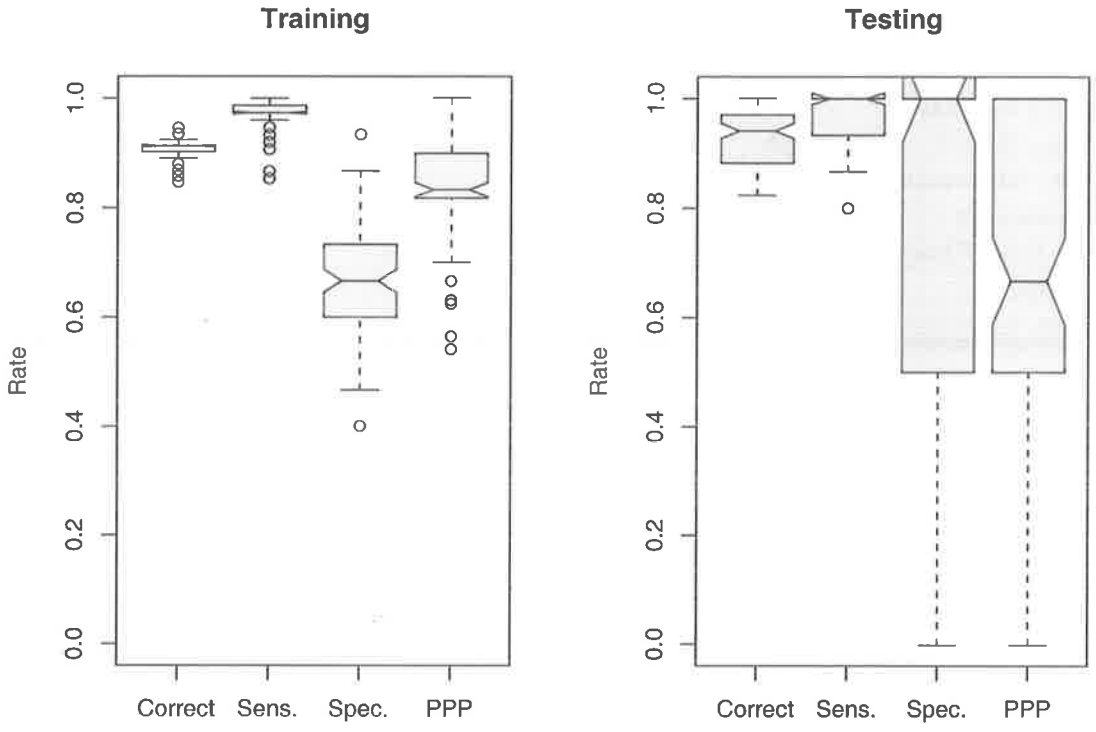


Figure 5.15. Microcystis *spp error rate, sensitivity, specificity and positive predictive power for 100 independent runs.*

5.3.4 Results of Species Prediction

Figure 5.15 shows the error rate, sensitivity, specificity and positive predictive power of the best classifier found, based on training results, for each of 100 independent evolutionary runs. Figure 5.16 shows the results for the filamentous algae. *Microcystis* spp are easier for the model to predict compared to the filamentous algae. *Microcystis* spp bloom events are more discrete compared to *Oscillatoria* spp and *Phormidium* spp events, tending to be near zero most of the time with discrete spikes lasting a few weeks of more than 50,000 cells. It could also be due to a clearer environmental preference of *Microcystis* spp.

The evolved rule sets were condensed to show the conditions under which the model predicts the algae species presence. The current database can be explained with a small number of classification conditions. This is clearly not an exhaustive list of the conditions preferred by the algal species, but rather a simplification of the conditions in Kasumigaura under which the data indicates that the species will be present. Different runs of the algorithm produce different conditions by considering different lake attributes. This is due to the algorithm learning on data containing redundant attributes. In different runs it substitutes different attributes to make

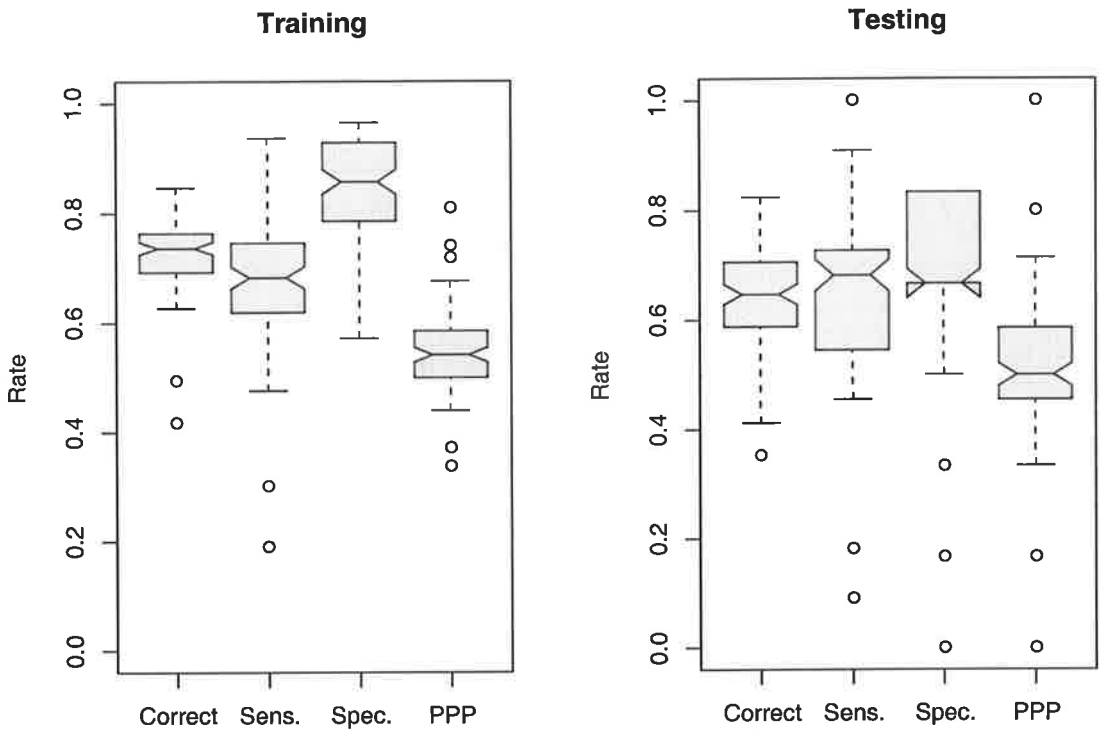


Figure 5.16. Oscillatoria *spp and* Phormidium *spp error rate, sensitivity, specificity and positive predictive power for 100 independent runs.*

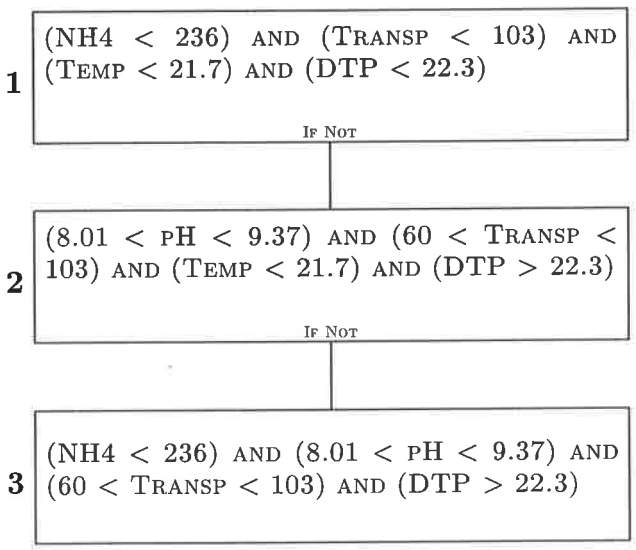


Figure 5.17. *A condensed classifier for the presence of* Oscillatoria *spp and* Phormidium *spp in Lake Kasumigaura.*

1 (TEMP > 29) AND (DTP > 74.2) AND (PH > 8.15)

Figure 5.18. *A condensed classifier for the presence of* Microcystis *spp in lake Kasumigaura.*

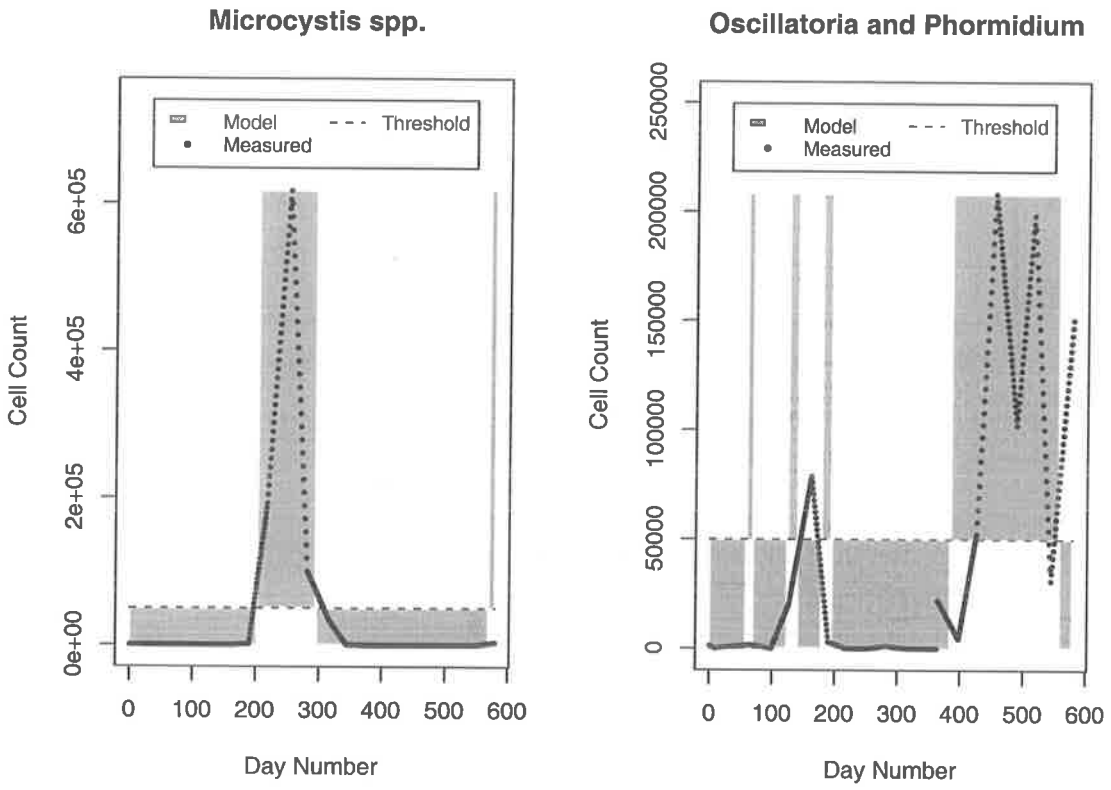


Figure 5.19. *Comparison between measured and predicted abundance of colonial (*Microcystis *spp) and filamentous (*Oscillatoria *spp and* Phormidium *spp) in 1986 and 1993*

the prediction. Since the inter-run variance of errors is not extreme, this acceptably allows the examination of hypothesis based on different attributes which nevertheless achieve similar error rates.

Figure 5.17 shows a typical ruleset found for *Oscillatoria* spp and *Phormidium* spp and Figure 5.18 for *Microcystis* spp. The time series plots on the testing set are shown in Figure 5.19.

Figure 5.19 shows the temporal distribution of model errors on the unseen test sets. These plots were made by interpolating the lake data to produce daily values for

the input and output variables. The 50,000 cell cutoff is drawn to show when the species are considered to be in bloom. The *Microcystis* spp model makes few errors on the test years used. The filamentous model significantly misses the lake dynamics on several occasions. It predicts two peaks for the 1986 algal bloom, one early and one late, though both miss the interpolated cell counts for this peak, and it misses the actual measured peak altogether. The other error is a slightly early onset of the 1993 peaks. Given the noise and difficulty of the problem, these error rates are quite acceptable. There is no temporal component in the model evaluation, and so missing a peak by a few days can induce a higher error than missing a peak altogether, which raises interesting questions about appropriate error measures. This study has conformed to the types of error measures used in previous studies which have utilized this data, however, there are other possabilities.

The *Microcystis* spp models consistently predict blooms of *Microcystis* spp under extremely high water temperatures and pH levels, as shown by the model in Figure 5.18. Oscillatoria is predicted most often under low secchi depths, indicating the presence of an algal bloom, and the absence of extremely high temperature and pH levels, as shown in Figure 5.17. Nutrient preferences of the two species are harder to contrast, although a preference for phosphorus levels not being extremely low in both cases is discernible. Blue-green algae have some ability to fix gaseous nitrogen from air, and it is possible that the blue greens are out competing other species under nitrogen limited conditions in lake waters.

5.3.5 Discussion of Species Prediction Results

Although the data used in this study contain a lot of sampling noise the model is still able to discover patterns in the measured data that relate the data to the occurrence of certain abundance levels of the algae species. The basis of the model predictions is clear and easily understood.

The rules discovered by the model are consistent with an algal succession based on decreasing phosphorus levels in Kasumigaura. In both case studies, only the rules for the prediction of filamentous algae contain positive predictions based on phosphorus levels not being high. Both models also agree that *Microcystis* spp is favoured by high pH and temperatures. This may be an environmental niche that *Microcystis* spp prefers, as suggested by many authors, eg. [211, 193, 107, 212].

5.4 Conclusions

Knowledge representation is an important issue for inductive learning algorithms. Representations that allow direct expression of learnt knowledge can be used to examine the underlying hypothesis of a learnt model. The *modus operandi* of the model can then be compared to domain theory.

In this chapter a number of different models have been evolved for the prediction of algae in a hypereutrophic lake in Japan. Aquatic ecosystem data present a challenging problem for machine learning due to the noise, complexities, and interactions of the measured data. The domain is also of interest because there is still a lack of knowledge about the relationships between physical and chemical properties of freshwater systems and their biological productivity.

The evolutionary models produced for ecosystem prediction perform no worse than those produced by artificial neural network approaches, they are to some extent, however, comprehensible. The SASME framework allows the choice of representation to be made so that information about the problem at hand can be discovered and communicated.

Evolutionary methods appear well suited due to their global rather than greedy approach to model formation. This allows the arbitrary conditions to be placed on the models while they are being evolved, and for representations to be chosen to suit the modeling tasks of interest.

Chapter 6

Conclusion

> ...when the dust has settled, it is usually found that the new technique is neither a miraculous cure-all nor a complete disaster, but rather an addition to the analyst's toolkit which works well in some situations and not in others.
>
> C Chatfield, [44, page 446]

6.1 Summary of Thesis

This thesis has addressed the problem of evolving mixed symbolic and numeric representations by developing a method of self-adaptive, symbiotic model evolution, SASME. The two parts of the representation were evolved as symbionts, each part mutualistically dependent on the other. The symbolic part was modified by a self-adaptive procedure which adapted the rates of discrete mutations during the course of the evolution. The numeric part of the representation was simultaneously evolved with a self-adaptive evolutionary strategies algorithm.

A discrete rule set model which explicitly represents exceptions was evolved by the SASME framework. The numeric component defined partition values for the observed continuous attributes, and these partitions defined symbols with which the rule set operated. The numeric vector optimised the state space partitioning, while the symbolic structure evolution optimised the rule set topology.

In Chapter 4 the SASME algorithm was applied to the evolution of control rules for the cart-pole problem. The algorithm successfully optimised rule set topologies whilst optimising state space partitions to solve this problem. The produced rule sets contain information about the learning problem. The rule sets were applied to the two-pole problem by including an evolvable relationship in the rule premises. This relationship allowed the rule sets to make decisions based on the relative angles of the two poles. The symbolic rule sets produced were comprehensible and able to solve this difficult problem. A final cart-pole problem was formed by removing the velocity information from the observed state. To solve this variant, a novel

recurrent rule structure was developed which made decisions based on its previous actions. The algorithm evolved the rate at which it discounts decisions made in the past. This problem was also solved and explicit rule sets produced showing how the evolved solutions were controlling the system.

In Chapter 5 the SASME rule sets were applied to the evolution of rule sets for ecosystem prediction. The algorithm formed models relating water quality data to chlorophyll-a concentrations. The evolved models explicitly displayed the knowledge they learnt about the system. Different runs of the algorithm produced models with similar predictive accuracy but with different rules. The algorithm was then applied to the problem of discovering why the observed patterns in algae species abundances had changed, and the models produced were compared to theory. The algorithm was able to produce comprehensible rule sets for the prediction of algae without domain knowledge.

6.2 Conclusions of Thesis

There are clear benefits in the utilisation of knowledge representations which enable the transfer of knowledge learnt by evolutionary learning methods. This thesis has demonstrated several times the ability of the SASME algorithm to evolve discrete rule set structures and optimise parameters for those structures in order to solve machine learning problems. The products of the evolutionary search can then be examined and the knowledge learnt by the evolutionary learner is explicitly represented.

