

**CONSTRUCTIVIST ARTIFICIAL INTELLIGENCE
WITH
GENETIC PROGRAMMING**

by

Kalyani Govinda Char

A Dissertation Presented to the

Department of Electronics and Electrical Engineering

University of Glasgow, United Kingdom

In Partial Fulfillment of the
Requirement for the Degree

DOCTOR OF PHILOSOPHY

October 1998
(Revised)

copyright © 1998

Kalyani Govinda Char

ProQuest Number: 13815581

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 13815581

Published by ProQuest LLC (2018). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

GLASGOW
UNIVERSITY
LIBRARY

GLASGOW UNIVERSITY
LIBRARY

11291 (copy 1)

Dedication

To my parents.

*'Whatever way they come to Me,
wherever they may be,
it is always in My path they go,
and I return their love just so.'*

The Bhagavad Gita

A Thought!

*'Knowledge is extremely pure and humble.
Access it straight instead of filtering it out
through a dirty filter.'*

Kalyani Govinda Char

Acknowledgments

First and foremost, my thoughts go to my parents who gave me the education and the encouragement throughout my career. My dad, being a scientist, has always believed in a scientific outlook and approach. I am indebted to my parents for everything they gave me. They have been my constant source of inspiration and motivation to continue with research while my coming across several hurdles. I strongly believe that parent's blessings are the greatest wealth that anyone can possess and are the best references to come up in life. I am extremely fortunate to be born in an excellent family that taught me the values of truth, kindness, dignity and love for animals and of course to the mankind. My brother Dr. Dwaraka Nath Char and my sister Dr. Malathi Char are real doctors in life. They wanted me to be different. I chose computing and I am happy with my choice. I am grateful to my brother and his family and my sister for giving me the support and the encouragement to continue in the path that I have chosen.

I am also very fortunate to meet and know excellent people in the research community. I have great appreciation for Dr. Daniel Polani from the university of Informatik-Mathematik, Germany for his interesting discussions on self-organizing neural networks and systems. He is dedicated to science and has been extremely keen on my work. His encouragement made me very strong and gave me the confidence to put new ideas with clarity on paper.

Dr. Walter Alden Tackett not only gave me the motivation and the strength to reach my goal but also an opportunity to be a co-author to one of his excellent publications in the Handbook of Evolutionary Computation, OUP, US. I will always remember his help and advice. In my opinion he belongs to the category of brilliant people who would strive to help anyone, anywhere.

My sincere thanks to Professor. John Koza for providing student's travel grants that enabled me to attend the GP-97 conference. I got excellent feedback from many researchers that in turn helped me a lot although it was a hurried preparation for presentation on my part.

I also appreciate the encouragement for my research work from Professor. Koza Dr. Fogel, Dr. Tom Ray, Dr. Hugo de Garis, Dr. Riccardo Poli, Professor. Goldberg, and from many others whom I met at the conference.

Although I have not met Dr. Gruau I should say that he is highly innovative and an extremely broad-minded person who would do anything to promote research. Earlier I missed an opportunity to go to France when I was given an offer to work with him for three months.

My sincere thanks to Dr. Adam Fraser for his patience in discussing the GP kernel to great depths. This in turn helped me implement my own kernel with neural networks and GP.

Doctorate, I think, is a difficult endeavor though it could be extremely interesting. I was thinking of a project in the field of artificial intelligence but was not sure as to where and how to start. Luckily I came across the genetic programming (GP) paradigm. I was excited and introduced GP to the department. I am happy to say that my efforts have brought in a whole body of research to this department. Various groups have been thriving since then. Unfortunately I could initiate my project only in March 96 after the facilities became available to me. I decided to use GP in the context of AI. I have been very lucky to get good feedback from my own publications to guide me further. Professor. Vaario's work on constructivist AI fascinated me and I thought that GP could be a potential tool for building such systems. I have suggested the basic approach. I am sure this work and also research in AI will be pursued from this department in the future. I appreciate the help given by Professor. Shimohara and Professor. Vaario from ATR laboratories, Japan in sending me their valuable papers and also for their encouragement.

I am grateful to Dr. Roger Waterfall at the university of Manchester, UK and to Dr. Mervyn Curtis at the university of Nottingham, UK, for their help and encouragement before I took up research.

I sincerely acknowledge the help given by Dr. Muir in providing PCs for running my simulation programs.

Thanks to Dr. Davie and also to the technical staff Sam-Kilgour, Raymond McCall, Jon-Trinder, Bridget Sweeney and Stephen Gallagher for their help.

Coming to the thesis, I would like to thank Professor Murray-Smith for going through the chapters and providing useful comments in improving the presentation.

Glasgow had some attractions for me most notably, the smart blue-tits and squirrels asking for nuts in the park, their soft touch and the friendly doggies, Sheila, Poppy, Kerry, and Prince.

My landlord Tom Dickson and his family have been a great help in providing a very decent accommodation and an atmosphere for my research work.

Finally, my friends especially Mary, George and Sheila, Manu Ahluwalia, Tassos Fragos, John Moxey, Wye Teoh, John Parle, Scott Ray and my well-wishers deserve appreciation.

Kalyani Govinda Char

UK, October 1998

List of Publications

1. K. Govinda Char, Constructive Learning with Genetic Programming, EuroGP-98, France.
2. K. Govinda Char, Constructivist AI with GP (Late-breaking paper), the Genetic-Programming Conference, GP-97, Stanford, USA.
3. K. Govinda Char, Constructivist AI with GP, IJCAI97, the International Joint Conference on Artificial Intelligence, Nagoya, Japan.
4. K. Govinda Char, AI-Revisited, SOCO97, the ICSC Symposium on Soft Computing, France.
5. K. Govinda Char, A Novel Approach to Artificial Intelligence, Tainn'97, the Sixth Turkish Symposium on Artificial Intelligence and Neural Networks, Baskent University, Turkey.
6. K. Govinda Char, Modeling Self-organization - Approaches and a Comparison with Evolutionary Methods, BCEC97, the International Conference on Bio-Computing and Emergent Computation, Skovde, Sweden, published by World Scientific Publishing.
7. K. Govinda Char, Evolution of Structure and Learning with Genetic Programming, IWANN'97, the International Workshop on Artificial Neural Networks, Canary Islands, Spain, published in the Lecture Notes in Computer Science.
8. K. Govinda Char, Modeling Self-Organization- a Comparison, GWAL'97, the Second German Workshop on Artificial Life, University of Dortmund, Hans Bommerholz, Germany.
9. K. Govinda Char and Walter Alden Tackett, Pattern Recognition (section F1.6) in the Handbook of Evolutionary Computation, 1997, Oxford University Press, USA.
10. Walter Alden Tackett and K. Govinda Char, Genetic Programming Applied to Image Discrimination (section G8.2) in the Handbook of Evolutionary Computation, 1997, Oxford University Press, USA.
11. K.G. Char, Self-organization with Adaptive Learning, ICML'96, the International Conference on Machine Learning, Bari, Italy.
12. K.G. Char, A Learning Rule for Emerging Structures, WCNN96, the World Congress on Neural Networks, San Diego, USA.
13. K.G. Char, Emergence of Structures in Self-organizing Neural networks using Genetic Programming and Cellular Encoding techniques, Tainn96, the National Conference on

Artificial Neural Networks, Istanbul, Turkey.

TABLE OF CONTENTS

Dedication.....	i
A Thought!.....	ii
Acknowledgments.....	iii
List of Publications.....	vi
List of Figures and Tables.....	xii
Abstract	xiii

CHAPTER 1	CONSTRUCTIVISM AND AI	1
1.1	Introduction	1
1.1.1	Genetic algorithms	3
1.1.2	Genetic programming	4
1.1.3	Artificial Life	4
1.2	Contributions of the research	5
1.3	Outline of the research	8

CHAPTER 2	BACKGROUND AND THE NEW PERSPECTIVE OF AI	11
2.1	A brief history of Artificial Intelligence	11
2.2	The new perspective of AI	13
2.2.1	The Credit Assignment Problem	13
2.2.2	The Knowledge Acquisition Bottleneck	14
2.2.3	Memory Indexing Problem	14
2.2.4	The Problem of Scaling	14
2.2.5	The Representation Design	15
2.2.6	The Software Crisis	15
2.3	Artificial Intelligence seen as Emergent Intelligence	15
2.3.1	Evolutionary Algorithms	16
2.3.2	Reactive Systems	17
2.3.3	Eclectic Hybrids	17

2.3.4 Automatic Programming	18
2.3.5 Constructivist Systems	19
CHAPTER 3 EVOLUTIONARY COMPUTATION	21
3.1 An overview	21
3.1.1 Genetic Algorithms	24
3.1.1.1 Population	26
3.1.1.2 Fitness Evaluation	26
3.1.1.3 Selection	27
3.1.1.4 Trait Inheritance and Recombination	29
3.1.1.5 Mutation	30
3.1.1.6 The Schema Theorem and the Building Block Hypothesis	30
3.1.1.7 Deception and Royal Road	32
3.1.1.8 Hybrid Algorithms	32
3.1.1.9 Parallelism in the Genetic Algorithm	32
3.1.2 Genetic Programming	33
3.1.2.1 Population	33
3.1.2.2 Fitness Evaluation	35
3.1.2.3 Selection	36
3.1.2.4 Trait Inheritance and Recombination	36
3.1.2.5 Mutation	37
3.1.2.6 The search mechanism	37
3.1.2.7 The Schema Theorem and the Building Block Hypothesis: the GP analogy	39
3.1.2.8 Deception and Royal Road: the GP analogy	41
3.1.2.9 Hybrid Algorithms	41
3.1.2.10 Parallelism in Genetic Programming	41
CHAPTER 4 EVOLUTION OF STRUCTURE AND LEARNING	43
4.1 Introduction	43

4.2	Connectionist Networks	45
4.3	The role of evolutionary algorithms in connectionism	46
4.3.1	The Genetic Algorithm approach	47
4.3.1.1	Network Induction	48
4.3.1.2	Induction of Learning	53
4.3.2	The Genetic Programming approach	59
4.3.2.1	Network Induction	59
4.3.2.2	Induction of Learning	60
4.4	Discussions	63
 CHAPTER 5 SELF-ORGANIZING NEURAL NETWORKS		67
5.1	Introduction	67
5.1.1	The Kohonen Self-organizing Feature Map: the characteristics and the Learning Rule	67
5.1.1.1	Performance Criteria	69
5.1.1.2	The Problems and Limitations	70
5.1.2	The Growing Cell Structures	71
5.1.3	The Enhanced Feature Map: Modeling Lateral Interactions	71
5.1.4	Incremental Grid Growing	72
5.2	The Evolutionary Approach	72
5.2.1	The Genetic Algorithm Approach	73
5.2.1.1	The Genotype-Phenotype Mapping	73
5.2.1.2	The fitness function	76
5.2.2	The Genetic Programming Approach	77
 CHAPTER 6 SIMULATION RESULTS		80
6.1	The Framework	80
6.2	The Problem	80
6.2.1	The environment	80
6.2.2	The task	81
6.2.3	The basic steps	81

6.3 The Genetic Programming Approach	82
6.3.1 The General Approach	83
6.3.1.1 Sample Programs	85
6.3.1.2 The issues	86
6.3.2 The Modular Approach	89
6.3.2.1 Advantages of Modularity in the context of learning	91
6.3.2.2 Sample Programs	93
6.3.2.3 The Fitness criterion	94
6.3.2.4 Co-evolution of Structure	103
6.5 A Comparison: GA vs. GP	104
6.6 Discussions	106
CHAPTER 7 CONSTRUCTIVIST AI WITH GENETIC-PROGRAMMING	108
7.1 The Background	108
7.2 The GP Approach	111
7.3 A Comparison between the two Approaches	113
7.4 Discussions	114
CHAPTER 8 SUMMARY, CONCLUSIONS AND DIRECTIONS FOR FURTHER RESEARCH	116
8.1 Summary and Conclusions	116
8.2 Possible Applications with the proposed Approach	118
APPENDICES	
A Two different models for the pattern recognition task.	
B The initialization file for the GP run.	
C The graphs of GP run for various parameters.	
BIBLIOGRAPHY	

List of Figures and Tables

Figures and Tables	Description
3.1	An evolutionary cycle.
3.2	A population of genotypes.
3.3	The decoding of the genotypes.
3.4	The Roulett wheel selection scheme.
3.5	Single-point crossover.
3.6	Mutation of a genotype.
3.7	The Lisp expression and the Parse tree.
3.8	A sample genetic program.
3.9	Crossover in GP.
5.1	The Kohonen Feature Map.
5.2	Neural network encoded in a connection matrix.
5.3	Interaction between the GA and the Feature Map.
5.4	A transcription rule.
5.5	A chromosome evolving a network connections.
6.1	The two models used for the evolution of learning.
6.2	Self-organization for the signals drawn from a circular model with the standard Kohonen rule (Appendix-A).
6.3	Self-organization for a square model with the Kohonen rule (Appendix-A).
6.4	An illustration of the ADFs.
6.5	An example of the types of ADFs for evolving a Kohonen-type of learning.
7.1	The life cycle of an adaptive system.
7.2	The hierarchical representation of computational models.
7.3	Cellular operators for neural network creation.
7.4	A comparison between the Rule-based and the GP approaches.

Abstract

Learning is an essential attribute of an intelligent system. A proper understanding of the process of learning in terms of knowledge-acquisition, processing and its effective use has been one of the main goals of artificial intelligence (AI). AI, in order to achieve the desired flexibility, performance levels and wide applicability should explore and exploit a variety of learning techniques and representations. Evolutionary algorithms, in recent years, have emerged as powerful learning methods employing task-independent approaches to problem solving and are potential candidates for implementing *adaptive* computational models. These algorithms, due to their attractive features such as implicit and explicit parallelism, can also be powerful meta-learning tools for other learning systems such as connectionist networks. These networks, also known as artificial neural networks, offer a paradigm for learning at an individual level that provide an extremely rich landscape of learning mechanisms which AI should exploit.

The research proposed in this thesis investigates the role of genetic programming (GP) in connectionism, a learning paradigm that, despite being extremely powerful has a number of limitations. The thesis, by systematically identifying the reasons for these limitations has argued as to why connectionism should be approached with a new perspective in order to realize its true potentialities. With genetic-based designs the key issue has been the encoding strategy. That is, how to encode a neural network within a genotype so as to achieve an optimum network structure and/ or an efficient learning that can best solve a given problem. This in turn raises a number of key questions such as:

1. Is the representation (that is the genotype) that the algorithms employ sufficient to express and explore the vast space of network architectures and learning mechanisms?
2. Is the representation capable of capturing the concepts of hierarchy and modularity that are vital and so naturally employed by humans in problem-solving?
3. Are some representations better in expressing these? If so, how to exploit the strengths that are inherent to these representations?
4. If the aim is really to automate the design process what strategies should be employed so as to minimize the involvement of a designer in the design loop?

5. Is the methodology or the approach able to overcome at least some of the limitations that are commonly seen in connectionist networks?
6. Most importantly, how effective is the approach in problem-solving?

These issues are investigated through a novel approach that combines genetic programming and a self-organizing neural network which provides a framework for the simulations. Through the powerful notions of constructivism and micro-macro dynamics the approach provides a way of exploiting the potential features (such as the hierarchy and modularity) that are inherent to the representation that GP employs.

By providing a general definition for learning and by imposing a *single* potential constraint within the representation the approach demonstrates that genetic programming, if used for construction and optimization, could be extremely creative. The method also combines the bottom-up and top-down strategies that are key to evolve ALife-like systems.

A comparison with earlier methods is drawn to identify the merits of the proposed approach.

A pattern recognition task is considered for illustration. Simulations suggest that genetic-programming can be a powerful meta-learning tool for implementing useful network architectures and flexible learning mechanisms for self-organizing neural networks *while* interacting with a given task environment. It appears that it is possible to extend the novel approach further to other types of networks.

Finally the role of flexible learning in implementing adaptive AI systems is discussed. A number of potential applications domain is identified.

Chapter 1

Constructivism and AI

Constructivism, as applied to artificial intelligence (AI), is the notion that adaptive behavior can be constructed through the interaction of primitive elements and processes. Whether an evolutionary algorithm such as genetic programming offers a way to extend this notion to construct flexible learning mechanisms that are vital to building adaptive AI systems is the focus of this research. This chapter will outline the contributions of the proposed research.

1.1 Introduction

Learning- the process through which knowledge is acquired, organized, refined and effectively used is an essential attribute of an intelligent system. The aim of machine learning (ML) has been to understand the nature of learning and to implement learning capabilities in machines. These goals are central to the research in artificial intelligence in building adaptive and flexible computational models capable of working in complex task environments. Conventional AI systems, in particular, knowledge-based systems, had almost no learning capabilities as these employed knowledge representation and search techniques that relied on explicit knowledge (Angelene, 1993). Generally, the methods encoded the domain knowledge and also the problem solving knowledge within the problem-solver explicitly rather than having the problem-solver learn these. This led to two major issues. Firstly, to find good representations for representing knowledge accurately is an extremely difficult task due to the nature of the knowledge itself. Knowledge, in general, is voluminous, and hard to characterize and represent accurately. Also it constantly changes and it is difficult to infer how much knowledge is needed to solve a given problem (Rich and Knight, 1991). Secondly, to find effective techniques capable of dealing with the knowledge having the above characteristics. As a result, these computational models, despite being successful in well-defined task domains failed to perform effectively in unpredictable and non-static task environments. It was realized that in order to be flexible and adaptive the models should have the capability to acquire knowledge during the process of problem solving and use it effectively. In other words, the need for

learning became evident. Over the years a number of new machine learning paradigms emerged. These include ID3 (Quinlan, 1986) which is a method of inducing decision trees from the contents of a given data set, rule induction software CN2 (Clark and Niblett, 1989; Greab and Narayanan, 1998) used for symbolic data mining, neural network data mining methods, other inductive learning techniques (Someren and Verdenius, 1998) and the evolutionary algorithms (EAs) that are based on Darwinian principles of natural selection. Evolutionary algorithms such as genetic algorithms (GAs) (Holland, 1975; 1992; Goldberg, 1989), evolutionary programming (EP) (Fogel, 1992; 1994) and genetic programming (GP) (Koza, 1993) became very popular for the reasons that these employ a different approach to problem solving. Their powerful features mainly include their ability:

1. to solve problems using representations and operators that are task-independent allowing the task-specific knowledge to emerge during the course of problem solving. This approach to problem solving avoids the reliance on explicit knowledge.
2. to conduct parallel searching over a large complex search space due to their implicit and explicit parallelisms.

These features along with expedience make the algorithms generally applicable to a variety of problems over a wide range of domains. A further advantage with evolutionary algorithms are that these can be hybridized with other machine learning methods such as ID3 complementing each other in their performance (Carter and Narayanan, 1998).

As learning paradigms evolutionary algorithms are potential candidates for implementing adaptive AI systems. On the other hand, these algorithms on their own are not powerful enough to solve all types and classes of problems. A few examples include hard learning problems (also known as type-2 learning problems) (Clark and Thornton, 1993; Thornton, 1994) and those that can only be learnt incrementally such as in language learning (Elman, 1991). In hard learning problems the learning refers to possible 'relationships' among the input variables instead of the variables themselves. This makes the learning extremely difficult. In language learning with neural networks it has been observed that the network fails to learn complex grammar when both the network and the input remain unchanging. However, when either the input is presented incrementally, or the network begins with

limited memory that gradually increases, the network is able to learn the grammar. AI models, therefore, should explore a wide variety of learning methods in order to be applicable over a wide range of problem solving environments. Connectionist networks, for example, provide powerful ways of solving certain classes of problems (Clark and Lutz, 1992). The network models are taught rather than programmed and they solve a problem by learning a set of internal representations. However, connectionism has shown limitations that mainly appear to stem from a number of rigid assumptions and inflexible approaches through which the networks and their learning rules are implemented (Govinda Char, 1998). The space of network architectures and learning being extremely large, evolutionary algorithms have been very successfully employed for network induction and learning for a number of complex task domains.

AI, in order to achieve real flexibility and performance levels, should exploit the strengths of various representations and strategies and the potentialities of integrating these.

The subsections ahead will provide a brief introduction to genetic algorithms, genetic programming and Artificial Life (Langton, 1989) as these relate to the work in this thesis. Chapter four discusses connectionist networks in detail.

1.1.1 Genetic algorithms

Genetic algorithms encode solutions to a problem through a representation, typically a string of symbols, the genotype. Generally the length of the string is fixed. Each genotype represents a point in the search space. A number of genotypes are randomly produced to form a population. Each of the corresponding points in the search space is evaluated by an appropriate evaluation function that gives a higher scores to those nearest the solution sought. The next generation is generated from the present population by selection and reproduction. The fitter individuals are selected and a new generation of genotypes is derived using crossover and mutation. The crossover operator works by choosing two parent genotypes, selecting a crossover point along the length of the genotype at random and swapping parts of the genotypes. The offspring inherits the genetic material from both parents. The mutation operator changes some symbols on the genotype at random. These

operations generate new points in the search space and the fitness is expected to improve over the course of a number of generations.

1.1.2 Genetic programming

The notion behind genetic programming (GP) is that a great variety of problems from different fields can be reformulated as problems of program induction. Genetic programming provides a way of searching through genetic algorithms the space of possible computer programs. GP uses a population of programs that are expressed as LISP S-expressions. These in turn can be depicted as rooted point-labelled trees with ordered branches. The genotype is a tree of variable length, size and complexity that is composed of the function and terminal sets for the problem domain. The recombination of trees is by crossover where complete subtrees of two parent trees are swapped to exchange genetic material. This also results in syntactically correct offspring. The output of the program is the value returned by the S-expression composed of the whole tree. To initiate the GP run, a set of function and terminal sets appropriate to problem domain are chosen and used to create the random trees. Usually the depth of the trees is controlled. Each of the S-expressions is evaluated based on a certain fitness measure that is appropriate to the problem in hand. The parents for the next generation is selected based on their fitness.

The tree representation has several advantages: the search space is not limited as in the case of a fixed-length string representation. When the size and the complexity of the solutions are not known in advance a tree representation is highly desirable. Also it allows any hierarchy in the problem solving process to be expressed naturally. Further the representation can be extended to incorporate modularity with automatically defined functions (ADFs) (Koza,1994).

1.1.3 Artificial Life (AL)

Artificial-Life is the study of man-made systems that exhibits behavioral characteristics of natural living systems. It attempts to synthesize life-like behaviors within computers and artificial media. Artificial intelligence (AI) tends to simulate high-level problem solving behavior through computational models. AL, in contrast start from the bottom-up to understand how primitive low level processes can produce emergent complex behavior.

Researchers have attempted to define *emergence* in different ways. As an example, Harvey (1993), tries to give a general definition for emergence. Something can be characterised as emergent relative to an initial given description if:

1. a system can be set up which corresponds completely to this initial given description.
2. a new description of the behavior of the system can be made which 'is useful' or 'makes sense' to an observer, and makes use of concepts outside those originally given.

Some (Chalmers, 1990; Vaario, 1993 and others) conceptualize emergence in terms of achieving a high-level complex behavior through the interactions of low-level elements and processes. The notion of emergence is *key* to AL work. Typically, the behaviors are simulated through robots in different environments. Evolutionary approaches are a common theme.

1.2 Contributions of the research

Evolutionary algorithms have proven to be powerful search and optimization methods due to their attractive properties such as the implicit and explicit parallelism, and robustness. However, it is argued that these algorithms lack the creativity needed to build adaptive systems that are endowed with properties such as adaptation, self-replication, and self-organization (Vaario, 1993) that are characteristics of ALife-like systems.

Constructivism is the notion that adaptive behavior can be constructed through primitive elements and processes. The proposed research through a novel approach illustrates how this notion can be extended to evolve flexible learning with genetic-programming. The approach involves two phases.

1. Integrating GP with a powerful learning paradigm such as the connectionist networks. In hybridizing, it is vital to understand the limitations of each of the paradigms (the components) that constitute the hybrid in order to implement a computational model that is highly effective in problem solving in known/unknown environments. It is crucial that the components of the hybrid need to be appropriately chosen and also combined in ways

such that they complement each other in the task of problem solving. Connectionist networks and evolutionary algorithms offer a paradigm for emergence at two different levels, viz., at an individual and at the population level. Connectionist networks support synchronic emergence or emergence over levels. At a given time a host of low-level computations take place and can be interpreted as a complex high-level functioning when observed from another level. Evolutionary algorithms support diachronic emergence, that is emergence over time. Primitive computational systems, over time, gradually evolve towards greater complexity. The hybrid has to support emergence at both levels in order to be effective. The proposed research through a novel approach shows how genetic programming can naturally be combined with connectionist networks. Whether such a combination can enable evolution of flexible learning mechanisms is investigated (Govinda Char, 1997a).

2. Understanding the role of genetic programming as a meta-learning system for connectionist networks.

Recently, optimal network topologies have been evolved with GP (Zhang and Mühlenbein, 1993; Poli, 1997). So far there have been very few attempts to evolve network learning rules (Radi and Poli, 1998). Radi and Poli have succeeded in evolving rules that are faster and more effective as compared to the standard back-propagation learning. The novel method that is proposed in this thesis employs an entirely different approach and a strategy that allows the network and learning to evolve *during* the process of problem solving. This strategy in turn raises a number of important questions such as:

- a. Should there be a general definition for a connectionist learning rule?
- b. Should there be any constraints involved in implementing a learning rule? If so, how and where should these be imposed?
- c. Does the implementation entail other potential strategies?

The current research has systematically addressed these issues to illustrate how flexible learning rules can evolve *while* interacting with a given task environment. The approach involves providing a very general definition for a connectionist learning rule (irrespective

of the type of network architecture), imposing a single potential constraint within the GP's representational structure and employing a potential strategy for constructing and combining the components of the learning rule. The single potential constraint in concert with the proposed strategy creates a paradox for the GP to be creative and enables flexible learning rules to emerge. This approach offers the evolutionary paradigm an open-endedness in terms of the architecture and also the node activation function that can be made to evolve through appropriate primitives. Although a self-organizing neural network is used as a framework it appears that the approach has a potential to be extended to other types of networks. The key aspects are: first, it attempts to exploit the powerful features (that is hierarchy and modularity) of GP's representation. Second, by providing a general definition for learning and imposing a single potential constraint within the representation the method creates a paradox for GP to be creative. Third, it combines the bottom-up and top-down strategies that are vital to generate complex behavior. Fourth, it offers a way to interpret the evolved rules through modular elements.

The simulations suggest that GP can be a powerful meta-learning tool capable of exploring an extremely rich landscape of learning techniques.

The aim, finally, is to understand/investigate the potentialities of the proposed hybrid in adaptive AI systems. As discussed in section 1.1 the reliance of conventional AI systems on explicit knowledge led to a number of problems limiting their applicability. Intelligence, if conceptualized as an adaptive behavior, can be constructed through primitive elements and processes (Vaario, 93). Such an assumption allows one to explore a wide range of paradigms and techniques that can work in task-independent ways. A recent model of an adaptive AI system employing constructivist strategy has been discussed. This model incorporates adaptation in the form of development, neural plasticity, natural selection and genetic changes and has been highly successful in implementing powerful autonomous systems. How GP can naturally incorporate all these forms of adaptation and how the notion of constructivism can be extended to implement adaptive AI systems through flexible learning are discussed.

1.3 Outline of the research

The chapters are organized as follows.

Chapter two provides background information on conventional artificial intelligence. In particular, the focus is on the knowledge-based systems that rely on explicit knowledge for problem solving. The problems associated with the reliance on explicit knowledge are highlighted and the importance of learning is emphasized. How AI, in its new perspective, has been successful in avoiding such reliance through new paradigms and techniques is discussed. A few such models that include evolutionary algorithms, reactive systems, eclectic hybrids, automatic programming techniques and finally constructive systems are briefly described to illustrate the notion of emergence.

Chapter three first provides a brief overview of the field of evolutionary computation. The advantages of using evolutionary algorithms over conventional techniques are briefly discussed. Genetic algorithms and genetic programming are discussed in greater detail for two main reasons. First, to provide good background information to those who are new to this field. Secondly, due to its relevance to the proposed research where the focus has been to investigate the role of genetic programming as a meta-learning system.

Chapter four emphasizes the role of connectionist networks in implementing powerful learning mechanisms for complex problem solving tasks. The learning at an individual level is vital to a system whether natural or artificial. The learning at sub-symbolic levels provide an extremely rich landscape that has not been fully explored. The recent methods in neuro-evolution and genetic-connectionism seem to provide an answer in searching this large, complex space of possible network architectures and learning rules. After providing a general introduction to connectionist networks, the problems associated with their design are discussed in detail. Induction of network architecture and learning are considered in the contexts of genetic algorithm and genetic programming. A comparison with other recent methods is drawn. The advantages of GP approach are identified. The key assumptions and the approach for induction of learning are briefly stated.

Chapter five discusses self-organizing neural network that provides a framework for subsequent simulations. The characteristics and the performance criteria of the network and learning rule are discussed. Evolutionary and non-evolutionary methods for achieving self-organization are described to highlight the advantages of the latter approach. How genetic programming can be used to implement similar learning rules and its advantages are explained.

Chapter six demonstrates the evolution of learning rules for self-organizing neural networks with genetic programming. The key assumptions, the issues and the implications are stated. How flexible learning rules can be evolved *while* interacting with a task-environment is illustrated. The approach provides a general definition for learning, imposes a single potential constraint within the GP's representational structure and employs a potential strategy. Due to the general definition for learning and the general approach, it appears that the method can be extended to other networks such as feed-forward and the recurrent networks. Further, the node activation function can evolve allowing learning for non-homogenous networks. A sample program illustrates how the network architecture can be evolved with a compatible grammar such as the cellular encoding (Gruau, 1993). The simulations emphasize the importance of automatically defined functions (ADFs) in implementing flexible network architecture and learning rules and also in achieving the comprehensibility of the rules that evolve. Finally the advantages are summarised.

Chapter seven aims at illustrating the role of flexible learning in implementing computational models of adaptive AI systems. To understand the underlying principles a recent computational model (Vaario, 1993) that employs the idea of emergent behavior is described. The global behavior is achieved through the interaction of local behavioral rules. The model consists of a neural network that grows in a dynamic environment and incorporates adaptations through development, neural plasticity, natural selection and genetic variations. A set of production rules describe the interactions at different hierarchical levels. This model could implement potential autonomous systems. How the above forms of adaptations can naturally be realized with GP is discussed. The flexible learning mechanisms that evolve with the GP-hybrid might replace the production rules. A

comparison is drawn. These suggest that GP if used for construction and optimization can be extremely creative.

Chapter eight finally provides a summary of the research and conclusions and discusses potential application areas for the GP-hybrid.

Conclusions

Learning is crucial for achieving adaptive behavior. AI should explore and exploit the strengths of various learning methods and strategies to build adaptive computational models. The work in this thesis focuses on employing genetic programming as a meta-learning tool for implementing flexible networks and learning rules for self-organizing neural networks. The aim is to understand how the powerful notion of constructivism can be effectively extended to such domains. The need to exploit the strengths of GP's representation is stressed. The role of such hybrids in AI should be investigated by applying these to complex task environments.

Chapter two will discuss conventional AI and the associated problems in detail.

Chapter 2

BACKGROUND AND THE NEW PERSPECTIVE OF AI

This chapter provides background information relating to conventional Artificial Intelligence (AI) and discusses some of the issues that have imposed limitations in achieving the broad goals of AI. The advantages of employing recent paradigms in overcoming the above limitations and the new perspective of AI are discussed.

2.1 A brief history of Artificial Intelligence

The ability to *learn* being fundamental to any intelligent behavior, the goal of Artificial Intelligence (AI) research (Schank, 1987) has been to create computational models to study human intelligence in terms of learning and problem solving (Newell and Simon, 1963 and many others). Conventional symbolic AI systems typically employed top-down strategies and had very limited learning capabilities as the entire knowledge for problem solving along with the domain knowledge were programmed into the systems. These models, being deductive by nature were too rigid and specialized though these were very successful in tackling well-defined problems. It was realized that flexible systems with capabilities of learning were needed to solve a wide range of problems in complex and non-static environments. Accordingly, such systems should possess the abilities to acquire new knowledge, to automatically generate their algorithms, to develop new solutions by drawing analogies to old ones or through discovery and to improve with experience. That is, to acquire the ability to draw inductive inferences from the information given to them (Michalski, Carbonell and Mitchell, 1986). Hence to understand the nature of learning and to implement learning capabilities in machines also became the goals of AI research. With some learning capabilities the later versions of AI systems overcame some of the earlier limitations and brittleness through the creation of inductive AI systems. Connectionism, the subsymbolic approach, seemed to offer an alternative to symbolic AI in terms of providing a fundamentally new view for knowledge representation and inference. Connectionist networks are massively parallel interconnected networks of simple (usually adaptive) elements which mimic the biological nervous systems (Lippmann, 1987). Working on a bottom-up strategy, these networks constitute a radically different

approach to computation and exhibit some of the important properties such as association, generalization, parallel searching, adaptation to changing environments that are common characteristics of natural systems. One of the most important properties of these networks is their ability to learn from examples. These networks were capable of solving problems where the algorithmic approach was infeasible because of the difficulty in expressing and specifying the sequence of steps (hard-to-write-algorithms). Nevertheless, these networks had their own limitations such as the inability to express the problem solving process in symbolic natural language for humans to interpret. Symbolic systems, on the other hand, were more successful in mimicking high-level human thinking. Novel symbolic learning systems such as ID4 (an extension of ID3), C4.5 (Quinlan, 1993) and CN2 capable of displaying learning characteristics similar to connectionist networks emerged over the years. It was realized and argued (Minsky, 1990) that Artificial Intelligence must employ hybrid approaches that combine different paradigms to take advantage of the strengths of each of the paradigms, each with its own justification. That is, to combine the expressiveness and versatility of symbolic representations with the fuzziness and the adaptive capability of connectionist representations overcoming the constraints that were inherent to either of these paradigms. Towards this goal the symbolic and subsymbolic paradigms were integrated and the hybridized models (Honavar and Uhr, 1994 and many others) were able to successfully tackle a number of difficult problems. These suggested that the two paradigms could complement each other in the process of problem solving.

The implications are that the potentialities of these paradigms need to be fully exploited and combined with other related paradigms to achieve the broad goals of AI. That is, to create computational systems that can not only exhibit intelligence similar to those seen in natural systems but can out-perform these systems in the task of problem solving from a wide range of domains. The research in the subtopics of AI and in Machine Learning in recent years along with the development of new computer architectures indicate a tremendous progress in the field of AI.

The section ahead will discuss some of the key issues with the conventional AI systems and focus on the new perspective of AI through the current trends.

2.2 The new perspective of AI

Conventional AI assumes *intelligence* to be a combination of knowledge in symbolic form and techniques that can manipulate this knowledge. As a consequence of this assumption the AI models rely on explicit knowledge requiring the problem solving knowledge to be placed within the problem solver using some representation (Angelene, 1993). These methods, generally known as strong methods, are rich in task-specific knowledge and have been efficient in solving problems in well-defined domains. Explicit knowledge guides the search mechanism during problem solving. This reliance on explicit knowledge, however, has resulted in a number of issues that, in turn, have imposed limitations on the model's capabilities in tackling problems that are complex and non-static in nature (Brooks, 1986; Vaario, 1993). The issues are discussed in the following subsections.