The use of self-adaptive mutation rates for the discrete structures and the associated continuous vectors allows the evolution to simultaneously optimise the two components. As each component discovers new innovations, and presents to its symbiont a new avenue of exploration, the self-adaptive mutation rate of the components can adjust to exploit the new environment. Thus the self-adaptive evolutionary search is able to evolve solutions to difficult problems and still maintain a level of comprehensibility in the supplied solutions, as shown in this thesis.

The application of self-adaptive mutation rates for discrete operators appears to allow rates of applications of different operators to adjust to the topology of the structure they are operating on; they allow the operator rates to be a function of the generation, and this is shown in this thesis to produce superior evolutionary dynamics to most, and equivalent to the best, fixed operator rates. However, self-adaptive operators require no domain tuning. Empirically they are shown to provide satisfactory results in all the test problems used in this thesis.

The SASME approach allows the representation to learn models in disparate problem domains. The learnt models from measured data can be shown to contain useful information about the problem domains from which they were derived. The SASME evolved rule sets provide a method for elucidation of ecological information, providing knowledge about model predictions of ecosystem behaviours. This allowed

hypothesis to be derived about the causes of species successions from predictive models of those species. Such problems are common in the emerging field of ecoinformatics.

A diverse range of discrete structures can be used by the SASME algorithm. This allows representations to be constructed which elucidate problem information and which can still solve difficult problems. Relational information can be incorporated in rule sets and descriptive models produced which use those relationships. Non-Markovian problems can be addressed by the inclusion of evolved discounts of past output in the set of attributes with which the model makes predictions. The recurrent rule set models developed in this thesis proved to be useful tools for the discovery of controllers in a difficult non-Markovian control problem. The SASME algorithm provides the first comprehensible model of the two-pole control problem. The evolved model is nevertheless able to control the system as well as the neurocontrol approaches.

6.3 Final Words and Future Work

The proposed SASME framework has been employed exclusively for the evolution of rule lists with exceptions in this thesis. The design of the algorithm, however, is such that any discrete structure could be evolved. The implementation of the algorithm has been carried out in Objective-C using the Swarm libraries,[1] with the two different parts of the solution coded as separate objects. This allows any discrete structure to be used as long as the object matches the required interface. The structure object notifies the parameter object how many parameters have to be adapted, and how many discrete mutation methods are defined for the object. The evolution proceeds in the way outlined in Chapter 3. When there is no discrete object the program reduces to an evolutionary strategy algorithm for parameter optimisation.

Several discrete structures have been coded, or borrowed from libraries. Neural networks have been evolved by the algorithm using a set of discrete mutations similar to those defined for the rule sets,[2] however other types of mutations are possible, including partial training by back propagation as is done in EPNet [255], and modification of transfer functions. The rule set object has been implemented to use objects as its consequence part. This allows arbitrary models to be performed as the consequence of a rule. So far, linear equations and fixed topology neural networks have been coded.

Future developments of the structures evolved by SASME will be based on the kinds of problems which the algorithm is asked to solve. The idea behind the flexibility of choosing representations is that knowledge can be learnt by the algorithm and

[1] *http://www.swarm.org*

[2] Currently only addition and deletion of network nodes have been created.

communicated. The discrete structure should be chosen so that evolved solutions to problems are able to provide some insights into the problem which is being solved. The nature of the insights being searched for will drive the development of the representations.

Appendix A

Convergence Results for Evolutionary Strategies

This appendix considers some of the convergence results obtained for evolutionary strategies, and what they mean for parameter settings in the algorithm.

A.1 Global Convergence

These results consider convergence of the $(1+1)$ ES, that is the algorithm where one parent is replaced by it's single offspring if and only if the offspring has a better fitness. The extension of the first set of results to the $(\mu+\lambda)$ case are trivial since the population based results will be at least as good as the $(1+1)$ case. Unfortunately, there are few results for convergence of the (μ, λ)-ES. Note that a $(1,1)$-ES strategy is equivalent to random search.

The criteria of interest in analyzing convergence of the ES approach is how long it takes the algorithm to enter a vicinity of radius ϵ of the optima in question (global or local). So far results relating to the rate of convergence are only available in very limited examples.

Following Schwefel and Bäck [209], consider the $(1+1)$-ES starting at initial point $x^{(0)}$ with a mutation operator $x^{(t+1)} = x^{(t)} + Z$ where Z is a n-dimensional normally distributed vector with stochastically independent components along each of the axis with the same *constant* variance σ^2. If $\boldsymbol{I}_n$ denotes the n-dimensional unit vector then Z can be written $N(\mathbf{0}, \sigma^2 \boldsymbol{I}_n)$. The simplification of Equation 2.5 to the case of using a single variance σ is common in convergence results. It means that the lines of equal probability in the objective function update equation, Equation 2.5, are circles (n-spheres) and not ellipsoids.

The convergence theorem presented by Schwefel and Bäck requires some weak assumptions on the objective function. In practice these assumptions are very unlikely

not to hold to for problems of interest. An optimisation problem for which the ES convergence algorithm holds is called regular.

Definition A.1 (Regular Optimisation Problem) *The optimisation problem*

$$f(\boldsymbol{x}^*) = min\{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathcal{M} \subset \mathbb{R}^n\}$$

is regular if and only if

1. $f(\boldsymbol{x}^*) > -\infty$
2. $\boldsymbol{x}^* \in \text{int}(\mathcal{M})$
3. $\mu(\{\boldsymbol{x} \in \mathcal{M} \mid f(\boldsymbol{x}) \in U_\epsilon(f(\boldsymbol{x}^*)\}) > 0 \ \forall \epsilon > 0$

where μ is the Lebesgue measure, $\text{int}(\mathcal{M})$ *the set of internal points of $\mathcal{M}$, and U_ϵ an ϵ-environment of it's argument. One calls f the objective function, $f(\boldsymbol{x}^*)$ the global minimum, and $\boldsymbol{x}^*$ the solution to the optimisation problem.*

Requirement 1 is obvious. Requirement 2 is used to simplify the analysis without making any strong requirements on the objective function. The last requirement (3) prevents the objective function from having isolated global optima, which would not be able to be reached with a probability greater than zero [209].

The following theorem, which is a trivial application of the Borel-Cantelli lemma [37], is stated by Schwefel and Bäck, [209].

Theorem A.1 *Let $\epsilon > 0$ and $p_g = p(\boldsymbol{x}_{(g)} \in \{\boldsymbol{x} \in \mathcal{M} \mid f(\boldsymbol{x}) \in U_\epsilon(f(\boldsymbol{x}^*)\})$ be the probability that a population of the $(1+1)$ ES has reached the point $\boldsymbol{x}_{(g)}$ at iteration g, the objective function value belonging to which is closer to the goal $f(\boldsymbol{x}^*)$ than ϵ. Then, assuming*

$$\sum_{g=0}^{\infty} p_g = \infty \tag{A.1}$$

implies that

$$p(\lim_{g \to \infty} (f(\boldsymbol{x}_{(g)}) - f(\boldsymbol{x}^*) = 0) = 1$$

for any starting point $\boldsymbol{x}_{(0)} \in \mathcal{M}$.

Condition A.1 will be satisfied when the probability of generating a solution in the ϵ-neighbourhood of the optimal solution does not vanish. Condition A.1 in Theorem A.3 is made clearer in the following lemma [209]

Lemma A.1 *If $\mathcal{M} \subseteq \text{support}(f_Z)$, where f_Z denotes the probability density of the random vector Z of the mutation operator, and $\mathcal{M}$ is bounded, then Equation A.1 is valid.*

Where $\text{support}(f(x))$ is the set of values $\{x \mid f(x) \neq 0\}$. This result is an extension of a general random search result [10][page 87] [198]. The theorem states that

the optimum will be found with probability one given the reasonable criteria on the search space presented above. Lemma A.1 will also hold when the mutation variance is bounded.

It should be noted that Theorem A.3 only holds for $(1 + 1)$ selection, which implies $(\mu + \lambda)$ convergence. For (μ, λ)-ES the sequence of solutions F_g consisting of the best solution at generation g does not converge, although it does enter the ϵ-neighbourhood of the global optima infinitely often (see Theorem 2.2 in [200]). This happens because the necessary condition for global convergence, that a solution has a finite probability of generating an offspring in the vicinity of the global optima, will mean that any parents in the vicinity of the global optima in a (μ, λ)-ES have a probability of 1 of generating no children in that vicinity at some later generation. A simple hall of fame containing the best ever found solution (which would not be put back into later generations) would allow the construction of a sequence that would converge with probability 1 and would seem an intuitively obvious step when implementing a (μ, λ) algorithm on an optimisation problem. That is, the best ever solution is considered, and not the current populations best solution, as the result of the (μ, λ)-ES. This obvious strategy is employed throughout this thesis.

The results for convergence can be summarized by the fact that if any EA solution has a finite probability of generating any other solution then the ϵ-neighbourhood of the global optima will be entered with probability 1 after a finite number of iterations. The argument is identical to that for convergence of the elitist-GA and many other optimisation procedures. Some of the criteria can be weakened [200], although most implementations which can be shown to converge will do so under the conditions presented. For the ES method, the requirement that any solution has a finite probability of generating a solution in the region of the global optima can be made exact, and implies that the variance of the mutation distribution is bounded below (and above) by some fixed positive bound[1]. Although this was effectively proposed by Schwefel in terms of satisfying computational limitations and ensuring that the variance is not zero, it is rarely implemented in such a way on modern computers [201].

The problem of what happens in the limit when the variation is not bounded below was answered in the negative by Rudolph [201]. Rudolph constructs a fitness function which the ES provably does not reach the global optima with probability 1. He states the following theorem

Theorem A.2 *A randomly initialized* $(1 + \lambda)$*-EA with a self-adaptation method resembling the* $1/5$*-success rule does not converge with probability* 1 *to the global optimum of a continuous objective function in general.*

Where essentially the $1/5$-success rule means that the variance is not bounded below. The result holds for Cauchy or Gaussian mutation distributions. The result means that even elitist evolutionary algorithms with self-adaptation do not always enter

[1]The variance, σ, is required to be in a compact positive set

the region of the global optima in finite-time, in contrast to other evolutionary algorithms [201].

While comforting to know that as time tends to infinity the optima will be found, not many computer scientists have the patience to wait that long. Instead results on convergence speed are desirable. Global convergence results do not say as much as might be expected about the likely utility of an optimisation procedure. The problem is that proof of entering the global optima in finite time with probability 1 says nothing about the probability of entering the global optima in some fixed finite number of generations G. More important than global results are results which analyses expected convergence velocities.

To maintain the theoretical possibility of entering the region of the global optima, the self-adaptation must be updated to be bounded below. Any such lower bound will affect convergence velocities to an optima when the distance to the optima is of the same order as the lower bound. This suggests choosing the lower bound to be in the same order as the desired precision of the solution. While such a scheme would maintain in theory the convergence properties of the self-adaptive evolutionary algorithm, there is little practical reason to implement the scheme.

A.2 Convergence Speed of Evolutionary Strategies: The 1/5 Success Rule

To estimate convergence speed one needs to assume a fitness function. In the analysis of ES's a sphere model is the most frequently used nonlinear model for the objective[2](eg. [10, page 85]), that is

$$f_{\text{sphere}}(\boldsymbol{x}) = c_0 + c_1 \cdot \sum_{i=1}^{n} (x_i - x_i^*)^2 = c_0 + c_1 \cdot r^2$$

where $\boldsymbol{x}^* = (x_1^*, \ldots, x_n^*)$ denotes the minimum, r is the Euclidean distance between the trial solution $\boldsymbol{x}$ and the optimum solution $\boldsymbol{x}^*$ and c_0 and $c_1 \neq 0$ are arbitrary constants determining the position of the centre of the sphere in $\mathbb{R}^n$ and the "steepness" of the sphere respectively. The sphere model represents the simplest non-linear function, and it is argued that a sphere model can approximate the local topology of a local minimum of any fitness function. Sometimes the class of objective functions is extended to arbitrary strongly convex functions [199], however the sphere model will be used here for simplicity and brevity.

If the mutation operation is denoted by the application of some (stochastic) function

$$m(\boldsymbol{x}) = \boldsymbol{x}' \tag{A.2}$$

[2]objective function and fitness function may be used interchangeably

where the objective update equation,Equation 2.5, is simplified to consider only one variation parameter again, that is

$$x_i' := x_i + \sigma^{(g)} \cdot N_i(0,1) \quad \forall i \in \{1,\ldots,n\} \tag{A.3}$$

then the $(1+1)$ algorithm becomes:

$$x^{(g+1)} = \max(x^{(g)}, m(x^{(g)})) \tag{A.4}$$

The original convergence rate results for the sphere model were presented by Rechenberg[3] [185]. Rechenberg also considered a linear corridor model, where the objective function is modeled as

$$f_{\text{corridor}}(\boldsymbol{x}) = c_0 + c_1 \cdot x_1$$

where $\forall i \in \{2,\ldots,n\} : -b/2 \leq x_i \leq b/2$. The corridor model represents a corridor of width b where improvement is made only by moving along the x_1 axis [10, page 85]. The convergence rate of the stochastic ES is defined as the expectation of the distance γ covered towards the optimum by mutation (ignoring recombination), ie

$$\varphi = \int p(\gamma) \cdot \gamma \, d\gamma \tag{A.5}$$

where $p(\gamma)$ denotes the probability for a mutation to cover a distance γ towards the optimum.

Convergence rates are expressed in terms of dimensionless normalized quantities, $\sigma_1' = \sigma_1 \cdot n/b$, $\varphi_1' = \varphi_1 \cdot n/b$, $\sigma_2' = \sigma_2 \cdot n/r$ and $\varphi_2' = \varphi_2 \cdot n/r$. For the $(1+1)$-ES, and for $n \gg 1$ we have the following convergence rates for the two models [10, page 85][79, page125]:

$$\varphi_1' = \frac{\varphi_1'}{\sqrt{2\pi}} \left(1 - \sqrt{\frac{2}{\pi}} \frac{\sigma_1'}{n}\right)^{n-1} \approx \frac{\sigma_1'}{\sqrt{2\pi}} \exp\left(-\sqrt{\frac{2}{\pi}} \sigma_1'\right) \tag{A.6}$$

$$\varphi_2' = \frac{\varphi_2'}{\sqrt{2\pi}} \exp\left(-\frac{\sigma_2'^2}{8}\right) - \frac{\sigma_2'^2}{4} \left(1 - \operatorname{erf}\left(\frac{\sigma_2'}{\sqrt{8}}\right)\right) \tag{A.7}$$

where $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt$ is called the error function [10, page 85]. These results are obtained by applying Equation A.5 i.e., by integrating the useful distance covered by the mutation and the probability of covering that distance over the success area for mutations. Note that there is no chance of the solution not improving, since the elitist $(1+1)$ strategy is being employed.

Using Equation A.6 and Equation A.7 it is possible to find the optimal standard deviations $\sigma_i'^*$ that maximize the convergence rate and the corresponding optimal convergence rate $\varphi_i'^*$ by setting

$$\left.\frac{d\varphi_i'}{d\sigma_i'}\right|_{\sigma_i'^*} = 0$$

[3] They are presented in Rechenberg's doctoral thesis, which is published in German and unsighted by the author, who can't read German anyway. The work is extensively cited and reproduced in English by Schwefel, [207, 208], and Bäck, [10] and others.

$i \in \{1,2\}$ [10, page86]. Rechenberg also calculated the expected values for the probabilities of an improvement occurring, i.e.

$$p_i = \mathcal{P}\{f_i(m(\boldsymbol{x})) \leq f_i(\boldsymbol{x})\} \qquad (i \in \{1,2\};\ n \gg 1)$$

with the mutation function $m(\cdot)$ in Equation A.2. Equation A.8 is the probability of the child replacing the parent in Equation A.4. The resulting probabilities for the corridor and sphere models are:

$$p_1 = \frac{1}{2}\left(1 - \sqrt{\frac{2}{\pi}}\frac{\sigma_1'}{n}\right)^{n-1} \approx \frac{1}{2}\exp\left(-\sqrt{\frac{2}{\pi}}\sigma_1'\right) \tag{A.8}$$

$$p_2 = \frac{1}{2}\left(1 - \operatorname{erf}\left(\frac{\sigma_2'}{\sqrt{8}}\right)\right) \tag{A.9}$$

Using σ'^* and φ'^* in Equation A.8 and Equation A.9, it is possible to find the optimal success probabilities p_i^*. The values found by Rechenberg are [185][10, page86]:

$$\begin{array}{lllllll} \sigma_1'^* & = & \sqrt{\frac{\pi}{2}} & \approx & 1.253 & \sigma_2'^* & \approx & 1.224 \\ \varphi_1'^* & = & \frac{1}{2e} & \approx & 0.184 & \varphi_2'^* & \approx & 0.2025 \\ p_1^* & = & \frac{1}{2e} & \approx & 0.184 & p_2^* & \approx & 0.270 \end{array} \tag{A.10}$$

The 1/5-success rule is based on these results. By noting that the application of optimal step sizes results in a (optimal) success probability, $p_i'^*$ of $\approx 1/5$, it can be seen that an algorithm that maintains a probability of generating a successful offspring of $\approx 1/5$ will maintain an optimal rate of convergence, $\varphi_i'^*$ for both model objective functions, i.e. $i \in \{1,2\}$. Rechenberg used the term *evolution window* to describe the order of magnitude that the step size can vary within and still maintain a convergence velocity at least 1/2 that of the optimal. For the model functions in Equation A.7 and Equation A.6 it is found that the step size can vary an order of magnitude from the optimal and maintain an acceptable rate of convergence. This allows us to say that the algorithm is not particular sensitive to the value of the step size when it is around the optimal [10, pages 86–87], and the approximation of 1/5 is sufficient. The alternative would require a different probability of success to be obtained to each new problem in order to maintain a reasonable rate of convergence.

The 1/5-success rule is theoretically interesting, but it has a number of problems in practice. If the objective function is such that the success rate is never above 1/5 the search will stagnate. It also provides no mechanism by which individual step sizes σ_h can be handled so that the mutation vector can be scaled to the scale of the relevant axis.

A.3 Convergence Speed of Evolutionary Strategies: The $(\mu \dot{+} \lambda)$-ES

Schwefel extended the convergence rate theory to apply to the $(1, \lambda)$-ES strategy, with a single standard deviation which does not change, i.e.

$$\hat{\boldsymbol{x}}^{(h)} = m(\boldsymbol{x}^{(h)}) \quad \forall h \in \{1, \ldots, \lambda\}$$

where the mutation of each component of $\boldsymbol{x}^{(h)}$ is:

$$m(x_i^{(h)}) = x_i^{(h)} + \sigma \cdot N(0, 1)$$

and the selection method for the $(1, \lambda)$ method means that Equation A.4 will become:

$$\hat{\boldsymbol{x}} = \max(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\lambda)})$$

where $\hat{\boldsymbol{x}}$ will be the parent vector of the next generation. The results are extended by looking at the distance that the average of the μ best offsprings makes towards the optimum, and the theory derived by Schwefel was general enough to account for the $(\mu \dot{+} \lambda)$-ES.

In this case, let $p_i(\gamma)$ be the probability that the ith best offspring covers the distance γ towards the optimum, then the probability that the average of the best μ offspring will cover a distance γ towards the optimum will be

$$p(\gamma) = \frac{1}{\mu} \sum_{i=1}^{\mu} p_i(\gamma)$$

The analog of Equation A.5 will then become

$$\varphi = \int_{\gamma=\gamma_{\min}}^{\infty} p(\gamma) \cdot \gamma \, d\gamma \tag{A.11}$$

where $\gamma_{\min}$ will be 0 in the $(\mu + \lambda)$ case, to reflect the fact that the elitist method does not ever have make negative progress, and it will be $-\infty$ for the (μ, λ) case. Schwefel presents the following expression for the probability of the ith individual covering a distance of γ towards the optimum:

$$p_i(\gamma) = \lambda C_{i-1}^{\lambda-1} \cdot p_{\gamma_j=\gamma} \cdot p_{\gamma_j<\gamma}^{\lambda-i} \cdot p_{\gamma_j>\gamma}^{i-1} \tag{A.12}$$

where $p_{\gamma_j=\gamma}$ is the probability of offspring number j covering the distance γ exactly, $p_{\gamma_j>\gamma}$, is the probability of j covering a distance greater than γ and $p_{\gamma_j<\gamma}$ is the probability of it covering less. The interested reader is referred to Bäck for more discussion [10, pages 88-89]. Combining Equation A.11 and Equation A.12 allows the statement of Schwefel's convergence rate theorem:

Theorem A.3 (Convergence velocity of $(\mu \dot{+} \lambda)$-ES) *Let $\varphi_{(\mu \dot{+} \lambda)}$ denote the expectation of the progress rate of the population average for a $(\mu \dot{+} \lambda)$-ES using one*

single standard deviation and no recombination or self-adaptation. Assuming a minimization task, $\varphi_{(\mu\dagger\lambda)}$ *can be calculated according to*

$$\varphi_{(\mu\dagger\lambda)} = \frac{\lambda}{\mu} \int_{\gamma=\gamma_{\min}}^{\infty} \left[\gamma \cdot \sum_{i=1}^{\mu} \mathrm{C}_{i-1}^{\lambda-1} \cdot p_{\gamma_j=\gamma} \cdot p_{\gamma_j<\gamma}^{\lambda-i} \cdot (1 - p_{\gamma_j<\gamma})^{i-1} \right] d\gamma \qquad \text{(A.13)}$$

where $\gamma_{\min} = 0$ *for a* $(\mu + \lambda)$*-ES and* $\gamma_{\min} = -\infty$ *for a* (μ, λ)*-ES.*

Unfortunately Equation A.13 in Theorem A.3 can't be solved to calculate analytical convergence velocities in most cases, including the sphere and corridor models.

In the case of the $(1, \lambda)$-ES, Schwefel was able to calculate the maximum of the ratio φ_i^*/λ_i and arrived at $\lambda \approx 5$ for the sphere model and $\lambda \approx 6$ for the corridor model. This led Schwefel to suggest a ratio of μ/λ of at least $1/7$ to maintain a sufficient convergence speed [10, pages 90-91].

Beyer provides some results for (μ, λ)-ES for the N-dimensional sphere model without recombination [23], and with recombination [21]. Beyer also extends the convergence rate ideas in Theorem A.3 to account the self-adaptive $(1, \lambda)$-ES[4] [24]. The self-adaptive theory confirms Schwefel's setting of the learning rate parameter $\tau \propto 1/N$, and Beyer hypothesizes that Schwefel's empirical finding that self-adaptation works best when intermediate recombination of the strategy parameters is employed (see discussion and footnote on page 21) might be explained by his *genetic repair* hypothesis of intermediate recombination [21, 24].

[4]With a single self adaptive parameter σ.

Appendix B

No Free Lunch Theorems

This appendix presents the no free lunch theorem, and discusses some of its implications.

Evolutionary algorithms are usually applied when existing algorithms cannot be applied, or they simply fail to give satisfactory answers or fail to supply answers in reasonable time frames. The performance of a given search, optimisation or learning algorithm on a given problem is usually unknown. The computability and complexity of a given problem are an estimate of how hard it is to solve a problem.

A fundamental concept in computational complexity is the Church-Turing Hypothesis:

Hypothesis B.1 (Church - Turing Hypothesis) *The class of decision problems that can be solved by any reasonable model of computation is exactly the same as the class of decision problems that can be solved by Turing machine programs.*

A number of related points can be noted:

1. If a given problem can be solved by a finite procedure (ie. is computable, see note on page 2) then it can be solved by a universal Turing machine.
2. It is not necessary to build a new machine for solving every new problem.
3. Modern computers are essentially universal Turing machines.

The Church-Turing result gives us optimism that evolutionary algorithms can solve a lot of computable problems by automatically generating solutions on a Turing equivalent computational device. The No Free Lunch (NFL) result reduces our hubris somewhat.

The NFL theorem by Wolpert and Macready [248] sets up a mathematical framework in which to evaluate search algorithms and then proceeds to show that without domain specific information, there is no way to justify claims of one search algorithm's efficiency over another. The theorem can be stated simply:

Theorem B.1 (No Free Lunch) *For any pair of algorithms a_1 and a_2,*

$$\sum_f P(\, d_m^y \mid f, m, a_1) \;=\; \sum_f P(\, d_m^y \mid f, m, a_2)$$

where a_i, $i \in \{1, 2\}$ is an algorithm mapping previously visited points in the search space and producing a new, unvisited point d; $P(\, d_m^y \mid f, m, a_1)$ is the conditional probability of generating a particular previously unvisited point d_m after iterating a_i for m iterations on a cost function f. Then Theorem B.1 says that the conditional probabilities for visiting any two points will be the same over all cost functions f regardless of the algorithms chosen a_i. An obvious corollary of Theorem B.1 is that given any performance measure $\Phi(d_m^y)$, the average over all cost functions f of $P(\Phi(d_m^y) \mid f, m, a)$ is independent of the algorithm used a. In other words, no algorithm performs better than all others on all cost functions, and on average, they all perform equally [78].

Consider two finite spaces $\mathcal{X}$ and $\mathcal{Y}$. Each point in $\mathcal{X}$ has a cost associated with it which is in $\mathcal{Y}$ and is given by $f(x)(\in \mathcal{Y})$. f is a single valued function $f : \mathcal{X} \to \mathcal{Y}$. Wolpert and Macready [247] consider all functions which take values from $\mathcal{X}$ to $\mathcal{Y}$. On two modest sized spaces of 100 elements each, this would represent 100! different possible functions. If a search algorithm knows nothing about the functions it is searching, then it could be confronted with any one of these functions. Wolpert and Macready show that all blind search algorithms perform exactly the same in this situation.

NFL results say something intuitive about blind search, and some ramifications will now be informally discussed. Culberson relates the following anecdote [51]:

> In the movie *UHF*, there is a marvelous scene that every computing scientist should consider. As the camera slowly pans across a small park setting, we hear a voice repeatedly asking "Is this it?" followed each time by the response "Nah!". As the camera continues to pan, it picks up two men on a park bench, one of them blind and holding a Rubik's cube. He randomly gives it a twist, then holds it up to his friend to repeat the question/answer sequence yet again.

The depicted scene typifies the concept of blind search. The searcher has minimum information about the space he is searching in. All search strategies which propose only new states of the cube will perform equally in terms of the expected number of iterations before the problem is complete.

When the environment returns a value for the state that the problem solver has proposed we are still no better off. To see this, consider instead of a fixed problem an adversary who assigns a value to each proposed solution. The value will be the value previously assigned if the solution has been previously proposed (something which the conditions of the NFL, Theorem B.1, forbids, since algorithms which go over old ground can be provably demonstrated to be inferior to algorithms which only ever test new ground!). If the solution has not been previously proposed, the adversary assigns it a random value. If the adversary chooses from an appropriate distribution

than it can generate with equal probability a function from the class of functions of interest. It is reasonable to suspect that there will be no best search method when confronted with the adversary. By assuming nothing about the function being searched the NFL theorem essentially allows an adversary to choose the evaluations of solutions.

Appendix C

Classifier Systems: The Michigan Approach

This appendix describes Hollands learning classifier system (LCS).

The LCS was designed to operate in a reinforcement learning environment [118, 130, 165, 216], such as that in Figure 2.4, page 26. A reinforcement learning environment is one where the environment returns information about a models output at some point. The only information available to the learning algorithm is therefore *performance related.* A classifier consists of a DETECTOR:MESSAGE pair. The DETECTOR attempts to match symbols presented to the classifier by the environment and the MESSAGE part of the classifier specifies an action to perform if the classifier is successfully activated. The MESSAGE part of the classifier consists of a fixed length string from the ternary alphabet $\{0, 1, \diamond\}$. The $\diamond$ symbol represents a don't care at that loci of the string. The binary alphabet was chosen to maximize schema processing. Figure C.1 shows an example pattern and some of the detectors that would match it.

Holland likens the system to an office where at the beginning of each day there are a number of memo's on a message list to be processed. Each classifier corresponds to a desk that can process a certain type of memo. At the end of each day, the desks perform an action based on the memo's processed that day. An action can be either posting a message on the message list for the next days processing, or

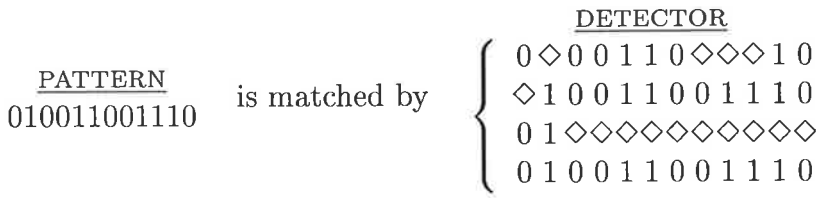


Figure C.1. *An example of a pattern and the classifiers that would match the pattern*

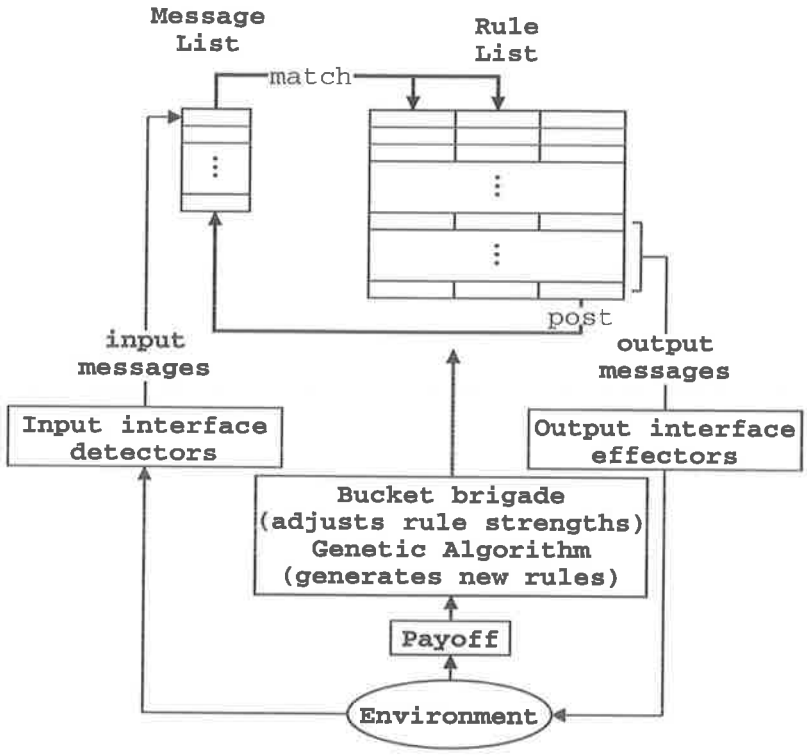


Figure C.2. *A Holland style classifier system ([118, page 464])*

performing some action on the environment via the effectors. The environment is assumed to give only sparse, intermittent payoff to the learning algorithm. The payoff is divided amongst the classifiers by an algorithm known as the *bucket brigade* algorithm. New rules are generated from old by a genetic algorithm. The situation is shown in Figure C.2, [118].

Each matching DETECTOR is assigned a *bid* value depending on how specific the classifier is and the expected payoff of the classifier. The number of wildcard elements in the classifier determines how specific it is—the more $\diamond$ symbols the less specific. The more specific the classifier, the higher it's bid. The expected pay off, or strength, of the rule is decided by the credit apportioning algorithm, the bucket brigade, and depends on how successful the rule has previously been in getting payoff from the environment. The bid b_i of rule i is usually a proportion of the rules strength s_i, where the proportion is dependent on the rules specisifity, specisifity_i, and is usually set to:

$$\text{specificity}_i = \frac{\text{Number of specified bits in rule } i}{\text{Number of bits in the rules}}$$

and the bid strength of rule i would then be:

$$b_i = \text{specificity}_i \cdot s_i$$

In the CS-1[1] classifier presented by Holland [115], the top ten classifiers are chosen

[1]CS stands for Cognitive System [115]

and randomly weighted according to bid strength to determine the winning classifier. The winning classifier has it's MESSAGE performed.

The bucket brigade algorithm works as follows. If a rules bid is successful then its strength is reduced by an amount equal to the bid:

$$s_i(g+1) = s_i(g) - b_i \tag{C.1}$$

All rules which posted messages to the message list that were matched by rule i get their strength increased by the bid of rule i, so that if rule j had posted a message matched by i it would have:

$$s_j(g+1) = s_j(g) + \frac{b_i(g)}{n} \tag{C.2}$$

where n is the number of classifiers which posted messages that are matched by i [122, section 4.1.5]. Note that the message list is cleared at the end of each time period ("day"). The analogy used by Holland is that of an economy where information is bought and sold by the rules and payments are made from the rules strength. The system recoups strength via reinforcement from the environment. If rule i performs an action on the environment that results in some reward, it has its strength boosted by an amount equal to that award. As rules grow stronger, they make stronger bids [118].

The bucket brigade algorithm is motivated partly to help the formation of *default hierarchies*. Default hierarchies are discussed in section 2.3.2 on page 30. The bucket brigade promotes the coupling of different rules by dividing payoff amongst rules with the bidding system. The algorithm promotes the formation of increasingly specific, accurate rules.