2.2.1 The Credit Assignment Problem

Knowledge-based AI systems typically employ a representation, that is some kind of data-structure to explicitly represent knowledge and the goals and an algorithm that can effectively manipulate this knowledge. The algorithm that performs the problem solving is referred to as a problem solver. The credit assignment problem (Minsky, 1967) highlights the issue of how to convert the feedback of problem-solving into information about how to manipulate a knowledge structure internal to the problem solver. The two forms of credit assignments are: the global and the local credit assignments. The global credit assignment problem is to determine the fact that there is an error in the internal knowledge structure. Typically this is determined by explicit goals or an evaluation function within the problem solver. In the case of explicit knowledge global credit assignment is determined by the inability of the problem solver to correctly solve the problem at hand. The local credit assignment problem is the identification of the components of the internal structure that are erroneous. In conventional AI systems the explicit knowledge of how to identify the faulty structure is also added to the knowledge base (Schank and Leake, 1989). This knowledge is often task-specific and relates the feedback from the task environment directly to the faulty components. The subsection ahead on evolutionary algorithms explains the problem of credit assignment more clearly in terms of a representational structure and the fitness function .

2.2.2 The Knowledge Acquisition Bottleneck

The difficulty in determining how a program should interact with an expert to extract the expert's knowledge to incorporate it into the problem solver (Hayes Roth et al. 1983) is known as the 'knowledge acquisition bottleneck'. In general terms, the problem of extracting sufficient knowledge from the task environment external to the problem solver and incorporating them into the problem solver is the bottleneck. The expert's knowledge needs to be properly represented within the format of the representational structure in order to be effective. With knowledge-based AI techniques the knowledge of 'how to acquire' the task-specific knowledge is also needed to be supplied explicitly to the problem solver. (Davis, 1979).

2.2.3 Memory Indexing Problem

For task-specific applications knowledge is stored in memory as 'experience' in terms of the instances of problem solving. This knowledge-base as a result is quite large. A particular piece of knowledge (as experience) is retrieved whenever it is appropriate for problem solving through some task-specific memory indexing scheme. These methods include Case-Based Reasoning (CBR) (Kolodner, 1989) and Explanation-Based Learning (EBL) (De Jong and Mooney, 1986 and others). The problem with a large knowledge-base is that at any instant only a small percentage of the knowledge is relevant for problem-solving but this knowledge has to be accessed by searching a large space each time the need arises. The memory indexing schemes take up a prohibitive time for searching a large knowledge base. This time can be reduced to some extent by allowing the possibility of retrieving similar knowledge with a specific index. Also when the task changes, the knowledge-base will have to be re-indexed suggesting the inflexibility of such memory indexing schemes.

2.2.4 The Problem of Scaling

The accuracy of the problem solver depends entirely on the accuracy of the explicit knowledge in the knowledge base. The need for the quantity of explicit knowledge increases exponentially to meet the accuracy even by an order of magnitude. Moreover, for complex problems the quantity of knowledge to represent the task environment may be prohibitive. The necessary level of accuracy of explicit knowledge may be unachievable.

The knowledge-base for ‘common-sense’ internal to a problem solver could be enormous even for simple problems (Lenat, Prakash and Shepard, 1986).

2.2.5 The Representation Design

It is required to represent a task in ways that not only reduce the explicit knowledge to a manageable level but provide maximal computational benefits. To achieve this goal various representations were developed. These include production systems (Newel and Simon, 1981), Predicate calculus (Nilson, 1980), fuzzy logic (Zadeh, 1965), connectionist networks (Rumelhart and McClelland, 1986), semantic networks (Brachman, 1979), frames (Minsky, 1975), conceptual structures (Sowa, 1984), scripts (Schank and Ableson, 1977), the multiple representation of generic tasks (Chandrashekharan, 1986) and others. The directed design of the representations implies *a priori* knowledge of the task and the algorithmic ways of tackling it forcing a human to remain in the problem solving loop.

(See Angelene, P., (1993) for the rest of the references for the above subsections).

2.2.6 The Software Crisis

Computer science is based on assumptions that everything could be predefined and then executed by following a predefined set of instructions. That is, the program does not change once it has been written. Software comprises a set of instructions designed to perform a particular task and the instructions are executed blindly. This principle has led to the so-called software crisis (Vaario, 1994): “The more complex software becomes, the exponential more time it takes to finish”. The recently developed Genetic Programming based on the principle of evolution has been successful in tackling the crisis.

2.3 Artificial Intelligence seen as Emergent Intelligence

New AI models have approached AI with a different perspective by avoiding the model’s reliance on explicit knowledge. These models are typically based on the notion of emergence. Traditionally, the notion of emergence involves the idea of a system behaving in a way which cannot be predicted through some simpler linear combination of low level units.

In the context of evolutionary algorithms, based on the abilities of empirical credit assignment, the emergent intelligence relies on two main assumptions about computational problem solving (Angelene, 1993).

1. The task environment itself is often a more concise representation for knowledge specific to the task than any internal representation of the explicit knowledge.
2. Direct interaction of a simple problem solver with the task environment permits the task environment's inherent constraints to be expressed naturally in the problem solver during the problem solving process. As a result, pertinent task-specific knowledge *emerges* from the interaction of the problem solver with the innate constraints of the task environment. Emergent intelligence thus avoids the problems associated with explicit knowledge by removing explicit knowledge.

The next section will briefly discuss few recent models that differ from conventional AI in their problem solving approach.

2.3.1 Evolutionary Algorithms

Evolutionary algorithms such as genetic algorithms (Holland, 1975; Goldberg, 1989), evolutionary programming (Fogel, Owens, Walsh, 1966; Fogel, 1992) and evolution-strategies (Rechenberg, 1973; Schwefel, 1981; Bäck, Hoffmeister and Schwefel, 1991) are population based search and optimization techniques inspired by natural evolution. These algorithms belong to a class of weak methods that use task-independent representations and operation. Being population based, they are capable of simultaneously searching a large space of potential solutions. Task-specific knowledge is acquired while solving a problem (Angelene, 1993). The important characteristics of evolutionary algorithms that enable task-independent way of problem solving are: firstly they model the task environment in terms of a 'fitness function' that maps an individual of a population into a real number which is then fed back to the problem solver. This minimal feedback provides a strong separation between task environment and the problem solver, avoiding reliance of the problem solver on explicit knowledge. Secondly the operators that manipulate the representational structures in evolutionary algorithms are representation-specific rather than task-specific enabling the evolutionary algorithms to be applicable to a wide range and type

of problems. Most importantly, the evolutionary algorithms employ an '*empirical credit assignment*' for local credit assignment by creating variations in the representational structure through the representation-specific operators. Over time the fitness of individuals in the population improves and the search becomes constrained towards the regions of individuals that have higher fitness in tackling the problem at hand. The empirical credit assignment allows the evolutionary algorithm to adapt its search dynamically in the problem space allowing the task-specific knowledge to emerge from the interaction of problem solver with the task environment.

2.3.2 Reactive Systems

Brooks (1991) has proposed a new approach to AI known as behavior-based AI to demonstrate that complex intelligent behavior can be easily produced by systems which have simple 'reactive' behaviors with regard to the environmental events. The key idea is that the world is its own best model and the representations are formed through interactions with the world. By introducing the notions of *situatedness* and *embodiment* Brooks has demonstrated how mobile robots can be made to generate robust behavior in uncertain and unpredictable environments. Situatedness means that the robots are situated in the physical world directly influencing the behavior of the system. Embodiment refers to the fact that the robot has a body and experiences the world directly. The result of their actions are fed back on their own sensory inputs. Intelligent behavior stems from the situation in the world, the signal transformation within the sensors, and the physical coupling of the robot with the world. The intelligence emerges through the system's interaction with the environment and also interactions among its own components.

2.3.3 Eclectic Hybrids

The conventional AI typically employed representations that were neat. That is, the solutions to a problem could be expressed in terms of data-structures that were easily interpretable and modifiable. This enabled easier modification of the representational structures and the methods were well suited in the context of Engineering-oriented AI applications. Cognition, on the other hand, is a result of a blend of representations. In its new perspective, AI combines various representations such as geneticism, connectionism, reactivism and hybridism, all capable of mimicking the processes in natural systems to

realize the notion of emergence. AI is thus characterised in terms of the '*representational eclecticism*' (Thornton, 1993) which is simply the idea that effective cognition may involve mixing and matching representational strategies in an opportunistic fashion. Representational eclecticism implies a rejection of the idea that cognitive representations will necessarily be 'neatly structured' and/or 'elegant'. Recent methods typically combine a number of paradigms that employ various representations to build complex AI systems that can adapt to unpredictable environments.

2.3.4 Automatic Programming

In recent years, automatic programming techniques have been developed with the goal of overcoming the software crisis. To make computers learn to solve problems without being told how to solve a problem through a set of instructions has been one of the main goals of Machine Learning. The recently developed Genetic Programming (GP) paradigm has been successful in achieving this goal. In the context of problem solving the approach recasts or reformulates a given problem as requiring the discovery of a computer program that produces some desired output when presented with particular inputs. That is, GP stresses the fact that many seemingly different problems can be reformulated as problems of program induction. Genetic programming provides a way to do program induction by searching the space of possible computer programs for an individual program that is highly fit in solving the problem at hand using the fitness information. The search process is domain-independent and employs the Darwinian selection mechanism. GP in essence uses genetic algorithms to search the space of computer programs. The representational structures that undergo adaptation in GP are hierarchically structured computer programs, typically expressed as LISP S-expressions built in terms of the function and terminal sets (that is the variables) of the problem domain. The size, shape and the contents of the programs can change dynamically during the process of problem solving. The hierarchy enables a hierarchical problem solving process similar to the top-down approach. Also, the computational effort and complexity could be reduced with automatically defined functions (ADFs) that enable reuse of code through modularity. Apart from solving a variety of interesting, non-trivial problems GP has been successful in evolving programs that generate complex behaviors. The question that naturally arises is whether GP is a viable tool for combining other paradigms to create hybrids that are more effective in solving complex problems. Research in

recent years has suggested that GP can successfully evolve neural network structures and weights (Zhang and Mühlenben, 1993; Poli, 1996). Recently GP has been applied to the evolution of connectionist learning rules for feed-forward networks (Radi and Poli, 1998). These rules are found to perform better in terms of speed and generalization. The work that is suggested in this thesis employs a different approach to evolve network and learning for self-organizing neural networks. Because of its general approach it might be possible to extend it to other types of neural networks. The approach emphasizes on exploiting the strengths of the representation and applying a few clever strategies. The role of GP in evolving flexible learning rules and their implications in the context of artificial intelligence need to be understood (Govinda Char, 1997b; 1997c).

2.3.5 Constructivist Systems

The reactive systems approach where a human is included in the design loop has its limitations in terms of the complexity of the design process and also in providing solutions to predefined tasks (Vaario, 1992). When the number of possible behaviors increases the complexity of the system increases exponentially. The predefined solutions will not suffice. Instead the system should itself find solutions through adaptation to a given environment. *Intelligence* in this context is viewed just as an *adaptive* behavior (Vaario, 1994). How to model intelligence as an adaptive behavior becomes the obvious issue. This assumption through which intelligence is viewed as an adaptive behavior adds a new dimension to AI and paves a way to a plethora of new approaches for building intelligent systems for uncertain and unpredictable environments. Also, an adaptive behavior can be constructed gradually from primitive components through the processes of development and evolution. The AI in this new context is known as ‘constructivist’ AI. The computational model is based on the Artificial Life approach (Langton, 1989; Langton, Taylor, Farmer and Rasmussen, 1992). The behavior of an individual (an agent) is based on neuron-like computational elements and emerges as a result of local interactions among similar elements and with the environment. The interesting aspect of the modelling method is in its approach to construct artificial neural networks that resemble biological morphogenesis and phylogenesis of nervous system. The networks are non-homogeneous in the sense that the neurons have different characteristic properties depending on the development process. The

computational model is production-rule-based and does not employ any traditional neural network learning algorithm to realize a complex behavior.

Conclusion

The reliance on explicit knowledge and the associated problems suggest that there is a need to identify new techniques that are task-independent and have the ability to acquire the task-specific knowledge during problem solving. AI, in its new perspective is based on the notion of emergence in some form (an organism or some form of learning or a complex behavior). It clearly rejects the idea of a predefined set of instructions capable of evolving systems that can exhibit intelligence similar to natural systems and possibly capable of out-performing these. Instead it proffers a wide range of techniques and models employing paradigms that mimic natural processes to create systems that are based on natural processes. In general, these systems are endowed with properties such as self-replication, adaptation and self-organization that are characteristics of Artificial Life.

The next chapter will focus on evolutionary computation.

Chapter 3

Evolutionary Computation

This chapter provides a short overview of the field of evolutionary computation. The genetic algorithm and the genetic programming paradigms will be discussed to a greater detail and compared in terms of the representations they use and their approach to problem solving.

3.1. An overview

The term ‘evolutionary computation’ (EC) refers to computation and problem solving with evolutionary algorithms that offer a number of advantages over traditional techniques. These advantages are multifold (Fogel, 1997) including the simplicity of the approach, its robust response to changing environments, its flexibility and its applicability to a wide range of problems in various domains. Evolutionary algorithms employing different representations with a variety of representation-specific operators and selection methods have been successfully applied to a wide range of difficult problems (where the variables typically interact in a nonlinear way). However it has been established that a particular algorithm or a representation would not hold across all problems. Some of the representations could be more effective as compared to others depending the type of problem that is being addressed.

The most important aspect of evolutionary approach to problem solving is their simplicity and expedience. Evolutionary algorithms typically encode solutions to a given problem through various representations which form the population for the search mechanism. These representations include fixed-length or variable-length strings, hierarchical trees, and others. The solutions evolve over time through effective manipulation of these representations via genetic operators that mimic the mechanisms of selection, crossover and variation. Being population based algorithms, these explore many different possibilities simultaneously (that is in parallel) during the search in the space of solutions. In addition, evolutionary methods due to their ability to perform credit assignment have the following advantages over the traditional techniques and the standard weak methods. Firstly, evolutionary

algorithms model the task environment as a fitness function that maps each individual in a population into a real number. The search mechanism sees only this number to guide the search. These algorithms employ ‘empirical credit assignment’ (Angeline, 1993) that works by creating novel structural variations in the population through probabilistic application of representation-specific operators and maintains the best solutions (local maxima/and or minima) as the search progresses. Secondly, the genetic operators are representation-specific rather than task-specific, allowing the task-specific knowledge to emerge from the interaction of the algorithms with the task environment.

Together, the representation, the genetic operators and the fitness function dictate the ultimate success of any evolutionary model. Based on different representations, types of genetic operators, and problem solving approaches, various evolutionary models have been developed. Some of these models are briefly discussed.

Evolution strategies (ESs) were first introduced by Rechenberg (1973) and was further developed (Schwefel, 1981; Bäck, Hoffmeister, and Schwefel, 1991). The emphasis in these techniques is on the set of behaviors of an individual in the population rather than on the acquisition of structures with high fitness. The search space is a space of potential behaviors. An individual is composed of a set of behaviors and the fitness function rates the behavior, each of which is a feature, of the individual, and the interaction between the features is unknown. ES systems use a fixed-length real-valued string as representation and employ both crossover and mutation operators (see, sections ahead) to manipulate the string. These systems have proved to be quite effective in solving parameter optimization problems.

Evolutionary programming (EP) was independently developed by Fogel, Owens and Walsh (1966). EP models reproductive relationship between a species behavior in successive generations. Although Fogel (1992) used a form of mutation as the reproductive operator, generally EP systems are not committed to any specific representation or operators. Fogel has given an excellent exposition on aspects of evolution that are important to model to achieve computational effects. EP remains an active area of research.

Genetic algorithms (GAs), one of the most popular and widely used search and optimization methods was developed by John Henry Holland of the University of Michigan, in the 1960s and 1970s. The idea behind genetic algorithms was to identify and model mechanisms of natural adaptation and apply these computational models for solving engineering problems. Genetic algorithms emphasise structure and its manipulation for modelling adaptation and evolution. These algorithms typically use a binary-valued, fixed-string representation and use crossover and mutation as reproduction operators. A mathematical basis was provided later by Holland for understanding the importance of genetic recombination to evolution and adaptation, through the *Schema Theorem* and *the Building Block Hypothesis* (Holland, 1975; Goldberg, 1989). This work led to his landmark book, *Adaptation in Natural and Artificial Systems*. Much work has been done on the theoretical foundations of genetic algorithms (see, e.g., Holland, 1975; Goldberg, 1989a; Rawlins, 1991; Whitley, 1993a; Whitley and Vose, 1995).

Genetic Programming (GP) has recently been introduced by Koza and Rice (1993). GP is basically a genetic algorithm for evolving computer programs that can solve problems. Genetic programming uses programs, in the form of recursive tree structures, as the basic representation. The genetic programs are subsets of LISP program tailored to particular domains and employ syntax preserving crossover for reproduction. This paradigm has proved to be highly successful in tackling many difficult problems (Kinnear, 1994). As with genetic algorithms, efforts have been made to explain genetic programming in terms of the *Schema Theorem* and *Building Block Hypothesis* (Angeline, 1993; Tackett, 1994; O'Reilly and Oppacher, 1994).

Other adaptive programming paradigms include Tierra (Ray, 1991) and FOIL (Quinlan, 1990). Ray pioneered a unique programming paradigm through the creation of Tierra system, a world which consists of assembly language programs that represents the organisms. Tierra breeds digital organisms which vie for memory and CPU time as metaphor for food and sunlight. It loosely emulates a shared memory MIMD computer with a 5-bit zero-operand instruction set. Each organism, that is an assembly language program has its own virtual CPU with registers, stack, program counter, and flags. The system is initialized with a single self-replicating 'ancestor' program residing in memory. This program copies itself into a

block of free memory and executes a special instruction which write-protects the new 'child' copy and allocates to it a virtual processor. The reproductive cycle then begins anew. As memory runs out organisms that are oldest or most defective are deleted to make room. Mutation helps to maintain diversity in the population as the generations progress. Organisms that finally survive adapt a variety of survival and competition mechanisms. These include code optimizations and biological properties such as parasitism and immunity.

FOIL system generates declarative code in the form of Prolog programs using nested loop that perform hill climbing. The outer loop generates the clauses, that is, lines of program code whereas the inner loop generates the literals for each clause. The program generation process is driven by a heuristic measure during the construction of a clause. The heuristic examines the mutual information gained due to the repartitioning of the training data through addition of a literal. If this addition does not change partitioning then it conveys no information whereas if it creates a more accurate partition then it increases mutual information. FOIL has been successfully tried on a number of machine learning problems.

Together, evolution strategies, evolutionary programming, genetic algorithms and genetic programming form the backbone of evolutionary computation. The potential advantages of evolutionary computation over the standard computational mechanisms are highlighted in the article by Forrest (1991). The following section describes genetic algorithms.

3.1.1. Genetic Algorithms

Genetic algorithms basically encode the solutions to a problem on a simple chromosome-like data structure which forms the individuals in the population. This population of chromosomes representing the genotypes is decoded into phenotypes that are evaluated for their fitness. Typically, fitness proportionate selection is used to select parents from this population. A genetic recombination (crossover) operator is applied to the selected pair of parents to produce offspring that form the next generation. An excellent introduction to genetic algorithms is given by Whitley (1990), Michalewicz (1992), Srinivas (1994), and Mitchell (1996).

An evolutionary cycle consisting of ‘evaluation-selection-recombination-creation’, running on a population (having a fixed sized N) of genotypes, is shown in figure 3.1. Parents are selected based on their fitness from the population of individuals. Genetic recombination (crossover) is applied to pairs to create offspring which will be inserted into the new population forming the next generation of individuals. The evolutionary cycle corresponds to a search through a space of potential solutions and repeats for a number of generations. Each genotype represents a point in the search space. Such a search requires balancing objectives that are conflicting: exploiting the best solutions and exploring promising regions of the search space. The flow diagram explains the process.

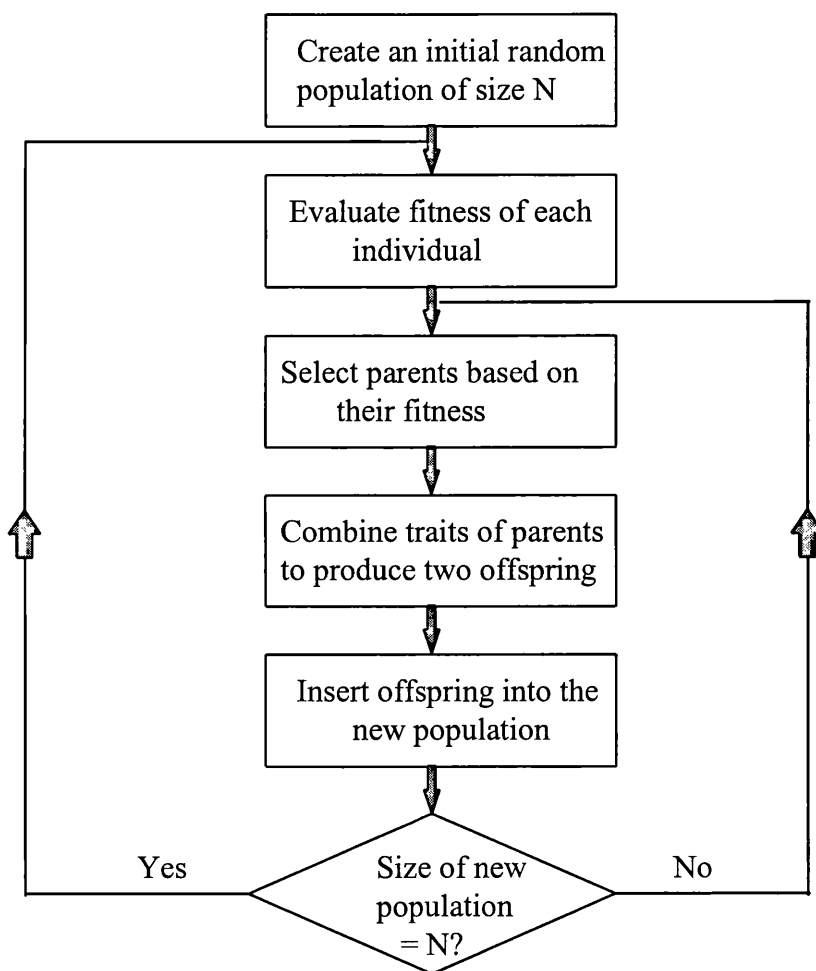


Figure 3.1: An evolutionary cycle.

The various stages of the evolutionary cycle are now explained in detail.

3.1.1.1. Population

The first step in implementing a genetic algorithm is to create a random population of genotypes. The size N of this population is fixed. In a *canonical genetic algorithm*, a genotype is typically a binary string of fixed length, though other representations abound. The size of the search space is related to the number of bits in the problem encoding. As an example, for bit string encoding with length l the search space is 2^l and forms a hypercube whose corners are sampled by the genetic algorithm. It is essential to maintain the diversity in the population, as this diversity is the driving force for the search mechanism. Some of the recent versions of genetic algorithms (Whitley and Starkweather, 1990) use a smaller distributed population in place of a single large population. A distributed search has been shown to improve the search mechanism and has provided better performance in terms of accuracy and consistency on a large range of problems including a set of *deceptive problems* (Baker and Grefenstette, 1989). A sample population of four genotypes is shown in figure 3.2.

Genotype label	Genotype
1	0 0 0 0 1 0 0 0
2	0 0 1 0 0 0 1 0
3	1 1 0 0 0 0 1 0
4	0 0 1 1 0 1 1 0

Figure 3.2: A population of genotypes

3.1.1.2. Fitness Evaluation

The genotype is decoded into a phenotype. The genotype in natural systems is the genetic blueprint, that is, strings of DNA. The genotype when decoded gives rise to the phenotype, that is, the individual with the characteristics (such as height) dictated by the genetic blueprint. The fitness of the individual is measured in terms of the ability and the strength of the individual to survive under a set of extremely diverse conditions and still compete for the goal. In engineering problems, depending on the nature of the optimization problem to be solved, the phenotype can represent any parameter and hence is totally problem dependent. Further, the phenotypic representation can be direct such as a real-valued parameter of a

function, control parameters for a process control application, strategies in a game, etc., or indirect such as a neural network architecture or a learning rule which are further evaluated for their fitness in solving particular tasks. As an example (from Mechalewicz, 1992), consider an optimization of a simple function of a real variable 'x', defined as:

$$f(x) = x \cdot \sin(10\pi \cdot x) + 1.0 \quad (3.1)$$

The genotype, in the form of a binary string, when decoded, yields the phenotype 'x' of the above function, which is further evaluated for its fitness in solving the given function. In this particular problem, the aim was to find 'x' from the range [-1..2] which maximizes the function $f(x)$. The fitness of the phenotype in maximizing $f(x)$, in turn, decides the chances of the particular genotype to reproduce and survive to compete in further generations. Genetic algorithms interchangeably use the notion of the evaluation function and the fitness function. These are very well explained in Whitley (1990). The fitness can be defined in terms of maximizing or minimizing a function. In the former case, the goal is to reach a higher value whereas in the latter case it is to reach the lowest possible value, as in the case of minimizing an error function. Figure 3.3 shows a sample population of four genotypes, for a problem which has an integer-valued optimization parameter. Each genotype is 8-bits long and is decoded into its phenotype. The fitness values for an arbitrary task are shown.

Genotype label	Genotype	Phenotype	Fitness
1	0 0 0 0 1 0 0 0	8	10
2	0 0 1 0 0 0 1 0	34	20
3	1 1 0 0 0 0 1 0	194	60
4	0 0 1 1 0 1 1 0	54	30

Figure 3.3: The genotypes are decoded to form integer phenotypes with given fitness values.

3.1.1.3. Selection

A variety of selection schemes can be used to select individuals from the given population. Some of the selection schemes are superior to others. A canonical genetic algorithm uses a fitness proportionate selection scheme. Figure 3.4 illustrates a roulette wheel selection scheme.

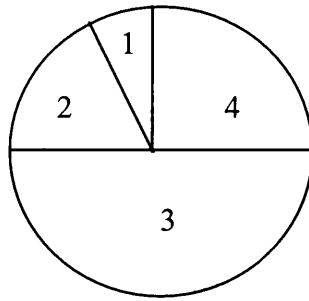


Figure 3.4: Mapping individuals onto the slots of a roulette wheel.

In this scheme, the population is viewed as a mapping on to a roulette wheel, where each individual is allocated a space in proportion to its fitness. Thus individuals with higher fitness are allocated wider slots and have a greater chance of being selected during the spin of the wheel. By repeatedly spinning the roulette wheel individuals in proportion to their fitness are chosen to form the new generation. The number of times an individual is expected to reproduce is given by: f/f_{av} , where f is the fitness of the individual and f_{av} is the average fitness of the individuals in the population. As an example, individual 3 will be selected for reproduction with a probability P_{ind} of:

$$P_{ind} = 60 / (10 + 20 + 60 + 30) = 60 / 120 = 50\% \quad (3.2)$$

Fitness can also be assigned based on a genotype's rank in the population (Baker, 1985), (Whitley, 1989) or by sampling methods such as *tournament* selection (Goldberg, 1990b). One of the popular methods used recently is the *k-tournament* selection originally introduced by (Wetzel, 1979). In this selection scheme k individuals are drawn from the population for replacement. The most fit individual among these are chosen as the 'winner' of the tournament and becomes a parent for the next generation. This process is repeated for the population size. The tournament selection method has been shown to out-perform the roulette wheel selection method by maintaining the diversity in the population. Also, tournament selection is more amenable when implementing parallel genetic algorithms (Mühlenbein, 1987) and (Tanese, 1989). Rank based selection schemes, along with distributed populations, have shown very promising results for a broad range of problems. Another selection scheme is the steady-state selection where only a few individuals are

replaced in each generation, usually least fit individuals being replaced by offspring resulting from crossover and mutation of fittest individuals.

3.1.1.4. Trait Inheritance and Recombination

After selection the parents are crossed over to form two offspring. As an example, consider two genotypes, representing the parents. The fragments between the two parents are swapped. The parents and the offspring are shown in figure 3.5 along with a 1-point crossover. The crossover operator randomly chooses a point on each of the parents and crosses over the parts of strings to produce the offspring. This operation, in effect, mimics sexual reproduction in natural systems. The offspring when decoded inherit the traits of the parents. The effect of crossover is to generate new sample points in the search space and thus maintain the diversity in the population. The strength of genetic algorithm lies in the crossover operator and how effective is this operator in exchanging structural information between the parents and also in exploiting problem-specific information.

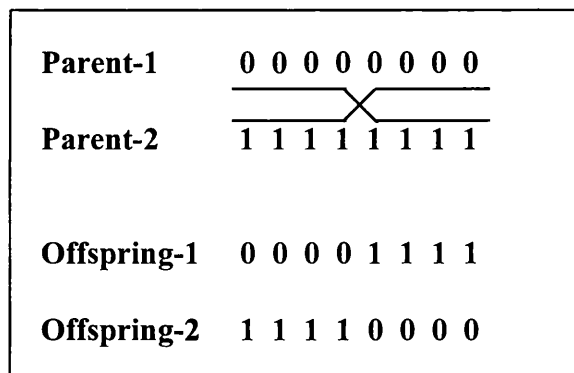


Figure 3.5: Crossing over the parents to produce offspring-1 and offspring-2.

The crossover operator is applied with a probability of typically 0.6. The effects of a variety of other types of crossover operators such as a 2-point crossover (Schulze-Kremer, 1992) and uniform crossover (Ackley, 1987; Syswarda, 1989) have been studied and seemed to perform very well in some problem domains providing further insights into the GA search mechanism.

3.1.1.5. Mutation

After recombination, the mutation operator, a unary operator, can be applied to the offspring. The mutation operator will flip some of the bits on a chromosome rarely. Mutation not only introduces new traits into the population but also can prevent the possible loss of diversity at given bit positions in a string.

Offspring	0	0	0	0	0	0	0	0
Offspring	0	0	0	0	1	0	0	0
(Mutated)					↕			

Figure 3.6: Mutating an offspring at locus 5.

Figure 3.6 illustrates an offspring mutated at locus 5. Unlike crossover, the probability of applying mutation to the population is very low, of the order of 1%. DeJong (1988), through extensive experimental studies, has proved that such a minimal mutation during the mating stage can avoid the search from getting trapped into ‘local optima’, that is, on to solutions that are only locally optimal rather than globally optimal.

3.1.1.6. The Schema Theorem and the Building Block Hypothesis

According to the *Schema Theorem*, genetic algorithms work by discovering and combining good *building blocks*, known as schemata (the substrings) that contribute a high fitness to the genotypes that contain these blocks. A schema is a string of the alphabet of genes with the wild cards (or ‘*don’t cares*’) at certain positions. As an example, the 10-bit string 11***1***1 is a schema. There are 2^{10} different instances of this schema. Thus a schema represents a hyperplane or subset of the search space. A schema is characterised by an *order* and a *defining length*. The *order* is the number of bits that are defined in a schema. The order defines the unique characteristics of a schema. The *defining length* is the distance between the first and the last string positions and defines the compactness of information contained in a schema. The above schema has an order of 4 and a defining length of 9. The schema has a fitness at the time of evaluation which is defined as the average fitness of all strings in the population matched by the particular schema. A schema with an ‘above average’ fitness survives and propagates in larger numbers to the next generation in comparison with those

that have a fitness ‘below average’. This is decided during the process of selection. The genetic algorithm seeks optimal performance through short, low order and high fitness schemas, the hypothesis known as the *Building Block Hypothesis*. Short low-order schemas are less susceptible to crossover disruptions and more likely to maintain and transmit the valuable information in the population. Forrest (1993) explains the notion underlying this hypothesis. The genetic algorithm initially detects biases toward higher fitness in some low-order schema and converges on this part of the search space. Over time, it detects biases in higher order schemas by combining information from low-order schemas by means of crossover and eventually converges on a small region of the search space that has high fitness. In the context of search, Holland has shown that for a population of the size N the number of schemas or the traits that are simultaneously searched for is around N^3 and has referred to this property as *implicit parallelism*. Although the genetic algorithm would seem to be explicitly evaluating the number of strings or chromosomes in the population, it is actually estimating the average fitness of a much larger number of schemas. Consequently, Holland argues that a genetic algorithm assigns credit not to the strings in the population but to schemata of the population. These concepts are well explained in Whitley (1990), Rawlins (1991), Michalewicz (1992), and Mitchell (1996). Crossover and mutation can also destroy and create instances of schemas as explained below:

- The effects of crossover on schema: Consider two schemas:

A = (**11****) and

B = (11*****1)

Assume a population that contains these schemas. The crossover places the two portions of each of the schemas in different offspring. By choosing a crossover site at locus ‘6’, schema ‘A’ survives the crossover and propagates to the next generation whereas the schema ‘B’ does not survive the crossover. Thus short, low order schemas are less likely to be disrupted by the crossover operator.

- The effects of mutation on schema:

Consider the schema $A = (**11****)$. The mutation operator flips 0s to 1s and vice-versa. A flipping at locus 4 or at locus 5 can destroy the schema. A mutation can also recreate the lost schema, for the same reasons.

3.1.1.7. Deception and Royal Road

Epistasis is a term that refers to non-linear interactions of genes. Recent work has shown that the schema theorem does not apply to problems with epistasis, suggesting that the structure of some problems can mislead the genetic algorithm. These problems are called *deceptive*. Goldberg (1989c) has introduced this concept in terms of the notion of *hardness*. To study deception, synthetic problems were constructed which assign specific fitness values to specific bit patterns to exercise precise control over the problem structure. The idea was to investigate the formation of a high fitness string from building blocks with a low average fitness. Such *Royal Road* functions are the opposite of deceptive functions. They are constructed in such a way as to be easy for the genetic algorithm to solve and compare the best performance of the genetic algorithm with the theoretical predictions. Forrest and Mitchell (1993), while analysing such problems, noticed that highly fit building blocks get attached, by coincidence, to adjacent unfit building blocks which propagate throughout the population, a property known as *hitch-hiking*. The effect was that the genetic algorithm failed to converge as expected in probability to theoretical predictions. This problem was overcome by the insertion of *introns* in the genetic algorithm (Forrest and Mitchell, 1993a).

3.1.1.8. Hybrid Algorithms

Hybridizing genetic algorithms with the other optimization techniques has yielded better results in many optimization problems. These hybrid algorithms can be computationally more expensive. Such an approach combines local hill-climbing with global hyperplane sampling. Davis (1991) and Mühlenbein (1991) have studied hybrid algorithms and have shown that these can outperform the standard genetic algorithm.