The generation of rules in a Holland style classifier is conducted by a GA which periodically replaces low strength rules in the population with new rules based on the current rules in the population [118]. The use of the internal memory structure is not always necessary. DeJong notes three reasons why an internal memory (message list) might not be used [57, page 630]:

1 The application does not need the extra computational complexity.
2 The halting problem: how many internal actions are allowed to be performed before the next external action must be?
3 Most of the work in traditional machine learning does not use an internal memory.

An internal memory will increase the complexity of the model and make it a more powerful computational engine than a simpler "stimulus-response" inference engine.

Nearly every part of the suggested classifier system has been altered and alternative methods tested [246].

1 Pattern Matching. Instead of detectors matching exactly the symbols supplied by the environment, partial matches are possible.

2 Credit Assignment. Many different credit assignment methods have been tried, such as: Methods which reward all classifiers who were active since the last payoff event equally *(epochal)*; methods where classifiers pay a part of there strength to classifiers active on the previous step *(implicit bucket brigade)*.

Although default hierarchies have been observed in applications [246, 94], it has been noted that their development and stability can be problematic [197, 215]. When a rule that forms part of a hierarchy is modified it can cause the complete collapse of the entire hierarchy. Rules which occur early in a long hierarchy chain are difficult to reinforce because of the number of iterations required to move payoff up the chain to the early classifiers [246].

Holland's classifier system is a complicated structure with many interacting processes determining its success. Wilson writes [240]:

> ...efforts to realize the framework's potential have met with mixed success, primarily due to difficulty understanding the many interactions of the classifier system that Holland outlined.

Despite this, there are a range of applications which Holland style classifiers have successfully been applied to such as control problems [94], letter recognition problems [87] (but see [69]) and others [246]. In response to the complexity of the LCS suggested by Holland, Wilson proposed a simplified "zeroth level" classifier system, ZCS [240]. ZCS simplifies Holland's classifiers by removing the message list, simplifying the bucket brigade, removing the bid competition and removing the measurement of specificity from the system. Wilson asserts that specificity is not desirable in classifier systems [239], and ZCS does not use it.

Wilson's credit assignment algorithm is related to the machine learning method of (tabular) Q-learning [240, 61]. Q-learning and temporal difference (TD) methods share a common goal with CFS research, although stemming from two different traditions in artificial intelligence research, the behaviourist tradition in the case of TD and Q-learning and the symbolic approach in the case of CFS [61]. When a classifier system is drastically simplified by

1 removing its internal message list.
2 forbidding "don't care" symbols (ie. the $\diamond$ symbol in Figure C.1 on page 163).
3 allowing only one condition and one action on each classifier.
4 allowing the population to contain every possible condition-action pair (so that a GA is not required).

and using the implicit bucket brigade where a classifier is selected probabilistically from the match set on the basis of its strength and pays a proportionate amount of its strength to the previously fired classifier, while receiving a proportion of the subsequent classifiers strength, ie.:

$$s_i(g+1) = (1-\alpha) \cdot s_i(g) + R + \alpha \cdot s_j(g+a) \tag{C.3}$$

where $s_i(g)$ is the strength of classifier i at generation (iteration) g, α is the proportion of a classifiers strength that gets paid to other classifiers, R is the reward from the environment after application of classifier i's action and classifier j is the

classifier whose action is performed at the next time step $g + 1$. It is beyond the scope of this thesis to review the Q-learning literature, however it is interesting to note that Equation C.3 corresponds to the Q-learning update formula with the discount rate γ set to one and the strength (Q-value) of an action updated by the action which is actually performed at $g + 1$ rather than the action corresponding to the maximum Q-value. These changes are minor, and Dorigo *et al* argue that Q-learning extensions often make the Q-learning algorithm more similar to the full CFS algorithm [61].

Wilson extends the ZCS classifier system to update the method in which classifiers are evaluated. In an algorithm referred to as XCS, Wilson allows each classifier to maintain a prediction of its accuracy, but the fitness of the classifier is based on a measure of the classifiers prediction estimate accuracy [241, 242, 243, 245, 244, 42]. The strength parameter is removed, and classifiers are chosen from the action set on the basis of their predicted value. Results of using the method indicate that XCS can generate more general rules than ZCS. Wilson also modifies the method the GA uses to generate new rules.

Extensions to the LCS algorithm's mentioned so far include using fuzzy sets and rules for the classifiers [50, 224, 171, 35, 36], real values for classifiers [244, 36], messy codings [149] and S-expressions for rule conditions [150].

To the authors knowledge, Michigan style learning classifiers have only been implemented using the genetic algorithm evolutionary paradigm.

Appendix D

Schema Theorem and Representation

The defining feature of genetic algorithms is the choice of binary strings to represent solutions. This choice is popular due to Holland [117], although others, including Bremerman [76, page 311][41] and Friedberg [88, 89], also used a binary representation.

D.1 The Genetic Algorithm

Holland [117] proposed modeling the operators and structures of a search method off an idealized understanding of genetic processes. The chosen representation most closely simulates a unicellular, *haploid* organism [10, page 16]. He called his method genetic algorithms (GAs). A population of solutions are modified by crossover, mutation and inversion operators and the best of the resulting solutions are chosen for the next generation. The operators are a simplification of the more complex behaviour which underlies natural genetic systems [170].

The essential components of the GA in common with all evolutionary methods are reproduction, mutation, competition and selection. The unique aspect, as proposed originally by Holland, is the representation of solutions as binary strings. The canonical genetic algorithm is implemented as follows:

1 To implement a genetic algorithm the problem of interest must be able to be defined as a fitness function, $F(\cdot) \in \mathbb{R}$, that represents the value of any given solution. The value assigned is clearly dependent on what the problem is and why we are solving it. Without loss of generality, we will assume the problem to be to find $\max(F(\cdot))$[1]

[1] A minimization problem redefines $F(\cdot)$ to be $-F(\cdot)$, although the roulette wheel selection method requires $F(\cdot) > 0$ this is easily remedied by choosing the probability of selection to be $1-p$ where p is the calculated probability. If $\exists\ \boldsymbol{x}, \boldsymbol{y} : F(\boldsymbol{x}) < 0, F(\boldsymbol{y}) > 0$ then some other trick needs

2 A population, $\mathcal{P}(0)$, of P candidate solutions,

$$\boldsymbol{x}^{(k)} \in \mathcal{P}(0),\ k \in \{1, 2, \ldots, P\}$$

are generated as binary strings, $x_i^{(k)} \in \{0, 1\}$. This typically involves forcing the representation onto the problem. For example [71], if the problem were to find a scalar y which minimizes

$$g(y) = y^3$$

then a finite range of values of y would be selected and then the minimum value would be assigned $\{0, 0, \ldots, 0\}$ and the maximum would be $\{1, 1, \ldots, 1\}$. The desired precision would then be used to determine the size of the solution vectors, ie. $\boldsymbol{x}^{(k)} \in \{0, 1\}^n$. Biologically inspired terminology is usually used. Solutions are referred to as *chromosomes*. Bits in the chromosome are referred to as *genes* and the alternative values at a position, or *locus*, are referred to as *alleles*. The binary string is the *genotype* of the solution and in this example the real value y which the binary string $\boldsymbol{x}^{(k)}$ codes for is the *phenotype*.
If $\mathcal{D}$ is the domain of interest of $F(\cdot)$, then let $d : \{0, 1\}^n \to \mathcal{D}$, where d is usually not one-to-one or onto the domain of $F(\cdot)$. We let the function $f(\cdot)$ be the function resulting from applying the fitness function $F(\cdot)$ to the transformed binary string, ie $f(\boldsymbol{x}) = F(d(\boldsymbol{x}))$, $\boldsymbol{x} \in \{0, 1\}^n$. We also refer to $f(\cdot)$ as the fitness function.

3 The solution vectors in the population,

$$\boldsymbol{x}^{(k)} \in \mathcal{P}(g),\ k \in \{1, 2, \ldots P\},\ g \in \mathbb{N}$$

are decoded and assigned a fitness, $f(\boldsymbol{x}^{(k)})$, according to the objective function. $g \in \mathbb{N}$ is the generation number.

4 Each chromosome is assigned a probability of reproduction proportional to it's fitness relative to the fitness of the other solutions in the population $\mathcal{P}(g)$. If $\forall \boldsymbol{x} : f(\boldsymbol{x}) > 0$, then this can be accomplished by *roulette wheel* selection where the probability of selection of $\boldsymbol{x}^{(k)} \in \mathcal{P}(g)$ is

$$p_k = \frac{f(\boldsymbol{x}^{(k)})}{\sum_{j=0}^{P} f(\boldsymbol{x}^{(j)})},\ \boldsymbol{x} \in \mathcal{P}(g)$$

5 With probability p_k a solution is chosen for reproduction. Chosen chromosomes reproduce by the genetic operators of mutation, crossover and (rarely) inversion, Figure D.1.

Mutation With a set probability p_{mut} a bit in a solution $\boldsymbol{x}^{(k)}$ is flipped. p_{mut} is often set to an arbitrary small value between 0.01 and 0.001 [71].

Crossover With probability p_{cross} crossover is performed. In a one-point crossover operation a uniformly random number, r, between 1 and n is chosen. The first r bits of a chosen solution are combined with the last

to be used.

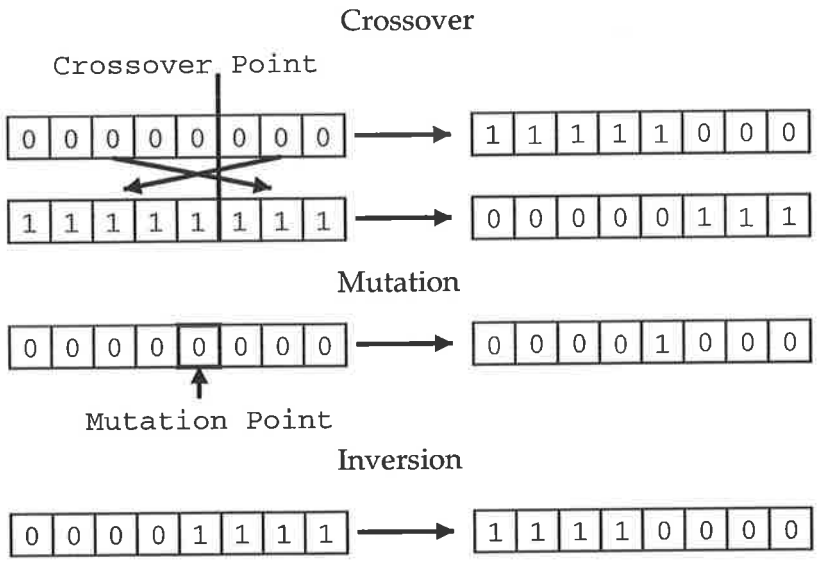


Figure D.1. *Genetic operators for binary representations.*

$n - r$ bits of another chosen solution to create a new child offspring. A second child offspring is created from the remaining bits. In a two-point crossover operation two randomly chosen points are used. In uniform crossover two offspring are created where with probability 0.5 each bit of the first offspring comes from one of the parents. The second offspring contains the bit from the parent not contributing to the first. p_{cross} is usually arbitrarily set between 0.6 and 0.95 [71].

Inversion With a set probability p_{inv} a part of a chromosome has it's order reversed. Inversion is rarely used because it rarely produces offspring with high fitness, it was, however, suggested in Holland's original work [117].

6 The genetic algorithm is terminated when a desired solution is found or when a set number of generations have elapsed.

The classical argument about how genetic algorithms work is based on three components:

1 A large population of solutions are initialized randomly to provide random sampling of the search space
2 Individuals with a high fitness are preserved through selection. This biases sampling of the search space towards areas of higher than average fitness
3 Portions of different strings, called "building blocks" are combined onto the one string by the process of crossover, thereby exploiting the parallelism provided by maintaining a population of solutions

We now review the details of the above argument.

D.2 The Schema Theorem

The motivation behind the use of a binary representation is inextricably intertwined with the notion of a schema. Consider an alphabet of symbols A with $\# \notin A$ (using the notation of Holland [117]). We consider fixed length strings from the alphabet A and we define the # symbol to be a wild card or don't care symbol, meaning that we do not care what symbol is at that particular loci. The set of fixed length strings formed by the union of the alphabet A and the symbol # are referred to as schema.

Consider the alphabet $A = \{0, 1\}$ and the schema $\{00\#\#\}$. The schema $\{00\#\#\}$ represents all of the strings $\{0000\}, \{0010\}, \{0011\}$ and $\{0001\}$. Holland recognized that the fitness of an individual with schema $\{00\#\#\}$ gave an implicit evaluation of the expected fitness of the schema. That means that an evaluation of $\{0000\}$ gives some information about the fitness of $\{00\#\#\}$, $\{\#0\#\#\}$, $\{\#0\#0\}$ and so on [71]. Holland called this *implicit parallelism* because it implied that a single evaluation could give information about a large number of schema [81] at the same time (ie. in parallel).

Definition D.1 (A Schema) *A schema is a similarity template describing a subset of strings with similarities at certain string positions*[2]

A binary representation of length n will have 3^n possible different schema, since it is an instantiation of the ternary alphabet $\{\#, 0, 1\}$. A binary string of length n describes 2^n schema, since each bit can have one of two possible values, either it's binary value or the # symbol. A population of P chromosomes will then be sampling between 2^n and $P \cdot 2^n$ different schema, depending on the number of different alleles in the population[3]. Where m is the number of # symbols in the schema, the order of the schema is defined to be the number of fixed positions in the schema, or $n - m$ [10].

Definition D.2 (Order of Schema) *The order of a schema is equal to the number of fixed positions in the schema, ie* $o : \{0, 1, \#\}^n \rightarrow \{0, 1, \ldots, n\}$ *such that:*

$$o(H) = \mid \{i \mid h_i \in \{0, 1\}\} \mid$$

For a binary string of length n, there are[4]

$$C_m^n$$

[2]Schemata plural of schema can be illustrated as hyperplane partitions of the n-dimensional hypercube where each vertex represents one of the 2^n different possible states of the individuals [22].

[3]In fact the number of schema sampled will be strictly less than $P \cdot 2^n$ whenever $P > 1$.

[4]C_m^n is the combinatorial operator, $C_m^n = \frac{n!}{(n-m)!\ m!}$, also written $\binom{n}{m}$

different possible schemae of order m represented [57, page 619]. Hence a single solution can implicitly evaluate a considerable number of different schema (hyperplanes, partitions) in the search space.

Holland proposed that different representations in genetic algorithms could be compared by considering the number of schema that the algorithm would process using that representation. For example, a decimal string of length 6 can represent $10^6 = 1\ 000\ 000$ distinct objects. Similarly a binary string of length 20 can represent about the same number of objects $2^{20} = 1048576$. However the number of possible schemata processed in a decimal string will be $11^6 = 1771561$ where as the binary string will process $3^{20} = 3.49 \times 10^9$ [117, 81]. Holland proved that schema processing is maximized when $|A| = 2$ and a binary representation was suggested as the universal representation [117, 71].

The problem was then to decide how to allocate trials amongst the different schema. Holland likened the problem to the gaussian two-armed bandit problem [116][117, page 75–83][79, page 115]. In the gaussian two-armed bandit problem a poker machine[5] with two arms is considered which provides a different normally distributed random payoff upon pulling either arm. The payoff from the first arm has mean μ_1 and variance σ_1^2, and the second arm has μ_2 and σ_2^2. The problem is how to best allocate trials between the two arms in order to minimize the expected loss from pulling the wrong arm. If we perform n_1 pulls of the first arm and n_2 pulls of the second, then we will have an observed mean payoff of $\bar{x}_1$ for the first arm and $\bar{x}_2$ for the second. If we let the function $q(n_1, n_2)$ be the probability that after pulling arm 1 n_1 times and arm 2 n_2 times the observed mean value for the first arm is greater or less than the observed mean value of the second when the true mean is not, ie

$$q(n_1, n_2) = \begin{cases} \Pr(\bar{x}_1 > \bar{x}_2) & \text{if } \mu_1 < \mu_2 \\ \Pr(\bar{x}_1 < \bar{x}_2) & \text{if } \mu_1 > \mu_2 \end{cases} \tag{D.1}$$

then the two-armed bandit problem is to minimize the expected loss function,

$$L(n_1, n_2) = [q(n_1, n_2)n_1 + (1 - q(n_1, n_2))n_2] \cdot \mid \mu_1 - \mu_2 \mid \tag{D.2}$$

Holland proved the following theorem about minimizing equation D.2 [116][117, pages 77–78][94, page 37][79, page 116]

Theorem D.1 *Given N trials to be allocated to two random variables with means $\mu_1 > \mu_2$ and variances σ_1^2 and σ_2^2, respectively, and the expected loss function described in equation D.2, the minimum expected loss results when the number of trials allocated to the random variable with the lower observed average payoff is*[6]

$$n^* \sim b^2 \, ln \left[\frac{N^2}{8\pi b^4 \, ln \, N^2} \right] \tag{D.3}$$

[5] Referred to as a slot machine in the USA

[6] the asymptotic notation $A(t) \sim B(t)$ indicates that for arbitrary functions $A(t), B(t)$ of the same variable t, we will have $\lim_{t \to \infty} \frac{A(t)}{B(t)} = 1$ [117][37, page 37]

where $b = \sigma_1/(\mu_1 - \mu_2)$. *The number of trials to be allocated to the random variable with the higher observed average payoff is* $N - n^*$.