3.1.1.9. Parallelism in the Genetic Algorithm

In natural systems millions of individuals exist and work in parallel. In principle, such a parallelism can exist in any population based computational systems. Different models have been tried to exploit parallelism in different ways. Further, with a proper selection scheme

such as a tournament selection scheme, the evaluation and crossover operations can be shown to occur in parallel. Whitley (1993b), Goldberg and Deb (1991), Tanese (1989) have described the various parallel models. The parallel genetic algorithms combine the hardware speed of parallel processors and the software speed of intelligent parallel searching and have been successfully applied for function optimization and combinatorial optimization problems (Mühlenbein, 1992).

3.1.2. Genetic Programming

Automatic programming, that is, computer programs automatically writing computer programs, has been an active area of research in the field of artificial intelligence. Evolutionary computation techniques have been tried with limited success to automate program induction. Evolutionary programming was applied (Fogel, Owens, Walsh, 1966) to evolve computer programs in the form of finite-state machines. Cramer (1985), Fujiki and Dickinson (1987) succeeded in evolving computer programs with genetic algorithms. Recently, Koza and Rice (1993, 1994) have successfully applied genetic algorithms for program induction, that is, for breeding computer programs for solving problems. In this context, genetic programming is also referred to as a genetic algorithm for program discovery. The notion that any task can be recast or reformulated as the problem of requiring the discovery of a computer program that can solve the given task led to the genetic programming paradigm. Furthermore, computer programs being universal can be applied to solve problems in any domain. Thus the GP paradigm is applicable to a wide variety of problems in diverse areas. The following sections give a background of genetic programming.

3.1.2.1. Population

Genetic programming employs almost the same evolutionary cycle as in Figure 2.1 for evolving computer programs. The individuals (the genotypes) in the population, that is, the genetic programs are composed of a set of domain-specific functions and terminals known as ‘primitives’ that are effective in solving the problem. The programs are represented as trees that are recursively composed of all possible combinations of these primitives. Koza evolves LISP programs, that is, LISP S-expressions, that can be expressed as ‘parse trees’ (Pagan, 1981). Lisp is chosen for its simplicity and convenience as all operations in this programming language can be implemented as function calls. These expressions while ensuring

syntactically correct programs, can be evaluated on-the-fly. A simple Lisp expression (for the equation $E = mc^2$, where m is the mass of a particle and c is the velocity of light) and its parse tree, with one function (*) and two terminals (c and m), is shown below.

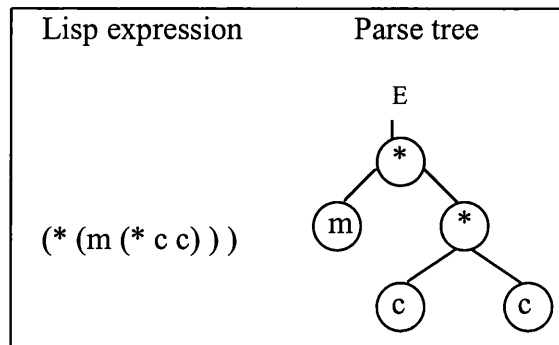


Figure 3.7: The Lisp expression $(* (m (* c c)))$ and its Parse tree.

In LISP, the operators (that is, the functions) precede their arguments. The arguments can themselves be functions again, calling other functions depthwise, recursively. The phenotype, depending on the context, is the behavior, or semantics, of the computer program. The concept of program induction is explained through an example. Consider the ‘symbolic regression’ problem, where the aim is to evolve a program that represents an equation for fitting a curve for a given set of data points. With genetic programming, the first step is to define the primitives, that is, the set of appropriate functions and terminals for the problem domain. The function set and the terminal sets for this problem are defined as:

$$F[s] = \{+, -, *, \%\}; T[s] = \{X\};$$

where the functions in the function set $F[s]$ are the arithmetic operators, each taking two arguments. The division operator is a protected division operator (Koza, 1993). The terminal is a global variable that can be assigned the data values. The role of designer expertise is crucial, as a propitious choice of the primitives and the test suite greatly influence the performance of GP (see, Kinnear, 1994; O’Reilly and Oppacher, 1995). Once the primitives are defined, the next step is to create a population of trees with the function set as the internal nodes and the terminal set as the leaf nodes. Depending on the problem domain, various problem-specific functions can also be defined (Koza 1993, 1994). Angeline (1993) refers to these as “different languages” for solving different set of problems. The primitives can also include a variety of functions for iteration and recursion. A sample program

composed with the above primitives, is shown in figure 3.8. This program represents an individual in the population.

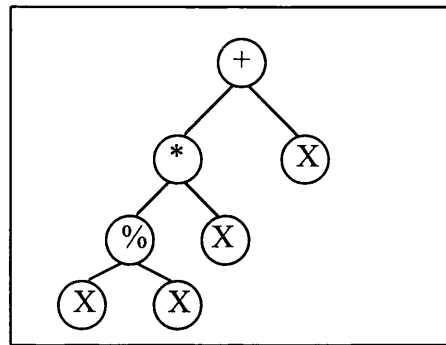


Figure 3.8: A sample genetic program

Because the trees are recursive, their depths and the complexity vary. This makes the GP paradigm very interesting as the complexity of the solutions increase over time instead of remaining constant as in the case of the genetic algorithm. The search space is a hyperspace consisting of all possible compositions of functions that can be recursively composed of the set of functions and terminals. As with the genetic algorithms, the diversity and the size of the population are important factors that contribute to successful program induction. Genetic programming typically uses a steady state model (Reynolds, 1992) instead of a generational model, maximizing the diversity in the population and also minimizing memory resource.

3.1.2.2 Fitness Evaluation

In the course of the search for a correct program, many such candidate programs will have to be executed in a simulated environment and assessed for their fitnesses. If the simulated environment happens to be the representative environment, it should enable the programs to work correctly for unseen data as well, that is, to learn to generalize from the simulated environment. The fitness is the only information that the algorithm has to search for potential solutions and is exceptionally important for successful program discovery. The evolutionary mechanism, through its individuals, will ruthlessly exploit the fitness function. Any bugs in the fitness function can be recognized only by examining the individuals as they evolve. These concepts are discussed in detail in Koza (1993, 94) and in Kinnear (1994). For the symbolic regression problem, each program is run on a number of fitness cases (a set of inputs for which the correct output is known). The program is assessed for its fitness depending on how well it performs on each of the fitness cases it encounters.

3.1.2.3 Selection

Genetic programming typically uses a tournament selection scheme for selecting the parents for reproduction. The selection is fitness proportionate as the programs evolve according to their fitness in solving the given problem and are generally evaluated directly. Other selection methods stated earlier in the GA context can also be employed.

3.1.2.4. Trait Inheritance and Recombination

The genotypes, that is the genetic programs are manipulated by crossover to produce offspring that inherit the traits from the parent programs points as shown in figure 3.9. The programs are rooted, point-labelled trees with ordered branches. Genetic programming employs syntax preserving crossover to retain the validity of the programs. The crossover operator is 'blind' in the sense that it uses a probabilistic bias to choose the crossover point. Typically, an internal node (with a probability of 90%) on each of the parent programs is selected at random. The subtrees of the two programs are swapped over at the crossover point resulting in two offspring. Also, the crossover operator does not contain the problem specific knowledge thus reflecting the power of the GP in evolving potential solutions. A variety of crossover operators such as 'hoist' (Kinnear, 1994) and 'greedy recombination operator' (Tackett, 1994) have been shown to improve GP performance. Altenberg (1994) has discussed self-crossover and modular crossover operators in an attempt to generate structural regularity in the programs. O'Reilly (1995) has described a variety of crossover operators and their effects on the performance of GP.

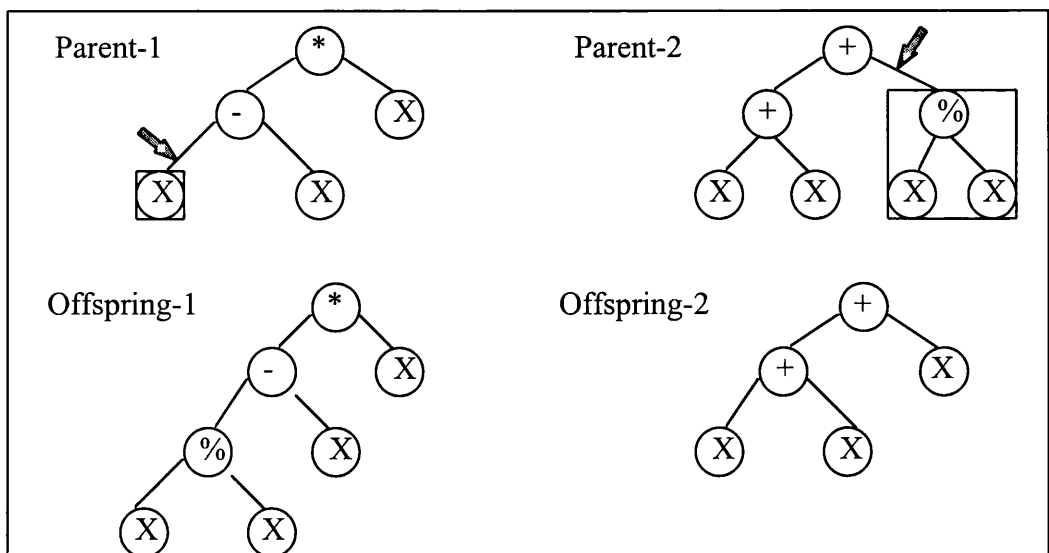


Figure 3.9: Crossing over the parents to produce two offspring.

3.1.2.5. Mutation

Mutation in genetic programming involves introducing a new subtree in an existing individual program. This is done by removing a subtree of the program and replacing the same with a newly created random tree. Koza (1993), through empirical demonstration, argues that the role of mutation is insignificant in genetic programming. By choosing a very large population, Koza stresses that the recombination operator is sufficient in maintaining the diversity in the population. Tackett (1994) explains this in the context of the genetic algorithm. Considering a fixed location on a fixed-length genotype, if this location contained the same value for all the individuals in the population, then mutation is the only way to change the value. In the case of GP, there is no concept of a 'fixed locus' in the genotype as programs of new sizes, shapes and complexity are generated continuously. The only way of losing the values would be if a member of the function set or the terminal set were to disappear completely from the population. This is extremely unlikely with a large population, where there will be hundreds of each type of nodes. Hence the canonical GP does not use the mutation operator. Currently a variety of mutation operators are being investigated along with the recombination operator (O'Reilly, 1995).

The canonical GP employs only the crossover operator to introduce diversity in the population. Most of the individuals in the initial population will normally have poor fitness values. Over time, through selection and crossover, the average fitness of the population will increase and the GP is likely to converge to a solution with the maximum fitness. The performance of GP should be assessed over a number of runs to see any changes in parameters.

The non-canonical GP includes innovative ideas in devising new operators, finding new applications and strategies and extended research in the canonical GP.

3.1.2.6 The search mechanism

Genetic programming being a genetic algorithm for program discovery, the search is for a program in a large space of computer programs. The search mechanism is again a directed search with fitness function (acting as a heuristics) guiding the search. In contrast, hill-climbing is a memoryless, local search method where the successors of current state are

generated and evaluated according to the heuristics (a function). If the successor state having the best heuristic value (closer to the goal) is better than the current state, that successor state is chosen to become the new current state, and the process is reapplied. Otherwise the process terminates. Thus the algorithm converges to the top of the nearest hill in the fitness landscape (for maximization) or seeks the bottom of the closest valley (minimization). There is no guarantee that the local extremum is global extremum. Simulated annealing (Kirkpatrick, Gelatt and Vecchi, 1983) is an extension of hill-climbing. The idea of simulated annealing comes from physics where a temperature is initialized to a predetermined starting value and reduced gradually according to a cooling schedule, to zero eventually. Inferior moves are given chances of being accepted. When the temperature is zero the search behaves in the same way as that of the hill-climbing, the difference being that a neighbour is accepted under a probability which is related to the temperature. If it is rejected then another unexamined neighbour is tried. This strategy increases the probability of locating the highest point in the search space. The effectiveness of simulated annealing depends on the definition of the domain-dependent neighbourhood function. The cooling schedule plays an important role in convergence. An alternative method is to store all states which have been heuristically evaluated but not expanded in a priority queue. These states are ordered according to their heuristic value, with the best being first, and the resulting algorithm is called 'best-first-search'. Although the algorithm is guaranteed to find the globally optimum value, the priority queue can grow exponentially with the depth of the search performed. Beam search (Lowerre and Reddy, 1980) is very much like best-first-search with the exception that the priority queue (memory) is set at some size limit. Thus there is a limit to which states the search can be backed up to. Tackett (1994) argues that there is a strong correlation between 'beam search' and genetic programming with a fixed population serving as memory and fitness to stochastically assign priority. In this context, Tackett explains that the states of the genetic 'search tree' are the expression. Rather than being the successors of a single initial state, the initial population of N expressions are N randomly generated states each of which is visited by being created and evaluated, as an expression for fitness. A cloning operator is employed which maps a state of the search tree into itself. This is important as it enables the search to 'remember' a state from generation to generation allowing search to back up to that state. The selection operator along with the recombination, cloning and possibly the mutation operator creates the successors of expressions in batches of N new states called a generation

replacing the previous generation gap (De Jong 1975;1993). The population of N expressions is analogous to a beam search priority queue with limited size N, ordered by fitness (Tackett, 1994).

3.1.2.7 The Schema Theorem and the Building Block Hypothesis: the GP analogy

Genetic programming evolves programs in the form of LISP S-expressions. These expressions are represented hierarchically as their parse tree. Does this hierarchy in the structure of a program (that is, the solution) represent the hierarchy in problem solving as well? Is there a hierarchical process in GP? If so, does the search mechanism exploit this hierarchy while searching the space of potential candidate programs? O'Reilly (1995) addressed these issues by first providing a clear distinction between a hierarchical process and hierarchical solutions and argues that GP may be proceeding in a manner of a hierarchical process for a number of reasons. O'Reilly defines the hierarchical process as a process that identifies and promotes useful primary elements, combines them into composite, modular, reusable, and successfully higher-level components of a hierarchy and guides high-level component assembly into a hierarchical solution. This approach is analogous to an efficient bottom-up design method. A hierarchical solution has a combination of hierarchical structures and control. The hierarchical control is the execution of the task through the accomplishment of a series of subtasks. The subtasks can themselves be recursively subdivided. This approach in essence reflects a top-down strategy. In programs, hierarchical structure is the existence of nested levels of procedures and functions. O'Reilly suggests the following reasons to conjecture that GP might be proceeding in a hierarchical way.

- Because GP's solutions are hierarchical, the process that produced these solutions may also be hierarchical.
- A hierarchical process, by introducing efficiency in the search mechanism, may enable finding solutions. It is easier to correctly complete a simple subtask than a complex task. Also, the same subtask if needed, can be reused.
- GP crossover depends on a hierarchical representation of programs. If swapping over of subtrees can be assumed as swapping subtasks, the crossover mechanism may be responsible for the exploration and combination of sub-control.

- GP may be employing the human design approach which in turn requires a hierarchical process.
- Hierarchical processes are ubiquitous in evolution. GP being a simplified model of evolution, is a hierarchical process.
- Because GP is a specialized GA, one can speculate that a building block behavior may occur in GP as well. If so, it may be possible to develop a schema theorem and a building block hypothesis for GP.

It is argued that GP in its canonical form does not exploit a hierarchical process to obtain hierarchical solutions. The solution may be hierarchical because the primitive chosen for the problem may implicitly encode the knowledge about the task decomposition and execution. It is questioned whether GP, on its own, is able to evolve high-level primitives by successful combination of low-level primitives and proceed to combine these high-level primitives to obtain a hierarchical solution. Defining these primitives as ‘general purpose’ primitives and through empirical demonstration O’Reilly proves that GP on its own lacks the power of a hierarchical process. By developing a Scheme theorem for GP (GPST) and Building Block Hypothesis (BBH) she concludes that GP may not be conducting its search exploiting a true hierarchical process. Earlier Koza (1993) attempted building new primitives by extracting and encapsulating a portion of the program and defined these as ‘define-building-blocks’ which are given a name on-the-fly. When a program containing this module is evaluated, the definition of the module in the module library is used. Angeline (1993) creates such primitives through the ‘compression’ operator and extends his system as ‘GLIB’, for Genetic Library Builder. These operators dynamically modify the representations during the run. Tackett (1994), while applying GP to a complex real-world problem on automatic target recognition, has demonstrated the evolution of successful building blocks that duplicate in the population at higher frequencies. Koza (1994) has introduced an extension to the Canonical GP and defines these representations as ‘Automatically Defined Functions’ (ADFs). These functions coevolve dynamically during a run enabling the GP to solve complex problems efficiently. That is, the primitives for the ADFs are defined initially by the designer but the ADFs evolve in terms of these primitives during the run. This approach is similar to the hierarchical decomposition of task into subtasks, though ADFs explicitly do not control a

hierarchical process. Unlike the ‘compression’ and ‘define-building-block’ operators, ADFs maintain the representation structure static.

3.1.2.8 Deception and Royal Road: the GP analogy

By formulating a class of constructional problems, Tackett (1994) tries to create simple GP analogies to ‘Royal Road’ and ‘Deception’ problems common to the studies of classical genetic algorithm. In these constructional problems, the fitness is based on the syntactic form of the expression rather than semantic evaluation. The reason is to allow a precise control over the fitness structure in the space of expressions. A particular target expression is assigned a ‘perfect’ fitness while the sub-expression resulting from the hierarchical decomposition will have intermediate fitness. If the intermediate fitness values increase monotonically with the complexity of the sub-expression, such problems are defined as ‘Royal Road’. Alternatively, if some intermediate expressions have lower fitness than the sub-expressions they contain, they are defined as ‘Deceptive’. Thus, the credit for partial solutions is precisely controlled to control the problem complexity. Through a special recombination operator, and different selection schemes, Tackett has empirically demonstrated the effects of these on building blocks and search.

3.1.2.9 Hybrid Algorithms

O’Reilly (1995) has compared GP to alternative algorithms by solving exactly the same class of problems. Stochastic Iterated Hill Climbing (SIHC) and Simulated Annealing (SA) were found to out-perform GP in some cases, suggesting that synthesising a localised search strategy into GP will complement its global, population-based search and improve it.

3.1.2.10 Parallelism in Genetic Programming

As with the genetic algorithms, there have been efforts to parallelize GP (Koza, 1993; Poli, 1996). The advantages sought are in terms of massively parallel evaluation as the evaluation function can be distributed over a number of processors and also in linear speed up. Poli, in a recent work describes a new form of genetic programming which is suitable for the development of fine-grained parallel programs. Known as PDGP (Parallel Distributed Genetic Programming) that is based on graph-like representation for parallel programs which is manipulated by crossover and mutation operators which guarantee syntactic correctness of

the offspring. The advantage is that PDGP can be seen as a paradigm to optimize acyclic graphs which need not be interpreted as programs but as designs, semantic nets, neural network topologies and so on.

(Some of the references are available at the bibliography section. See Tackett (1994), Angelene (1993), Whitley (1993) and Mitchell (1996) for the rest of the references).

Conclusion

Evolutionary computation (EC) through its population based approach offers a different method of problem solving with a number of advantages as compared to conventional techniques. Whether evolutionary algorithms can be effectively hybridized with other paradigms such as connectionist networks and the role of such models in the domain of AI need to be investigated. The next chapter discusses few possible approaches.

Chapter 4

Evolution of Structure and Learning

Evolutionary algorithms in recent years have been shown to be quite successful as learning systems on their own and also as meta-learning systems for other paradigms such as connectionist networks. A novel approach is proposed to demonstrate how the genetic programming paradigm can naturally be combined with connectionist networks to synthesize potential connectionist learning *while* interacting with a given environment. The assumptions, the justifications and the approach are discussed.

4.1 Introduction

As the ability to learn is entwined with intelligent behavior, learning is desirable for both natural and artificial systems. Artificial systems typically aim at forms of learning that resemble human learning through a variety of computational models some of which are inspired by nature. These include models of connectionist networks that mimic, in some respects, the information processing mechanisms in natural systems through the implementation of brain-like structures, associated learning algorithms and evolutionary methods that work on Darwinian principles. The success in learning to solve a given problem depends primarily on two factors: firstly, on the type and the complexity of the problem itself and secondly, on the efficiency of the learning mechanisms in tackling the given level of complexity. Certain types of problem are easily amenable to conventional algorithms whereas some preclude an algorithmic approach for a solution. Typically, the process of problem-solving is considered as a search in the solution space. This space can be small, easily understood and interpretable. Alternatively it can be very large, poorly understood and highly complex. Evolutionary algorithms such as genetic algorithms, being population based search methods, are capable of simultaneously searching large, complex spaces and have been successfully applied to machine learning problems (Michalski, 1986; De Jong, 1988). The reasons for opting to use evolutionary paradigms as learning systems are due to their attractive properties such as the implicit and explicit parallelisms, robustness as well as the expedience (Goldberg, 1988). Also, the processes of natural evolution and natural

genetics are well known for centuries. In contrast, the fundamental mechanisms in the brain are still unknown.

The learning at an individual level (phenotypic learning) is vital to any system, whether natural or artificial, and the learning mechanisms at subsymbolic levels provide an extremely rich landscape that needs to be explored and exploited for complex machine learning tasks. Connectionist networks offer an approach to learning at an individual level and have proven to be good at simulating different features of human-like learning, memory, detection of analogies or handling of similarities (Heistermann, 1990). They can learn to perform tasks for which computational algorithms may not exist (Turing, 1950) and are capable of learning from examples. The incorporation of connectionist networks and other machine learning paradigms offer flexibility to conventional AI systems in terms of knowledge acquisition and processing as the knowledge can be acquired during the process of problem-solving. However, the space of possible network topologies and network learning algorithms is extremely large and there are no standard design methodologies to implement the optimum network or the best learning algorithm for a given problem. Evolutionary algorithms due to their ability for parallel search in complex spaces are good candidates for neural network design (Branke, 1990; Yao, 1990; Schiffmann, Joost and Werner, 1992; Kuscu and Thornton, 1994; Balakrishnan and Honavar, 1995). Genetic Algorithms (GAs), Genetic Programming (GP), Evolutionary Strategies (ESs) and Evolutionary Programming (EP) have been shown to be quite successful in evolving optimum network architectures and also the network weights (Montana and Davies, 1989; Harp, Samad and Guha, 1989; Mühlenbein, 1990; Whitley and Bogart, 1990; Belew, McInerney and Schraudolph, 1990; Fogel, 1992; Degaris, 1993; Angelene, 1993; Zhang, 1995 and others). However, their role as meta-learning system for connectionist networks would be extremely interesting and is worth investigating. Two major approaches, namely, the neuro-evolution and the genetic-connectionism have emerged in this particular direction of research in recent years. Neuro-evolution (Whitley and Bogart, 1990; Belew, McInerney and Schraudolph, 1990; Fullmer and Miikkulainen, 1991; Torreele, 1991; Harvey, 1993; Moriarty and Miikkulainen, 1996) employ genetic algorithms for evolving and training neural networks. The chromosomes encoding neural network parameters such as connection weights, thresholds and connectivity are recombined based on principles of natural selection.

The selection process is guided by certain fitness measure for the problem in hand. The result is the evolution of network(s) capable of solving a given problem. Genetic algorithm, by evolving appropriate network weights replaces the standard network learning methods. Neuro-evolution is extendible to genetic programming as well. Genetic-connectionism (Chalmers, 1990; Baxter, 1992; Dasdan and Oflazar, 1993; Radi and Poli, 1997; Govinda Char, 1997, 1997a) uses evolutionary algorithms (such as genetic algorithms/ genetic programming) to search the space of network learning algorithms themselves. Further the methods offer ways of implementing useful network topologies.

The need for discovery of potential connectionist learning algorithms, new architectures, compatible grammars and encoding techniques cannot be over-emphasised as these will not only enable us to understand the neuromorphic systems to greater depths but also further the progress in artificial intelligence through their use in various domains. Evolutionary paradigms as meta-learning systems might prove to be potential tools for this endeavour.

The sections ahead will discuss some of the recent work in neuro-evolution and genetic-connectionism after providing a brief introduction to connectionist networks.

4.2 Connectionist networks

Connectionist networks, invariably known as artificial neural networks (ANNs), offer a radically different approach to computation through a network of processing elements that are often presented as a simplified version of the human neuron in the brain. These networks, inspired by the structure of the brain, are massively parallel systems that rely on dense arrangements of interconnections of these surprisingly simple processing elements. Parallelism, a distributed representation and distributed control are the key features of these networks. The network models have been used to address problems that are intractable and cumbersome with traditional methods (Dayhoff, 1990). The models being rule-implicit have the greatest potential in a number of areas such as speech and image recognition and in natural language understanding. The underlying processes require high computational rates and the current systems are far from equalling human performance. Also, they offer a framework that provides insight into how biological neural

processing may work. They are unique in their ability to adapt to changing environments and to operate on distributed fault-tolerant hardware. The networks typically consist of:

- A directed graph with a number of nodes (processing units) and a number of links connecting the nodes in different ways providing a variety of network topologies.
- A state variable associated with each node.
- A real-valued bias associated with each node.
- A real-valued weight associated with each link.
- A transfer function or node activation function for each node determining the state of the node as a function of its bias, the weights of the incoming links and the input variables associated with the input links.

The nodes sum their inputs via a set of synaptic weights (sum of the product of input variable and the associated weights) and pass on the resultant via the node activation function to yield an output. The networks are generally characterised by the architecture (that is, the number of neurones, the way they are arranged and connected via the synaptic weights) and the learning algorithms (that is, the learning rules that modify the weights) based on particular topologies.

4.3 The role of evolutionary algorithms in connectionism

Designing neural networks is a complicated task as it involves many variables, discrete and continuous, interacting in a complex manner and there are no heuristics to guide the design phase. Recent methods in which the networks can learn to configure themselves have gained prominence (Honavar, 1988; Ash, 1989; Fallman, 1990; Hall, 1990; Hirose, 1991; Smotroff, 1991; Sanger, 1991; Romaniuk, 1992). Two general approaches were identified: the destructive and the constructive methods. The destructive methods for network design start with a larger network and then prune off the excessive nodes and connections (nodes that are not actively used) to arrive at the optimal network size. The method is computationally expensive (Seitma and Dow, 1988). The constructive methods (Ash, 1989) start off with a small number of nodes and add nodes until the required performance is achieved. The method also has limitations in terms of computational time. These suggest a need for techniques that can automatically generate the optimal network

architecture (and optimal learning rules) in a short time and allow testing on a number of possible solutions. The search space of the possible connectionist network architectures is vast, deceptive and multimodal. Deceptive means that similar network architectures can have different performance. It is also possible that different network architectures can exhibit similar performance making the search space multimodal. The enumerative and random methods are highly inefficient in exploring such complex spaces. Also, the space of possible learning algorithms is extremely large to be explored by standard methods. Evolutionary algorithms have been employed to automate the design process mainly for two reasons. Firstly, due to their capability for parallel search in large, complex spaces and secondly, with the hope that the evolutionary approach might yield networks and learning mechanisms that are more flexible. Genetic methods definitely provide a robust and faster search procedure and are found to be excellent tools to automate the design process. Further these methods are amenable to parallel processing.

The major issue in genetic-based design of artificial neural networks is that of the encoding strategy or the mapping scheme. How should one encode the neural network architecture or the learning mechanism effectively in the genotype in order to achieve an optimal solution? In the context of network induction (the architecture) the encoding strategy should enable one to capture potentially useful designs for the given task and also provide the capability for generalization. In the context of learning it should enable evolution of efficient learning rules for a given task environment. Moreover, the evolutionary algorithms employ different genotype representations. Certain representations might help effective encoding strategies when compared to others.

A few encoding schemes with genetic algorithm and genetic programming in contexts of neuro-evolution and genetic-connectionism will be discussed in this chapter.

4.3.1 The Genetic Algorithm approach

Over the years genetic algorithms have been applied to connectionist networks in several ways:

- Given the architecture (that is, the number of layers, the number of nodes in each layer and the connectivity pattern) genetic algorithms have been used to determine the connection

weights (Whitley and Hanson, 1989; Montana and Davies, 1989; Mühlenbein 1989; Heistermann, 1989; 1990; Wilson, 1990; Whitely, Starkweather and Bogart, 1990; Belew, McInerney and Schraudolph, 1991; Karunanithi, Das and Whitley, 1992). Genetic learning is compared with other standard learning algorithms such as the back-propagation (BP) (Rumelhart, Hinton and Williams, 1986) that are susceptible to the problems of local minima (Mühlenbein, 1989).

- Given a standard learning rule for training the network, genetic algorithms have been successful in finding the architecture of the network (Harp, Samad and Guha, 1989; Miller, Todd and Hegde, 1989; Whitley and Bogart, 1990; Mühlenbein, 1990; Bornholdt and Graudenz, 1992; Romaniuk, 1993; Jacob and Rehder, 1993; Jones, 1993a; 1993b).
- Given both the architecture and the learning rule of the network, optimal parameters for the learning rule are found with the genetic algorithms (Belew, McInerney and Schraudolph, 1991).
- Given a standard learning algorithm, optimal training sets have been evolved with the genetic algorithm during structure evolution (Romaniuk, 1993).
- Given the architecture of the network, genetic algorithms are employed to find the fittest learning rule based on certain fitness measures (Chalmers, 1990; Dasdan and Oflazar, 1994).

4.3.1.1 Network induction (Neuro-Evolution)

Optimization of neural network architectures or finding a minimal network for a particular application is important as the complexity of a network will dictate the speed and accuracy of the learning and its overall performance. Generally the size of the network should be as small as possible but sufficiently large to ensure the sufficient fitting of the training set along with a capability for generalization.

With genetic methods the encoding mechanism that encodes the neural network (the *phenotype*) into a string (the *genotype*) is crucial. The way in which the coding should be realized is not straightforward (Nolfi and Parisi, 1994). In most models the representations of the genotype and phenotypic forms coincide. That is, the inherited genotype directly and literally describes the phenotypic neural network. These direct encoding methods, also known as strong specification schemes are good at capturing the connectivity patterns within

small networks very precisely facilitating rapid evolution of finely optimized, compact architectures (Miller, Todd and Hegde, 1989). However, the scheme has led to the problem of scalability as the number of bits of information to encode a neural network increases exponentially with the number of neurons. For larger networks the direct encoding scheme increases the search space exponentially for the evolutionary process to be effective (Kitano, 1990). Also, the direct genotype-phenotype mapping scheme is biologically implausible. In biological mapping the phenotypic behavior emerges as a result of non-linear interactions among the genes. Indirect mapping schemes, typically encode a set of instructions in the genotype for network development. The network architecture can be specified by growth rules (Mjølness and Sharp, 1987), by sentences of a formal language (Mühlenbein and Kindermann, 1989) or by a graph generation grammar (Kitano, 1990) and grammar based encoding, such as cellular encoding (Gruau, 1993). The latter mapping schemes have yielded better network architectures with shorter and compact genotypes overcoming the problem of scalability. The evolved neural networks have been shown to outperform networks with fixed architectures (Schiffmann, Joost and Werner, 1992; Kitano, 1990; Whitley, Starkweather and Bogart, 1990; Wong, 1994; Maniezzo, 1994; Nolfi and Parisi, 1994).

A number of novel neuro-evolution techniques have emerged recently. These employ a variety of potential encoding strategies for evolving neural networks capable of dealing with complex problem domains. A few of these will be described and discussed briefly.

a. Fullmer and Miikkulainen (1991) have proposed an encoding mechanism that is loosely based on marker structure of biological DNA. The advantage of this mechanism is that it allows all aspects of network structure including the number of nodes and their connectivity to be evolved through genetic algorithm. Thus every aspect of network architecture is controlled by evolution. Previous approaches were rigid in this respect yielding networks that were either inefficient or incapable of performing the required task. The marker-based encoding represents a chromosome as a homogenous string of integer values that is manipulated by the genetic algorithm. Similar to biological DNA markers separate individual node definitions. Each definition contains all the information that the nodes need to carry out the computations. The number of layers or the degree of connectivity emerge from

individual node functions instead of being specified *a priori*. The number of nodes in the network depends solely on the number of start/end marker pairs found in the chromosome. Each node definition contains the identification of the node, its initial activation value, and a list specifying its input sources and weights. A neuron may receive input from other nodes, from the sensors and from its own output. The number of connections are determined by the distance between the start and the end markers, allowing each node to use as many or as few inputs as it requires. The chromosome is implemented as a linear list but is treated as a continuous circular entity, that is a node definition may begin near the end of the list and continue at the beginning of the list. Node definitions are not allowed to overlap. Also, the method avoids disruption due to crossover that might yield invalid phenotypes. The effect of mutation depends on where it takes place. It is smooth if mutation occurs on the weights while resulting in significant changes on the markers. The nodes are evaluated in the order in which they are read off the chromosome.

It is demonstrated that the networks are capable of evolving high-level behavior similar to that of finite-state automata. In addition the networks are able to develop an internal world model by evolving an understanding of their sensory inputs and actions.

b. Moriarty and Miikkulainen (1996) have developed an efficient neuro-evolution system called SANE (Symbiotic, Adaptive, Neuro-Evolution) with good scaling properties. Unlike most approaches to neuro-evolution where each individual is a complete network, SANE's individuals are single neurons (hidden neurons in a three-layered network). Each neuron acts as a subcomponent with specialized features and is an object of evolution. The authors argue that evolution at the neuron level promotes population diversity and allows SANE to better evaluate these subcomponents as parts of the final solution. Neurons are defined in bitwise chromosomes that encode a series of connection definitions, each consisting of an 8-bit label and a 16-bit weight field. The absolute value of the label determines where the connection is to be made. The neurons connect only to the input and the output layer. If the decimal value of the label, D , is greater than 127, then the connection is made to output unit $D \bmod O$, where O is the total number of output units. Similarly if D is less than 127, then the connection is made to the input unit $D \bmod I$, where I is the total number of input units. The weight field encodes a floating point weight. Once each neuron has participated in a

sufficient number of networks, the population is ranked to the average fitness values. It is argued that SANE could be implemented with a variety of different neuron encodings and architectures that allow recurrence.

c. SANE is found to be suitable in solving simpler tasks in just a few generations. The reason is that evolving individual neurons often produces a more efficient genetic search. In complicated tasks and those requiring high precision SANE has been found to be inefficient and slow. This problem is addressed by implementing a hierarchical SANE that integrates both the neuron level and network level of evolution in a single framework (Moriarty and Mikkulainen, 1996a). An outer-loop network-level evolution is incorporated on top of SANE neuron population. Thus two separate populations are maintained: a population of neurons and a population of network blueprints. The neuron population provides efficient evaluation of the building blocks, while the population of network blueprints learns effective combinations of these building blocks. Initially the population of blueprints is random resulting in a similar random combinations as performed in SANE. As the blueprint population is evolved, the neuron combinations become more focused towards the best networks. The hierarchical approach thus combines the early efficient exploration of SANE with the late exploitation of the network-level approaches. The hierarchical SANE employs an encoding mechanism that is an extended version of that of the SANE.

d. Another interesting approach is the incremental design of neural networks through artificial evolution (Harvey, 1993). Harvey presents a novel methodology for the design of complex systems through genetic algorithm. Through a framework known as SAGA (Species Adaptation Genetic Algorithm), Harvey has demonstrated that genetic algorithms can be made to work in ill-defined task domains where the search space can increase in complexity indefinitely. The key aspect of this approach is the evolution of real-time recurrent neural networks through variable-length genotypes. The networks are considered as dynamical systems rather than tools to perform computations from input to output. Also, the evolution takes place in a genetically converged population. The framework is applied to evolution of control systems for mobile robots engaged in navigational tasks using low-bandwidth sensors. The encoding mechanism employs two chromosomes. One of these is a fixed-length bit string encoding the position and size of visual receptive patches. Three 8-bit fields per

patch are used to encode the radii and polar co-ordinates of the camera's circular field of view. The other chromosome is a variable-length character string encoding the network topology which is interpreted sequentially. First the inputs units are coded for, each preceded by a marker. For each node the first part of its gene can encode node properties such as threshold values. This is followed by a variable number of character groups, each representing a connection from that node. Each group specifies whether it is an excitatory or a veto connection and then the target node indicated by a jump type and jump size. The jump type allow for both absolute and relative addressing to avoid invalid phenotypes. The internal and the output nodes are handled in a similar way with their own identifying genetic markers. The scheme allows for a variable number of hidden nodes. The crossover operator is designed carefully to cope with the variable-length genotype.

In the above methods both the network architecture and the connection weights are genetically determined. It has been argued that these methods can only yield a network that is entirely innate and there is no learning (Parisi, Cecconi and Nolfi, 1989). The alternative approach has been to train the evolved networks with a standard network learning algorithm such as the back-propagation algorithm. However, back-propagation has a number of drawbacks. Firstly, it has a scaling problem. Although it is highly suitable for simple training problems its performance falls off with problem complexity and makes it unfeasible for many real-world applications. Secondly, it tends to become stuck at local minima (opting to choose local rather than the global solutions). Thirdly, it fails to handle discontinuous node transfer functions. This precludes its use on common node types and simple optimality criteria (Montana and Davies, 1990). Genetic algorithms are employed to evolve weights in these cases. In some cases, they are used to evolve a good set of initial weights that can be further modified by a standard learning algorithm (Miller, Todd and Hegde, 1989). The problem of scaling has been overcome by using modular networks consisting of a number of independently trained sub-networks (Mühlenbein, 1990; Happel and Murre, 1994, and others).

Modularity is an important aspect in problem solving and especially, in the design of neural networks as it offers a number of advantages (Mühlenbein, 1990; Ossen, 1990; Nadi, 1991; Smieja and Mühlenbein, 1992; Happel and Murre, 1994, and others). Firstly, it allows

a complex problem to be expressed in terms of simpler sub-problems as modular components. Secondly, the modules and their interactions are easily interpretable. Thirdly, it offers a natural way of dealing with scalability and finally the learning time can be reduced considerably when compared to that in flat (non-modular) networks. It has been shown experimentally that the global learning algorithms such as the backpropagation algorithm fail to converge when applied to modular networks (Mühlenbein, 1990) suggesting that the algorithm could not make use of the structural information. This is a drawback as the application-specific information can be coded in the structure but the algorithm fails to use this. Also, for larger networks the learning time grows exponentially. With modular networks each of the modules can be trained quickly for a specialized task. It is possible that the modular elements can learn separately in a hierarchical way (Nadi, 1990). The recent work by Happel and Murre (1994) suggest a number of design principles for designing modular networks with genetic algorithm and investigates the relations between structure and function. The results suggest better learning and generalization capabilities of evolved modular network architectures. How effective is an encoding scheme in implementing modularity is an important question. The schemes described above (a, b, c, d) do not seem to address the issue of modularity nor alternative learning methods.

The question that naturally arises is whether there are better connectionist learning rules that can replace the evolutionary algorithm while providing a greater insight into the space of learning mechanisms and how effective are the evolutionary algorithms in searching the space of these rules?

4.3.1.2 Induction of Learning (Genetic-Connectionism)

Genetic-connectionism (Chalmers, 1990) is the idea of using evolutionary algorithms such as genetic algorithm to search the space of potential connectionist learning rules. Learning and evolution are the two fundamental forms of adaptation where the notion of *emergence* plays a key role. This notion has been expressed in a number of ways through a variety of definitions by various researchers (Harvey, 1993), (see chapter two). Others (Chalmers, 1990; Vaario, 1993) conceptualise emergence in terms of achieving a complex high-level behavior as a result of combining simple low-level computational mechanisms in simple ways. In the context of this definition both evolutionary methods and connectionist

systems offer a paradigm of *emergence*. The kinds of emergence found in genetically based systems differ from those found in connections systems (Chalmers, 1990). Connections systems support *synchronic emergence* or emergence over levels: at a given time a host of low-level computations takes place which when looked at from another level can be interpreted as a complex high-level functioning. By contrast, the genetic-based systems support *diachronic emergence*, that is emergence over time; primitive computational systems gradually evolve towards greater complexity. The road to achieving synchronic emergence through evolutionary methods is to loosen the connection between the genotype and the phenotype. When the genotype encodes high-level features directly and symbolically there is no room for synchronic emergence. To achieve synchronic emergence the phenotypic characteristics need to emerge indirectly from the genetic information. This also enables an open-ended search as the relationship between the genotype and the phenotype is indirect and emergent.

When a process of learning evolves through the process of evolution, the evolution is seen as a second-order adaptation that produces individual systems that are not immediately adapted to their environment but that have the ability to adapt themselves to many environments by the first-order adaptive process of learning. Thus the learning mechanisms themselves are the objects of evolution. Based on the encoding strategy the synchronic and diachronic levels may be distinct or may merge.

Further, recent studies on the effect of learning on evolution (Hinton and Nowlan, 1987; Belew, 1990) suggest that learning that is acquired during a lifetime (individual learning) alters the shape of the search space in which evolution operates. That is, learning can be very effective in guiding the search, even when the specific adaptations that are learned are not communicated to the genotype. In the context of connectionist learning it is necessary to investigate its effect on evolution in order to understand how learning and evolution interact.

The following sections will discuss how genetic algorithms were used in evolving a number of neural network learning rules by encoding the dynamic parameters of the network in the genotype and subjecting these to selection pressures.

A few cases are discussed. In the first case, a supervised learning rule is evolved for a single layer feed-forward architecture. The evolved learning rule needs to associate specific input patterns with specific output patterns. The desired output patterns are presented to the network as a training signal. In the second case a supervised learning is evolved for local binary neural networks. The architecture is flexible. In the third case an unsupervised learning rule is evolved for a fixed architecture. The desired output is not known in this case. The evolved learning rules need to induce this information.

- **The supervised learning rule**

Chalmers (Chalmers, 1990) employed a fully connected single-layer feed-forward network with sigmoid output units and with a built-in biasing input to allow for the learning of the thresholds. A maximum connection strength of twenty was imposed, to prevent possible combinatorial explosion under some learning procedures. The network is known to have powerful learning rules such as the delta rule for supervised learning tasks. The aim was to see whether such rules could be evolved. As it is not possible to express all kinds of weight-space dynamics under a single encoding the dynamics are constrained. The constraints imposed in these experiments are that: the changes in the weight of a given connection should be a function of only information local to that connection, and the same function should be employed for every connection. For a given connection from input unit j to output unit i , local information includes four items:

a_j - the activation of the input unit j ;

o_i - the activation of the output unit i ;

t_i - the training signal on the output unit i ;

w_{ij} - the current value of the connection weight from input j to output i .

The genotype encodes a function F given by:

$$\Delta w_{ij} = F (a_j, o_i, t_i, w_{ij}) \quad (4.1)$$

A genotype of 35 bits was employed. This assumes that the function F to be a linear function of the four dependent variables and their six pair-wise products. The genotype

specifies the ten coefficients with the help of an eleventh scaling parameter. With this approach Chalmers succeeded in evolving a number of potential learning rules that included the well-known delta rule. The rules that evolved were evaluated for their fitness by testing them on a number of various linearly separable (Minsky, 1988) learnable tasks on different networks. The fitness of the learning rule is obtained by evaluating its performance on each of the tasks (environment) and taking the mean fitness error over all tasks. Whether the learning rules that evolve are specifically adapted for the tasks that are present during the evolutionary process or whether they are capable of learning a wide range of tasks that were not present during the evolutionary process depends on the diversity of the environment.

A more recent method (Baxter, 1992) based on a similar approach evolves local binary neural networks (LBNNs) consisting of interconnected binary nodes operating in a discrete time, synchronous fashion. The nodes are divided into three classes, that is input, hidden and output nodes. However, the architecture, that is which nodes are connected, is completely unrestricted, rather than layered as in the back-propagation networks. The network operates in two phases, the training and the testing. During the training phase the input and the output nodes are clamped by the environment, whereas during the testing phase only the input nodes are clamped. The network weights include fixed and learnable weights. The learnable weights are adapted according to a local learning rule. The networks are represented as bit strings. It is assumed that each network in the population has the same number of nodes, n , which is fixed for the whole evolutionary run. The network architecture is specified by determining which nodes are connected by non-zero weights, whether those weights are learnable or fixed, and for the fixed weights, their values. This information is coded using three bits for each pair of node in the network. The method allows to employ a uniform length bit strings avoiding problems due to crossover that are typically seen in variable-length strings. The learning rule is a Boolean function of two variables and the training of the network is totally supervised. The network is evaluated based on its ability to learn a number of Boolean functions. The aim of this work is basically to demonstrate that a network's learning ability must primarily be a property of its architecture, and not some sophisticated method of setting its weight. The work is in contrast to the back-propagation networks that have a complex algorithm to set weights, but limited architectures.

- **The unsupervised learning rule**

These experiments employ the Self-Organizing-Map (SOM) architecture (Kohonen, 1990). Kohonen raises a number of questions on the process of self-organization: firstly, are there possibly many optimal algorithms that lead to similar organization produced by the Kohonen rule? Secondly, does the Kohonen algorithm ensue from some more general principles? Thirdly, can the principle also be expressed for a more general structure?

Dasdan (1993) uses a genetic algorithm to evolve a number of unsupervised learning rules such as a Kohonen learning rule (Please refer to Chapter 5 for details). In this case the target value of the exemplars is not known. These experiments suggest that there exists a number of potential unsupervised learning algorithms that are capable of enforcing topological ordering similar to that achieved by the Kohonen learning rule. The equation for the weight adaptation is given by:

$$\Delta w_{ij} = F(w_{ij}, x_j, t, y_j) \quad (4.2)$$

where:

w_{ij} - the current value of the connection weight;

x_j - the signal on the input node;

t - the training iteration number;

y_j - the correlation between the signal x and m , m being the weight associated with the output neuron.

The final equation that evolved included a scaling parameter and fifteen other coefficients. A number of potential learning rules similar to those of the Kohonen rule were evolved along with the Kohonen rule. With the SOM architecture, the definition of the optimal mapping is still unclear. A mean error of the Euclidean distances between input patterns and the weight vectors of their winning cells at the output was used as a criterion for the optimal mapping. A map with the smallest error value was the best map with highest fitness.

In all these cases the evolved learning rule(s) are *subsequently* applied to adapt the network weights in order to evaluate their fitness (that is the fitness of the learning rules). The rules that adapt the weights effectively survive. It is to be noted here that the synchronic and the diachronic levels are distinct.

An excerpt from Chalmers (Chalmers, 1990) that emphasizes the criticality of the coding strategy is included here:” The encoding strategy is crucial. Whether it should allow for many possible weight-space dynamics or should the space be constrained using a priori knowledge? How could we possibly find a coding of possible dynamics that includes as possibilities all the diverse learning algorithms proposed by humans today? The above experiments employed small networks and the relevant information was known in advance that a simple quadratic formula can provide a good learning mechanism. The encoding of more ambitious mechanisms such as the back-propagation may not be so simple but would need highly complex genetic coding or else a simple but very specific coding that is rigged in advance to allow back-propagation as a possibility. When we do not know the form of a plausible learning algorithm in advance- and this is the most interesting and potentially fruitful application of these methods- the problem of coding strategy becomes vital. Only so much can be coded into a finite-length bit string. One way around the limitation of pre-specified coding of dynamic possibilities would be to move away from the encoding of learning algorithms as bit-strings, and instead encode algorithms directly as function trees. In recent report, Koza (1990) has demonstrated the potential of performing genetic-style recombination upon function-tree specification of algorithms. This method of “genetic programming” uses recombination and selection in a fashion very similar to traditional genetic methods, but with the advantage that under evolutionary pressures such function-trees may become arbitrarily complex if necessary. This open-endedness may be a good way of getting around the limitations inherent in fixed genetic coding. Furthermore, the method is a very natural way of encoding dynamic, algorithmic processes of the kind we are investigating here..”

The following sections will investigate the role of genetic programming in contexts of neuro-evolution and genetic-connectionism.

4.3.2 The Genetic Programming approach

Genetic Programming (GP) encode possible solutions to a problem as *programs* that, when executed are the candidate solutions to the problem. These programs are expressed as parse trees and consists the terminal and the function sets of a given problem environment. The search algorithm that is used in GP is the classical genetic algorithm. With appropriate terminals, functions and/ or interpreters standard GP can go beyond the production of tree-like programs (Poli, 1996). Its role in connectionism has yielded powerful insights into the design and learning aspects of neural networks and will be the focus of the next few subsections.

4.3.2.1 Network induction (Neuro-Evolution)

GP has been successfully applied to evolving neural network architectures along with network weights (Koza, 1993). However, the method does not provide a general approach to implement standard networks nor a mechanism for finding networks with minimum complexity (Zhang and Mühlenbein, 1993). Recently alternative methods have emerged and a few will be discussed.

a. The individual structures that undergo adaptation in genetic programming are hierarchically structured computer programs. These programs can be expressed as LISP S-expressions that can be graphically depicted as rooted, point-labelled trees with ordered branches. With such a representation for the genotype a variety of encoding schemes is possible. For instance, the representation can indirectly encode a rewriting grammar such as the cellular encoding (CE) (Gruau, 1993). This grammar is interpreted in a recursive manner generating a family of related networks. The advantage of this approach is that larger networks can be evolved with a very compact code providing a wide range of possible network architectures. The method, being highly successful in the evolution of Boolean networks (both standard and modular types), has recently been applied for designing network architectures with real-valued weights (Friedrich and Moraga, 1996).

b. Network architecture and weights have been optimized simultaneously with an evolutionary approach known as the Breeder Genetic Programming (BGP) (Zhang and Mühlenbein, 1993). The genotype of each network is represented as a tree whose depth and

width are dynamically adapted to the particular application by specifically defined genetic operators. The weights are trained by next-ascent hillclimbing search and employs a fitness function that quantifies the principles of Occam's razor. Occam's razor states that unnecessarily complex models should not be preferred to simpler ones. Hence the method prefers a simple architecture to a complex one. However scaling problems were observed due to the direct encoding scheme employed. That is, the genotype directly encodes the network architecture. The problem of scaling can be overcome by grammar encoding. Again the disadvantage with grammar encoding is that the genotype must be converted to phenotype every time the weights are trained. Direct encoding schemes are preferable in this context. The recent trend has been towards more compact representation schemes which can exploit the advantages of both direct and indirect encoding strategies.

c. Recently, an extended version of genetic programming known as Parallel Distributed Genetic Programming (PDGP) is claimed to be highly suitable for development of parallel programs (Poli, 1996). The method allows symbolic and neural processing to be combined in a natural way through a graph-like representation. PDGP uses a direct representation of graphs which, although not completely general, allows the definition of crossover operators which always produces valid offspring in an efficient way. Each node on the graph is assigned a physical location on a multi-dimensional grid with a pre-fixed shape and limiting the connections between the nodes to be upwards. Also connections can only be established belonging to adjacent rows, like the connections in a standard feed-forward multi-layer network. The limitations of this method has been the increased computational effort to develop programs with weighted links as the operators used by PDGP are ineffective in optimizing the network weights. However it is successful in optimizing the topology and also in discovering a variety of complex network architectures.

4.3.2.2 Induction of Learning (Genetic-Connectionism)

The existing connectionist learning algorithms despite being quite effective in tackling a wide range of problems have a number of limitations. It appears that the limitations are due to different types of rigidity. The rigidity could be in terms of:

- the architecture where the connectivity of most of the networks is fixed in advance or in a mental rigidity. That is, the assumption that the node's activation function must be a real

number and that activation should be combined using weighted sums and sigmoid functions. These assumptions limit the universality of the learning algorithms (Fletcher, 1990). Fletcher has shown that nodes in the networks can contain any (bounded) data-structure and any processing function appropriate to the problem at hand.

- the way in which the learning algorithms are defined or in the approach through which they are implemented. Should there be a general definition for a learning algorithm (Govinda Char, 1997a)?
- the constraints. Is the rigidity due to the type of constraints imposed (Govinda Char, 1997b)?
- the convenience. Is the required flexibility not achievable because of a tendency to use simple methods in order to avoid complexity instead of trying to tackle it (Ciff, Harvey and Husbands 1992)?

In addition, most of the connectionist learning algorithms hardly resemble learning in natural systems. Should this problem be again attributed to the above facts? Also, whenever the learning algorithm is known a priori, the designer implicitly has a notion about the way the algorithm is going to behave and also sometimes the likely outcomes. In the context of artificial intelligence such a strategy cannot be very fruitful when it comes to situations where the learning algorithm has to deal with unpredictable environments. These clearly suggest that learning algorithms need to evolve to suit the situation.

Genetic algorithm though was successful in evolving a variety of potential connectionist learning rules mostly had the architecture and the type of node activation function fixed a priori. The approach limits the potential of evolutionary algorithm in searching a much larger space of learning rules. Certain strategies, if employed appropriately might allow the desired flexibility and open-endedness.

So far a few researchers (Bengio et al., 1994; Radi and Poli, 1998) have attempted evolution of learning with genetic programming. Radi and Poli (1998) have used GP to discover new supervised learning algorithms. GP allows direct evolution of symbolic learning rules with their coefficients (if any) rather than the simpler evolution of parameters of a fixed learning rule. A feed-forward network with input, hidden and output layers is used to

explore a larger space of rules using different parameters and different rules for the hidden and the output layers. The results suggest that the evolved rules are faster (converge in a few epochs) and have better generalisation capability than the standard back-propagation learning when tested on a number of sample problems.

The method proposed in this thesis uses an entirely different approach. It is based on combining two potential strategies: the bottom-up and the top-down. Also, the method attempts to exploit the strengths that are inherent to the representation that GP employs, that is the aspects of hierarchy and modularity that the representation offers. Most importantly, it is based on providing a very general definition for learning and on the imposition of a single potential constraint within the representation. The framework that is used is a self-organizing neural network. The assumptions and the justifications will be stated first.

- **The assumptions**

1. A learning rule is defined simply as *a sequence of interacting concepts* such as association, competition, co-ordination and adaptation. This definition is necessary to proceed further.
2. The network weight adaptation is an *integral* part of the representational structure, that is the genetic programs and hence the evolutionary process. This strategy is indeed a potential *constraint* and is the *key* to the simulations. In the context of the first assumption, the network weight adaptation can be thought of as an abstract symbolic concept.

These assumptions are justified based on the facts that learning in natural systems also entails evolution of symbolic concepts and their proper sequencing. Also, the neural structures in the brain adapt while forming concepts, that is, *while* interacting with the environment.

- **Why Genetic Programming?**

The first assumption entails evolution and sequencing of concepts appropriately so as to yield potential learning rules. Genetic programs are tree-structured symbolic expressions. That is, the programs are hierarchical LISP S-expressions. This hierarchy should enable the

sequence that is needed in the above definition of a learning rule. The interactions of the concepts could be as a result of the above hierarchy itself and also due to the effects of the recombination of genetic programs. The recombination basically swaps the subtrees.

Genetic programming is the *mechanism* that is required to evolve the macro-concepts such as the concept of winning, competition, co-ordination and adaptation through its primitives. The primitives are the function and the terminal sets for the GP run. These primitives implicitly represent the micro-concepts that GP will employ to form the macro concepts and sequence them appropriately to yield a potential learning rule. The approach thus employs the notion of *micro-macro dynamics* in realizing *emergence*.

The second assumption is *vital* to the simulation work. In earlier approaches with genetic algorithms (Chalmers, 1990; Dasdan, 1993), learning rules were evolved and *subsequently* adapt the network weights. The two levels of adaptation, that is, the synchronic and the diachronic levels were distinct. With the proposed approach the concepts evolve *while* interacting with the environment. The two levels of adaptation merge as the weight adaptation is an *integral* part of the representational structure itself. The learning rules in this case need to adapt the weights effectively in order to evolve. Although this is a new constraint the paradox is that it will force GP to evolve potential concepts. Thus the constraint provides an *implicit motivation* for the evolutionary paradigm to be creative.

This leads us to a *key* question: In the context of problem solving should a learning rule evolve to adapt the network weights effectively? or should it adapt the network weights effectively in order to evolve? The subtleties need to be understood to appreciate the depth and the consequences.

4.4 Discussions

The sections discussed the role of evolutionary algorithms, in particular, genetic algorithms and genetic programming in connectionist networks. The focus was on neuro-evolution and genetic-connectionism, the two main approaches that have emerged in recent years. The strength of the DNA based encoding (Fullmer and Miikkulainen, 1991) is that it allows all aspects of network structure including the number of nodes and their connectivity, to be

controlled by evolution. The number of layers and their connectivity are not defined a priori but emerge from the node definition. The designer has to carefully craft the definition of the node and appears to have some implicit notion about the way the networks might evolve. Also, it is not clear whether the encoding scheme is applicable to all types of networks such as recurrent networks. SANE (Moriarty and Miikkulainen, 1996) evolves a population of neurons with specialized features and combine them to form a network. The advantages are the quick evolution of networks and good scale-up properties. However, the type of network is fixed in advance. Again the node definitions are carefully crafted by the designer. It is slow in dealing with complex problems. This problem is overcome in hierarchical SANE (Moriarty and Miikkulainen, 1996a). The hierarchy and its implementation has to be decided by the designer. The methods employ a fixed length chromosome. The incremental design suggested (Harvey, 1993) has the advantage of having a variable-length chromosome and applicable to recurrent networks. It also allows a flexibility in architecture. The chromosomes are again carefully designed. It is not clear whether the networks that evolve are optimum.

None of the above methods have addressed the modular aspects in network design that is vital.

In the context of genetic-connectionism, Chalmers (1990) employs a known architecture to evolve learning mechanisms. Baxter (1992) has managed to remove the restriction in fixing the architecture by an efficient encoding scheme. The constraint though is that each network in the population should have the same number of nodes. Dasdan (1993) employs a similar approach as that of Chalmers for evolving unsupervised learning for a fixed architecture. All the above approaches employ fixed-length chromosomes limiting the search space.

With genetic programming, grammar-encoding (Gruau, 1993) is highly suitable for network design in general and modular networks in particular. The chromosome is compact allowing good scale-up properties. It is not clear whether the networks are optimum in terms of the weights. Breeder Genetic Programming (Zhang and Mühlenbein, 1993) evolve optimised networks and weights simultaneously. However, the grammar encoding used in these methods are extremely time-consuming. The graph-based approach (Poli, 1996) seems to

overcome the problems associated with grammar encoding. Again, the crossover operator has to be carefully designed to yield valid networks. Whether modularity can be implemented remains to be seen.

Coming to the learning aspect of the networks, the type of network architecture is fixed in advance in the approach suggested by Radi and Poli (1998). Despite, the advantages of this method are faster convergence time and better rules as compared to the standard back-propagation learning rule. It is not clear whether the approach could be extended to other types of neural networks.

The novel approach proposed in this thesis (Govinda Char, 1997a) has several advantages. The representation structure (the genotype) can be varied in shape, size and the complexity allowing for an open-ended search. The approach is based on providing a general definition for learning and involves a single potential constraint within the representation. The evolutionary paradigm has all the options open to it in terms of the network architecture, the node activation function and the type of learning it can evolve. The flexibility in network architecture can be achieved by incorporating a technique for morphogenesis allowing GP to induce a variety of network architectures. The grammar of cellular encoding (Gruau, 1993) is highly compatible with genetic programming and flexible in implementing any type of neural network. Although the focus in this thesis is on using a self-organizing neural network as a framework for the purpose of demonstration, it appears that the approach can be extended to other types of networks. The reasons being, firstly, a general definition for learning is provided irrespective of the type of network. Secondly, the grammar is flexible enough to generate different types of network architecture. Also, it allows a possibility of evolving the node activation functions through appropriate primitives. It is unlikely that natural systems employ the same type of node activation function at various subsymbolic levels. These advantages offer a flexibility that allows for an open-ended search for the evolutionary paradigm. The aim is to see how one can extract maximum information from the paradigm (in terms of learning and problem-solving) just by imposing a *single potential constraint* while leaving every other option open to the paradigm to choose. Further, learning in natural systems involves logical primitives. Connectionist learning should have the freedom to choose for logical primitives based on a given situation. Such primitives can

be easily included with the proposed approach. Finally, the learning rules that evolve should be interpretable in symbolic terms. A standard connectionist learning rule can be easily expressed (and explained) in terms of few statements in natural language. This can be achieved through automatically defined functions (ADFs) (Koza, 1994) in genetic programming as demonstrated in the next two chapters. Finally, the hierarchy and modularity (through ADFs) are inherent features of the representation. Given an appropriate grammar for network generation, the designer need not craft the genotypes carefully nor worry about the effects of crossover disruption. If the aim is to really automate the design process the human involvement in the design loop has to be minimized. The greatest advantage as compared to other methods is that the network and the learning can evolve simultaneously while interacting with a given environment. Grammar encoding is slow which, of course, is a disadvantage.

Conclusion

It seems that the rigidity in connectionist learning algorithms can be avoided in a number of ways. The first of these involves providing a very general definition for a learning rule, for instance, as a sequence of interacting concepts. Secondly, by imposing a potential constraint and leaving most of the options such as the network architecture, node activation function and the type of learning open to the evolutionary paradigm. The novel approach suggested in this thesis has several advantages when compared to other evolutionary approaches. It exploits maximally the strengths that are inherent to the representation that the evolutionary paradigm (GP) employs. These are: the hierarchy and modularity which are very important in problem solving. Further, it is based on powerful notions such as micro-macro dynamics and constructivism and offer a way of combining bottom-up and top-down strategies.

Chapter 5 will discuss a self-organizing neural network that is used as a framework for further simulations.

Chapter 5

Self-organizing Neural Networks

Topological feature maps are ubiquitous in the brain. These maps are formed as a result of the process of self-organization that involves the basic principles of competition and coordination among the cells in the brain and have been successfully modelled by artificial neural networks. This chapter provides an overview of some of the computational models that have been effective in capturing and simulating the process of self-organization accurately. The role of evolutionary paradigms in the evolution of such models and the advantages of evolutionary approach are discussed.

5.1 Introduction

Topological feature maps are ubiquitous in the brain (Knudsen et al. 1987) and show up in a localization of cortical activity by sensory stimuli (Tavan et al. 1990). These maps are characterised by the fact that sensory signals that are closer will cause excitations in the nearby regions of the cortical plane. An example of a topological map is the retinotopic map in the visual cortex. The basic principles for the self-organization of topological feature maps from sensory input have been established (Malsburg and Wilshaw, 1977; Malsburg, 1976) and involve competition among the neurons of the map for maximal response and cooperation of the neighbouring neurons. Later a simple algorithm demonstrating these principles was developed (Kohonen, 1982a, b; 1984) and was successfully applied to a variety of problems that included vector quantization (Schweizer et al. 1991), biological modelling (Obermayer, Ritter and Shulten, 1990), combinatorial optimization (Favata and Walker, 1991), processing of symbolic information (Ritter and Kohonen, 1989) and for motor control in robotics (Ritter and Shulten, 1988a, b; Ritter and Kohonen, 1989). A few models that have been successful in simulating the process of self-organization will be discussed.

5.1.1 The Kohonen Self-organizing Feature Map: the Characteristics and the Learning Rule

The Kohonen feature map is a two-layered network as shown in Figure 5.1. The first layer is the input layer consists of a number of cells (neurons) each taking on a

corresponding value from the input pattern. The second layer, the competitive layer is typically organised as a two-dimensional grid of cells. The two layers are fully interconnected as each input unit is connected to all of the units in the competitive layer through an associated reference vector (Dayhoff, 1990).

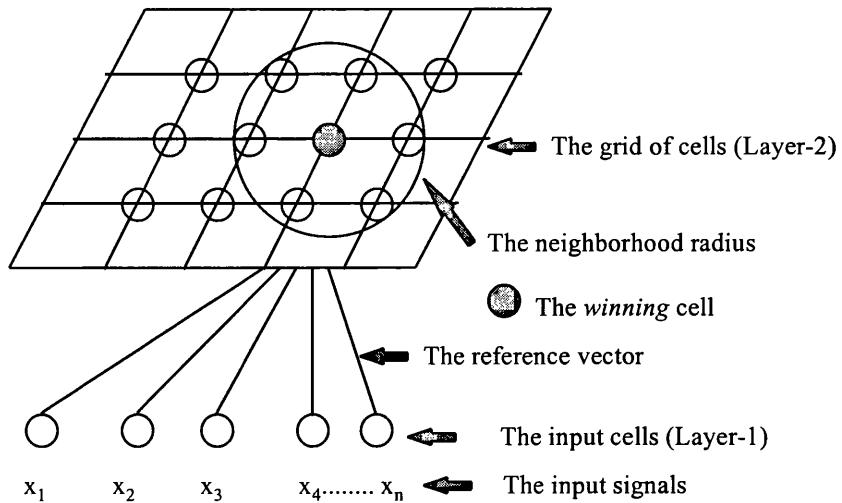


Figure 5.1 The Kohonen Feature Map

That is, an n -dimensional reference vector associates each of the cells on the competitive layer with an n -dimensional input signal. Other cells on the competitive layers are also associated in a similar manner. When an input pattern is presented, each unit in the first layer takes on the value of the corresponding entry in the input pattern. The units on the second layer then sum their inputs and compete to find a single winning unit (the *winner*). The reference vector determines the cell that is maximally sensitive to a particular input signal based on the Euclidean distance of the signal from the reference vector. The learning algorithm organizes the cells on the second layer into local neighborhoods that act as feature classifiers on the input data. Thus the reference vectors of neighboring units are near each other if the signals are close. In essence, a given set of reference vectors divides the input vector space into regions with a common nearest reference vector. These regions are commonly known as *Voronoi* regions and the corresponding partition of the input vector space is denoted *Vornoi* partition. The network learns in an unsupervised manner from a stream of input signals. The n -dimensional input vector is denoted by $\mathbf{x} = (x_1, x_2, \dots, x_n)$ with real-valued components taking values in the subspace $\mathbf{V} \in \mathcal{R}^n$. Exemplars of such vectors are repeatedly presented to the network. The values of \mathbf{x} are drawn randomly according to a given probability distribution. For each presentation the best matching cell (the *winner*) is determined according to the minimum value of the Euclidean distance $\|\mathbf{x} - \mathbf{w}_n\|$ where $\mathbf{w}_n \in \mathcal{R}^n$

represents the n -dimensional reference vector. The weights are then adapted according to the rule:

$$\mathbf{w}_n(t+1) = \mathbf{w}_n(t) + \eta(t)g(\mathbf{n} - \mathbf{n}_0, t)(\mathbf{x} - \mathbf{w}_n(t)) \forall \mathbf{n} \quad (5.1)$$

where ‘ t ’ is the update time. The parameter $\eta(t)$ is the learning rate. The function $g(\mathbf{n} - \mathbf{n}_0)$ is typically a Gaussian given by:

$$g(\mathbf{n} - \mathbf{n}_0) = \exp(-p), p = \|\mathbf{n} - \mathbf{n}_0\|^2 / 2\Delta^2 \quad (5.2)$$

and is essential for the success of the algorithm. Equation (5.2) has a maximum value (normalized to unity) when \mathbf{n} coincides with \mathbf{n}_0 (the winner cell) and decays to zero at larger distances. The steepness of the decay is characterized by the width parameter Δ . Thus the *winner* cell on the network is maximally adapted and the surrounding cells are adapted to a lesser extent depending on the distance $\|\mathbf{n} - \mathbf{n}_0\|$. The function g induces a lateral inhibition among the neurons. The learning rate $\eta(t)$ and $\Delta(t)$ are initially large but reduce monotonically as the learning progresses according to some cooling schedule. The learning (“winner takes most”) is distinguished from a competitive learning where only the *winner* is adapted (“winner takes all”). Such networks are capable of generating interesting low-dimensional representations of high-dimensional input data.

5.1.1.1 Performance Criteria

Self-organizing networks have mainly three performance criteria. The importance of each criterion may vary based on the application (Fritzke, 1993).

- **Topology preservation**

The mapping from input space to the output space is said to be topology-preserving if similar inputs are matched to identical or neighboring cells and neighboring cells have similar reference vectors. The first property ensures that small changes in the input vector will cause correspondingly small changes in the position of the winner unit. Such a mapping is robust against distortions of inputs and highly desirable while dealing with noisy data. The second property ensures the robustness of the inverse mapping. That is, when the dimension of the input mapping is higher than the dimension of the network the mapping reduces the data dimension but usually preserves the important similarity relations among the input data.

- **Modelling of probability distribution**

A set of reference vectors is said to model the probability distribution, if the local density of the reference vectors in the input vector space approaches the probability density of the input vector distribution. This property is desirable for two reasons. First, it is possible to get an implicit model of the unknown probability distribution underlying the input signals and second, the network becomes *fault-tolerant* against damage as every cell is only responsible for a small fraction of all input vectors.

- **Minimization of quantization error**

The *quantization error* for a given input signal is the distance between the signal and the reference vector of the *winning* cell. A set of reference vectors are said to be *error minimizing* for a given probability distribution if the mean quantization error is minimized. This property is important if the original signal needs to be reconstructed from the reference vector which is common to vector quantization. This error needs to be small for efficient self-organization.

5.1.1.2 The Problems and the Limitations

The specific architecture (the two-dimensional rectangular grid) that the Kohonen network employs imposes limitations on the process of self-organization (Ervin et al. 1995; Polani, 1995; Zavrel, 1996) due to a number of reasons. Firstly, the convergence of the self-organizing process to a stable state is only guaranteed if the learning parameter and the neighborhood radius are slowly decreased during learning. Otherwise the network weights may perpetually change when patterns are presented. There is no guarantee that the imposed schedule for the decrease of the adaptation is optimally organized despite a stable state. Secondly, during the adaptation the grid may get tangled or collapse into a single point depending on the combination of data and initial parameters, resulting in a severe distortion of distances in the map. Thirdly, the rectangular grid structure does not allow proper adaptation to the input signal spaces that have non-rectangular distributions. Finally, the convergence of Kohonen networks is slow and they easily get stuck at local minima. The remedy to these problems lies in the creation of network structures that can better adapt to the structures of the input space.