The proof is long and can be found in Holland [116][117, pages 78–83]. Equation D.3 can be rewritten as

$$N - n^* \sim N \sim \sqrt{8\pi b^4 \ln N^2} \cdot e^{\frac{n^*}{2b^2}}$$

indicating an exponentially growing number of trials $N - n^*$ to be allocated to the observed best (see [117, page 83]) [10, page 127].

The above analysis will also hold for the k-armed bandit. Holland likened the problem of allocating trials to schema to that of allocating trials in the k-armed bandit problem. In order to minimize the expected loss, it was proposed to allocate an exponential number of trials to the observed best solutions in order to minimize the expected loss while sampling schema [79, page 117][94, page 30–31].

Reproduction will increase the frequency of successful schema in the population by selecting those individuals with higher probability. Mutation will occasionally disrupt schema and is seen as a slow background operator which ensures that no schema are irrevocably lost to the population. The effect of crossover will depend on the *defining length* of the schema.

Definition D.3 (Defining Length of Schema) *The defining length of a schema is defined to be the number of symbols between the first non-# symbol and the last non-# symbol, ie* $\delta : \{0, 1, \#\}^n \rightarrow \{0, 1, \ldots, n-1\}$ *such that*

$$\delta(H) = max\{i \mid h_i \in \{0, 1\}\} - min\{i \mid h_i \in \{0, 1\}\}$$

Short low order schema are less likely to be disrupted than longer schema by the typical one or two point crossover operators. This leads to the central theorem for genetic algorithms and how they work, the schema theorem [117, 94][10, page 125–126], which describes the change in the number of copies of different schema in the population during the execution of the genetic algorithm described on pages 169–171, with one-point crossover.

Theorem D.2 (The Schema Theorem) *Short low order above average schemata receive exponentially increasing trials in subsequent generations.*

$$x_i(g+1) \geq x_i(g)\frac{f(\boldsymbol{I_i})}{\overline{f}}\left(1 - p_{\text{cross}}\frac{\delta(\boldsymbol{I_i})}{n-1}\left[1 - \frac{x_i(g)}{P}\right]\right)(1 - p_{\text{mut}})^{o(\boldsymbol{I_i})} \quad \text{(D.4)}$$

where $x_i(g)$ is the number of copies of schema $\boldsymbol{I_i}$ at generation g. P is the population size. f_i is the average fitness of the fixed length binary strings containing schema $\boldsymbol{I_i}$. $\overline{f}$ is the average fitness of the entire population, that is

$$\overline{f} = \frac{\sum_i x_i f_i}{\sum_j x_j} \quad \text{(D.5)}$$

n is the length of the bit string. $o(\boldsymbol{I}_i)$ is the number of non # symbols in the schema $\boldsymbol{I}_i$, ie the order of the schema, Definition D.2. $\delta(\boldsymbol{I}_i)$ is the defining length of schema $\boldsymbol{I}_i$ (Definition D.3) and p_{cross} and p_{mut} are the probabilities of mutation and crossover respectively.

The proof of the schema theorem is straight forward and can be found in Holland [117], Goldberg [94, page 28–33], Bäck [10, pages 123–126] and elsewhere. Theorem D.2 says that the observed number of a particular schema in the next generation will increase (decrease) according to a multiplicative equation, Equation D.4, which depends on the fitness and order of the schema. In particular, schema with low order (short) and above average fitness will increase in number exponentially as the number of generations increase. Equation D.4 will increase the number of copies of schema $\boldsymbol{I}_i$ when $f(\boldsymbol{I}_i) > \overline{f}$ and $\delta(\boldsymbol{I}_i)$ and $o(\boldsymbol{I}_i)$ are small.

The schema theorem gives us a concrete result about what is happening during a genetic algorithm run. It provides the reason for the choice of a binary representation, namely *the principal of minimum alphabets*; binary strings maximize the number of schema the algorithm processes. With theorem D.1 it provides a reason for choosing fitness proportionate selection; to optimally allocate trials amongst schema. It provides reasons for choosing crossover methods; crossover is chosen to minimize disruption to schema and hence to conform to the building block hypothesis

Hypothesis D.1 (Building Block Hypothesis) *Successively better solutions can be generated by combining useful parts of extant solutions.*

It also dictates the mutation probabilities; mutation is a background operator which is infrequent so that it does not significantly disrupt schema processing, however it must be maintained in the algorithm to ensure that no schema are lost to the population. The schema theorem is seen as the fundamental theorem for how genetic algorithms work [94, page 33][79, page 117]. Genetic algorithm work by optimally allocating trials in a manner which maximizes the implicit parallelism of the method. The schema theorem states that the method will increase the frequency of short above average fitness schema exponentially when they are rare in the population [2].

D.3 Problems with Schema Analysis

The previous analysis is important since it has guided genetic algorithm design since it's conception with Holland in 1975. The theoretical analysis has many short comings, some of which we will now consider.

The stated goal of Theorem D.1 "...is to discover a procedure for distributing an arbitrary number of trials ...so as to maximize the expected payoff" [117], however a number of problems with both the derivation and utility of the Gaussian k-armed bandit problem to this end have been identified:

1 Holland considers only strategies where each arm has been pulled n times and the remaining $N - 2n$ trials are to be optimally allocated. It is not clear what relevance this subset of strategies has to the general optimisation problem of k-armed bandits [155] and by extension to the problem of allocating trials in a GA.

2 The fact that equation D.3 on page 173 is independent of the standard deviation σ_2 of the arm with the lower payoff is surprising. A higher value of σ_2 would indicate that more work would be required to establish which arm has the higher mean than otherwise. Macready and Wolpert state that "...the fact that Holland's result is independent of σ_2 is *ipso facto* reason to suspect it." [155].

3 In fact, it has been shown that a simple greedy algorithm based on a Bayesian [112] update from prior pulls performed better than Holland's strategy [155]. The derivation of Theorem D.1 assumes that the expected loss for allocating trials is unconditioned instead of being conditioned on the previous trials, which it should correctly be [155]. Theorem D.1 is the solution to the wrong problem [79, page 116–117].

4 In examining the problem of allocation of trials, the assumption is always that the fitness distribution of schema is normal[7]. This requires the central limit theorem to be applied, which in turn will require a sufficient number of independent evaluations of the schema. Only then can the separate fitness evaluations of the schema be assumed to be normally distributed. Most GAs codings violate the assumption of independent evaluations, and the frequency of schema in the GA population will frequently be less than that required for the central limit theorem, invalidating the assumption of a normally distributed fitness evaluation of the schema [79, page 118].

Altenberg notes that it has been shown that the link between the schema theorem and performance of the GA is based on the building block hypothesis. The basic idea is that by allocating exponentially many trials to promising schemata the GA is searching promising area's of the search space and thereby more likely to find better solutions [2]. The promising area's of the search space corresponds to above average schemata, and the assumption is that the genetic algorithm produces high quality solutions from above average schemata. That is, the fitness of children and parent is correlated. But this requirement is independent of the schema theorem [2].

Fogel writes that [79, page 117]:

> In light of Macready and Wolpert (1998), there now appears to be no support for viewing the schema theorem as having fundamental importance. The theorem simply describes the expected number of each schemata at the next generation under proportionate selection when each complete solution is assigned a specified fitness value.

The above problems go some way to explaining the proliferation of schema-theorem defying GAs in the literature.

[7]The assumption is always that the distribution of the payoffs in the k-armed bandit problem is normal

In Baker (unsighted [15]) and later Whitley [235] reported the first experiments using a rank-based selection method, where the probability of selecting an individual was independent of its fitness value, and instead dependent on its rank (according to fitness) in the current population. Rank based selection helped slow down the convergence of the genetic algorithm and stop domination of the method by a few super individuals whose fitness would be so far higher than the rest of the population that they become over represented in the consecutive population and the method quickly convergences. Other selection methods are often used, not necessarily in agreement with the schema theorem [27]

The *principal of minimal alphabets* is based on the idea that a binary alphabet can process 3^n schema, where as a k-ary alphabet which encodes the same number of points as a binary alphabet of length n, ie 2^n points, will have a length of

$$n' = n \cdot \frac{\ln 2}{\ln k}$$

and will process $(k+1)^{n'}$ schemata. This number is always less than 3^n for $n > 2$ [10, page 128]. Interpreting genetic algorithms as schema processing machines gives that a population of size N will process $\mathcal{O}(N^3)$, which is the *implicit parallelism* result [94, pages 40–41][10, page 128]. Antonisse [7] has proposed a different interpretation of the # character. Rather than representing any single symbol, Antonisse proposed that the # symbol can be interpreted as representing all subsets of available symbols. So the schema [0#02] in a ternary alphabet would be the sets {[0002] [0102]}, {[0002] [0202]}, {[0102] [0202]} and {[0002] [0102] [0202]} as the # symbol indicates a (0 or 1), (0 or 2), (1 or 2) and (0 , 1 or 2). Viewed in this way implicit parallelism is maximised by higher cardinality alphabets.

D.3.1 Operators and Representations in Evolutionary Computation

Fogel [81] shows that no particular representation of a genetic algorithm can be universally preferred over any other by showing that under appropriate mappings all operators can be modified to give the exact same behaviour on any representation.

Where we use an EP/ES phenotypic representation on a problem with a enumerable solution space[8] we note the following, following an argument similar to that of Fogel [78]. We will let one representation be genotype space, G, and the other phenotype space, P, where the fitness function is applied directly to members of P. EP and ES typically represent solutions as elements of P. We will suppose that there is an invertible mapping T between G and P. We will now show the equivalence of the GA and EP representations.

[8]The result should extend to an infinite space by considering a finite number of subsets of the real coded EP/ES algorithm as corresponding to a binary structure in genotype space, as noted in [81].

If $T(\cdot)$ is a mapping from genotype space G to phenotype space P, $T : G \rightarrow P$, then the population of an evolutionary program could be considered as $T(x), \forall x \in \mathbb{P}$, where $\mathbb{P} \subset G$ is the current population. The probabilistic operators of a genetic algorithm map elements of G to other elements of G. If $O(\cdot)$ is the operator mapping applied at each generation, then $O(x), \forall x \in \mathbb{P}$ is the child population generated from the current population. That is, it is the next generation. $O(\cdot)$ generates a probability distribution over elements of G where the probability is the probability of $x \in G$ being generated by the operator $O(\cdot)$. In an evolutionary programming approach we can define the operator O_{ep}, a map from $P \rightarrow P$ by using an inverse mapping T^{-1} to map solutions in P to the genotype space G and then applying $O : G \rightarrow G$ and then $T : G \rightarrow P$. By choosing the operator $O_{\text{ep}} \equiv T^{-1} \circ O \circ T$ we have identical behaviour between the GA and the EP/ES algorithm. Similarly, by choosing $O \equiv T \circ O_{\text{ep}} \circ T^{-1}$ we can map the genotype to the solution, the solution to another solution and back to the equivalent genotype.

A similar argument can be constructed for any crossover operator which acts on n parents. Note that the mapping T will not be invertible when different genotypes map to the same phenotype. When the EP algorithm visits such a phenotypic point, the behaviour of the two algorithms diverge.

Reasoning like this we can argue that there is no difference between the representations chosen by the different approaches, GA and EP/ES. The difference lies in the operators, since the operators can undo any mapping if carefully constructed. The argument will hold for any problem representations for which we can build invertible maps between. This leaves the question of how to best to represent a solution. Part of the answer to this question will depend on why we are solving the particular problem, that is, what we hope to achieve and find in solving it. Another part of the answer might follow from the next section on the emergence of evolvability in EC.

Appendix E

Future Development of the SASME Framework

> Despite this partisan flavor, the book shines in the few paragraphs where Bentley pauses to discuss some of the limitations of the systems. "We cannot prove that evolution will find us a good solution—but it almost invariably does. And we certainly cannot predict the solutions that evolution generates," he notes as a caveat to everyone planning to use genetic programming to solve world peace.
>
> *http://slashdot.org/books/02/03/04/195222.shtml*

> As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.
>
> Albert Einstein

> In theory there is no difference between theory and practice. In practice there is.
>
> Yogi Berra

This appendix details some possible future directions for the work presented in this thesis.

E.1 Theoretical Analysis of the SASME Algorithm

The SASME framework presented in this thesis was justified through heuristic arguments concerning the nature of self-adaptive evolutionary processes. The principal arguments used to justify the components of the new evolutionary search approach can be summarized as follows:

Search Method

It was argued that evolution is a suitable search method for the learning problems used in this thesis. Whilst evolution is not guaranteed to find the best

solution to the problem at hand, it is guaranteed to autonomously propose solutions; this is the nature of the evolution algorithm discussed in Chapter 1. The solutions proposed by evolution often have the appearance of design, without there being a designer. Hence, it is argued, evolution is an appropriate framework for problems where little is known about the search space, and little is known about how to find good solutions in that space.

A similar argument can be made for using evolution with structures which are able to represent the solution in a manner which elucidates some of the characteristics of the problem which are of interest.

Representation

Rule lists with exceptions are used as the symbolic structure for the problems tackled in this thesis. It is argued that these rule sets are capable of representing information that the evolutionary search process discovers. Further, the addition of exceptions to rules allows the evolutionary search to make small modifications to the behaviour of the phenotype—it is widely stated that this is an important property of structures amendable to evolutionary search.

Self-adaptive Search

Self-adaptation is argued to be one solution to the problem of evolutionary parameter setting. By allowing the mutation step sizes and probability of application of discrete mutations to be self-adapted the algorithm gains the flexibility to adapt the search strategy to the extant structures in the population in any given problem domain. The utility of this approach is investigated empirically on a specified problem in Chapter 4 and found to automatically find a suitable level of discrete mutation operations.

These arguments provide a rationale for the development and implementation of the SASME algorithm. The algorithm is then used to find good solutions to some difficult machine learning problems, demonstrating the power of heuristically motivated algorithms. Indeed, many new machine learning methodologies owe their existence and form to heuristic argument rather than arguments grounded in any kind of mathematical analysis. The development of the entire class of naturally-inspired algorithms could be considered to be consistent with this statement, including Simulated Annealing, Evolutionary Computation, Artificial Immune Systems [114] etc.

However, it is interesting to review what analysis of the approach can show.

E.1.1 Schema Analysis

The well-known schema theorem has been extensively used as a basis for the analysis of different evolutionary algorithms. The original schema theorem discussed in Appendix D was developed for the binary representation and the canonical GA. The theorem itself describes the relationship between the number of short, low

order schemata in one generation compared to the next under proportionate selection, one point cross-over and point mutation. Algorithms which propose different operators and representations often have new schema theorems derived for them [228, 177, 176, 178, 158, 175, 174]. Such theorems can provide some understanding of the macroscopic behaviours of the evolutionary process. Holland's schema theorem provides proof that the GA allocates exponentially many trials to short, low order schema. Along with Theorem D.1, it was believed that this showed that the GA search was in some way optimal. That is, the GA optimally allocated trials amongst competing short, low order schema.

This latter interpretation of the schema theorem is now untenable. The NFL theorem (Appendix B) and the arguments presented in Appendix D independently show this view cannot persist. The performance of a search algorithm cannot be assessed independent to the domain within which it is applied. The usual assertion about the schema theorem is that it describes the propagation of useful structures in the population between generations—the principal issue is justifying why the structures which have been used are useful to the problem at hand. The theorem does not require there to be a correlation between the fitness of parents and the fitness of offspring, instead this becomes implicitly assumed when justifying the analysis. Some work has been done on using parent-offspring fitness correlations to develop schema theorems which describe the trajectories taken by those solution components which do give rise to correlations between parent and offspring fitness [2].

This does not entirely negate the utility of schema theorems, however. The theorems can be viewed as descriptive of certain macroscopic parameters of algorithmic behaviours. As such, they can provide some light on some aspects of the trajectories that the evolutionary search takes. When combined with knowledge about the search space, this could lead to predictive statements about algorithmic performance. Should the development of the SASME framework in the future incorporate schema-like theorems for analysis, this is the approach which would be first attempted.

There are two obstacles which need to be overcome in developing a schema analysis of the SASME algorithm. The first is the definition of schema for non-binary strings. This has been tackled extensively in tree-based GP schema theorems [174], and some of these approaches could be used as a basis for an analysis in SASME. A more difficult obstacle is dealing with the self-adaptive components of the SASME approach, and the number of operators. Whilst the operator set could be simplified, self-adaptation does not appear to be amendable to schema analysis, and is central to the SASME framework. A final issue with schema analysis is that a schema theorem will not shed any light on the question of whether the algorithm will solve problem X in reasonable time, it will only provide a toolkit with which analysis of the algorithms behaviour on problem X can be performed.

E.1.2 Convergence Analysis

Most analysis questions about algorithms centre on the asymptotic behaviours of the algorithm and bounds on worst case scenarios. Interesting questions such as bounds on time and memory required to discover a solution, and predictions of solution quality are the motivation for this kind of analysis [163].

In many evolutionary algorithms, convergence to the neighborhood of the optimal solution as time tends to infinity can be proven under general conditions, as presented and discussed in Appendix A. For the SASME framework, if the rule set is held constant, the parameter adaptation is a standard (μ, λ)-ES then the global convergence caveat discussed in Appendix A will hold. That is, the parameter optimisation will not converge to the global optima with probability 1 when the fitness landscape of the parameter space is continuous. However, global convergence is not a particularly useful property without information on convergence velocity.

The convergence velocity analysis in Appendix A will likewise hold when the rule structure is held constant. Where the local topology of the search space is assumed to be a sphere, there can be a reasonable level of confidence that the parameter optimisation procedure is efficiently approaching the optimum values. The goal of the algorithm would then be to find a good set of parameters with which the problem can be solved, not to find the optimal set of parameters. When the rule set is not held constant it can be seen that the local topology of the search space seen by the self-adaptive parameter evolution will change as the rule set changes. Towards the end of a run it can be expected that the rule set is remaining unchanged from one generation to the next.

The evolution of the discrete component of the SASME model is more difficult to analyse. It can be noted that it is the evolution of a tree-based representation, and as such much of the analysis which has been developed for GP may be applicable. The principal question concerning the evolution of the discrete model is the utility of self-adapting the discrete operator rates. This question was addressed empirically in Chapter 3. Future work may look at methods to quantify when this self-adaptation is beneficial in terms of time to solution.

The interaction between the parameter evolution and the discrete structure evolution is the most interesting and difficult aspect of SASME from a theoretical viewpoint.

E.1.3 Further Empirical Validation

The most likely future developments in the understanding of the SASME framework will come from further empirical studies of the interactions of the different components of the framework. The algorithm was not designed with theoretical analysis in mind, and as such is not readily amendable to a mathematical treatment. Analysis similar to that conducted in Chapter 3 may tease out some relationships between

the different components.

E.2 Extentions of the Framework

A number of extensions to the SASME algorithm presented in this thesis are being considered for future work. One promising area is the utilization of domain specific operators as well as representations in the framework. Information theoretic measures such as those used in classic rule induction systems are a possibility. The inclusion of more discrete structures, and perhaps populations of mixed structures, is also possible. The following section presents a recipe for including new discrete structures in the SASME framework.

E.2.1 A Recipe for Evolving New Representations with SASME

The algorithm presented in Chapter 3 is general and applicable with a wide variety of discrete structures. The implementation used in this thesis utilized a rule set with exceptions as the discrete structure. To implement a different discrete structure the following steps need to be undertaken:

1 The structure has to be implemented along with the operators which will manipulate it. In an object-oriented programming language the structure and operators can be encapsulated as an object. The interface between this object and the SASME object needs to communicate the number of operators which have been implemented, and the number of parameters which are required to realize the structure. In a neural network, for example, each weight in the network is a parameter, and the number of weights is returned as the number of parameters. As a minimum an add hidden node and delete hidden node operator needs to be implemented. When affected, either of those operators will modify the number of weights which the model requires. The structure needs to inform the SASME object what parameters have been deleted, or the values of new parameters which have been created.

2 The SASME object initializes a parameter object suitable for the structure, containing the number of parameters required by the model and the number of operators. The bounds for the parameters will depend on the nature of the structure, and also need to be communicated. The values of the parameters can likewise be communicated by the structure object.

3 The SASME object creates a population of parameter objects and their associated structure objects. The objects are evaluated by the SASME object interacting with the problem environment, and then the population is evolved by the method described in Chapter 3. The SASME object sets the value of τ and τ' in the ES equations on the fly according to the size of the parameter object. The parameter object contains the self-adaptive mutation step sizes.

The structure object may also need to communicate directly with the problem environment, for example, when one of the operators uses information from the environment to guide the creation of the structure. Future work will look at operators which can adjust both the parameters and the structure, for example, using heuristics to adjust the weights after a node has been deleted from a neural network. The network could be partially trained by a back-propagation algorithm to smooth out the effects of the discrete operator. This may be needed to maintain parent-offspring fitness correlation.

Clearly a range of structures can be coded in this way. Future work will attempt to categorize what problems a particular structure is well suited for.

Bibliography

[1] *Handbook on Evolutionary Computation.* IOP Publishing Ltd and Oxford University Press, 1997. Release 97/1.

[2] Lee Altenberg. The Schema Theorem and Price's Theorem. In Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 23–49. Morgan Kaufmann, San Mateo, CA, 1995.

[3] Edgar Anderson. The irises of the Gaspe Peninsula. *Bulletin of the American Iris Society*, 59:2–5, 1935.

[4] P. J. Angeline. A historical perspective on the evolution of executable structures. *Fundamenta Informaticae*, 36(1–4):179–195, August 1998.

[5] Peter J. Angeline. Multiple interacting programs: A representation for evolving complex behaviors. *Cybernetics and Systems*, 29(8):779–806, 1998.

[6] Peter J. Angeline and Jordan B. Pollack. Coevolving high-level representations. In C. G. Langton, editor, *Artificial Life III*, pages 55–71. Addison-Wesley, Reading MA, 1994.

[7] J Antonisse. A new interpretation of schema notation that overturns the binary encoding constraint. In J D Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 86–91, San Mateo CA, 1989. Morgan Kaufmann. Reprinted [76, pages 558–563].

[8] J. Armstrong and F. Collopy. Error measures for generalizing about forecasting methods - empirical comparisons, 1992.

[9] S. Augier, G. Venturini, and Y. Kodratoff. Learning first order logic rules with a genetic algorithm. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *The First International Conference on Knowledge Discovery and Data Mining*, pages 21–26, Montreal, Canada, 20-21 1995. AAAI Press.

[10] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, 198 Madison Avenue, New York, New York 10016, 1996.

[11] Thomas Bäck, Frank Hoffmeister, and Hans-Paul Shwefel. A survey of evolution strategies. In Richard K. Belew, editor, *Proceedings of the Fourth International Conference on Genetic Algorithms and their Applications*, pages 2–9, San Diego, California, 1991. Morgan Kaufmann Publishers.

[12] Thomas Bäck, G. Rudolph, and Hans-Paul Schwefel. Evolutionary programming and evolution strategies: Similarities and differences. In D.B. Fogel and W. Atmar, editors, *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 11–22, San Diego CA, 1993. Evolutionary Programming Society.

[13] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[14] Thomas Bäck and Hans-Paul Schwefel. Evolutionary computation: An overview. In T. Fukuda, T. Furuhashi, and D. B. Fogel, editors, *Proc. 1996 IEEE Conf. Evolutionary Computation (ICEC'96)*, pages 20–29, Nagoya, Japan, May 20–22, 1996. IEEE Press, Piscataway NJ.

[15] James Baker. Adaptive selection methods for genetic algorithms. In John Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 101–111. L. Erlbaum, 1988, original proceedings 1985, 1985.

[16] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming—an Introduction : on the automatic evolution of computer programs and its applications.* Morgan Kauffmann, Heidelberg, 1998.

[17] Andrew G. Barto, Richard Sutton, and Charles Anderson. Neuronlike adaptive elements that can solve difficult learning problems. *IEEE Transactions on Systems, Man, And Cybernetics*, 13(5):834–846, September/October 1983.

[18] R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, N.J., 1957.

[19] Forrest H. Bennett III, John R. Koza, David Andre, and Martin A. Keane. Evolution of a 60 decibel op amp using genetic programming. In *First International Conference on Evolvable Systems (ICES-96)*, October 7–8 1996.

[20] Tom Berman and Sara Chava. Algal growth on organic compounds as nitrogen sources. *Journal of Plankton Research*, 21(8):1423–1437, 1999.

[21] H.-G. Beyer. Toward a Theory of Evolution Strategies: On the Benefit of Sex – the $(\mu/\mu, \lambda)$-Theory. *Evolutionary Computation*, 3(1):81–111, 1995.

[22] H.-G. Beyer. An alternative explanation for the manner in which genetic algorithms operate. *BioSystems*, 41:1–15, 1997.

[23] Hans-Georg Beyer. Toward a theory of evolution strategies: The (μ, λ)-theory. *Evolutionary Computation*, 2(4):381–407, 1995.

[24] Hans-Georg Beyer. Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation*, 3(3):311–347, 1996.

[25] H Birru, K Chellapilla, and S. S. Rao. Local search operators in fast evolutionary programming. In *Proceedings of the 1999 Congress on Evolutionary Computation, Washington D.C., USA*, volume 2, pages 1506–1513, Piscataway NJ, July 6–9 1999. IEEE Press.

[26] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. *http://www.ics.uci.edu/~mlearn/MLRepository.html.*

[27] Tobias Blicke and Lothar Thiele. A comparison of selection schemes used in genetic algorithms. Technical report, Seiss Federal Institute of Technology (ETH), Computer Engineering and Communication Networks Lab, Gloriastrasse 35, 8092 Zurich, Swizerland, June 1995. Nr. 11, Version 1.1b.

[28] Jason Bobbin and Friedrich Recknagel. Mining water quality time series for predictive rules of algal blooms by genetic algorithms. In *Proc. of 1999 Modelling and Simulation Society of Australia and New Zealand Conference (MODSIM'99)*, volume 3, pages 691–696, Hamilton, New Zealand, 6-9 December 1999. The Modelling and Simulation Society of Australia and New Zealand Inc.

[29] Jason Bobbin and Friedrich Recknagel. Inducing explanatory rules for the prediction of algal blooms by genetic algorithms. *Environment International*, 27(2–3):237–242, September 2001. Full version of conference paper [28].

[30] Jason Bobbin and Friedrich Recknagel. Knowledge discovery for prediction and explanation of blue-green algal dynamics in lakes by evolutionary algorithms. *Ecological Modelling*, 146(1–3):253–262, December 2001.

[31] Jason Bobbin and Xin Yao. Automatic discovery of relational information in comprehensible control rules by evolutionary algorithms. In *Proc. of 1999 Australia-Japan Workshop, Canberra, Australia*, 23-26 November 1999.

[32] Jason Bobbin and Xin Yao. Evolving rules for nonlinear control. *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation, Vienna, Austria*, 1999.

[33] Jason Bobbin and Xin Yao. Automatic discovery of comprehensible control rules by evolutionary algorithms. In Masoud Mohammadian, editor, *New Frontier in Computational Intelligence and its Applications*, volume 57 of *Frontiers in Artificial Intelligence and Applications*, pages 197–202. IOS Press, Amsterdam, 2000. Full version of conference paper [32].

[34] Bela Bollobas, Gautam Das, Dimitrios Gunopulos, and Heikki Mannila. Time-series similarity problems and well-separated geometric sets. In *Symposium on Computational Geometry*, pages 454–456, 1997.

[35] Andrea Bonarini. An introduction to learning fuzzy classifier systems. In Lanzai et al. [148].

[36] Andrea Bonarini, Claudio Bonacina, and Matteo Matteucci. Fuzzy and crisp representations of real-valued input for learning classifier systems. In Lanzai et al. [148].

[37] E J Borowski and J M Borwein. *Dictionary of Mathematics.* Collins, 1989.

[38] George E P Box. Evolutionary operation: A method for increasing industrial productivity. In *Applied Statistics*, volume 6, pages 81–101, 1957. Reprinted page 121 [76].

[39] Martin Braun. *Differential Equations and Their Applications*, volume 15 of *Applied Mathematical Sciences.* Springer-Verlag, Heidelberg, 3rd edition edition, 1986. This is an excellent book.

[40] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and Regression Trees.* Statistical/Probability Series. Wadsworth International Group, New York, 1984.

[41] H J Bremermann, M Rogson, and S Salaff. Global properties of evolution processes. In H H Pattee, E A Edlsack, L Fein, and A B Callahan, editors, *Natural Automata and Useful Simulations*, pages 3–41. Spartan Books, Washington D.C., 1966. Reprinted [76, page314].

[42] Martin V. Butz and Stewart W. Wilson. An algorithmic description of XCS. IlliGAL report no. 2000017, University of Illinois at Urbana-Champaign, April 2000.

[43] *Congress on Evolutionary Computation 2000*, 16–19 July 2000.

[44] C Chatfield. Neural networks: Forecasting breakthrough or just a passing fad? *Journal of Research Statistics Society B*, 56(3):409–446, 1994.

[45] K Chellapilla. Evolutionary programming with tree mutations: Evolving computer programs with crossover. In J. Koza *et al.*, editor, *Proceedings of the Second Annual Conference on Genetic Programming*, pages 431–438, Stanford University, CA, Jul 13–16 1997. Morgan Kaufmann.

[46] K Chellapilla. Automatic generation of nonlinear optimal control laws for broom balancing using evolutionary programming. In *Proceedings of the 1998 IEEE Conference on Evolutionary Computation*, pages 195–200, Orlando, Florida, 1998.

[47] Kumar Chellapilla. Evolving nonlinear controllers for backing up a truck-and-trailer using evolutionary programming. *Proceedings of the 1998 IEEE Conference on Evolutionary Computation*, pages 195–200, 1998.

[48] Kumar Chellapilla and David Fogel. Two new mutation operators for enhanced search and optimization in evolution programming. In B Bosacchi, J C Bezdek, and D B Fogel, editors, *SPIE's International Symposium on Optical Science, Engineering, and Instrumentation, Conference 3165: Applications of Soft Computing*, pages 260–269, July 27th – Aug 1st 1997. San Diego, CA.

[49] P. Compton and R. Jansen. Knowledge in context: A strategy for expert system maintenance. In C.J.Barter and M.J.Brooks, editors, *AI'88: Proceedings of the second Australian Conference in Artificial Intelligence*, pages 292–306, Berlin, 1989. Springer-Verlag.

[50] O. Cordón, F. Herrera-Viedma, and M. Lozano. Genetic algorithm and fuzzy logic in control processes. DESCAI Technical report #decsai-95109, ETS de Ingeniería Informática, Universidad Granada, March 1995.

[51] Joseph C. Culberson. On the futility of blind search. Technical Report TR 96-18, University of Alberta, Edmonton, Alberta, Canada, July 1996.

[52] Paul J Darwen. *Co-Evolutionary Learning by Automatic Modularisation with Speciation.* Phd thesis, University College, The University of New South Wales, Australian Defence Force Academy, Northcotte Drive, Canberra, 18th July 1996.

[53] Charles Darwin. *On the Origin of Species by means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* Oxford University Press, Walton Street, Oxford, 2nd edition, 1860.

[54] Richard Dawkins. *The Blind Watchmaker.* Penguin Books, 1986.

[55] I. De Falco, A. Iazzetta, and E. Tarantino. An evolutionary system for automatic explicit rule extraction. In CEC00 [43], pages 450–457.

[56] Kenneth De Jong. Learning with genetic algorithms: An overview. *Machine Learning*, 3:121–138, 1988.

[57] Kenneth A. De Jong. *Genetic-Algorithm-Based Learning*, volume III of *Machine Learning*, chapter 21, pages 611–638. Morgan Kaufmann, 1990.

[58] Kenneth A. De Jong and William M. Spears. Learning concept classification rules using genetic algorithms. *Learning and Knowledge Acquisition*, 1989.

[59] Daniel Dennet. *Darwin's dangerous idea.* Penguin Books, 1995.

[60] Sašo Džeroski, Ljupčo Todorovski, Ivan Bratko, Boris Kompare, and Viljem Križman. Equation discovery with ecological applications. In Alan H. Fielding, editor, *Machine Learning Methods for Ecological Applications*, chapter 7. Kluwer, 1998.

[61] Marco Dorigo and Hugues Bersini. A comparison of Q-learning and classifier systems. In *Proceedings of from Animals to Animats, Third International Conference on Simulation of Adaptive Behavior (SAB94)*, August 8–12 1994.

[62] Saso Dzeroski, Luc De Raedt, and Hendrik Blockeel. Relational reinforcement learning. In *International Workshop on Inductive Logic Programming*, pages 11–22, 1998.

[63] Senchi Ebise, Morihiro Aizaki, Masaaki Hosomi, Hideaki Ozawa, Toshio Iwakuma, Noriko Takamura, Takayoshi Kawai, Yukihiro Nojiri, Takehiko Fukushima, Takayuki Hanazato, and Kazuho Inaba. *Environmental Data for Lake Kasumigaura*, chapter 1: Limnological Data for Lake Kasumigaura. National Institute for Environmental Studies, 16-2 Onogawa, Tsukuba, Ibaraki 305 Japan, 1994.