5.1.2. The Growing Cell Structures

To overcome the limitations imposed on the resulting mappings by the predetermined structure and size of the Kohonen's model the Growing Cell Structures method (Fritzke, 1991) was introduced. This model differs from the Kohonen's model in some respects. Firstly, new cells can be added or removed during the process of self-organization. Secondly, the weight adaptation rule, although almost the same as that proposed by Kohonen, has two important differences. The adaptation strength is constant over time and two different adaptation parameters are used for the *winning* cell and the neighboring cells respectively. Only the *winning* cell and its *direct* neighbors are adapted. These choices eliminate the need to define a cooling schedule for any of the model parameters. The advantages of this approach are that the size as well as the structure of the final neural network are determined automatically from the input data. The network size is not pre-defined but grows until a performance criterion is met. As the cells are grown based on the pattern space, the method enables many clusters to cover the dense regions of the input space and a few at the sparsely occupied regions. The true structure of the data set will be reflected in the cluster structure more accurately including cluster boundaries and hence allows for data visualization. The Kohonen's models do not provide such information as there are no cluster boundaries on the map.

5.1.3. The Enhanced Feature Map: Modelling Lateral Interactions

Both the above approaches rely on an external supervisor to find the maximally active unit, that is, the *winning* cell. To be biologically realistic the algorithm should be reduced to local computations and interactions among the cells. The lateral interaction weights between the cells can also be made to self-organize along with the external input weights (Miikkulainen, 1991; Sirosh and Miikkulainen, 1995). Each cell in the neural network is assumed to have three sets of inputs: the excitatory input connections that supply external inputs to the cell; short-range lateral excitatory connections from close neighbors on the map; and long-range inhibitory connections from within the map. Also it is possible that the external and the lateral connections of the same cell follow two different rules of weight modification. The lateral weights are modified by a Hebb rule keeping the sum of the weights constant whereas the external input weights are modified according to the normalized Hebbian rule. This

enables the computations to be localized to each cell and its connections. Such a map can autonomously self-organize without a global supervisor.

5.1.4 Incremental Grid Growing

The usefulness of a map depends on how accurate it is in representing the input space. This space may be arbitrarily non-convex and discontinuous and may contain high-dimensional clusters. Further the real world data sets often contain distinct but non-obvious subsets of data. The standard learning algorithm fails to delineate the boundaries of such groupings. The incremental grid-growing algorithm (Blackmore, 1993) is based on an incremental approach, but avoids the difficulties of an arbitrarily connected graph structure as it retains a regular 2-dimensional grid at all times. The algorithm is briefly explained. Initially the feature map grid consists of four connected nodes with weight vectors chosen at random from the input. Each main iteration of the algorithm consists of three main steps:

1. Adapting the current grid to the input distribution through the usual feature map self-organizing process.
2. Adding nodes to those areas in the perimeter of the grid that inadequately represent their corresponding input area.
3. Examining the weight vectors of neighbouring nodes and determining whether a connection between the nodes should be deleted from the map, or whether a new connection should be added.

The new structure is re-organised, and the process continues until a predetermined maximum number of nodes has been reached. Thus the algorithm by employing effective heuristics enables the non-convexities, discontinuities and clusters in the data set to be represented explicitly on the two-dimensional structure of the map during the process of organization.

5.2 The Evolutionary Approach

The above models employed non-evolutionary, standard programming techniques. In general, the space of possible neural network architectures and learning for a given problem domain could be very large. As discussed in chapter four, evolutionary algorithms have been successfully applied in neural network design in several ways. Genetic algorithms have

yielded improved topologies of self-organizing neural networks capable of better adaptation characteristics (Polani and Uthmann, 1992; Hämmäläinen, 1995). Novel learning algorithms for the self-organizing process providing better performance measures have also been synthesised using genetic algorithms (Dasdan and Oflazar, 1994).

The following sub-sections will briefly describe the genetic algorithm approach for optimizing the Kohonen feature map. The advantages of employing the genetic programming paradigm in designing such networks is outlined.

5.2.1 The Genetic Algorithm approach

The sequence of steps for modelling the self-organization process with the genetic algorithm will be discussed first.

5.2.1.1 The Genotype-Phenotype Mapping

The genotype-phenotype mapping scheme is crucial for evolving optimal topological structures. The mapping schemes dictate the convergence time in terms of the search space and also the scalability of the evolving neural networks. Two different mapping schemes for evolving an optimum topological structure for a self-organizing neural network through genetic algorithms are illustrated and discussed briefly.

- The approach (Hämmäläinen, 1995) employs a connection matrix for encoding the network connections (direct encoding).

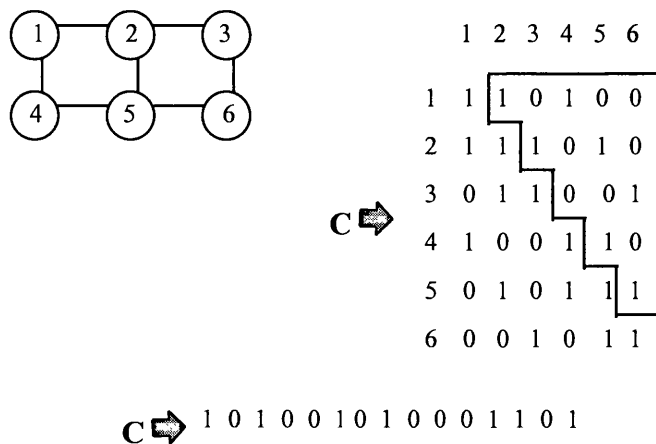


Figure 5.2: The Connection matrix C

Figure 5.2 shows the matrix. The network has six nodes (two rows and three columns). A 1 implies a connection between the nodes (every node is connected to itself). The upper (right) triangle of the matrix is used as the chromosome.

The genetic operations were crossover and mutation. The fitness of the individuals were tested using the following function for the trained network given by:

$$f = \sum_{j=1}^n \sum_{\substack{x_i \in F_j \\ i}} \|x_i - m_j\|^2 * (1 + M) \quad (5.3)$$

If the measure of the disorder $M=0$ then F_j is the Voronoi region of m_j , that is, for every $x \in F_j$ m_j is the nearest of all m_i 's. The maps that broke into several parts were penalised. Other fitness measures can also be employed. The problem with direct encoding of the network matrix into a chromosome has limitations in terms of its length, requiring longer chromosomes to encode larger networks (Kitano, 1990). As discussed earlier in chapter four, this will not only increase the search space but degrade the performance of the networks with their size. Also, the method assumes that the connectivity information encoded in the DNA to be in almost one-to-one correspondence. Indirect methods either apply simple rules to chromosomes or encode certain types of grammar that are compact and more suitable in overcoming the above limitations. Kitano's approach encodes a graph generation grammar that defines the growth of graphs. The grammar is an augmentation of Lindenmayer's L-system (Prusinkiewicz and Lindenmayer, 1990) that is designed to describe morphogenesis, that is the growth and migration of cells.

The next sub-section will discuss an indirect method.

- In this method (Polani and Uthman, 1993) the neural network is assumed to be an undirected graph that is optimized with a genetic algorithm. This results in optimizing the network topology. The interaction between the genetic algorithm and the Kohonen map is shown in figure 5.3. Every genotype defines the topology of a Kohonen net via a transcription rule. The network is trained with the standard algorithm and then subjected to a quality test which serves as a fitness function for the genetic algorithm.

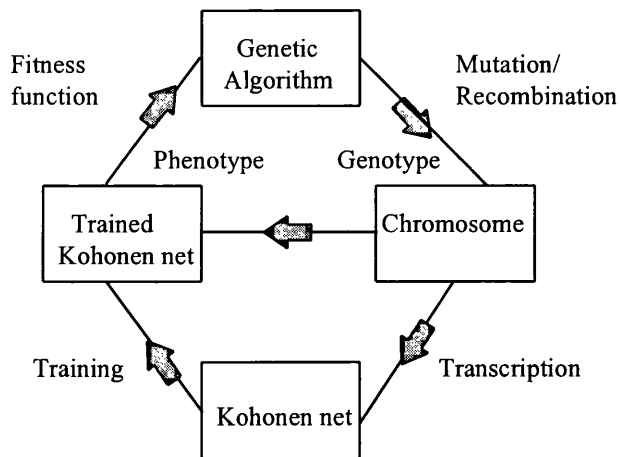


Figure 5.3: Interaction between the Genetic Algorithm and the Kohonen Feature Map.

The feature map is regarded as an undirected graph G where a weight vector $\in [0,1]^m = I^m$ is associated with every vertex. The transcription rule is described in figure 5.4. The rule yields a valid neural network with a unique topology for every chromosome.

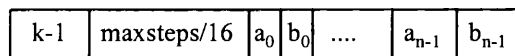


Figure 5.4: A transcription rule.

The transcription rule requires the chromosome to consist of 2 bytes + n double-bytes. The parameters a and b are the constants of the transcription process. The number k of vertices in the graph is given by the first byte +1 guaranteeing the net to have at least one vertex. The second byte multiplied by 16 yields the maximum number of transcription steps that may be done before the procedure stops (maxsteps). These two header bytes are followed by n double bytes each of which defines one transcription step. In every transcription step a vertex is connected with a different one. Whenever the rule tries to connect two vertices that are already connected or tries to connect a vertex with itself, the algorithm stops. The links determine the topological neighbourhood. An example showing the transcription algorithm and the resulting connectivity is illustrated in figure 5.5.

*/*transcription: k, n and maxsteps as defined above */*

```

from :=0; to :=0; step :=0; i :=0;
while step < maxsteps do /*transcription step */
  from := (to + a[i]) mod k
  to :=(from + b[i]) mod k

```

```

if (from = to) or (vertex[from] is connected with vertex[to])
  then exit while loop
connect vertex[from] with vertex[to]
step := step + 1
i := (i + 1) mod n
endwhile

```

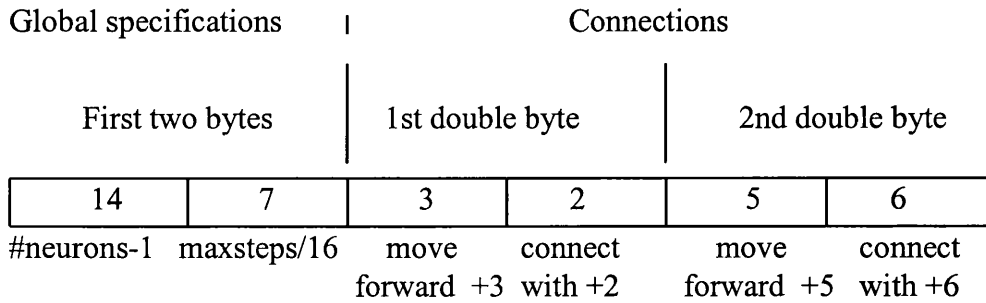


Figure 5.5: A two bytes + two double-bytes chromosome.

The topology resulting from the transcription rule is then trained with the standard Kohonen learning rule until a termination condition is satisfied. The genotype-phenotype mapping thus consisted of two phases: firstly, applying the transcription rule to the chromosome to yield a network topology. Secondly training this network with the standard learning rule. The trained net can then be regarded as the phenotypic expression.

5.2.1.2 The fitness function

The domain of the experiment chosen was for input signal spaces of a square $[0,1]^2$ and extended to toroidal and Möbius topology. For the sake of simplicity, a quality function which gives an estimate of the average distance from an arbitrary signal $x \in I^m$ to the nearest weight vectors of one of the vertices of G is chosen. For a sample of points $x_i \in I^m$, $i=1$ to $i=q$ the quality function is given by:

$$Q(G) = 1 / \left(\sum_{i=1}^q w^*(x_i) - x_i \right)^2 \quad (5.4)$$

where, $w^*(x_i) := w_v^*$ if \forall vertices $v: d_1(x_i, w_v^*) \leq d_1(x_i, w_v)$ holds. The vertex v^* is activated by sample x_i . $Q(G)$ is essentially a measure of the average distance from an input

vector to the vertex it activates. A smaller average distance yields a higher quality function, indicating a better adaptation to the input signal space. The weight adaptation in effect reflects the process of self-organization. The method yielded improved topologies with a simple fitness function.

The next subsection briefly describes the novel approach with genetic programming (GP). It combines both the bottom-up and top-down strategies for the evolution of network structure and flexible learning that effectively adapt the network weights. Chapter six will discuss the GP approach in detail through the simulation results.

5.2.2 The Genetic Programming Approach

The learning rule for a self-organizing neural network is based on a number of concepts such as competition, co-ordination, adaptation and so on. These concepts can be evolved (and represented) through different ADFs and be appropriately sequenced so as to enforce a topological ordering (Govinda Char 1997a; 1997b). Genetic programming has to evolve these ADFs (macros) through the supplied primitives (micros) and sequence them appropriately using the fitness function. Further, the network architecture can be made to evolve simultaneously with an additional ADF as terminal that might encode simple rules or a grammar. The result is the evolution of dynamic network structures while interacting with the task environment (the given signal space). The learning rules to adapt the network weights are evolved on-the-fly.

The advantages of GP approach over other approaches can be summarised under two different contexts.

1. GP vs. the non-evolutionary methods.

As discussed in chapter four, the space of possible neural network architectures and learning is extremely large. GP, being an evolutionary paradigm is capable of searching this space for optimality whereas the non-evolutionary methods (standard methods) have limited options. Moreover, standard methods do not allow the possibility of *evolving* potential learning rules while interacting with the task environment. GP offers a way that enables evolution of a variety of network architecture and learning for a given task environment

using only the fitness information. A learning rule is expressed in terms of a number of concepts. Each of these concepts can be implemented in terms of a module that can easily be implemented through an automatically defined function (ADF). Although the basic primitives are supplied to GP in terms of the function and terminal sets, it is not known *a priori* how these primitives will form an ADF or how the ADFs will combine to yield effective learning rules. Also, the designer has an option to define and include additional primitives that are employed in standard methods and observe their effects on the process of self-organization.

2. GP vs. GA

Genetic programming offers a number of advantages over the genetic algorithm in terms of the representation. The representational structure in the genetic programming is a variable length hierarchical tree structure. This offers a large search space for the network architecture and learning mechanisms to be explored by the evolutionary paradigm. Combined with automatically defined functions it allows the possibility of implementing hierarchical, modular elements capable of dealing effectively with the problem domain. The modularity allows the interpretability of the results as symbolic entities. When GP is employed for evolving an optimum neural network topology, the hierarchical representation itself, in some cases, might express the process of problem solving at a subsymbolic level (Koza, 1993). The tree representation is also highly suitable for certain types of grammar encoding to incorporate a biologically plausible developmental process. The encoding will enable overcome the problem of scalability through shorter genotypes.

In contrast, the genetic algorithm approach employed chromosomes of fixed length and the standard Kohonen learning rule for the weight adaptation phase. It is to be noted here that a variable length chromosome can be employed with the genetic algorithm approach as well. So far, there has been no attempts to co-evolve network structures along with learning.

Because of the representation of the genotype, typically as a string it is difficult to achieve/ implement the notions of hierarchy and modularity that are vital not only to problem solving but also in expressing the solutions. These features are inherent to GP due to the

representation it employs suggesting GP has definite advantages when compared to the standard and the genetic algorithm approaches.

Conclusion

The chapter mainly focused on the process of self-organization and discussed a few computational models that are highly effective in capturing the process. The limitations of the Kohonen's feature map and the attempts to overcome those limitations through various models were discussed in detail. Finally the role of evolutionary algorithms was emphasised by introducing the genetic algorithm and the genetic programming methods. Being population based and due to their implicit parallelism evolutionary algorithms are capable of searching an extremely large space of neural network architectures and learning for any task environment. In addition, evolutionary methods allow the network structures and the learning to emerge during the course of problem solving rather than being defined *a priori*. The standard methods have limited options in this context. A comparison with other methods highlights the potential of the GP approach in terms of achieving flexible network architecture and learning. The representation that GP employs has definite advantages in expressing solutions in terms of hierarchical modular elements.

Chapter six, through simulations, will demonstrate the role of GP in the evolution of flexible learning for self-organizing neural networks.

Chapter 6

Simulation Results

This chapter will demonstrate the role of genetic programming (GP) as a meta-learning paradigm for a self-organizing neural network. The detailed approach, the issues and the implications will be discussed. A pattern recognition task is considered to illustrate how learning mechanisms can evolve dynamically while interacting with a given environment.

6.1 The Framework

The simulations employ a self-organizing feature-map (Kohonen, 1990) as a framework to implement unsupervised connectionist learning algorithms. The Kohonen rule, belonging to the category of unsupervised learning rules, is expressed in terms of a number of concepts such as competition, co-ordination and adaptation. The feature map essentially consists of a number of cells (in a competitive layer) competing for a particular signal component from a given input signal space. The *winner* cell in the network is determined according to the minimum value of the Euclidean distance $\| \mathbf{x} - \mathbf{w}_n \|$, where \mathbf{x} and \mathbf{w}_n are the input and the reference vectors respectively. The learning rule, typically employs an external supervisor to find the winner and adapts its weights for maximal response. The result of the training is described as a process of *self-organization* that is capable of enforcing a topological ordering. Please refer to chapter five for details.

6.2 The Problem

6.2.1 The environment

The sample vectors are drawn from a two-dimensional signal space with real-valued components, taking on a value in a subspace $V \in \mathcal{R}^n$ with an unknown probability distribution. For the simulations, the sensory input stimuli are provided by a vector (\mathbf{x}, \mathbf{y}) with components distributed in a chosen subset of a square $[-1, +1]^2$. Two models as described in figure 6.1 are implemented. The first model consists of a circular ring with an inner radius 0.5 cm. and an outer radius 1.0 cm.. The second model consists of two disjoint squares from the first and the third quadrants of the square.

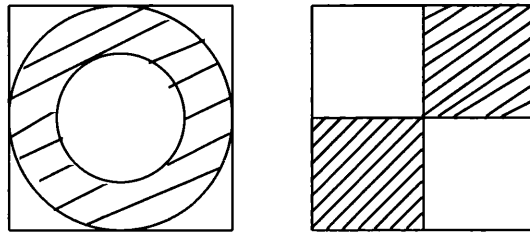


Figure 6.1: The input vectors are drawn from the dashed areas of the figures representing the two models.

6.2.2 The task

It is important to distinguish between the evolutionary task and the learning task (Hinton and Nowlan, 1987; Nolfi, Elman and Parisi, 1994; Harvey, 1996). These tasks occur over different time scales. Whereas an evolutionary task occurs from one generation to the next generation, learning is a change during the lifetime of a single individual (lifetime learning). To investigate how evolution influences learning and learning influences evolution it is necessary to identify whether the two tasks are same or different.

In the present work, the evolutionary task is to evolve effective components of learning rules (that is, the macros) and to sequence them appropriately. The learning task is to learn the correct mapping of the real-valued input vectors on to the space of reference vectors through the evolved rules. Thus, a behavior 'B' can be represented as a mapping $B: x \rightarrow W$, where, x is the real-valued input vectors and 'W' is the winner. If B^* = space of possible behaviors and w^* = space of weights associated with B^* , then the task T is a reinforcement function represented as: $T: B^* \rightarrow \mathcal{R}$, $T: w^* \rightarrow \mathcal{R}$.

In these simulations, the fitness of a learning rule depends on its performance in inducing the correct mapping that minimizes the quantization error and also in enforcing an effective topological ordering. The fitness of the evolutionary task is defined in terms of the fitness of the learning rule. A better learning rule will have a higher fitness to survive and reproduce.

6.2.3 The basic steps

The learning rule mainly consists of the following steps:

- Apply exemplars from the given input signal space for a number of epochs.
- Find the *winning* cell.

- Adapt the network weights according to the equations (5.1) and (5.2) (refer to chapter 5).

The learning rule, in essence, moves the two-dimensional reference vector towards the two-dimensional input signal so as to minimize the *quantization error*. The quantization error for a given input signal is the distance between the signal and the reference vector of the *winning* cell over a number of epochs. The training enforces a topological ordering where adjacent vectors in \mathcal{R}^n are mapped on adjacent (or identical) cells in the competitive layer. Further, adjacent cells in the layer will have similar position vectors in \mathcal{R}^n . Figures 6.2 and 6.3 (see Appendix-A) illustrate the topological ordering resulting from the standard Kohonen learning rule for the above two models.

The simulations ahead aim at evolving a Kohonen type of learning rule with the genetic programming (GP) paradigm. Whether GP is able to evolve the variety of concepts and sequence them appropriately is to be investigated.

6.3 The Genetic Programming approach

The *key* aspects of the simulation include

- providing a general definition for a connectionist learning rule as *a sequence of interacting concepts*.
- imposing a *single potential constraint* that the network weight adaptation should be an *integral* part of the representational structure, that is the genotype that the GP employs. In this context, the weight adaptation is seen as a symbolic concept, the adaptation process itself being subsymbolic.
- employing a *potential strategy* such as *micro-macro dynamics* that enables GP to realize the notion of *emergence* through its primitives. GP's primitives are the micro concepts that should enable it to form macro concepts.

Under these assumptions, GP is required to evolve the concept of a *winning* cell, induce the appropriate direction of weight adaptation for the given signal components and evolve a neighbourhood strategy such as a Gaussian to adapt this cell maximally compared with the rest of the cells in the network. Further, the concepts need to be appropriately sequenced.

The simulations initially use a network that has a fixed number of cells to investigate whether GP is capable of evolving any valid learning mechanism.

The major steps involved in preparing to apply genetic programming to a given problem (Koza, 1993) include determining the

1. set of terminals,
2. set of primitive functions,
3. fitness measure,
4. parameters for controlling the run,
5. method for designating a result and the criterion for terminating a run.

The first major step in preparing to use genetic programming is to identify the appropriate set of terminals and functions that construct computer programs that can be expressed in terms of LISP S-expressions. The search space is a space of possible programs which the genetic programming system will search. This space can become extremely large as the number of terminals increases. A rich enough set of functions and terminals will have to be chosen for the best performance (Kinnear, 1994). Further, as a meta-learning system for connectionist networks, genetic programs provide an extremely large landscape of potential concepts. Genetic programming, in this context, has an additional onus of evolving the needed concepts through its primitives and of sequencing them in ways that can yield valid learning mechanisms. Next, the fitness function that scores how well an individual performs on a given problem needs to be defined very carefully. Again, in the context of the evolution of connectionist learning rules, a valid learning rule will have a higher fitness. The population size and diversity are equally important to allow for a rich combination of possible concepts.

Two possible approaches (Govinda Char, 1997a; and 1997b) for the above mentioned task-domain have been attempted and will be described.

6.3.1 The General approach

The primitives for the GP run are:

Function set = { +, -, *, %, IFLTE, ABST, Adapt-x, Adapt-y};

The function set consists of the standard mathematical operators along with the protected division operator;

IFLTE (IF LESS THAN ELSE) is a comparison operator that can be employed by the GP to evolve the concept of a *winning* cell.

ABST returns an absolute value of an expression. GP might use this primitive to induce the direction of weight adaptation.

Adapt-x and Adapt-y adapt the network weights that are associated with the two-dimensional signal vector (\mathbf{x}, \mathbf{y}) . For instance, Adapt-x and Adapt-y might look like:

$\{ w[0][ix][iy] = \}$ and $\{ w[1][ix][iy] = \}$ respectively. GP has to induce the values to be substituted onto the right side of these expressions.

Terminal set = $\{ x, y, w[0][ix][iy], w[1][ix][iy], ix, iy, \text{delta}, \text{eps} \}$;

where x and y refer to the components of the two-dimensional signal vector (\mathbf{x}, \mathbf{y}) ; $w[0][ix][iy]$ and $w[1][ix][iy]$ represent the two-dimensional reference vector associated with each of the cells. The *location* of a cell is crucial for evolving the concept of the *winner* cell and is to be accessed via the co-ordinate variables ix and iy respectively. The parameters ‘delta’ and ‘eps’ represent the width parameter and the learning rate as stated in equations (5.1) and (5.2). No other information is provided. As the network weight adaptation is assumed to be an *integral* part of the genetic programs, GP will have to evolve the right concepts in the right sequence to adapt the weights effectively in order to minimize the quantization error.

The Fitness:

The fitness is a quality function $G(\mathbf{x}, \mathbf{y})$ given in terms of:

$$Error = \sum ABS(x - wix_{win}) + ABS(y - wiy_{win}) \quad (6.1)$$

for the winning cells over a number of epochs.

$$\text{The quantization error} = Error / (\text{number of cells}) \quad (6.2)$$

$$\text{The Fitness} = G(\mathbf{x}, \mathbf{y}) = 1 / (\text{The quantization error})^2 \quad (6.3)$$

The smaller the error the higher is the fitness of the genetic program. It is realized that although through equation (6.3) the quantization error can be minimized it does not help in topological ordering as it does not include any distance information either in terms of the weights of other cells with reference to the signal components (Euclidean distance) or the actual distance from the winner cell which is crucial for the process of self-organization. The fitness function as defined by equation (6.3) cannot be effective as such.

6.3.1.1 Sample programs

The initialization file for the GP run has to be created as a first step. This file contains the information about the various parameter settings for the GP run and is discussed in Appendix-B.

Population size: 500

Number of Generations: 50

Number of ADFs: 0

Creation Type: Variable

Maximum Depth at Creation: 6

Maximum Depth at Crossover: 17

Maximum Fitness: 1000

Number to Mutate: 0

The preliminary programs are shown.

Generation:3

((Adapt-x (x))

Fitness : 9

Structural Complexity : 2

Generation : 4

((IFLTE (Adapt-x (x) (y (wty (* (veps=0.1 (+ (Adapt-y (y) (% (% (Adapt-x (x) (- (ix (delta=0.5)) (x)))))))))))))))

Fitness : 15

Structural Complexity : 18

Generation: 6

(((+ (+ (- (wix (ABS (Adapt-x (x))) (% (wty (ix)) (- (wix (Adapt-y (y))))))

Fitness : 23

Structural Complexity : 14

Although the combination ((Adapt-x (x)) and ((Adapt-y((y)) is good for minimizing the quantization error, it does not guarantee a topological ordering. GP was not given any bias to form these combinations either. The fitness function was the only feedback that the GP had to induce this information.

6.3.1.2 The issues

- The primitives

1. When GP has *abstract* primitives such as ‘Adapt-x’ and ‘Adapt-y’ how should one decide the number of arguments for these?
2. How effective could such primitives be in adapting the network weights and in enforcing a topological ordering?
3. Should these primitives be functions or terminals?
4. Does the hierarchical tree representation that GP employs enable these primitives to be effective at all?

- The winner

The attempt is to evolve a Kohonen-type of learning rule. This rule typically employs an external supervisor to evolve the concept of the *winner*.

1. Will GP through its primitives evolve the concept of winner as such?
2. Will GP evolve the distance information that is needed to form a neighbourhood strategy for the winning and its surrounding cells, given just the above fitness function?

- The fitness measure

The definition for an effective fitness measure is crucial. Equation (6.3) does not include any distance information which is crucial for the process of self-organization. GP *on its own* will not be able to induce this information.

- The search space

What will be the effect of the random combination of all the primitives on the search space?

- The comprehensibility, interpretability and the translatability.

Finally, will the learning rules that evolve be comprehensible, interpretable and translatable? In the above experiment some of the programs managed to model the input signal distributions with a modified fitness function as defined by the equation (6.4) to some extent. This could be observed graphically. The difficulty was in comprehending and interpreting the programs. A possible explanation is that the above fitness measure may not be optimum. Kohonen's rule can be easily stated (and explained) in terms of a few statements in natural language. How could the same be achieved with the proposed approach? Also with the weight adaptations embedded within the representational structure itself (and hence within the evolutionary process) how should an expression such as:

IFLTE((Adapt-x (x((Adapt-y(y(wx (* ((wy(iy))))))))))))) be conceptualised ?

The macros adapt the network weights as and when they are invoked. The evolutionary process through its only feedback (that is the fitness) has to decide to evolve the right macros at the appropriate instances.

The above issues will be addressed before proceeding further.

The primitives:

The reason for including the primitives 'Adapt-x' and 'Adapt-y' is to enforce the network weight adaptation within the evolutionary mechanism. This constraint is imposed to make the evolutionary paradigm creative. It is indeed a *paradox* that will provide an *implicit motivation* to the evolutionary process to form and combine potential concepts. These concepts in turn need to adapt the network weights appropriately so as to enforce a topological ordering through the process of self-organization.

1. The arguments for the primitives are based on the dimensionality of the signal in these simulations.

2. The abstract primitives for network weight adaptation are not effective in enforcing the desired topological ordering as they are components of the complete tree (that is the genotype). Different subtrees might have the same abstract primitive with a totally different combination of concepts in the hierarchy below. The concepts associated with a certain subtree might adapt the network weights in ways that help the process of self-organization whereas with other subtrees they may nullify the effects and vice versa. A genetic program as a whole has only one fitness. The fitness in this case is due to effective network weight adaptation. There is no way GP can find the best program due to the lack of fitness information and as a result the approach precludes topological ordering.
3. The primitives as terminals will be totally ineffective.
4. As discussed (in point 2) the GP hierarchy does not support such primitives to be effective at all.

The winner:

1. GP might avoid the concept of the winner and opt for some local strategy that can still enforce topological ordering. This can happen despite providing all the necessary micros for evolving the concept of the *winner*. It is to be noted that this option cannot be *enforced* on GP as such. With an *explicit* fitness function (for the winner) GP should evolve the concept of the winner.
2. The primitive 'IFLTE' was supplied to evolve the concept of the *winner* cell. GP, instead, employs this primitive in other contexts. The search space could become extremely large for GP to be effective.
3. Again the notion of distance from the winner cell to other cells can be evolved only via an *explicit* fitness function.

The fitness measure:

The expression does not have enough information for GP to be effective. GP is unable to induce this information. The fitness measure needs to be cleverly defined.

The search space:

The search space can become extremely large for the evolutionary mechanism to be effective.

The comprehensibility and translatability:

With the weight adaptations as an integral part of the representational structure itself, it becomes extremely difficult to conceptualise and interpret the rules that evolve. Hence there is no question of translatability.

However, it is realized that this can be a potential approach that needs to be refined in order to appreciate the depths and the implications. The refinement in terms of modularity of concepts greatly enhances the interpretability of the learning mechanisms. Such modularity can be achieved with automatically defined functions (ADFs) (Koza, 1994) and is discussed in the next subsection.

6.3.2 The Modular approach

As the size and the complexity of the problems increase, decomposition of a problem becomes increasingly important. Problem decomposition not only enables efficient problem-solving but enhances the understandability of the process involved. In the context of genetic programming, automatically defined functions enable such problem decomposition through the definition of a number of potential functions and the hierarchy. Each of these automatically defined functions, known as the building blocks, have their own set of functions and terminals. These building blocks evolve during the run and can be used many times from any part of a computer program. Also, in hierarchical form any building block can call upon any other already-defined block. Figure 6.4 illustrates a program with ADFs.

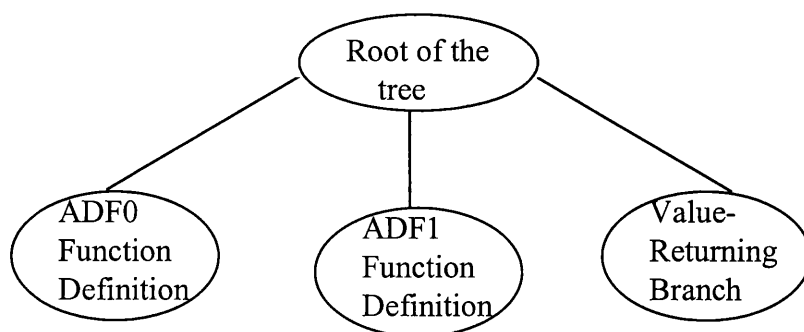


Figure 6.4: An S-expression with two function-defining branches and one value-returning branch

The S-expression with two function-defining branches and a value-returning branch is shown. GP will evolve function definition in the two function-defining branches ADF0 and ADF1 and will use one, two or none of the defined functions in the value-returning branch. Also, an S-expression with ADFs can have a number of value-returning branches. ADFs, in general, have shown to enhance the performance of the genetic programming in terms of the speed and the population size.

The role of ADFs is illustrated, as an example for an even-4 parity problem (Koza, 1993). The function and the terminal sets are:

$$F_b = \{ \text{AND, OR, NAND, NOR} \} \text{ (four functions, each taking two arguments).}$$

$$T_b = \{D0,D1,D2,D3\} \text{ (four terminals).}$$

$$A_2 = \{ \text{ARG0, ARG1} \} \text{ (two dummy variables).}$$

$$A_3 = \{ \text{ARG0, ARG1, ARG2} \} \text{ (three dummy variables).}$$

The S-expression will contain both function definitions and calls to the functions so defined. The terminals from the sets A_2 and A_3 are used to define functions of two and three arguments respectively. These arguments serve as formal parameters to the defined functions. ADF0 branch in figure 6.4 will compose functions that will include the functions from the function set and the terminals from the set A_2 . The ADF1 branch will similarly compose functions that include the functions from the function set and the terminals from the set A_3 . The value-returning branch will consist of terminals from the actual terminal set T_b and the functions from the function set F_b and ADF0, and ADF1. Over the course of the run the ADF0 branch will evolve an XOR function. The ADF1 branch will also evolve some arbitrary function. The value-returning branch will however call the ADF0 branch (that is sufficient) to solve the even-4-parity problem with the result (the value-returning branch):

(ADF0 (ADF0 D0 D2) (NAND (OR D3 D1) (NAND D1 D3))). ADF0 is actually an XOR function that evolves during the run as:

(OR (AND ARG0 ARG1) (AND (NOT ARG0) (NOT ARG1)))).

Thus the ADFs are defined once but can be instantiated as many times as needed to solve the problem in hand. Also, the defined blocks can be used to solve problems of higher complexity. The blocks that evolve are interpretable and reusable. Koza's simulations used a population size of 4000 and 51 generations over 10 runs. One of the runs yielded the correct solution in generation 3. The number of individuals that are processed is shown to be less than half (Koza, 1994) when compared to the approach without ADFs for the same problem.

6.3.2.1 Advantages of modularity in the context of learning

As a meta-learning system, GP again seems to be more powerful with the ADF approach due to the following reasons.

1. Given the general definition for a learning rule as a sequence of interacting concepts, it is possible to modularise each of the macro concepts and make them interact through ADFs. For instance, one of the ADFs can be employed to evolve the concept of the *winner* while another can adapt the network weights. GP's primitives as micro-concepts form macro-concepts that, in turn, can be represented in terms of ADFS.
2. The approach enables the tractability and interpretability of the rules that evolve. The formation of concepts and their sequencing can be interpreted easily with the GP hierarchy.
3. As an expression with ADFs can have a number of value-returning branches, each of the ADFs can be assigned an *explicit* fitness function if needed.
4. The weight adaptation can still be an integral part of the representational structure.
5. The ADFs, as terminals can evolve, unlike in the case of the general approach. With a good strategy the abstract primitives can be quite effective in network weight adaptation. For instance, the abstract primitive while being a terminal ADF to the main function can act as a function to the rest of the terminal ADFs. This strategy will allow the abstract primitive to access the global variables (returned values) from each of the terminal ADFs to form a strategy for weight adaptation. In addition, the disruption due to the effects of crossover will be minimised and can be totally eliminated as the abstract primitive is a terminal in the context of the main program. Figure 6.5 illustrates this notion.