[64] V. Estivill-Castro. Collaborative knowledge acquisition with a genetic algorithm. In *Proceedings of the Ninth IEEE International Conference on Tools with Articial Intelligence*, pages 270–277. IEEE Computer Society Press, Los Alamitos, CA, 3.-8. November 1997.

[65] M. V. Fidelis, H. S. Lopes, and A. A. Freitas. Discovering comprehensible classification rules with a genetic algorithm. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 805–810, Piscataway, NJ, 2000. IEEE Service Center.

[66] R. A. Fischer. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936. Part II.

[67] Ian W. Flockhart and Nicholas J. Radcliffe. A genetic algorithm-based approach to data mining. In Evangelos Simoudis, Jia Wei Han, and Usama Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 299–302, Portland, Oregon, USA, 2-4 1996. AAAI Press.

[68] T. C. Fogarty and R. Huang. Evolving prototype control rules for a dynamic system. *Knowledge-Based Systems*, 7(2):142–145, 1994. KBS Letter.

[69] Terence C. Fogarty. Technical note: First nearest neighbor classification on frey and slate's letter recognition problem. *Machine Learning*, 9:387–388, 1992.

[70] David B. Fogel. An analysis of evolutionary programming. In D. B. Fogel and W. Atmar, editors, *First International Conference on Evolutionary Programming*, pages 43–51, La Jolla, CA, 1992. Evolutionary Programming Society.

[71] David B Fogel. An introduction to simulated evolutionary optimisation. *IEEE Transactions on Neural Networks*, 5(1):3–14, January 1994.

[72] David B. Fogel. *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence.* IEEE Press, Piscataway, NJ, 1 edition, 1995. 2nd edition [79].

[73] David B. Fogel. Phenotypes, genotypes and operators in evolutionary computation. In *Proceedings of the 1995 IEEE International Conference on Evolutionary Computation, Perth, Australia*, pages 193–198, Piscataway, 1995. IEEE Press.

[74] David B. Fogel. A "correction" to some cart-pole experiments. *Evolutionary Programming V*, pages 67–71, 1996.

[75] David B. Fogel. The advantages of evolutionary computation. In D. Lundh, B. Olsson, and A. Narayanan, editors, *Bio-Computing and Emergent Computation*, pages 1–11. World Scientific Press, Singapore, 1997.

[76] David B. Fogel, editor. *Evolutionary Computation : The Fossil Record.* IEEE, IEEE Press, 1998.

[77] David B Fogel. An overview of evolutionary programming. In Lawrence David Davis, Kenneth De Jong, Michael D Vose, and L Darrell Whitley, editors, *Evolutionary Algorithms*, volume 111 of *The IMA Volumes in Mathematics and its Applications*, pages 89–109. Springer-Verlag, New York, 1999.

[78] David B. Fogel. Some recent important foundational results in evolutionary computation. In K. Miettinen, M.M. Mäkela, P.Neittaanmaki, and K.Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, pages 55–71. John Wiley, Chichester, U.K., 1999.

[79] David B Fogel. *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence.* IEEE Press, Piscataway, NJ, 2nd edition, 2000. 1st edition [72].

[80] David B. Fogel and Adam Ghozeil. Using fitness distributions to design more efficient evolutionary computations. *Third IEEE International Conference on Evolutionary Computation(ICEC'96),Nagoya 1996*, pages 11–19, 1996.

[81] David B Fogel and Adam Ghozeil. A note on representations and variation operators. *IEEE Transactions on Evolutionary Computation*, 1(2):159–161, July 1997.

[82] Lawrence J. Fogel. A retrospective view and outlook on evolutionary algorithms. In B. Reusch, editor, *Computational Intelligence: Theory and Applications, 5th Fuzzy Days*, pages 337–342, Berlin, 1997. Springer-Verlag.

[83] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. Artificial intelligence through a simulation of evolution. In M. Maxfield, A. Calahan, and L. J. Fogel, editors, *Biophysics and Cybernetic Systems: Proceedings of the 2nd Cybernetic Sciences Symposium*, pages 131–155, Washington, D.C., 1965. Spartan Books. Reprinted [76, page 230].

[84] A S Fraser. Simulation of genetic systems by automatic digital computers. In *Australian Journal of Biological Sciences*, volume 10, pages 282–291, 1957. Reprinted [76, page 87].

[85] A S Fraser. The evolution of purposive behavior. In H von Foerster, J D White, L J Pererson, and J K Russell, editors, *Purposive Systems*, pages 15–23. Spartan Books, Washington D.C., 1968. Reprinted [76, page 109].

[86] A.A. Freitas. A survey of evolutionary algorithms for data mining and knowledge discovery. In A. Ghosh and S. Tsutsui, editors, *Advances in Evolutionary Computation.* Springer-Verlag, 2002. To Appear.

[87] P.W. Frey and D.J. Slate. Letter recognition using holland-style adaptive classifiers. *Machine Learning*, 6(2):161–182, 1991.

[88] R. M. Friedberg. A learning machine: Part I. In *IBM Journal of Research and Development*, volume 2, pages 2–13, 1958. Reprinted [76, page 145].

[89] R. M. Friedberg, B. Dunham, and J. H. North. A learning machine: Part II. In *IBM Journal of Research and Development*, volume 3, pages 282–287, 1959. Reprinted [76, page 157].

[90] George J. Friedman. *Selective Feedback Computers for Engineering Synthesis and Nervous System Analogy.* Master of Science in Engineering Thesis, University of California, Los Angeles, February 1956. Reprinted in [76].

[91] Naoshi Fujimoto and Ryuichi Sudo. Nutrient-limited growth of *Microcystis aeruginosa* and *Phormidium tenue* and competition under various N:P supply ratios and temperatures. *Limnology and Oceanography*, 2(42):250–256, 1997.

[92] Adam Ghozeil and David B. Fogel. Discovering patterns in spatial data using evolutionary programming. In J R Koza, D E Goldberg, D B Fogel, and R L Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 521–527, Cambridge,MA, 1996. MIT Press.

[93] Antonella Giani, Fabrizio Baiardi, and Antonina Starita. PANIC: A parallel evolutionary rule based system. In *Evolutionary Programming*, pages 753–771, 1995.

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

[95] David E. Goldberg, Jeffrey Hown, and Kalyanmoy Deb. What makes a problem hard for a classifier system? Technical Report IlliGAL Report No. 92007, University of Illinois, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801, May 1992.

[96] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. Technical Report AI96-248, 1 1996.

[97] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behaviour*, (5):317–342, 1997.

[98] Faustino J. Gomez and Risto Miikkulainen. Solving non-Markovian control tasks with neuro-evolution. In *International Joint Conference on Artificial Intelligence IJCAI*, pages 1356–1361, 1999.

[99] Stephen Jay Gould. *The Panda's Thumb: More Reflections in Natural History.* Norton, Chichester, New York, 1980.

[100] Stephen Jay Gould. *Bully for Brontosaurus.* Penguin Books, 1991.

[101] Stephen Jay Gould. *Life's Grandeur: The spread of excellence from Plato to Darwin.* Vintage, Random House, 20 Vauxhall Bridge Road, London SW1V 2SA, 1996. Published as *Full House* in the United States.

[102] John J. Grefenstette. A system for learning control strategies with genetic algorithms. In *International Conference on Genetic Algorithms*, pages 183–190, 1989.

[103] John J. Grefenstette. The evolution of strategies for multiagent environments. *Adaptive Behaviour*, 1(1):65–89, 1991.

[104] John J. Grefenstette. Strategy acquisition with genetic algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms*, chapter 14, pages 186–201. Von Nostrand, Boston, 1991.

[105] John J. Grefenstette, Connie Loggia Remsey, and Allan C. Shultz. Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5:355–381, 1990.

[106] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89, Cambridge, MA, 1996. MIT Press.

[107] G P Harris. *Plankton Ecology - Structure, Function, and Fluctuation.* Chapman and Hall, New York, 1986.

[108] Jukka Hekanaho. A GA-based approach to disjunctive concept learning. Technical Report TUCS-TR-72, 11, 1996.

[109] Jukka Hekanaho. DOGMA: A GA-based relational learner. TUCS Technical report no. 168, Turku Centre for Computer Science, Åbo Akademi University, Finland, May 1997. ISBN 952-12-0181-9.

[110] Jukka Hekanaho. GA-based rule enhancement in concept learning. In *Knowledge Discovery and Data Mining*, pages 183–186, 1997.

[111] D. T. Higdon. *Automatic Control of Inherently Unstable Systems with Bounded Control inputs.* Ph.D. Dissertation, Department of Aeronautics and Astronautics, Stanford University, 1963.

[112] Ray Hilborn and Marc Mangel. *The Ecological Detective.* Number 28 in Monographs in Population Biology. Princeton University Press, 1997.

[113] Lars Håkanson. On the principles and factors determining the predictive success of ecosystem models, with a focus on lake eutrophication models. *Ecological Modelling*, 121:139–160, 1999.

[114] Steven A. Hofmeyer and S. Forrest. Architecture for an artificial immune system. *Evolutionary Computation*, 7(1):45–68, 2000.

[115] J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-Directed Inference Systems*, pages 313–329, NY, 1978. Academic Press. Reprinted [76, pages 464–480].

[116] John H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal of Computing*, 2(2):88–105, 1973. Reprinted [76, page 443–460].

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

[118] John H. Holland. Using classifier systems to study adaptive nonlinear networks. In D. Stein, editor, *Lectures in the science of complexity, SFI Studies in the Science of Complexity.* Addison Wedley, 1989.

[119] John H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 2nd edition, 1992.

[120] John H. Holland. Genetic algorithms. *Scientific American*, 267(1):44–50, 1992.

[121] John H. Holland et al. What is a learning classifies system? In Lanzai et al. [148].

[122] John H. Holland, Keith J. Holyoak, Richard E. Nisbell, and Paul R. Thagard. *Induction: Processes of Inference, Learning, and Discovery.* MIT Press, Cambridge, Massachusetts, 1986.

[123] John H. Holmes. Learning classifier systems applied to knowledge discovery in clinical research databases. In Lanzai et al. [148], pages 243–261.

[124] K Hornik, M Stichcombe, and H White. Multi-layer feed-forward networks are universal approximators. *Neural Networks*, 3:359–366, 1989.

[125] David H. Horricks and Mark C. Spittle. Component value selection for an active filter using genetic algorithms. Internet journal thing ***, 1996.

[126] Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.

[127] Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.

[128] Fogel L. J., Angeline P. J., and Fogel D. B. An evolutionary programming approach to self-adaptation in finite state machines. In J. R. mcDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 355–365, Cambridge MA, 1995. MIT Press.

[129] Kenneth A. De Jong, William M. Spears, and Diana F. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 13:161–188, 1993.

[130] Leslie Pack Kaebling, Michael L Littman, and Abdrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[131] C. L. Karr. A cart-pole system. In *Handbook on Evolutionary Computation* [1], chapter D2.2, pages D2.2:1–D2.2:9. Release 97/1.

[132] James D. Kelley and Lawrence Davies. A hybrid genetic algorithm for classification. In *Proceedings of the Twelth International Conference on Artificial Intelligence IJCAI-91*, volume 2, 1991.

[133] J. Kelly and L. Davis. Hybridizing the genetic algorithm and the k-nearest neighbors classification algorithm. In *Proceedings of the fourth international conference on genetic algorithms*, 1991.

[134] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimisation by simulated annealing. *Science*, 220(4598):671–680, 1983.

[135] Donald Knuth. *The Art of Computer Programming: Fundamental Algorithms*, volume 1. Addison-Wesley, Reading, MA, 2nd edition, 1973.

[136] T Kohonen. *Self-organisation and Associative Memory.* Sringer-Verlag, Berlin, 2nd edition, 1989.

[137] Tim Kovacs and Pier Luca Lanzi. A Learning Classifier Systems Bibliography. Technical Report CSRP-99-19, School of Computer Science, University of Birmingham, 1999.

[138] J R Koza. Evolving programs using symbolic expressions. In N S Sridharan, editor, *Proc. of the 11th Intern. Joint Conf. on Artificial Intelligence*, pages 768–774. Morgan Kaufmann, San Mateo, CA, 1989. Reprinted [76, page578].

[139] John R. Koza. *Genetic Programming.* MIT, Stanford University. Cambridge, MA, 1992.

[140] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge, MA, 1994.

[141] John R. Koza, David Andre, Forrest H. Bennett III, and Martin A. Keane. Design of a 96 decibel operational amplifier and other problems for which a computer program evolved by genetic programming is competitive with human performance. In Mitsuo Gen and Weixuan Zu, editors, *Proceedings of l996 Japan-China Joint International Workshop on Information Systems*, pages 30–49, Ashikaga, 1996. Ashikaga Institute of Technology.

[142] John R. Koza, Forrest H. Bennett III, and David Andre. Using programmatic motifs and genetics programming to classify protein sequences as to cellular location. In *Seventh Annual Conference on Evolutionary Programming*. San Diego, March 26 1998.

[143] John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. Automated wywiwig design of both the topology and component values of electrical circuits using genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 123–131, Stanford University. Cambridge, MA, July 28-31 1996. The MIT Press.

[144] John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. Four problems for which a computer program evolved by genetic programming is competitive with human performance. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 1–10. IEEE Press, 1996.

[145] John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. Evolutionary design of analog electrical circuits using genetic programming. In *Adaptive Computing in Design and Manufacture conference (ACDM–98)*, April 21–23 1998.

[146] John R. Koza, Martin A. Keane, Jessen Yu, Forrest H. Bennett III, and William Mydlowec. Automatic creation of human-competitive programs and controllers by means of genetic programming. In *Genetic Programming and Evolvable Machines*, volume 1, pages 121–164. Kluwer Academic Publishers, The Netherlands, 2000.

[147] John R. Koza, Martin A. Keane, Jessen Yu, William Mydlowec, and Forrest H. Bennet III. Automatic synthesis of both the control law and parameters for a controller for a three-lag plant with five-second delay using genetic programming and simulation texhniques. In *Proceedings of the 2000 American Control Conference*, pages 453–459, Chicago, Illinois, June 28–30, 2000. Evanston, IL: American Automatic Control Council.

[148] Pier Luca Lanzai, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Learning Classifier Systems: An Introduction to Contemporary Research*, volume 1813 of *LNAI*, Berlin, 2000. Springer-Verlag.

[149] Pier Luca Lanzi. Extending the representation of classifier conditions part I: From binary to messy coding. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO-99*, volume 1, pages 11–18, San Francisco, CA 94104, USA, July 1999. Morgan Kaufmann.

[150] Pier Luca Lanzi and Alessandro Perrucci. Extending the represextending the representation of classifier conditions part II: From messy coding to S-expressions. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H.

Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 345–352, San Francisco, CA 94104, USA, July 1999. Morgan Kaufman.

[151] F.H.F. Leung, L.K. Wong, and P.K.S. Tam. Fuzzy model based controller for an inverted pendulum. *Electronics Letters*, 32(18):1683–1685, August 1996.

[152] K. H. Liang, X. Yao, Y. Liu, C. Newton, and D. Hoffman. An experimental investigation of self-adaptation in evolutionary programming. In *Evolutionary Programming VII: Proceedings of the 7th Annual Conference on Evolutionary Programming*, Lecture Notes in Computer Science, pages 291–300, Berlin, 1998. Springer-Verlag.

[153] Charles X. Ling and Ralph Buchal. Learning to control dynamic systems with automatic quantization. *Adaptive Behaviour*, 3(1):29–49, 1994.

[154] Juliet Juan Liu and James Tin-Yau Kwok. An extended genetic rule induction algorithm. In CEC00 [43], pages 458–463.

[155] W G Macready and D H Wolpert. Bandit problems and the exploration/exploitation tradeoff. *IEEE Transactions on Evolutionary Computation*, 2(1):2–22, 1998.

[156] Holger R Maier and Graeme C Dandy. Neural networks for the prediction and forecasting of water resources variables: a review of modelling issues and applications. *Environmental Modelling and Software*, 15:101–124, 2000.

[157] Keith E. Mathias and L. Darrell Whitley. Initial performance comparisons for the delta coding algorithm. In *IEEE Conference on Evolutionary Computation.*, volume 1, pages 433–438, 1994.

[158] Nicholas Freitag McPhee and Riccardo Poli. A schema theory analysis of the evolution of size in genetic programming with linear representations. In *European Conference on Genetic Programming*, pages 108–125, 2001.

[159] D. Michie and R. A. Chambers. BOXES: An experiment in adaptive control. In E. Dale and D. Michie, editors, *Machine Learning 2*, pages 132–152. Edinburgh University Press, Edinburgh, 1968.

[160] Richard Milner. *The Encyclopaedia of Evolution.* Facts on File, 1990. Forward by Stephen Jay Gould.

[161] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, 1996.

[162] Melanie Mitchell. Review of *darwin's dangerous idea* by Daniel C. Dennet. In *Complexity*, volume 2, pages 32–36. 1996.