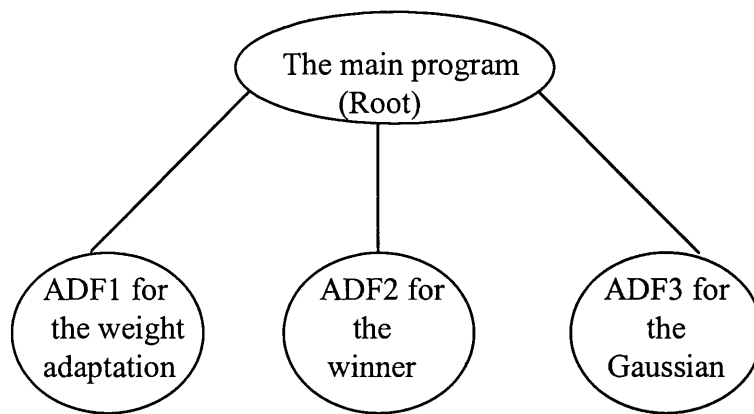


Figure 6.5: ADF1 is a terminal for the main function and can access global variables from ADF2 and ADF3.

ADF1 is an actual terminal to the main function while acting as a function (at the same time) to the rest of the terminal ADFs to form a strategy for the weight adaptation.

6. The search space becomes more focused (towards the regions of potential concepts) through the use of automatically defined functions enabling the evolution of valid learning mechanisms. This is vital if one expects to achieve a good performance in a reasonable amount of time with the evolutionary paradigm. It was discussed earlier in the general approach how GP can use the same primitives in different contexts making the search space very large and also yield learning rules that are incomprehensible.

7. Co-evolution of neural network structures along with the learning (Govinda Char 1996a; 1996c; 1997d) is essential for a variety of problem environments. Co-evolution in this particular context means the evolution of neural network architecture along with the evolution of learning mechanisms for the evolved architecture. The advantage with co-evolution is that an optimum architecture (and topology) might evolve for the task in hand. Also, the network size can be optimized and the learning for an optimized architecture is likely to be more efficient. In recent years, a number of approaches have been tried for designing the network architectures using genetic algorithms and genetic programming. These include encoding simple rules as well as a variety of complex grammars into the genotype. For instance, cellular encoding (CE, Gruau 1994) offers a context-free grammar that is compatible with the LISP S-expression and includes a variety of structure creating primitives for feed-forward and recurrent neural networks. The network creation is based on the process of morphogenesis. Genetic programming by employing such grammar might induce the type of network for particular type of inputs/signals including the temporal signals. In the latter

context, GP may have to opt for the primitives with recursive connections on the primitive elements, that is the cells. The cellular operators can be defined in terms of a terminal ADF. This strategy will allow the network structure to evolve first before proceeding to the evolution of learning at a higher level of hierarchy in the genetic program. Also, with the proposed approach applications that incorporate different types of architectures and learning at various hierarchical levels are feasible.

6.3.2.2 Sample programs

The initialization file for the GP run (see Appendix-B for details) is:

Population size: 500

Number of Generations: 50

Number of ADFs: 6

Creation Type: Ramped Half and Half

Maximum Depth at Creation: 4

Maximum Depth at Crossover: 4

Maximum Fitness: 1000

Number to Mutate: 0

The population size needs to be large enough to avoid the possibilities of missing any of the pre-defined ADFs in each generation and especially in the initial generation. It is to be noted here that each of the ADFs is predefined in terms of the different primitives. However these ADFs evolve in terms of the combination of the primitives *during* the run using the fitness information. That is, how the primitives will combine within a given ADF is not known a priori.

Initially the winner and the distance information, that is the distance of a cell from the winner, is provided in order to see whether GP is able to evolve the right adaptation strategy. The weight adaptation is through ADFs that is actually a function at a higher level of hierarchy allowing other strategies to evolve as terminal ADFs. As there is no standard measure for the process of self-organization, a maximum fitness of 1000 is assumed. This is based on the fact that the minimum quantization error that can be achieved with a given number of cells reaches a saturation point beyond which it cannot be reduced further. Only

further addition of cells can reduce the error. In other words, the network will have to grow dynamically in order to reduce the quantization error. In such cases, the parameters for the learning rule need to adapt dynamically as the structures grow (Govinda Char, 1996b; 1996e).

The details of the approach will be discussed now. The global variables are represented by glbADF1, glbADF2, glbADF3, glbADF4, glbADF5 and glbADF6. These variables return the value of the ADFs to the main program through the respective value-returning branches. Some of the ADFs can access the global variables associated with other ADFs. This enables interaction among the various ADF modules. GP has to choose the appropriate modules in the correct order for an effective self-organization. The ADF(s) for the weight adaptation, in particular, have a crucial role in terms of their number and the hierarchy. The designer has to decide the number of ADFs and their hierarchy in terms of defining them as functional or terminal ADFs whereas their hierarchy during problem-solving will be decided by the GP. The fitness criterion will be discussed first.

The Fitness criterion

The definition for the fitness measure needs to include the distance information. As GP is unable to induce this information on its own, it has to be accessed through a clever strategy. For a topological ordering, the weights associated with each of the cells need to move towards the weights of the winner. This difference can be checked and included in the fitness measure.

A few possible fitness measures and their effects will be investigated.

Consider the equation (6.4) for fitness measure.

$$\text{The Fitness} = G(\mathbf{x}, \mathbf{y}) = 1 / ((\text{The } \textit{quantization error})^2 * \text{diff}) \quad (6.4)$$

where the expression for the quantization error is the same as in the earlier case. The Euclidian distance is minimum for the *winner* cell when compared to the rest of the cells. The term ‘diff’ represents the (absolute value of the) average of the difference in the weights from that of the winner cell.

Reducing this difference enables the weight vectors to move towards the weights associated with the winning cell. GP should have the right information in terms of its fitness measure in order to induce and construct the valid components for the learning mechanism that it evolves. Further, the simulations with a fitness measure as expressed by the equation (6.3) show that the GP can totally avoid the weight adaptation phase which is crucial. One way of overcoming this problem is by forcing GP to enter this phase. This can be achieved by having a large quantization error (initially) that will reduce only if the weight adaptation phase is entered. A more natural way is to define the fitness itself in such a way that GP should naturally opt for the weight adaptation. This is possible through a fitness expression as defined by the equation (6.4). However, this expression does not always guarantee a topological ordering. The first term in the expression can become zero even in the initial generations if the signal components are assigned directly to the weight vectors as discussed earlier (see equation 6.3). Another possibility for the fitness measure is the equation (6.5).

$$\text{The Fitness} = G(\mathbf{x}, \mathbf{y}) = 1 / ((\text{The } \textit{quantization error})^2 + (\text{diff})^2) \quad (6.5)$$

The above equation ' although was effective in enforcing a topological ordering did not seem to be optimum. Moreover, the fitness definition yielded programs whose structural complexity was quite high.

Finally the following expression for the fitness measure is tried.

$$\text{The Fitness} = G(\mathbf{x}, \mathbf{y}) = 1 / ((\text{The } \textit{quantization error})^2 + (\text{diff})) \quad (6.6)$$

Equation (6.6) is found to be the optimum fitness function capable of enforcing a topological ordering with shorter programs and has been used throughout the simulations. Further, the results are consistent over several runs.

The ADFs can be combined in several ways. A few possibilities will be discussed now.

Case 1: The abstract primitive is defined as a function in ADF1. The ADFs are:

1. ADF1 (defined as a function) - contains a function 'Adapt' as one of its primitives along with the other global variables. This function takes two arguments in order to adapt the two-dimensional weight vector.
2. ADF2 and ADF3 (both defined as functions) evolve a strategy for network weight adaptation.
3. ADF4 (defined as a terminal) - evolves a strategy for the Gaussian.
4. ADF5 and ADF6 (both defined as terminals) evolve a strategy with the two-dimensional signal components and the corresponding weights.

A sample program:

Generation : 1

Best Of Generation was :

Main: ((ADF3 (ADF2 (ADF6 (ADF5) (ADF1 (ADF6 (ADF5)))))))

ADF1: ((Adapt (Adapt (Adapt (glbADF3 (glbADF5) (Adapt (glbADF4 (glbADF6))) (Adapt (glbADF2 (glbADF4))))))))

ADF2: ((* (+ (eps (eps) (* (glbADF4 (eps)))))))

ADF3: ((* (* (wiy (wiy) (+ (glbADF6 (wiy)))))))

ADF4: ((Exp ((dist))))

ADF5: ((- (ABS (wix)) (- (x (x)))))

ADF6: ((- (ABS (wiy)) (ABS (y))))

Fitness : 12

Structural Complexity : 44

For an effective self-organization, the x-components (ADF2, ADF5) need to combine with ADF4, the Gaussian function appropriately. Similarly the y-components (ADF3 and ADF6) need to combine and co-ordinate with ADF4. In this particular part of the simulation the value of 'delta', that is the width parameter is included in the variable 'dist'. It is observed that the components of the Kohonen rule are nearly evolved but they have not been combined properly. Further, it can be seen that the strategy with the 'Adapt' primitive cannot be effective at all. The 'Adapt' primitive adapts the two-dimensional weights in different ways in different parts of the same tree. The fitness information is lost. Also, it is observed that the programs become extremely large with a structural complexity in the

range of 500 and with a fitness of 30. This happens because GP has no clear direction or fitness information to guide it to be effective in the evolutionary mechanism.

Case 2: The simulation in this part employs five ADFs. The weight adaptation phase is implemented through two ADFs, ADF1 and ADF2. That is, the values returned by the global variables `glbADF1` and `glbADF2` get assigned to the x and y-directional weights respectively. The programs use a network of sixteen neurons (arranged on a 4*4 grid). The simulations are carried out to investigate the effects of various parameters on the fitness of the genetic programs and include the following graphs.

1. The number of epochs vs. the fitness.
2. The population size vs. the fitness.
3. The population diversity vs. the fitness.
4. The variation of the depth parameter vs. the fitness.
5. The number of epochs and the network size vs. the fitness evaluation time, that is, the time complexity.

The initialization file for the GP run (see Appendix-B for details) is shown.

Population Size : 500

Number of Generations: 50

Number of ADFs : 5

Creation Type: Ramped Half and Half

Maximum Depth at creation : 4

Maximum Depth at Crossover: 4

Maximum Fitness : 1000

Number to Mutate : 0

The population sizes in the initialization file were 300, 500, and 1000 respectively for various simulations. Please refer to Appendix-C for the graphs. The graphs are for:

1. The number of epochs vs. the fitness.

Refer to figure. 1 (Appendix-C). The simulations were run for varying epochs. The results suggest that for a network the fitness improves with epochs. The effects of varying the epochs from 500 to 2000 are shown. This suggest that the number of epochs is *key* to achieve

good solutions. A larger number of epochs enables better sampling of the signal space from which the components are drawn at random. Also, the weight adaptation is much more effective in minimizing the quantization error and improving the fitness.

2. The population size vs. the fitness.

Refer to figures. 2a and 2b (Appendix-C) Four sets of experiments were conducted. The simulations were run for population sizes of 300 and 500 (figure 2a) for 1500 epochs and for population sizes of 500 and 1000 (figure 2b) for 1000 epochs. The results were averaged over ten runs. The graphs suggest that a larger population size will result in a better fitness measure.

3. The population diversity vs. the fitness.

Refer to figures. 3a and 3b (Appendix-C). Four sets of experiments were conducted. The population was created with the variable and ramped half and half methods to investigate the effects of diversity on the fitness measure. For Figure. 3a the simulations used a population size of 500 and the number of epochs being 1500. For figure. 3b the population size was 1000 and the number of epochs were 1000. The results, averaged over ten runs, suggest that with the ADF approach the population diversity does seem to have a marginal effect on the fitness measure. It is likely that this effect is observable to a larger scale with the general approach where the depth parameter can be varied to a greater range.

4. The variation of the depth parameter vs. the fitness.

Refer to figures. 4a and 4b (Appendix-C). Two different sets of experiments were run for a population size of 500 for 1500 epochs and a population of 1000 for 1000 epochs respectively. Starting with an initial depth of 3 the depth at crossover was increased by 1 to investigate the effects of the depth parameter on the fitness measure. The results suggest that with the ADF approach an optimum depth of $3(\text{creation})/4(\text{crossover})$ and $3(\text{creation})/5(\text{crossover})$ gives the best results. Increasing the depth further decreases the fitness and have an adverse effect on the fitness measure.

5. The number of epochs and the network size vs. the fitness evaluation time, that is, the

time complexity. Refer to figure. 5 (Appendix-C) shows the amount of time (in minutes) required for evaluating a single individual for varying epochs. The same network, that is a network with sixteen neurons was used. The population size chosen was 500. The results although look almost linear need not be so if the depth at creation and at crossover are different. Also the evaluation time will increase considerably if the network size is increased.

The sample programs:

Generation 0

Initial random population

Average Fitness : 33.914

Best of Generation was :

Main: ((ADF1 (ADF1 (ADF2 (ADF3 (ADF3) (ADF1 (ADF3 (ADF3)) (ADF2 (ADF2 (ADF5 (ADF3) (ADF2 (ADF5 (ADF5))))))))))))

ADF1: ((* (+ (* (eps (glbADF3) (* (glbADF3 (wix))) (+ (* (eps (glbADF3) (+ (eps (glbADF3)))))))))))))

ADF2: ((* (+ (+ (eps (glbADF3) (+ (glbADF5 (glbADF3)))) (* (+ (wiy (eps) (+ (glbADF3 (eps))))))))))))

ADF3: ((Div (Div (Exp(Delta)) (Div (Dist(Delta)))) (Exp (Div (Dist(Delta))))))))

ADF4: ((ABS (ABS (- (wix (wix)))))))

ADF5: ((- (ABS (ABS (wiy))) (ABS (- (y (wiy)))))))

Fitness : 480

Structural Complexity : 69

The program illustrates how the primitives are combining to form the ADFs. The combinations seem to be quite effective. ADF4 is not seen in the main program. However, in this particular case the role of ADF4 as such is not clear. Also, ADF3 suggests that the GP is trying to evolve a Gaussian sort of function involving the distance and the width parameters.

Generation : 50

Average Fitness : 937.72

Best of Generation was :

Main: ((ADF1 (ADF3 (ADF1 (ADF2 (ADF3 (ADF5) (ADF4)))))))

ADF1: ((+ (+ (+ (+ (glbADF4 (glbADF3)) (+ (glbADF3 (glbADF3))) (+ (eps (glbADF4)) (+ (glbADF3 (glbADF3)))))))))

ADF2: ((+ (+ (glbADF3 (glbADF3)) (+ (glbADF5 (glbADF3))))))

ADF3: (Exp(Div (Div (Dist (Delta)))))

ADF4: ((ABS (ABS (wix)))))

ADF5: ((ABS (- (y (wiy)))))

Fitness : 959

Structural Complexity : 41

The program has been successful in evolving all the ADFs and sequencing them appropriately. GP opts for the weight adaptation phase naturally and succeeds in enforcing a topological ordering eventually.

It is to be noted that the possibility of multiple weight adaptation (that is, adapting a weight more than once in a single iteration) exists. It depends on the modules that evolve. GP can eventually enforce topological ordering either by employing multiple adaptations or it may, over the course of evolution, avoid the multiple instances of weight adaptation over a single iteration.

In the above simulations the information on the winner cell and the distance parameter were provided. In actual practice, these co-evolve. These can be evolved as follows.

1. The winner

The winner can be evolved with the following primitives.

The function set: { IFLTE, ABS, +, - };

The terminal set: { x, y, w[0][ix][iy], w[1][ix][iy], temp };

and with an explicit fitness function that is a minimum of equation (6.1). A sample program that uses two ADFs, ADF1 and ADF2 for evolving the concept of winner is shown.

The function and the terminal sets for ADF1 are:

Function set = { IFLTE}. //A single function taking four arguments.

Terminal set = { mismatch, minimum, minimum=mismatch, glbADF2}.

The variables:

mismatch = ABS(x-w[0][p][q]) + ABS (y-w[1][p][q]). (a)

minimum = 1e10; //A variable (to that holds the value of the winner) has a large value initially that will get replaced based on the comparison (with expression (a) for other cells) during the run.

minimum=mismatch is the replacement operator.

glbADF2 is the value returned from ADF2.

The function 'IFLTE' will compare the expressions for different cells and find the minimum value of the expression (a) that is a winner.

ADF2 is used basically to create the expression (a) itself and uses a kind of symbolic regression till the output of ADF2 matches that of expression (a).

The function and the terminal sets for ADF2 are:

Function set = { ADD, SUB, ABS} taking 2, 2, 1 arguments respectively.

Terminal set = { x,y, w[0][p][q], w[1][p][q]}.

The initialization file is:

Population Size : 300

Number Of Generations : 50

Number Of ADFs : 2

Creation Type : Ramped Half and Half

Maximum Depth at creation : 3

Maximum Depth at Crossover: 4

Maximum Fitness : 1000

Number To Mutate : 0

The depth parameter is held small. The 'IFLTE' primitive taking four arguments can make the programs extremely large otherwise.

Sample programs:

Generation 2:

Best of Generation was :

Main: ((ADF1 (ADF1 (ADF1 (ADF2)))))

ADF1: ((IF(minimum = mismatch (IF(minimum=mismatch (mismatch (IF (minimum= mismatch (minimum= mismatch (glbADF2 (glbADF2) (minimum) (IF (minimum (glbADF2 (IF (glbADF2 (glbADF2 (min (glbADF2) (IF (mismatch (glbADF2 (minimum= mismatch (mismatch)) (glbADF2)))

ADF2: ((- (+ (ABS (wix) (ABS (y))) (- (wiy (x))))

Fitness : 300

Structural Complexity : 38

Generation : 21

Best Of Generation was :

Main: ((ADF1 (ADF2)))

ADF1: ((minimum))

ADF2: ((- (+ (ABS (wix) (ABS (y))) (- (wiy (x))))

Fitness: 786

Structural Complexity : 12

The final expression for the fitness measure that is equation (6.6) should incorporate the fitness information for the winner either implicitly or explicitly.

2. The distance parameter

The function set: { SQR, +,-};

The terminal set: { ix, ixmin, iy, iymin};

where the terms ix, iy, ixmin, iymin, represent the location (the co-ordinates) of any cell and that of the winner cell. It should be noted that 'ixmin' and 'iymin' are also in the

process of evolution. Again, an explicit fitness function will have to be defined for evolving the correct distance parameter and included in equation (6.6).

6.3.2.3 Co-evolution of structure

A sample program is shown to demonstrate the evolution of structure along with learning. The program uses cellular operators (Gruau, 1994). Two basic operators tried are the 'PAR' and the 'SEQ' (refer chapter seven) for creating cells (neurons). These operators are defined as primitives in an ADF, that is ADF6. The simulations start with a single cell and grow cells dynamically using the operators during the run. It is essential that the width parameter has to be adapted accordingly. The task domain is the same as discussed earlier.

Generation : 1

Average Fitness : 21.33

Best Of Generation was :

Main: ((ADF2 (ADF1 (ADF3 (ADF4) (ADF1 (ADF3 (ADF6)))))))

ADF1: ((+ (+ (glbADF4 (+ (glbADF4 (wix))) (+ (wix (glbADF4)))))))

ADF2: ((+ (+ (glbADF3 (eps)) (* (glbADF5 (glbADF3))))))

ADF3: ((Exp (Exp (dist))))

ADF4: ((- (ABS (x)) (ABS (wix))))

ADF5: ((- (- (y (wiy)) (ABS (wiy)))))

ADF6: ((Seq2 (Par1 (END))))

Fitness : 30

Structural Complexity : 40

The simulation program for the meta-learning system has been developed in the C++ programming language with object oriented programming techniques. A steady state GP employing a tournament selection scheme is used. A tournament selection randomly selects a number of genetic programs from the population. The fitness value of each member of this group are compared with each other and the best replaces the worst. The tournament size is set to five. Adam Fraser's (Fraser, 1993) kernel is taken as a base on which the meta-learning kernel has been designed and implemented. The programs use a Windows 95 environment. A Pentium 233/300 has been found to be too slow for the

fitness evaluation even for a medium sized network. The reason being that each fitness evaluation requires the network weights to be adapted over a large number of epochs for a good performance (please refer to Appendix-C, figure. 1). For larger networks the fitness evaluations could be extremely time-consuming (as the number of neurons and the associated weights will also increase). However, it is to be noted that the learning rule that evolves for a smaller network has to be applicable for larger networks as well. A good strategy is to start with smaller networks (with an optimum number of cells) that can enforce a good topological ordering and apply the evolved rules to larger networks to achieve the desired performance.

The indications are that a parallel processing environment will be highly desirable to achieve the required performance with larger (optimum sized) networks. The fitness evaluation time can be considerably reduced based on the number of processors.

The graphics interface has to be dynamic to enable observation of the weight changes and their effects on the input/output mapping. The interesting point about this simulation is that the effects of various fitness measures can be observed graphically. Also it is impressive to observe GP opting for the Gaussian (the distance function) and its effects on the process of self-organization. Reducing the number of ADFs again leads to the problem of incomprehensibility of the evolved rules.

6.4 A comparison: GA vs. GP

Genetic algorithm, as discussed in chapter four, was employed to evolve learning rules for a feed-forward type of neural network capable of dealing with a different class of problems. The present work has mainly focused on unsupervised learning that are applicable to an entirely different class of problem domain. However, some comparison between the two approaches can be made in terms of a number of factors such as:

- **The topology and the node activation function**

With the GA implementation the network topology and the node activation function are specified *a priori*. The GP implementation is flexible. Although these simulations assumed

a fixed structure variable topology can be incorporated through a process of morphogenesis. The node activation functions might be allowed to evolve with the GP approach. Hence the networks can be non-homogenous where each cell can employ a different node activation function. Although a possible approach is suggested (in the co-evolution part) the simulations in the current work do not attempt these. It is likely that the GA approach also might allow for such a flexibility. It is not known whether any such work has been done so far.

- **The levels of adaptation**

The two levels of adaptation are distinct in the GA implementation. The learning rules evolve and *subsequently* adapt the network weights whereas with the GP, the learning level is embedded within the evolutionary level. That is, the weight adaptation is a part of the representational structure itself and hence an integral part of the evolutionary process. The key difference between the GA and the GP approaches is that in the case of GP the macro concepts including the concept of adaptation evolve *while* interacting with the given environment. The implications of the GP approach are profound. *If* the linear chromosome in the GA is also able to encode the concept of network weight adaptation the two approaches might then have some similarity and possibly the same implications.

- **The difference**

With GA, a learning rule is applied to different tasks to assess the fitness of the learning rule. A number of different networks with the assumed topology are set up to test the evolved rule for its fitness (that is, the fitness of the learning rule) on a number of learnable tasks. With the GP approach a variety of learning rules in terms of the node activation function and structure can evolve for a given task, although this has not been attempted as a part of the current work. As discussed earlier these might be possible with the GA approach also. It seems further work needs to be done in this direction as well.

The simulations with GP are based on a general definition for a connectionist learning rule as a sequence of interacting concepts. Encoding and decoding the genotype seems to be easier with the GP approach. The size and the complexity of the genotype is flexible. The learning mechanism can be evolved in terms of a number of potential modules and can be interpreted easily. The purpose of the experiments was to investigate whether the notions

of constructivism and micro-macro dynamics could be extended to the evolution of valid connectionist learning mechanisms. The simulation results have demonstrated that flexible learning mechanisms can be evolved with a general definition for learning and with a single potential constraint imposed within the representation that the GP employs.

6.5 Discussions

The attempts were aimed at evolving a Kohonen type of learning rule and to observe whether a topological ordering can emerge with the evolved learning rule. The simulations suggest that GP, as a meta-learning paradigm can be a potential tool. A number of issues were identified and addressed. The modular approach seems to be more powerful for the reasons discussed. To summarise, the proposed approach has the following advantages. The approach

1. suggests a way of naturally combining connectionist networks with the evolutionary paradigm.
2. by providing a general definition for a learning rule as a sequence of interacting concepts and by imposing a single potential constraint within the genotype is successful in implementing flexible learning rules for a self-organizing neural network. The constraint creates a paradox for the evolutionary paradigm to be creative. Further, the learning rules evolve while interacting with a given task environment.
3. allows for flexibility in terms of modularity and the rules are easily interpretable (and hence translatable) through the ADF modules.
4. suggests the possibility of co-evolution of structures and learning.

It would be interesting to investigate whether the proposed approach can be extended to feed-forward networks that employ a supervised learning rule. The fitness function definition should be easier as the target solution will be known a priori for a supervised learning rule. Also, learning for recurrent networks can be attempted as a part of the future work.

The representation that GP employs suggests that a hierarchical learning (different types at different levels) is feasible.

Whether the method allows designing non-homogenous networks, that is networks where different cells/neurons in the same network having different node activation functions can also be investigated further.

Recent work in GP has shown that most interesting problems need a sort of internal memory (Teller and Andre, 1995). The incorporation of memory into GP have shown performance improvements. The proposed method seems to be a natural way of incorporating memory into the GP paradigm.

The learning rules can evolve while interacting with a task environment. The system, nevertheless cannot be defined as purely reactive as it incorporates a network structure with adaptive weights forming some sort of memory and a representational structure. On the other hand it can safely be termed as an eclectic hybrid.

(See references, Govinda Char, 1996 a, b, c also in the context of evolution of learning).

Conclusion

The proposed approach, by providing a very general definition for a connectionist learning rule and imposing a single potential constraint offers a novel way to evolve flexible learning rules for a self-organizing neural network. The simulations demonstrated how such rules can be evolved while interacting with a given environment through the powerful notions of constructivism and micro-macro dynamics. Genetic Programming seems to be an excellent tool as a meta-learning system as it offers a natural way of combining connectionist networks, employing both bottom-up and top-down strategies.

The role of such flexible learning mechanisms in constructivist AI systems will be discussed in the next chapter.

Chapter 7

Constructivist AI with Genetic Programming

Constructivist AI conceptualizes intelligence as an adaptive behavior that can be constructed through primitive elements and processes. These co-ordinate effectively to achieve a global behavior, typically employing a bottom-up strategy. This chapter briefly describes a recent modelling method for constructing adaptive systems and explains how such systems can be evolved with the genetic programming technique by extending the notion of constructivism to the evolution of neural structures and learning. A comparison between the two approaches is drawn highlighting the merits of the latter approach.

7.1 The Background

Traditional Artificial Intelligence (AI) understands intelligence to be explicitly definable. Thus AI is seen as a combination of knowledge in symbolic form and techniques that can manipulate this knowledge. This view of AI, as discussed in earlier chapters, has led to a number of limitations in terms of its range of applicability. The new AI approaches, typically, view intelligence in terms of emergence. In this context, intelligence can be conceptualized as an adaptive behavior and can be constructed from primitive elements and processes that involve interactions with the environment. These primitive processes may be in the form of a set of rules that evolve structures and/or concepts. For instance, the rules may specify a sequence of operations such as cell division, differentiation, interactions among the cells to create a structure, or the interaction of the cell/the structure with the environment. Alternatively the rules may simply represent a sequence of macro-concepts that evolve to solve a given problem. Each of these macro-concepts in turn might be in terms of a combination of potential micro-concepts. The final complexity of the systems that evolve is unpredictable. Such models are known as *constructivist* AI systems (Vaario, 1994a) and possess a number of characteristic properties seen in ALife-like (Langton, 1993) paradigms. Vaario in his recent work (Vaario, 1994b; 1994c) has demonstrated how such adaptive systems can evolve with a *constructivist* approach and argues that intelligence cannot be taken as a describable fact but is a result of gradual evolutionary and developmental processes. The approach is briefly discussed. Figure 7.1 shows the life cycle

of an adaptive system, typically a nervous system illustrating the four forms of adaptation: Development, Neural Plasticity, Natural Selection and Genetic Changes.

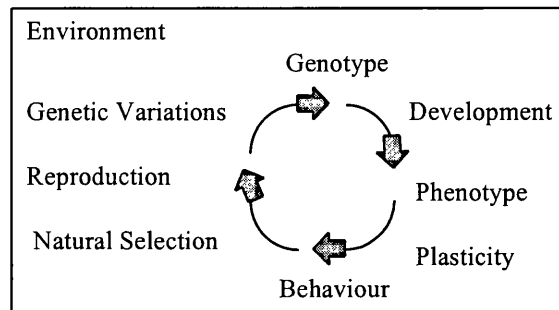


Figure 7.1: The life cycle of an adaptive system

The development of the neural network within the adaptive system and the information processing mechanism of the mature network are not separated from each other. The development process is important as:

- explicit design of complex systems is difficult suggesting a need for an ‘intrinsic design’ method to create complex systems such as neural networks.
- the development by itself is a form of adaptation filling the gap between the fast synaptic plasticity adaptation and the slow genetic-based adaptation.
- genetic code requires the development process to describe a complex structure.
- development implements the anatomical plasticity that can implement long-term memory.

The method employs the idea of emergent behavior where the complexity is reached without any global definition but using several local behavior rules that together reveal the global behavior. The neural network is grown towards a mature state gradually through a set of production rules. The fitness of the system as a whole is not just a function but the survival capability defined by a collection of selection processes which are functions of the current environment. The evolution is thus open-ended. With these four forms of adaptation Vaario has demonstrated how *emergent* phenomena can be realized based on atomic interactions. Figure 7.2 illustrates the hierarchical representation of computational levels. The computational model is based on a set of production rules inspired by the Lindenmayer system (Lindenmayer, 1970). The rules however are not a string of letters. Instead a set of

abstract objects which have their own production rules are defined. Each of these objects can have sub-objects within them executing a different set of production rules.

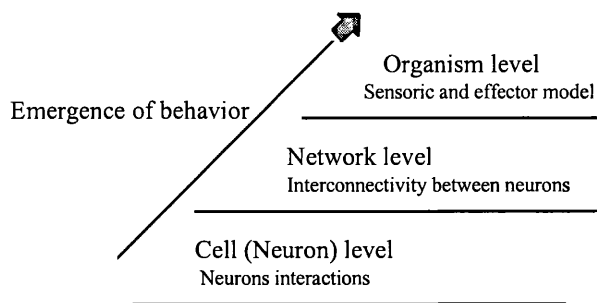


Figure 7.2: The hierarchical representation of computational levels.

The objects and their interactions can be seen at three different levels. The basic cell level, the neural network level and finally the organism level. A set of production rules describes at each level the local interactions between objects at the upper, lower or the same level. The production rule consists of a conditional part that switches the production rule on and off. The result of evaluation of a production rule can be the creation of a new object or the modification of its own attributes (a tuple of a key and a value). The rules are also expressed as attributes enabling them to change themselves. Eventually the behavior emerges. The key aspect of this approach was to model environmental adaptation using a Multilevel Interaction Simulation language (MLIS) which simulates the interactions at different organizational levels. The MLIS is different from traditional object oriented systems in the following aspects. In traditional systems an object is passive unless it receives a message for an action implying that the control mechanism for messages needs to be synchronized in the sense that the sender must know to whom and when to send the messages. In MLIS language the sent messages are broadcast to the environment where each object can check and act accordingly. There is no central control as each object is autonomously executing its own set of instructions in parallel with the rest of the objects. From these interactions the organism and the plasticity resulting in the behavior emerge. As an example assuming two organisms in an environment the production rule looks like:

Environment(Constraints,.....

 Organism1(Genetic Code, Neurons (attributes))

 Organism2(Genetic Code, Neurons (attributes)))

Constraints - define the environment depending on the production rules that are common for each organism.

Genetic Code- defines the organism depending on the production rules for cell divisions, axon-dendrite growth, *etc.*

Neurons- defines the initial cell for the growth process.

The execution of the Genetic Code production rule is done without an explicit definition for it. Eventually the desired behavior emerges through the interactions of the productions at various levels.

In the above hierarchy some of the cells can act as sensors, the others as effectors and the rest as the neurons in between the two layers. The sensors and effectors also adapt to the environment. The final structure not only is genetically predetermined but can be affected by the environment suggesting that the approach allows for the intelligent adaptation of the organism to its environment.

7.2 The GP approach

It is argued that evolutionary algorithms, despite being powerful search methods lack the creativity to evolve ALife-like systems (Vaario, 94c). The reasons seem to lie basically in the type of problem environment, in the approaches that are employed, and mainly due to the failure in imposing proper constraints within the evolutionary paradigms. In the preceding chapter, through simulations, it has been demonstrated how flexible connectionist learning mechanisms can evolve just by:

- providing a very general definition for learning as *a sequence of interacting concepts* and
- through the imposition of a single potential constraint that the neural network weight adaptation should be an integral part of the hierarchical tree representation that the GP employs.

The constraint creates a paradox for the evolutionary algorithm to evolve potential concepts capable of tackling the task environment. This suggest that a right constraint can

make evolutionary algorithms extremely creative. The type of constraints and their effects can be established by experimentation.

It was also illustrated through the simulations that the notion of constructivism could be easily extended to the genetic programming for the evolution of building blocks representing the components of connectionist learning rule(s). It is to be seen whether flexible AI systems can be constructed and realized through the proposed approach.

The four forms of adaptation shown in figure 7.1 can easily be implemented with genetic programming as well. The aim is to realize the notion of *emergence* in terms of the neural network structure and also the type of learning that evolve for a given task environment. These will be explained.

1. The development stage of the neural network can be implemented either through simple rules or through a process for morphogenesis such as cellular encoding (CE) (Gruau, 1994). The latter case, being a grammar-based encoding the cellular operators for cell division can be easily defined and included as the GP primitives. The representational structure in CE is compatible with that of GP. The neural network structures can emerge while in constant interaction with a given environment. GP will have to induce the network architecture for a given signal/input space by choosing the appropriate primitives and construct the network architecture. Three types of primitives for cell/neuron growth and the connectivity patterns are shown in figure 7.3.

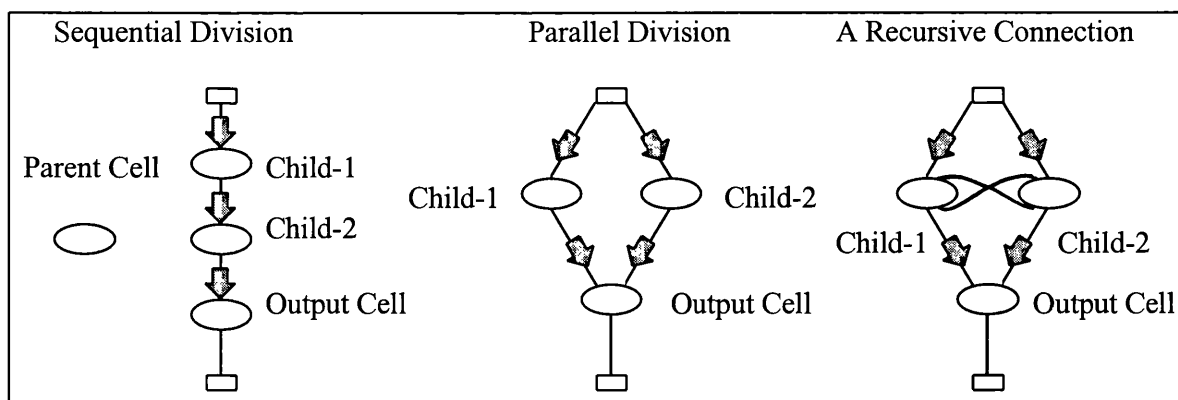


Figure 7.3: Sequential and Parallel Division of cells and two cells recursively connected

Cellular encoding basically employs three types of primitives. These are the SEQ, PAR and the REC primitives. SEQ divides a parent cell into two cells and connects them in sequence.

PAR divides the cells in parallel and inherits parallel links from the parent cell. The REC primitive provides a recursive connection between the two child cells, each with two output links. The first two figures show a feed-forward type of connection whereas the last figure illustrates a recursive connection. Based on the signals/ inputs GP will have to induce the right primitive to create the appropriate network structure for the task environment.

2. The simulations in the preceding chapter demonstrated how a network and learning can evolve while interacting with the task environment. The genetic programming system provides a paradigm to realize the notion of emergence at the levels of network creation and also the learning. The phenotypic characteristics emerge indirectly as a result of the genetic information (for the development) and also due to learning. The key point to note is that learning can evolve during the development.

3. The recombination of the genetic programs in terms of the swapping of sub-trees will yield the required genetic changes as a natural part of the evolutionary cycle.

4. The evolution itself will be at the highest level of adaptation incorporating natural selection.

7.3 A comparison between the two approaches

Models 1 and 2 in figure. 7.4 represent two different approaches to constructivist AI, and compare Vaario's and the GP approaches.

Characteristics	Model-1	Model-2 (GP)
Structure	Artificial neural network	Artificial neural network
Creation	Production Rules-based	Grammar based encoding
Learning	Production Rules-based	Evolve network learning
Type of Learning	Dynamic with the environment	Dynamic with the environment
Evolution	Open-ended	Based on a fitness value
Simulation language	MLIS	GP and ADFs (in LISP form)
Emergence	the structure as an organism	the structure and learning
Constructivism	mainly applied to structures	applied to structure/ learning

Figure 7.4: A comparison between the two computational models

7.4 Discussions

Model-1 has been successfully employed in implementing potential constructive AI systems (Vaario, 1993). It is observed from figure 7.4 that the four forms of adaptation can be easily incorporated within the genetic programming system. In addition, emergence is realized at the levels of both the network structure and learning. The final complexity of the system(s) that evolve is unpredictable. It is to be noted that the evolution in the case of model-2 is not open-ended but is bound by the fitness criteria that the user specifies. However, the proposed approach, by providing a general definition for learning and by imposing a single potential constraint within the representational structure provides an extremely large space (possible combinations) in terms of the type of network architecture, the node activation function and the type of learning that can be implemented. The advantage when compared with the reactive systems is that it can incorporate the development phase enabling the system to grow autonomously based on the need.

Model-2 with genetic programming has a potential for implementing constructivist AI systems that can be effectively employed in unpredictable/unknown situations.

Further, it has been argued that evolutionary paradigms are good at optimization but not creative enough to realize Alife-like systems. The reasons for these limitations need to be addressed. It basically seems to lie in the way evolutionary algorithms are employed, that is, in using these algorithms for optimization only rather than for construction and optimization.

It is to be realized that evolutionary algorithms could be extremely creative if appropriately combined with other paradigms such as connectionist networks.

The simulations employed a self-organizing neural network as a framework to attempt the evolution of valid learning rules in a dynamic environment. However, the goal should be to realize potential self-organizing systems that can adapt to the environment through individual and evolutionary adaptations. It should be possible to realize autonomous, self-organizing systems in terms of eclectic hybrids with proper constraints imposed within the hybrid in some form.

Conclusion

The aim was to investigate whether the proposed approach has the potential to realize constructivist AI systems in the form of an eclectic hybrid. Simulation results with genetic programming suggest that the notion of constructivism are easily extended to the evolution of neural network structure and learning. The advantage with this approach is that the learning rules will replace the production rules used in the first model. Further, the four forms of adaptation shown in the life cycle of an adaptive system are incorporated naturally within genetic programming. These suggest that GP, if used for construction and optimization, will yield powerful adaptive AI systems through hybrids.

The next chapter provides the concluding remarks and suggests some applications for the proposed approach.

Chapter 8

Summary, Conclusions and Directions for Further Research

This chapter provides a summary of the research with conclusions, suggests a few potential application domains and indicates directions for further research.

8.1 Summary and Conclusions

The research described in this dissertation has three main objectives.

First, to understand traditional knowledge-based systems and their limitations. The role of learning in problem solving and the need to investigate various representations and strategies are discussed in chapter one. In particular, the focus is on evolutionary algorithms that employ task-independent representations and operators to solve a wide range of problems in flexible ways. Constructivism, a powerful notion is introduced. It has been argued that AI, if seen as an adaptive behavior can be constructed through the interaction of primitive elements and processes. How an evolutionary algorithm such as genetic programming (GP) can be used to extend this notion to construct flexible learning is briefly discussed. Chapters two and three provide a background information on AI, its new perspective and on evolutionary computation.

Second, to investigate the role of genetic programming in connectionism. That is, how effective is genetic programming as a meta-learning tool in evolving connectionist network architectures and learning rules. A self-organizing neural network is chosen as a framework to focus on various aspects of network architecture and learning. The approach firstly involves identifying key issues in connectionism that have led to its limitations. How evolutionary algorithms offer a way to overcome these limitations is discussed in detail. In genetic-based design the encoding strategy is crucial. In the context of network induction, the strategy should not only capture useful architectures that are optimum for solving a problem in hand but allow for further generalisation. In the context of learning it should be able to discover a variety of potential learning mechanisms for the given task environment. The proposed research by focusing on earlier approaches and also on most

of the recent work in genetic-based design has systematically raised a number of key questions such as:

1. Is the representation (genotype) that the algorithms employ sufficient to express and explore the vast space of network architectures and learning mechanisms?
2. Is the representation capable of capturing the concepts of hierarchy and modularity that are vital and naturally employed by humans in problem solving?
3. Are some representations better in expressing these? If so, how to exploit the strengths that are inherent to these representations?
4. If the aim is really to automate the design process what strategies should be employed so that the involvement of a human in the design loop is minimum?
5. Is the methodology or the approach able to overcome at least some of the limitations of connectionist networks?
6. Most importantly, how effective is the approach in solving problems?

Chapter four has attempted to address these through detailed discussions and also through comparisons to most of the recent work with genetic algorithms and genetic programming. The merits of the novel approach that is proposed in this thesis are identified in terms of the representation and the strategy employed. The importance of modularity and hierarchy in problem solving and the need for potential strategies to exploit these are stressed. How genetic programming offers these through the representation and automatically defined functions is demonstrated through simulations in chapter six.

The experimental results demonstrate that genetic programming, if used for construction and optimization could be extremely creative in implementing potential learning mechanisms and also network architectures for self-organizing neural networks. Further, the proposed method combines the bottom-up and top-down strategies through the powerful notions of constructivism and micro-macro dynamics. The network architecture and the learning evolve while interacting with the task environment. As the approach involves a general definition for learning (irrespective of the type of network) and a single potential constraint *within* the representation (that is the genotype), it appears that it could be extended to other

types of networks as well. These suggest that connectionism has to be approached with a new perspective to realize its true potentialities.

Third, the aim is to identify the role of flexible learning in implementing adaptive AI systems. Chapter seven has discussed a recent model of a constructivist AI system that incorporates four forms of adaptation such as development, neural plasticity, natural selection and genetic changes. The novel method that is proposed in this thesis, can easily incorporate these within GP to build adaptive AI systems. A comparison is drawn between the two approaches.

The next subsection will discuss few applications with the GP hybrid.

8.2 Possible applications with the proposed approach

This concluding section suggests few applications. These include:

1. An extension to GP

Genetic programming is not Turing complete. That is, GP is not powerful enough to recognise all possible algorithms. The reason is attributed to the fact that GP has no inherent mechanism to implement a *state* or an internal memory. A number of interesting problems require a memory to be solved effectively. Indexed memory (Teller, 1994; Andre, 1994) is a simple way of implementing memory by adding few non-terminals such as ‘Read’ and ‘Write’ to GP. Adding these non-terminals result in a system that is Turing complete. Using indexed memory, a GP function can save past inputs and then use them appropriately as needed to tackle a given problem. It is argued that indexed memory helps GP in solving memory-critical problems.

The proposed approach might be extended to implement internal memory as it incorporates neural networks within the GP paradigm.

2. Self-organizing systems

The simulations attempted evolution of learning rules for self-organizing neural networks. The goal should be to evolve self-organizing systems for information processing and for

real-time applications. It should be possible to extend the notions of constructivism and micro-macro dynamics to the evolution of building blocks in terms of structure and learning that can self-organize at the systems level. The co-ordination of the building blocks based on various fitness criteria can be investigated.

3. Robotics

The approach can also be applied to the design of autonomous mobile robots that adapt to a given environment through automatic learning mechanisms (Zimmer and Puttkamer, 1994; Vaario, 1994; Balakrishna and Honavar, 1997). Robots, typically employ neural networks for learning. Networks that grow (or shrink) dynamically based on the environment have been shown to be more suitable than those with fixed architectures. The proposed approach allows for network development through a process for morphogenesis. It is possible to design autonomous system(s) where a few of the network modules can be made to act as sensors and some as effectors with the rest of the modules implementing the learning. The effects of modularity and hierarchy in network creation and learning can be investigated for different task-environments. Also, the learning in dynamic environments might be attempted as a part of the future work.

4. Adaptive pattern recognition

Pattern recognition (PR) is one of the most important components of an intelligent system. The traditional methods in pattern recognition have been inadequate to provide optimal solutions to a number of complex pattern recognition and classification tasks. Evolutionary algorithms, being powerful search and optimization methods, have been more successful in tackling complex problems (Tackett, 1994). PR, typically employs a number of techniques that use a variety of representations (Govinda Char and Tackett, 1996). The proposed approach provides a potential hybrid for adaptive pattern recognition and classification tasks. Initially the hybrid can be tried with difficult bench-mark problems.

5. Modular and hierarchical learning

Distributed artificial neural networks have found applications in natural language processing (NLP) (Miikkulainen 1991; Elman, 1993). The architecture (Miikkulainen, 1991) employs hierarchically-organized back-propagation modules (for processing words) communicating

through a central lexicon of word representations which is implemented as a feature map. The proposed method offers modularity in terms of network architectures and learning and has a potential in implementing NLP systems. The hierarchy is inherent to the representation that GP employs.

6. Reinforcement learning and learning for recurrent networks

A variant of supervised learning is the reinforcement learning that has been extensively used in many potential applications. Rather than giving the network the entire correct output some measure of how well the system is doing is presented. Recurrent networks, on the other hand model dynamical systems and employ a number of algorithms such as the recurrent back-propagation (Pineda, 1989) and back-propagation through time (Werbos, 1990). Evolution of architecture(s) and learning mechanism(s) for the above types of networks could be attempted in the future with the proposed method. GP, based on the environment and the fitness criteria might induce appropriate network architecture(s) and learning.

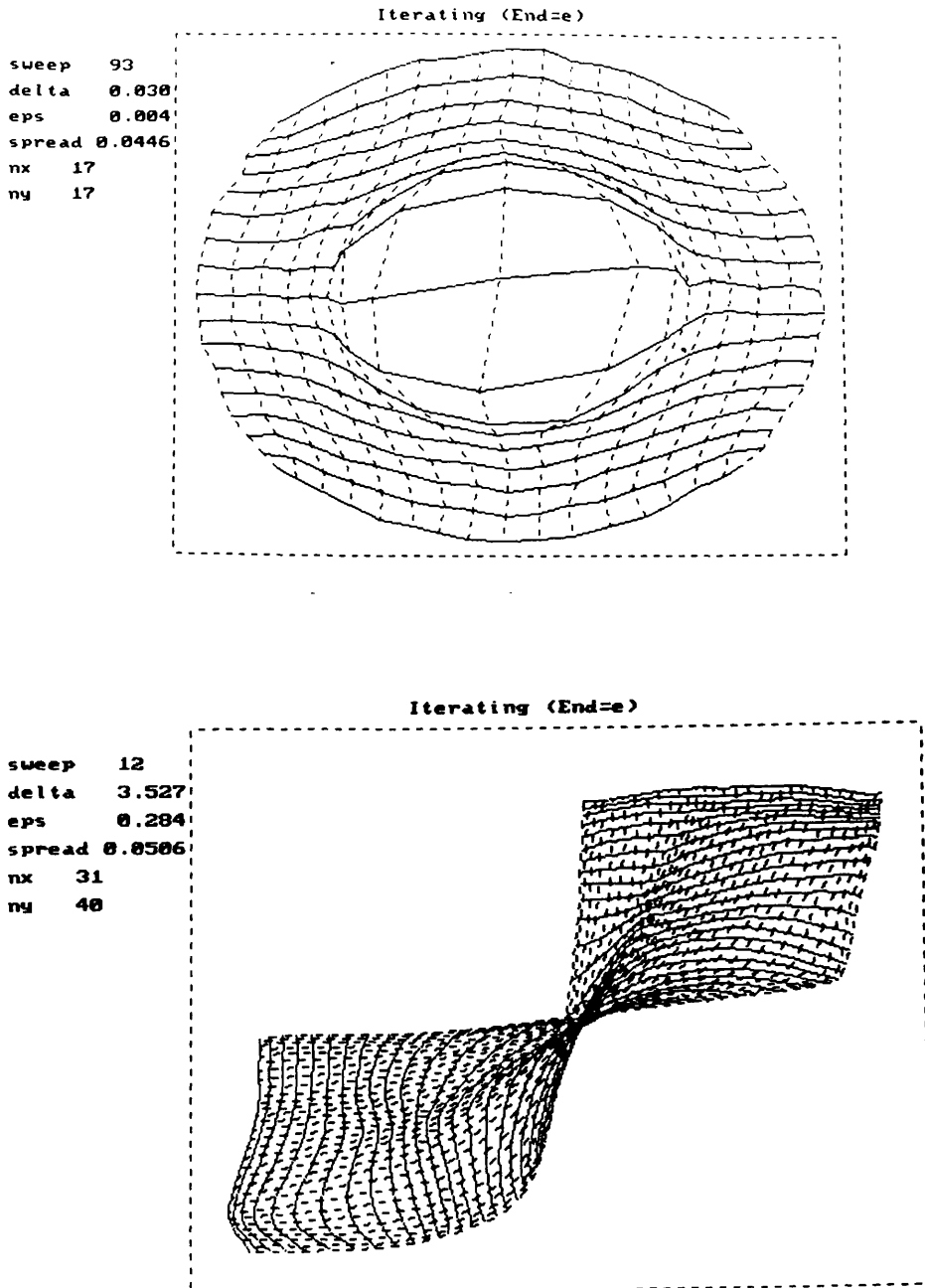
7. Incremental learning

Incremental learning techniques seem to be highly suitable in a number of applications. Connectionist networks and genetic programming offer ways of implementing incremental learning. Considering an example in language processing, it has been shown that neural network models are incapable of learning complex grammars when both the network and the input remain unchanging (Elman, 1991; 1993). However, when either the input was presented incrementally or the network begins with limited memory that gradually increases, the network was able to learn grammar and represent complex sentences. Further, the existing learning algorithms, in general, are inadequate for tackling a number of problems known as the hard-to-learn problems or the type-2 problems (Clark and Thornton, 1993). The components of the input data tend to relate in some way and the learning algorithms as such fail to identify the relationship. These problems can be solved if the data is re-coded to discover the regularities. GP has successfully solved these problems through incremental learning (Thornton and Kuscü, 1994). The proposed method allows for incremental learning and can be tried on type-2 problems.

8. Integrating hardware and software

Connectionist networks and evolutionary algorithms are potential candidates for parallelism. Recently Hardware Description Languages (HDL) (Hemmi, Mizoguchi, and Shimohara, 1994; Higuchi, 1994) have been innovated and successfully applied for evolving hardware (Ray, 1994; Hugo de Garis, 1993). GP might be a powerful tool in building integrated system(s) that combine neural hardware and software through an effective interface with HDL. Network structures can be implemented in hardware with HDL. GP's role will be in evolving the learning mechanisms. The approach is amenable to parallel implementation.

Appendix-A



Figures: 6.2 and 6.3 above show the process of self-organization with the standard Kohonen rule for the models shown in figure. 6.1. The networks grow starting from a single cell.

Appendix-B

Initialization File for the GP Run

Consider the file shown below:

Population size: 500

Number of Generations: 50

Number of ADFs: 6

Creation Type: Ramped Half and Half

Maximum Depth at Creation: 4

Maximum Depth at Crossover: 4

Maximum Fitness: 1000

Number to Mutate: 0

The aim is to create a random population of trees of different sizes and shapes and to have a diversity in the population. The random trees can be created basically by two methods, the 'full' and the 'grow' methods (Koza, 1993). The 'ramped half-and-half' generative method combines the two. In full method all the trees will have the same shape whereas in the grow method the shapes will vary. The mixed approach enables maintain the diversity of the population minimizing the chances of duplicating individuals. These duplicate individuals also waste the computational resources.

Please refer to pp. 92-94 (Koza, 1993) for the details.

Appendix-C

Figure 1 : Fitness Vs Epochs

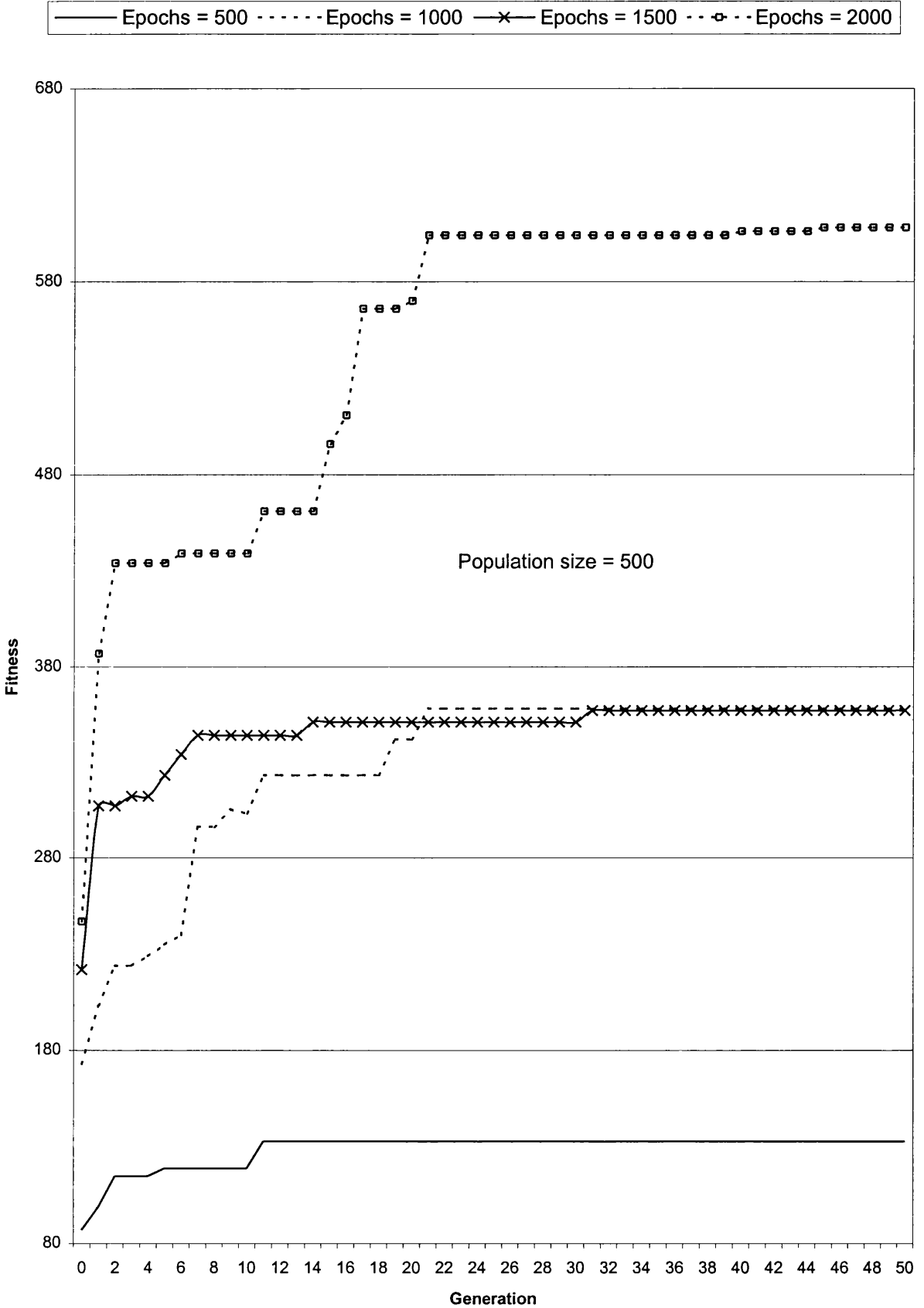


Figure 2a : Fitness Vs Population Size(Ramped half and half)

— Population = 500 ····· Population = 300

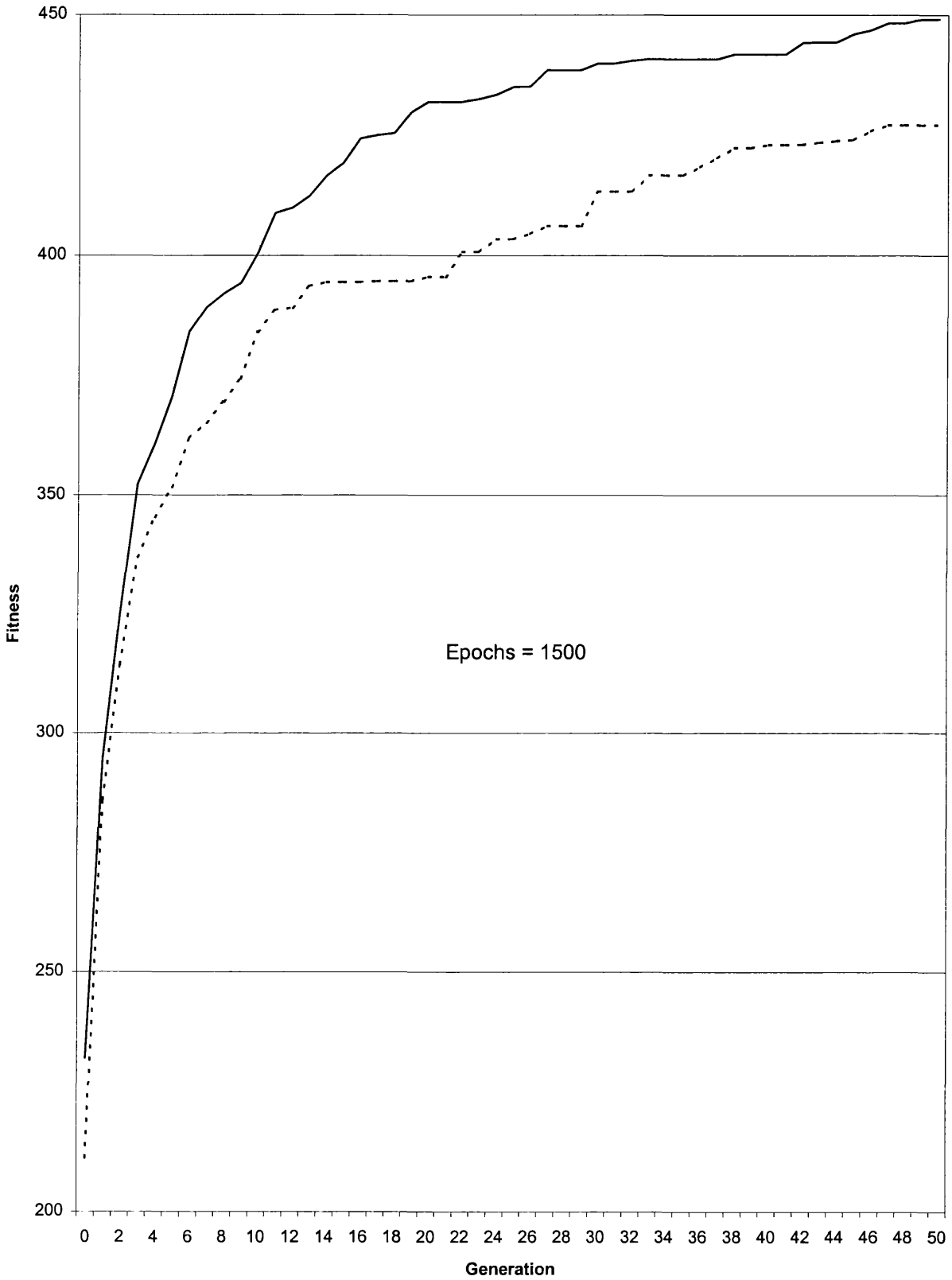


Figure 2b : Fitness Vs Population Size (Ramped half and half)

— Population = 1000 ····· Population = 500

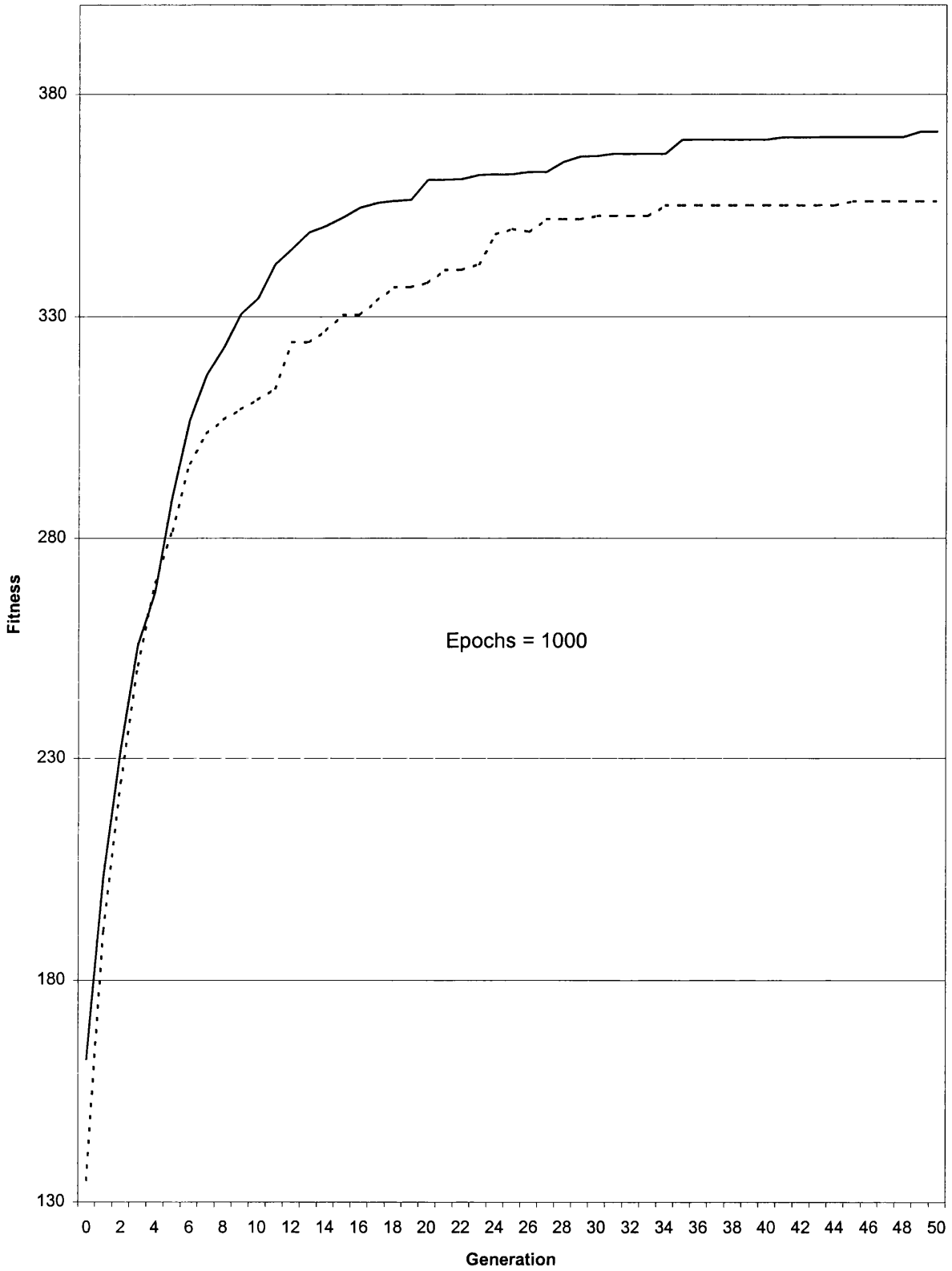


Figure 3a : Fitness Vs Population Diversity

— Ramped hald and half - - - - Variable

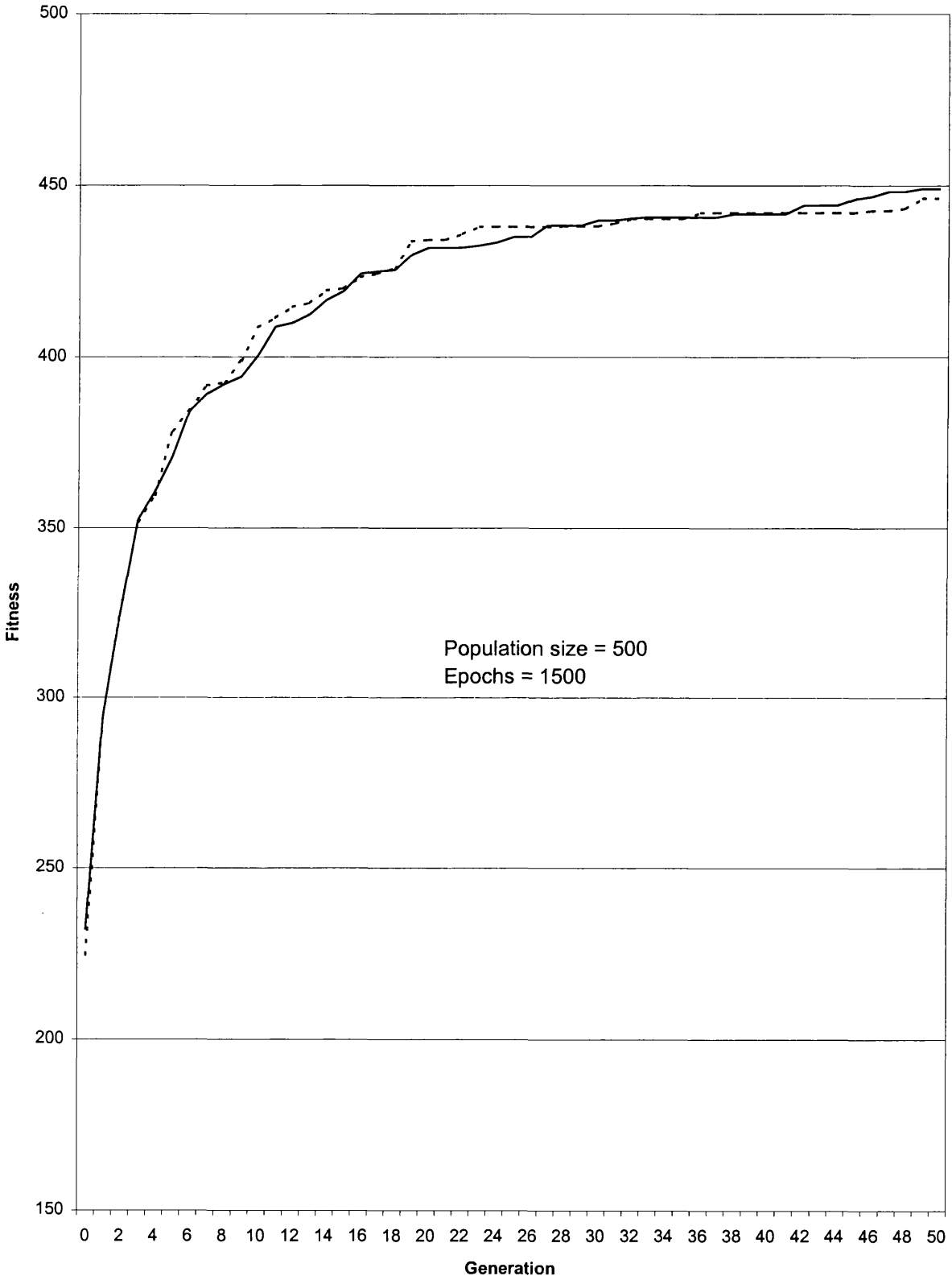


Figure 3b : Fitness Vs Population Diversity

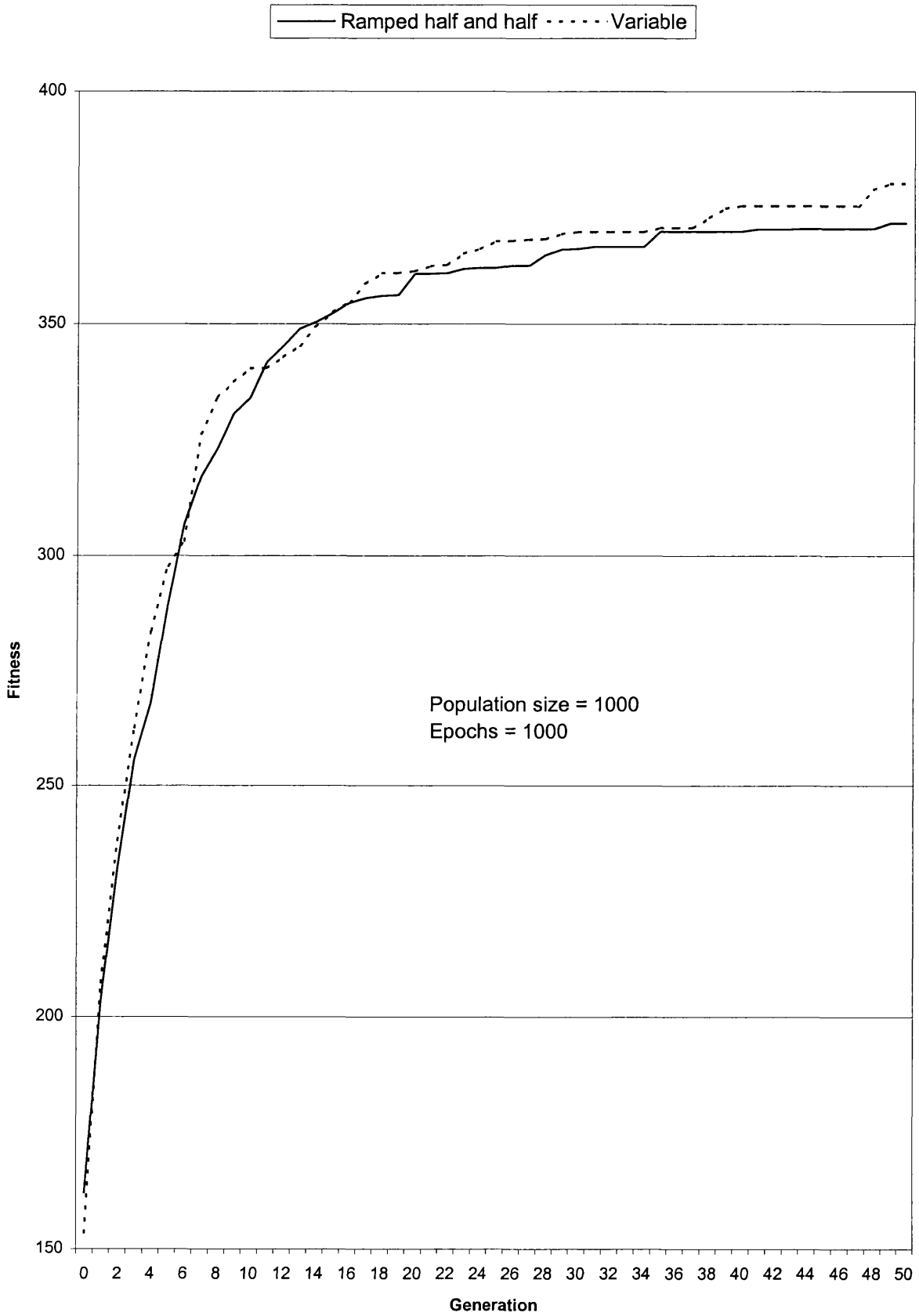


Figure 4a : Fitness Vs Depth

—○— Depth = 3(Creation) / 3(Crossover) - - ○ - - Depth = 3(Creation) / 4(Crossover)
—□— Depth = 3(Creation) / 5(Crossover) —×— Depth = 3(Creation) / 6(Crossover)