[163] Bernard M.E. Moret and Henry D. Shapiro. Algorithms and experiments: The new (and old) methodology. *Journal of Universal Computer Science*, 7(5):434–446, 2001.

[164] David E. Moriarty and Risto Miikkulainen. Learning sequential decision tasks. Technical report ai95-229, The University of Texas, Austin, Texas, January 1995.

[165] David E Moriarty, Alan C Schultz, and John J Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:241–276, 1999.

[166] Edgar Noda. Discovering interesting prediction rules with a genetic algorithm. In Una-May O'Reilly, editor, *Graduate Student Workshop*, pages 386–387, Orlando, Florida, USA, 13 1999.

[167] M.O. Odetayo and D.R. McGregor. Genetic algorithms for inducing rules for a dynamic system. *International Conference on Genetic Algorithms*, pages 177–182, 1989.

[168] Una-May O'Reilly and Franz Oppacher. The troubling aspects of a building block hypothesis for genetic programming. In L. Darrell Whitley and D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 73–88. Morgan Kaufmann, 1995.

[169] Akira Otsuki, Mohrihiro Aizaki, and Takayoshi Kawai. Long-term variations of three types of phosphorus concentrations in highly eutrophic shallow lake kasumigaura, with special reference to dissolved organic phosphorus. *The Japanese Journal of Limnology*, 48:1–11, December 1987. Kasumigaura – Characteristics of Water Quality and Ecosystem –.

[170] Raymond C Paton. Principles of genetics. In *Handbook on Evolutionary Computation* [1], chapter A2.2, pages A2.2:1–A2.2:9. Release 97/1.

[171] Carlos Andrés Peña-Reyes and Moshe Sipper. Applying fuzzy coco to breast cancer diagnosis. In CEC00 [43], pages 1168–1174.

[172] Daniel Polani and Risto Miikkulainen. Fast reinforcement learning through eugenic neuro-evolution. Technical Report AI99-277, University of Texas at Austin, 27, 1999.

[173] Daniel Polani and Risto Miikkulainen. Eugenic neuro-evolution for reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, San Francisco, 2000. Kaufmann.

[174] Riccardo Poli. Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. *Genetic Programming and Evolvable Machines*, 2(2):123–163, 2001.

[175] Riccardo Poli. General schema theory for genetic programming with subtree-swapping crossover. In *European Conference on Genetic Programming*, pages 143–159, 2001.

[176] Riccardo Poli and W. B. Langdon. A review of theoretical and experimental results on schemata in genetic programming. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391, pages 1–15, Paris, 14-15 1998. Springer-Verlag.

[177] Riccardo Poli and William B. Langdon. Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation*, 6(3):231–252, 1998.

[178] Riccardo Poli and Nicholas F McPhee. Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. Technical Report CSRP-00-14, 2000.

[179] Mitchell A. Potter. *The Design and Analysis of a Computational Model of Cooperative Coevolution.* Phd Thesis, George Mason University, Fairfax, Virginia, 1997.

[180] Mitchell A. Potter, Kenneth A. De Jong, and John J. Grefenstette. A coevolutionary approach to learning sequential decision rules. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA'95)*, pages 366–372. Morgan Kauffman, July 15–19 1995.

[181] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[182] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, 1993.

[183] J. R. Quinlan. Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[184] I Rechenberg. Cybernetic solution path of an experimental problem. In *Royal Aircraft Establishment*, 1965. Library Translation, 1122. Reprinted [76, page 301].

[185] I. Rechenberg. *Konvergenzraten von Random Search Vergahren zur globalen Optimierung.* Doctoral dissertation, Hochschule der Bundeswehr München, Germany, 1973. Not seen by author.

[186] F Recknagel. *Applied Systems Ecology: Approaches and Case Studies in aquatic ecology.* Akademie-Verlag, Berlin, 1989.

[187] Frieder Recknagel and Jürgen Benndorf. Validation of the ecological simulation model "SALMO". *International Revue Hydrobiology*, 67(1):113–125, 1982.

[188] Friedrich Recknagel. ANNA - artificial neural network model for predicting species abundance and succession of blue-green algae. *Hydrobiologia*, 349:47–57, 1997.

[189] Friedrich Recknagel, Jason Bobbin, Peter Whigham, and Hugh Wilson. Multivariate time series modelling of algal blooms in freshwater lakes by machine learning. In *Proceedings of the 5th WATERMATICS international conference on systems analysis and computing in water quality management*, pages 9.17–9.24, London, September 18–20 2000. International Water Association.

[190] Friedrich Recknagel, Mark French, Pia Harkonen, and Ken-Ichi Yabunaka. Artificial neural network approach for modelling and prediction of algal blooms. *Ecological Modelling*, 96(1–3):11–28, March 1997.

[191] Friedrich Recknagel, Takehiko Fukushima, Takayuki Hanazato, Noriko Takamura, and Hugh Wilson. Modelling and prediction of phyto- and zooplankton dynamics in Lake Kasumigaura by artificial neural networks. *Lakes & Reservoirs: Research and Management*, 3:123–133, 1998.

[192] Friedrich Recknagel, Masaaki Hosomi, Takehiko Fukushima, and Dong-Soo Kong. Short- and long-term control of external and internal phosphorus loads in lakes—a scenario analysis. *Water Resources*, 29(7):1767–1779, 1995.

[193] C S Reynolds. *The Ecology of Freshwater Phytoplankton.* Cambridge University Press, New York, 384pp, 1984.

[194] Debbie Richards, Vijaletchmee Chellen, and Paul Compton. The reuse of ripple down rule knowledge bases: Using machine learning to remove repetition. In *Proceedings of Pacific Knowledge Acquisition Workshop PKAW'96*, Coogee, Australia, October 23–25 1996.

[195] Debbie Richards and Paul Compton. Combining formal concept analysis and ripple down rules to support the reuse of knowledge. In *Proceedings of 1997 Conference on Software Engineering & Knowledge Engineering*, Madrid, 1997.

[196] M. Riedmiller. Concepts and facilities of a neural reinforcement learning control architecture for technical process control. *Journal of Neural Computing and Application*, (8):323–338, 2000.

[197] Rick L. Riolo. The emergence of default hierarchies in learning classifier systems. In *International Conference on Genetic Algorithms 3*, pages 322–327, 1989.

[198] Günter Rudolph. Parallel approaches to stochastic global optimization. In W Joosen and E Milgrom, editors, *Parallel Computing: From Theory to Sound Practice, Proceedings of the European Workshop on Parallel Computing (EWPC 92)*, pages 256–267, Amsterdam, 1992. IOS Press.

[199] Günter Rudolph. Convergence rates of evolutionary algorithms for a class of convex objective functions. *Control and Cybernetics*, 26(3):375–390, 1997.

[200] Günter Rudolph. Finite markov chain results in evolutionary computation: A tour d'horizon. *Fundamenta Informaticae*, pages 1–22, 1998. IOS Press.

[201] Günter Rudolph. Self-adaptation and global convergence: A counter-example. In *Proceedings of the Congress on Evolutionary Computation (CEC'99)*, volume 1, pages 646–651, Piscataway, 1999. IEEE Press.

[202] C. Sammut and J. Cribb. Is learning rate a good performance criterion for learning? In *Proceedings of the Seventh International Workshop on Machine Learning*, pages 170–178, Austin, Texas. Morgan Kaufmann.

[203] N. Saravanan and D. B. Fogel. Multi-operator evolutionary programming: A preliminary study on function optimisation. In P. J. Angeline, R. G. Retnolds, J. R. McDonnell, and R. Eberhart, editors, *Proceedings of the Sixth Annual Conference on Evolutionary Programming*, pages 215–221, Berlin, 1997. Springer.

[204] J. F. Scheafer. *On the Bounded Control of Some Unstable Mechanical Systems.* Ph.D. Dissertation, Department of Electrical Engineering, Stanford University, 1965.

[205] Tobias Scheffer. Learning rules with nested exceptions. In P. Brazdil, editor, *Proceedings International Workshop on Artificial Intelligence Techniques*, Brno, Czech Republic, 1995.

[206] Hans-Paul Schwefel. *Numerische Optimierung vin Computer-Modellen mittles der Evolutoinstrategie*, volume 26 of *Interdisciplinary Systems Research.* Birkhäuser Verlag, Basel, 1977. Not seen by author.

[207] Hans-Paul Schwefel. *Numerical Optimization of Computer Models.* John Wiley and Sons, Chichester, 1st edition, 1981. English translation of [206].

[208] Hans-Paul Schwefel. *Evolution and Optimum Seeking.* John Wiley and Sons, New York, 2nd edition edition, 1995.

[209] Hans-Paul Schwefel and Thomas Bäck. Evolution strategies II: Theoretical aspects. In G. Winter, J. Périaux, M. Galán, and P. Cuesta, editors, *Genetic Algorithms in Engineering and Computer Science, Proc. First Short Course EUROGEN-95*, pages 127–140. Wiley, Chichester, Las Palmas de Gran Canaria, Spain, December 4–8, 1995.

[210] Hans-Paul Schwefel, Günter Rudolph, and Thomas Bäck. Contemporary evolution stategies. In *Proceedings of the European Conference on Artificial Life*, Interdisciplinary Systems Research, pages 893–907, Basel, 1995. Birkhäuser Verlag.

[211] J Shapiro. Blue-green algae: why they become dominant. *Science*, 179:382–384, 1973.

[212] J Shapiro. Current beliefs regarding dominance by blue-greens: the case for the importance of pH and CO_2. *International Revue Ges. Hydrobiology*, 24:38–54, 1990.

[213] Alan C. Shultz. Improving tactical plans with genetic algorithms. In *Proceesing of IEEE Conference on Tools for Artificial Intelligence, TAI'90*, pages 328–334. IEEE Society Press, November 6–9 1990.

[214] Andy Singleton. Some assembly required. *Byte Magazine*, Februrary 1994.

[215] Robert E. Smith and David E. Goldberg. Reinforcement learning with classifier systems: Adaptive default hierarchy formation. *Applied Artificial Intelligence*, 6:79–102, 1992.

[216] Richard Sutton. *Reinforcement Learning: an Introduction.* MIT Press, 1998.

[217] Noriko Takamura and Morihiro Aizaki. Change in primary production in Lake Kasumigaura (1986-1989) accompanied by transition of dominant species. *Japanese Journal of Limnology*, 52(3):173–187, 1991.

[218] Noriko Takamura, Akira Otsuki, Morihiro Aizaki, and Yukihiro Nojiri. Phytoplankton species shift accompanied by transition from nitrogen dependence to phosphorus dependence of primary production in Lake Kasumigaura, Japan. *Archiv Hydrobiologie*, 1992.

[219] P. L. Tan, T. S. Dillon, and J. Zeleznikow. Representing exceptions in rule-based systerms. In C J Barter and M J Brooks, editors, *AI'88: Proc. of the 2nd Australian Joint Artificial Intelligence Conference*, pages 240–255, Berlin, Heidelberg, 1990. Springer.

[220] Tanja Urbančič and Ivan Bratko. Knowledge acquisition for dynamic system control. In B. Souček, editor, *Dynamic, Genetic and Chaotic Programming: The Sixth Generation*, pages 65–83. New York: Wiley, 1992.

[221] H. Vafaie and K. De Jong. Improving a rule induction system using genetic algorithms. In R Michalski and G Tecuci, editors, *Machine Learning IV: A Multistrategy Approach*, pages 453–470. Morgan-Kaufmann, 1994.

[222] Alan Varšek, Tanja Urbančič, and Bodgan Filipič. Genetic algorithms in controller design and tuning. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(5):1330–1339, September/October 1993.

[223] Peter M B Walker, editor. *Chambers science and technology dictionary*, 43–45 Annadale Street, Edinburgh EH7 4AZ, 1991. W & R Chambers Ltd.

[224] David Walter and Chilukuri K. Mohan. Cladia: A fuzzy classifier system for diease diagnosis. In *Congress on Evolutionary Computation*, pages 1429–1435, 2000.

[225] Mark Walter, Friedrich Recknagel, Craig Carpenter, and Myriam Bormans. Predicting eutrophication effects in the Burrinjuck reservoir (Australia) by means of the deterministic model SALMO and the recurrent neural network model ANNA. *Ecological Modelling*, 146:97–113, 2001.

[226] P. A. Whigham. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming : From Theory to Real-World Applications*, pages 33–41. Morgan Kauffmann, 1995.

[227] P. A. Whigham. Inductive bias and genetic programming. In *First International Conference on Genetic Algorithms in Engineering Systems : Innovations and Applications*, pages 461–466, UK, 1995. IEE.

[228] P. A. Whigham. A schema theorem for context-free grammars. In *1995 IEEE Conference on Evolutionary Computation*, volume 1, pages 178–181, Perth, Australia, 29 - 1 1995. IEEE Press.

[229] P. A. Whigham. Search bias, language bias and genetic programming. In *Genetic Programming*, pages 230–237. MIT Press, 1996.

[230] P. A. Whigham. An inductive approach to ecological time series modelling by evolutionary computation. *Ecological Modelling*, 2001. In Press.

[231] Peter A. Whigham. Induction of a marsupial density model using genetic programming and spatial relationships. *Ecological Modelling*, 131(2–3):299–317, 2000.

[232] Peter A Whigham and Friedrich Recknagel. Predictive modelling of plankton dynamics in fresh water lakes using genetic programming. In Les Oxley and Frank Scrimgeor, editors, *International Congress on Modelling and Simulation, MODSIM '99, 6-9 December 1999, Hamilton, New Zealand*, pages 679–684, 1999.

[233] Peter A. Whigham and Friedrich Recknagel. Evolving difference equations to model freshwater phytoplankton. In *2000 Congress on Evolutionary Computation, San Diego, USA*, Piscataway, NJ, 2000. IEEE Press.

[234] D. Whitley, K. Mathias, and P. Fitzhorn. Delta coding: An iterative search strategy for genetic algorithms. In D Whitley, K Mathias, and P Fitzhorn, editors, *Proceedings of Fourth International Conference on Genetic Algorithms*, pages 77–84, San Diego, CA, 1991. Morgan Kaufmann.

[235] Darrell Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. *Proc. of the Third Int'l Conf. on International Conference on Genetic Algorithms and Their Applications*, pages 116–121, 1989.

[236] Alexis P. Wieland. Evolving controls for unstable systems. *Connectionist Models: Proceedings of the 1990 Summer School*, pages 91–102, 1991.

[237] Hugh Wilson and Friedrich Recknagel. Advances in modelling and prediction of algal blooms in freshwater lakes by artificial neural networks. In *Proceedings of the International Congress on Modelling and Simulation MODSIM 97, Hobart, Tasmania*, volume 4, pages 1772–1777, 8–11 December 1997.

[238] Hugh Wilson and Friedrich Recknagel. Towards a generic artificial neural network model for dynamic predictions of algal abundance in freshwater lakes. *Ecological Modelling*, 146(1–3):69–84, December 2001.

[239] Stewart W. Wilson. Bid competition and specificity reconsidered. *Complex Systems*, 2:705–723, 1989.

[240] Stewart W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994.

[241] Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[242] Stewart W. Wilson. Generalisation in the XCS classifier system. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, San Francisco, CA, 1998. Morgan Kaufmann.

[243] Stewart W. Wilson. State of XCS classifier system research. Technical report no. 99.1.1, PREDICTION DYNAMICS, Concord, MA, March 18 1999.

[244] Stewart W. Wilson. Get real! XCS with continuous-valued inputs. In Lanzai et al. [148].

[245] Stewart W. Wilson. Mining oblique data with XCS. Technical report, University of Illinois at Urbana-Champaign, July 2000. Illinois Genetic Algorithms Laboratory, Technical Report No. 2000028.

[246] Stewart W. Wilson and David E. Goldberg. A critical review of classifier systems. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 244–255, San Mateo, CA, 1989. Morgan Kaufman.

[247] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[248] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimisation. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[249] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, September 1999.

[250] X. Yao and Y. Liu. Fast evolution strategies. *Control and Cybernetics*, 26(3):467–496, 1997.

[251] X. Yao and Y. Liu. Fast evolution strategies. In P. J. Angeline, R. G. Reynolds, J.R. McDonnell, and R. Eberhart, editors, *Proceedings of the Sixth Annual Conference on Evolutionary Programming*, pages 151–161, Berlin, 1997. Springer.

[252] X. Yao and Y.Liu. Fast evolutionary programming. In L. J. Fogel, P. J. Angeline, and T. Bäck, editors, *Proceedings of the Sixth Annual Conference on Evolutionary Programming*, pages 451–460, Cambridge, MA, 1996. MIT Press.

[253] Xin Yao. How does evolutionary computation fit into it postgraduate teaching. In *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 3, pages 1707–1713, Piscataway, NJ, USA, July 1999. IEEE Press.

[254] Xin Yao, Yon Liu, and Guangming Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102, July 1999.

[255] Xin Yao and Yong Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3):694–713, May 1997.

Index