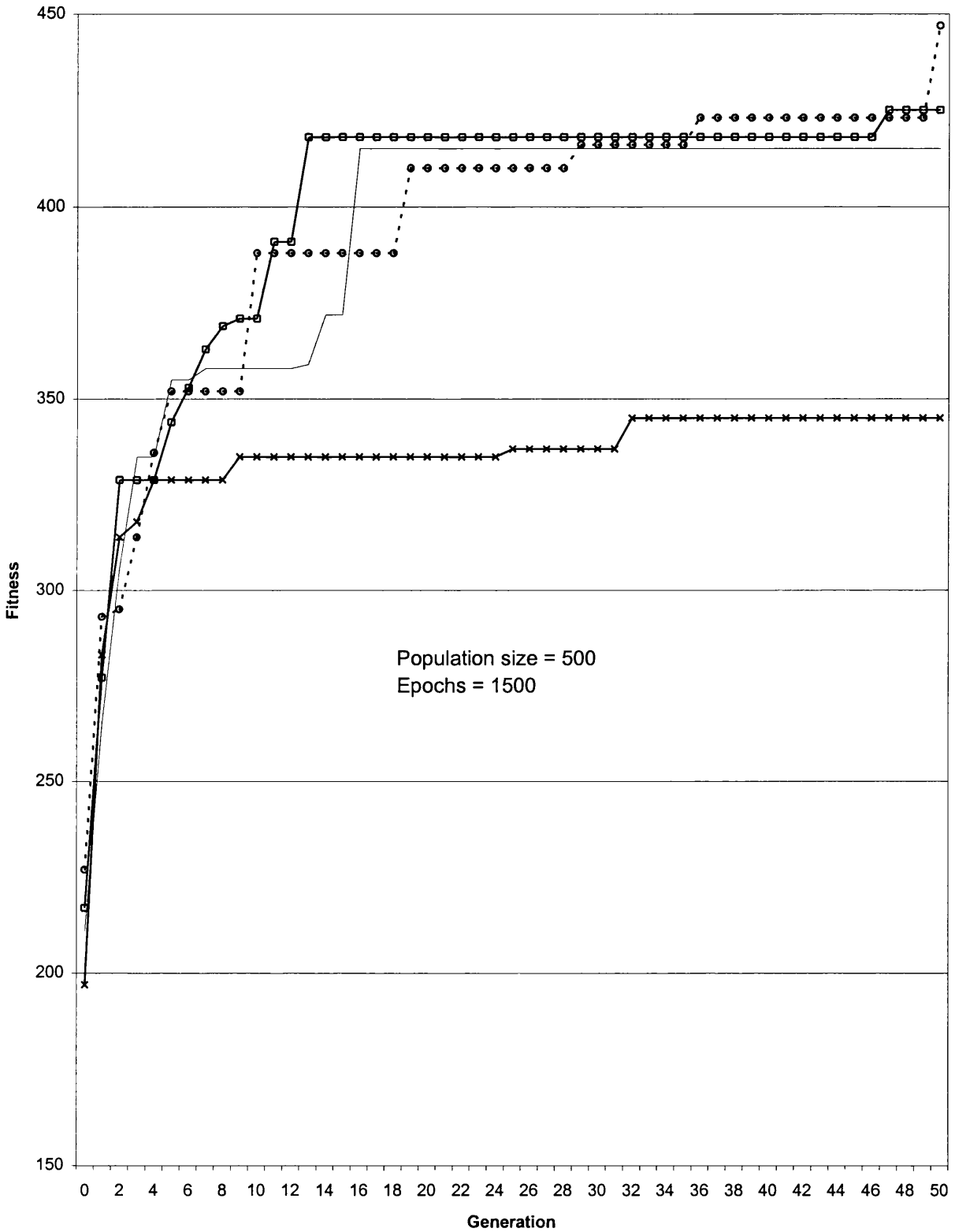


Figure 4b : Fitness Vs Depth

— Depth = 3(Creation) / 4(Crossover) - - o - - Depth = 3(Creation) / 5(Crossover)
—□— Depth = 3(Creation) / 6(Crossover) —x— Depth = 3(Creation) / 7(Crossover)

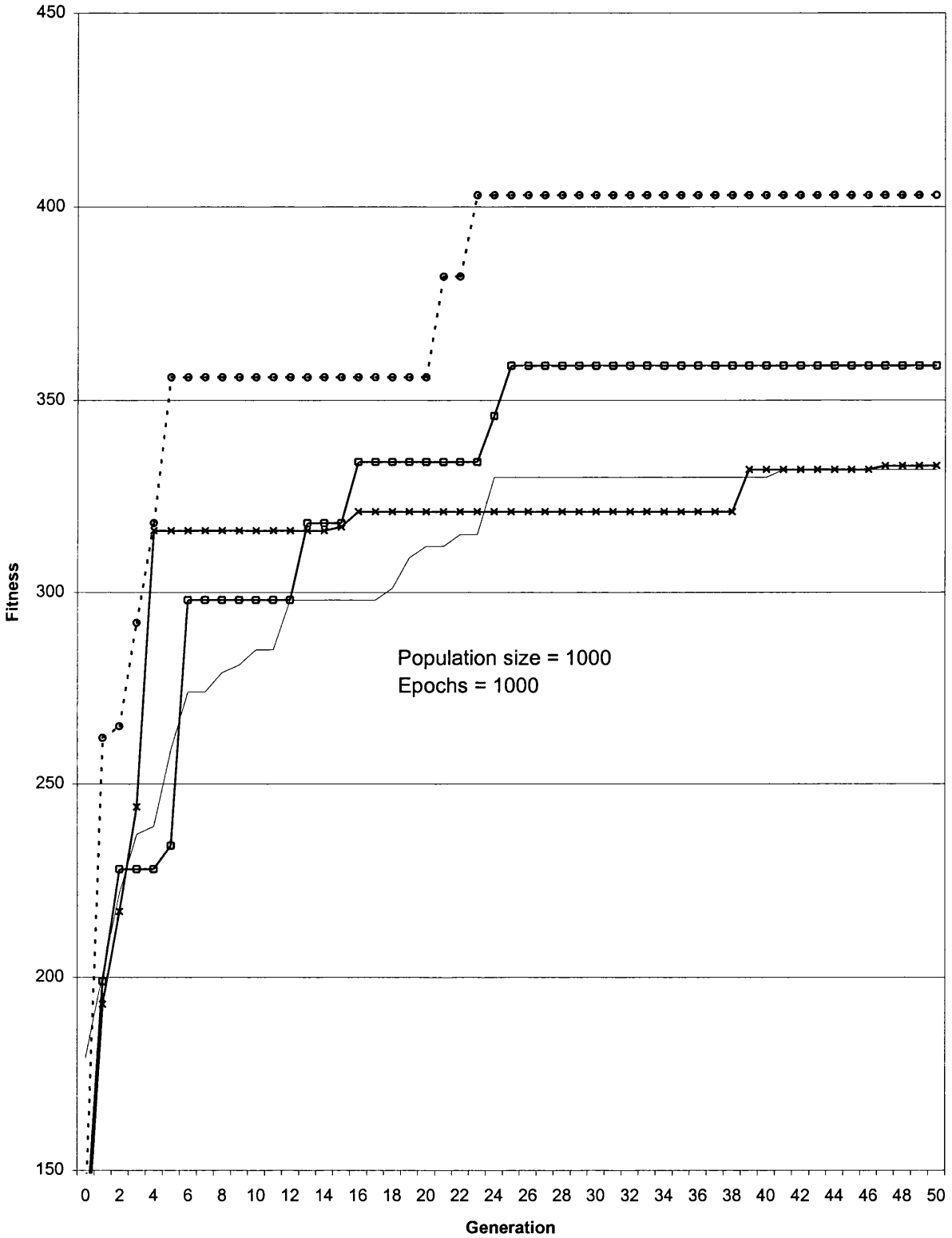
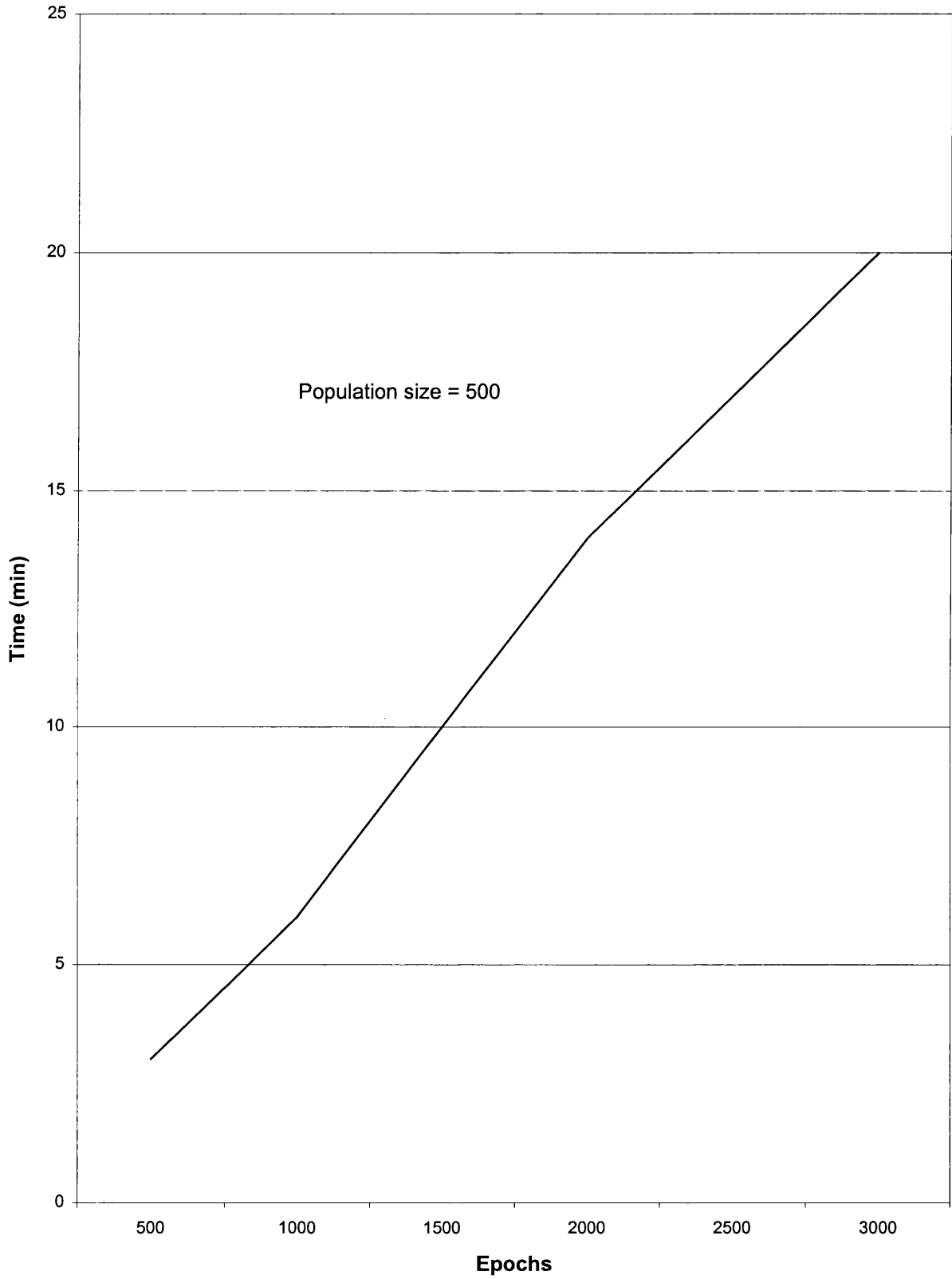


Figure 5 : Time Vs Epochs

— Time (min)



Bibliography

Angelene, P. (1993), *Evolutionary Algorithms and Emergent Intelligence*. Doctoral Dissertation, The Ohio State University.

Ash, T. (1989), Dynamic Node Creation in Backpropagation Networks, ICS Report 8901, University of California, La Jolla.

Bäck, T., F. Hoffmeister and H.P. Schwefel (1991). A survey of evolution strategies. In *Proceedings of the fourth International Conference on Genetic Algorithms*, R.K. Belew and L.B. Booker (eds.), San Mateo, CA: Morgan Kaufmann, pages 2-9.

Belew, R. K., J. McInerney and N. Schraudolph (1991), Evolving Networks: Using the Genetic Algorithm with Connectionist Networks, CSE Technical Report #CS90-174, University of California, San Diego.

Belew, R.K. (1989). When both individuals and populations search: Adding simple learning to Genetic Algorithm. Cognitive Computer Science Research Group. University of California, San Diego, La Jolla, CA.

Balakrisnan, K., and V. Honavar (1995), Evolutionary Design of Neural Architectures, A Preliminary Taxonomy and Guide to Literature, Technical Report CS TR # 95-01, Artificial Intelligence Group, Iowa State University.

Balakrishnan, K. and V. Honavar (1997), Spatial Learning for Robot Localization, *the Genetic Programming Conference, GP-97, Stanford, USA*.

Baxter, J. (1992). The Evolution of Learning Algorithms for Artificial Neural Networks, in: Green D. and Bossomair T. (eds.) *Complex systems*, IOS Press.

Blackmore, J. and R. Miikkulainen (1993), Incremental Grid Growing: Encoding High-Dimensional Structure into a Two-Dimensional Feature Map. In *Proceedings of the IEEE International Conference on Neural Networks*, San Fransisco, CA.

Branke, J. (1995), Evolutionary Algorithms for Neural Network Design and Training. In, *Proceedings of the First Nordic Workshop on Genetic Algorithms and its Applications*, Vaasa, Finland, 1995.

Brooks, R., A. (1986), Achieving Intelligence through Building Robots, AI memo 899, May 1986, MIT.

Brooks, R.A.(1991), Intelligence without representation. *Artificial Intelligence*, 47: pages 139-159.

Clark , P. and Niblett, T. (1989). The CN2 Induction Algorithm. In *Machine Learning Journal*, 3(4), pages 261-283, Netherlands, Kluwer, 1989.

Clark, A. and Lutz, R. (1992), *Connectionism in Context*, Chapter 1.

Carter, D.B. and Narayanan, A. (1998). Genetic algorithms for knowledge discovery in continuous data. Research report R374, Department of Computer Science, University of Exeter, UK.

Chalmers D.J. (1990), *The Evolution of Learning: An experiment on Genetic Connectionism*. In: *Proceedings of the 1990 Connectionist Models Summer School*, CA: Morgan-Kaufmann.

Clark, A., and C. Thornton (1993), *Trading Spaces: Computation, Representation and the Limits of Learning*, Cognitive and Computing Sciences, University of Sussex, Brighton, UK.

Cliff, D., I. Havely and P. Husbands (1992). *Incremental Evolution of Neural Network Architectures for Adaptive Behaviour*, CSRP 256, School of Cognitive and Computing Sciences, University of Sussex, Brighton, UK.

Dasdan A., and K. Oflazar, *Genetic Synthesis of Unsupervised Learning Algorithms* In: *Proceedings of the Second Turkish Symposium on Artificial Intelligence and Artificial Neural Networks*, Istanbul, June 1993.

Davis, R. (1979). Interactive transfer of expertise: Acquisition of new inference rules. *Artificial Intelligence*, 12, pages 121-157.

Dayhoff, J. E. (1990), *Neural Network Architectures, An Introduction*, Van Nostrand Reinhold, New York.

De Jong, K. (1988), *Learning with Genetic Algorithms: An Overview*, Machine Learning 3 pages 121-138, Kluwer Academic Publishers.

De Garis, H. (1992), *Exploring GenNet Behaviours, Using Genetic Programming to Explore Qualitatively New Behaviours in Recurrent Neural Networks*, ETL Laboratories, Japan.

Elman, J.L. (1992), *Distributed representations, simple recurrent networks, and grammatical structure*, Department of Cognitive Science and Linguistics, University of California, San-Diego.

Elman, J.L. (1991), *Incremental learning, or The importance of starting small*, CRL Technical Report 9101, University of California, San Diego.

Erwin, E, K. Obermayer and K. Shulten (1991). *Convergence Properties of Self-Organizing Maps*. *Artificial Neural Networks*, pages 409-414,

Fogel, L., A. Owens and M. Walsh (1966). *Artificial Intelligence through simulated evolution*. New York: John Wiley & Sons.

Fogel, D.B. (1992). *Evolving artificial intelligence*. Doctoral Dissertation, University of California, San Diego.

- Fogel, D. B.(1994). An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*.
- Fogel, D.B. (1997), The Advantages of Evolutionary Computation. In *Proceedings of BCEC97*, Sweden.
- Forrest, S. (1993), Genetic Algorithms: Principles of Natural Selection Applied to Computation, Science, Volume 261, August, 1993.
- Friedrich, M, C. Moraga (1996), An Evolutionary Method to Find Good Building -Blocks for Architectures of Artificial Neural Networks. In *Proceedings of IPMU'96*, pages 951-956, Granada, Spain.
- Fritzke, B. (1991), Unsupervised Clustering with Growing Cell Structures, IJCNN'91, Seattle, US.
- Fritzke, B. (1993), Kohonen Feature Maps and Growing Cell Structures- a Performance Comparison. In *Advances in Neural Information Processing Systems 5*, C.L. Giles, S.J. Hanson, J.D. Cowan (Eds.), Morgan Kaufmann, San Mteo, CA.
- Fritzke, B. (1993), Growing Cell Structures- A Self-organizing Network for Unsupervised and Supervised Learning, TR-93-026, May 1993, International Science Institute, Berkeley, CA.
- Fullmer, B. and Miikkulainen. R (1991). Using Marker-Based Genetic Encoding of Neural Networks To Evolve Finite-State Behaviour, in: *Proceedings of the First European Conference on Artificial Life (ECAL-91)*, Paris.
- Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading MA: Addison Wesley.
- Govinda Char, K. (1998), Constructive Learning with Genetic Programming, *EuroGP-98*, France.
- Govinda Char, K. (1997a), Constructivist AI with GP (Late-breaking paper), *the Genetic Programming Conference, GP-97, Stanford, USA*.
- Govinda Char, K. (1997b), AI-Revisited, *SOCO97, the ICSC Symposium on Soft Computing*, France.
- Govinda Char, K. (1997c), A Novel Approach to Artificial Intelligence, *Tainn'97, the Sixth Turkish Symposium on Artificial Intelligence and Neural Networks*, Baskent University, Turkey.
- Govinda Char, K. (1997d), Evolution of Structure and Learning with Genetic Programming. IWANN'97, *the International Workshop on Artificial Neural Networks*, Canary Islands, Spain, published in the Lecture Notes in Computer Science.

- Govinda Char, K. (1997e), A Novel Approach to Artificial Intelligence, *Tainn '97, the Sixth Turkish Symposium on Artificial Intelligence and Neural Networks*, Baskent University, Turkey.
- Govinda Char, K. (1997f), Modeling Self-organization - Approaches and a Comparison with Evolutionary Methods, BCEC97, *the International Conference on Bio-Computing and Emergent Computation*, Skovde, Sweden, published by World Scientific Publishing.
- Govinda Char, K. (1997g), Modeling Self-Organization- a Comparison, *GWAL '97, the Second German Workshop on Artificial Life*, University of Dortmund, Hans Bommerholz, Germany.
- Govinda Char and Walter Alden Tackett (1997h), Pattern Recognition (section F1.6) in the *Handbook of Evolutionary Computation*, 1997, Oxford University Press, USA.
- Govinda Char, K. (1996a), , Self-organization with Adaptive Learning, *ICML '96, the International Conference on Machine Learning*, Bari, Italy.
- Govinda Char, K. (1996b), A Learning Rule for Emerging Structures, *WCNN96, the World Congress on Neural Networks*, San Diego, USA.
- Govinda Char, K. (1996c), Emergence of Structures in Self-organizing Neural networks using Genetic Programming and Cellular Encoding techniques, *Tainn96, the National Conference on Artificial Neural Networks*, Istanbul, Turkey.
- Greab, R. and Narayanan, A. (1998). A comparison between symbolic and nonsymbolic data-mining techniques. Research report R377, Department of Computer Science, University of Exeter, UK.
- Gruau, F.(1993), The cellular development of neural networks: the interaction of learning and evolution, Research Report 93-04, Ecole Normale Superiure de Lyon, Cedex, France.
- Gruau, F. (1994), Efficient Computer Morphogenesis: A Pictorial Demonstration, Report No. 94-04-027, Santa Fe Institute, April 29, 1994.
- Hämäläinen, A. (1995). Using Genetic Algorithms in Self-organizing map Design. In *Proceedings of ICANNGA '95*, Ales, France.
- Happel, B.L.M. and Murre, J.M.J. (1994). Design and Evolution of Modular Neural Network Architectures. *Neural Networks*, vol 7, Nos. 6/7 pages 985-1004.
- Harvey, I. (1993). *The Artificial Evolution of Adaptive Behaviour*. Doctoral Dissertation, The University of Sussex, UK.
- Hayes-Roth, F., D. A. Waterman and D.B. Lenat (Eds.) (1983), *Building Expert systems*. Reading, Mass: Addison-Wesley.

Heistermann, J. (1990), *Learning in Neural Nets by Genetic Algorithms, Parallel Processing in Neural Systems and Computers*, (Eds.), R. Eckmiller, G. Hartmann and G. Hauske, Elsevier Science Publishing B.V, North Holland.

Hinton, G.E. and Nowlan, S. J. (1987). How learning can guide evolution. *Complex systems*, 1: pages 495-502.

Holland, J., H. (1975; 1992) *Adaptation in Natural and Artificial Systems*. Cambridge, MA: MIT Press.

Honavar, V., and Uhr, L. (1994), *Symbolic Artificial Intelligence, Connectionist Networks and Beyond*, TR-96-16, Iowa State University, Ames.

Kinnear, K.E. (1994), *Advances of Genetic Programming*, MIT Press.

Kitano, H., (1990), *Designing Neural Networks Using Genetic Algorithms with Graph Generation System*, *Complex Systems* 4, (1990) pages 461-476.

Kohonen, T. (1989), *Self-organization and Associative Memory*, volume 8 of Springer Series in Information Sciences. Springer-Verlag, Berlin, Heidelberg, New York, third edition, May 1989.

Kohonen, T. (1995), *The Self-organizing Maps*. Springer-Verlag, Heidelberg, 1995.

Koza, J., R. (1993), *Genetic Programming, On the programming of computers by means of natural selection*, MIT Press.

Koza, J., R. (1994), *Genetic Programming II, Automatic Discovery of Reusable Programs*, MIT Press.

Kuscu, I., and C. Thornton (1994). *Design of Artificial Neural Networks Using Genetic Algorithms: review and prospect*, University of Sussex, UK.

Langton, C.G., (1989). *Artificial Life*. Addison-Wesley.

Langton, C.G., C. Taylor, J.D. Farmer, S. Rasmussen (1992). *Artificial Life II*. Addison-Wesley.

Lippman, R. P. (1987). *An Introduction to Computing with Neural Networks*, IEEE ASSP Magazine, April, 1987.

Mechalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag.

Michalski, S., Carbonell, G., Mitchell, M. (1986). *Machine Learning*, Volume II, Morgan Kaufmann Publishers, Inc.

- Minsky, M. (1990). Logical vs. Analogical or Symbolic vs. Connectionist or Neat vs. Scruffy, in *Artificial Intelligence at MIT., Expanding Frontiers*, Patrick, H. Winston (Ed.), Volume 1, MIT Press 1990.
- Minsky, M. (1963). Steps towards artificial intelligence. In *Computers and Thought*, E. Feigenbaum and J. Feldman (eds.), New York: McGraw Hill.
- Mitchell, M. (1996). *An Introduction to Genetic Algorithms*, MIT Press.
- Miikkulainen. R. and Sirosh, J. (1995). A Unified Neural Network Model for the Self-organization of Topographic Receptive Fields and Lateral Interaction. In *Proceedings of ICANN'95*, Ales, France.
- Moriarty, D.E. and Miikkulainen. R (1994). Efficient Reinforcement Learning through Symbiotic Evolution. Department of Computer Sciences, The University of Texas at Austin, Austin.
- Moriarty, D.E. and Miikkulainen. R (1996). Hierarchical Evolution of Neural Networks. Technical Report AI96-242, Department of Computer Sciences, The University of Texas at Austin, Austin.
- Mühlenbein, H.(1990). Limitations of multi-layer perceptron networks- steps towards genetic neural networks, *Parallel Computing* 14 (1990), pages 249-260.
- Nadi, F. (1991). Topological Design of Modular Neural Networks. *Artificial Neural Networks*, T. Kohonen, K. Makisara, O. Simula and J. Kangas (Eds.) Elsevier Science Publishers B. V. (North Holland).
- Newell, A. and H.A. Simon (1963). 'GPS: A program that simulates thought', In *Computers and Thought*. E.A. Feigenbaum and J. Feldman (eds.), New York: McGraw-Hill, pages 279-293.
- Nolfi, S. and D. Parisi (1994). Genotypes for Neural Networks, Technical Report 94-06, Institute of Psychology, National Research Council, Rome, Italy.
- O'Reilly, and U., F. Oppacher (1995). *An Analysis of Genetic Programming*, Doctoral-Dissertation, Carleton, University of Ottawa.
- Ossen, A. (1990). Top-Down Learning in Modular Feed-Forward Networks. *Parallel Processing in Neural Systems and Computers*. R. Eckmiller, G. Hartmann and G. Hauske (eds.). Elsevier Science Publishers B.V. (North Holland).
- Polani, D. and T. Uthman, Training Kohonen Feature Maps in different Topologies: An Analysis using Genetic Algorithms, in: *Proceedings of the Fifth International Conference on Genetic- Algorithms*, 1993.

- Poli, R.(1997), Discovery of Symbolic, Neuro-Symbolic and Neural Networks with Parallel Distributed Genetic Programming, Technical Report, CSRP-96-14, School of Computer Science, University of Birmingham, UK.
- Puttkamer. E and U. R. Zimmer (1994), Realtime -learning on an Autonomous Mobile Robot with Neural Networks, *Euromicro '94* , Vaesteraas, Sweden.
- Quinlan, J. Ross. (1986). The induction of decision trees. *Machine Learning*, 1, pages 81-106.
- Radi, A.M., and Poli, R. (1997), Discovery of Neural Network Learning Rules Using Genetic Programming, Technical Report, CSRP-97-21, School of Computer Science, University of Birmingham, UK.
- Rechenberg, I.(1973). Evoultinsstrategie: Optimierung Tecechnischer Systeme nach Prinzipien der Biologischen Evolution, Frommann-Holzboog Verlag, Stuttgart.
- Rich, E., and Knight, K. (1991). *Artificial Intelligence*, Second Edition, McGraw-Hill, Inc.
- Ritter, H. (1991). Learning with the Self-Organizing Map, *Artificial Neural Networks*, T. Kohonen, K. Makisara, O. Simula and J. Kangas (Eds)., Elsevier Science Publishers, North Holland.
- Ritter, H., T. Martinetz, K. Schulten (1992). *Neural Computation and Self-Organizing Maps, An Introduction*. Addison Wesley.
- Romaniuk, S. (1993). Evolutionary growth perceptrons. In *Genetic Algorithms: Proceedings of the Fifth International Conference (GA93)*. , S. Forrest (ed.), San Mateo, CA: Morgan Kauffmann, pages 334-341.
- Schank, R.C. (1987). What is AI anyway? *The AI magazine*, 8 (4), pages 59-65.
- Schank, R.C., and D.B. Leake (1989). Creativity and learning in case-based explainer: in *Machine Learning: Paradigms and Methods*, J. Carbonell (ed.), Cambridge, MA: MIT Press, pages 353-386.
- Schiffmann, W., M. Joost, R. Wernwer (1992). Synthesis and Performance Analysis of Multilayer Neural Network Architectures, Technical Report 16/1992, University of Koblenz, Germany.
- Schwefel, H.P.(1981). *Numerical optimization of Computer Models*, John Wiley and Sons, Chichester.
- Smieja, F. (1994). The Pandemonium System of Reflective Agents. Report number: 1994/2. German National Research Center for Computer Science (GMD), Sankt Augustin, Germany.
- Someren, M. and Verdenius (1998). Introducing inductive methods in knowledge acquisition by divide-and-conquer in: D. Aha, R. Engels and F. Verdenius (eds.): *AAAI-ICML*

Workshop Developing Machine Learning Applications: problem definition, task-decomposition and technique selection, AAAI Technical Report.

Srinivas, M., and L. Patnaik (1994). Genetic Algorithms, A Survey, an IEEE, Computer.

Tavan, P., H. Grumüller, and H. Kühnel (1990). Self-organization of associative memory and pattern classification: recurrent signal processing on topological feature maps, *Biological-Cybernetics* 64, pages 95-105,

Tackett, W.A. (1994), *Recombination, Selection and the Genetic Construction of Computer Programs*. Doctoral Dissertation, University of Southern California.

Teller, A. and D. Andre (1995). Turing Completeness in the Language of Genetic Programming with Indexed Memory, Department of Computer Science, Carnegie Mellon-University, Pittsburgh.

Thornton, C. (1993). Representational Eclecticism A Foundation Stone for the New AI? School of Cognitive Science, University of Sussex, UK.

Thornton, C. (1994). Measuring the Difficulty of Specific Learning Problems, *Cognitive and Computing Sciences*, University of Sussex, Brighton, UK.

Vaario, J. (1993). Artificial Life as Constructivist AI, University of Tokyo.

Vaario, J. (1994). From Evolutionary Computation to Computational Evolution, ATR Laboratories, Japan.

Whitley, D. (1993). A Genetic Algorithm Tutorial, Technical Report CS-93-103, Colorado State University.

Whitley, D., T. Starkweather and C. Bogart (1990). Genetic algorithms and neural networks: optimizing connections and connectivity, *Parallel Computing* 14 (1990), pages 347-361.

Zhang, B., and Mühlenben, H. (1993). Genetic Programming of Minimal Neural Nets Using Occam's Razor. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, (Ed.), Stefanie Forrest, Morgan Kaufmann.

Zaverel, J. (1996). Neural Navigation Interfaces for Information Retrieval: Are They More than an Appealing Idea?. *Artificial Intelligence Review* 10, pages 477-504, Kluwer Academic Publishers, Netherlands.



DUNDAS & WILSON

MACM/KMC
Your ref

Dundas & Wilson CS

Privileged/Confidential information

may be contained in this facsimile and is intended only for the use of the addressee. If you are not the addressee, or the person responsible for delivering it to the person addressed, you may not copy or deliver this to anyone else. If you receive this facsimile by mistake, please notify us immediately by telephone. Thank you.

191 West George Street,
Glasgow G2 2LB
0141 222 2200 Telephone
0141 222 2201 Facsimile (Central)
DX GW345

Date: 14 December 1999

To: Debra Maddern - University of Glasgow

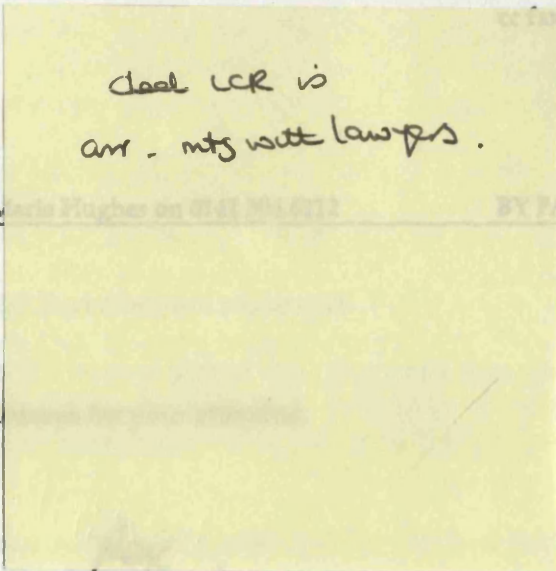
Fax no: 330 4920

cc:

From: Mark Morton

Subject: Dr KC Shar

Pages to Follow: 7



If unreadable please contact Anne M

BY FAX ONLY

Our Ref: MACM/KMC

Please see attached letter and encl

Regards

Mark A. C. Morton
Associate

Response on Clar follows —
(The WR had a chance to read it as I'm away this p.m.)

Debra
14.12

DUNDAS & WILSON

Privileged/Confidential information

may be contained in this facsimile and is intended only for the use of the addressee. If you are not the addressee, or the person responsible for delivering it to the person addressed, you may not copy or deliver this to anyone else. If you receive this facsimile by mistake, please notify us immediately by telephone. Thank you.

Dundas & Wilson CS

191 West George Street,
Glasgow G2 2LB
0141 222 2200 Telephone
0141 222 2201 Facsimile (Central)
DX GW345

Date: 14 December 1999
To: Debra Maddern - University of Glasgow
cc:
From: Mark Morton
Subject: Dr K C Shar
Pages to Follow: 7

Fax no: 330 4920
cc fax no.


If unreadable please contact Anne Marie Hughes on 0141 304 6112

BY FAX ONLY

Our Ref: MACM/KMG

Please see attached letter and enclosures for your attention.

Regards


Mark A. C. Morton
Associate

Rex
Response or Clar follows —
(The WR had a chance to
read it as I'm away this p.m.)
Debra
14.12

DUNDAS & WILSON

Our ref MACM/KMG
Your ref

Dundas & Wilson CS

Ms Deborah Maddern
Administrative Assistant
University of Glasgow
Main Building
Glasgow
G12 8QQ

191 West George Street
Glasgow G2 2LD
0141 222 2200 Telephone
0141 222 2201 Facsimile
DX 561475
GLASGOW 16

14 December 1999

Dear Deborah

Dr KC Shar

Thank you for your letter of 9th December and enclosures.

For the purpose of this letter I have assumed that the panel was charged with the responsibility of investigating the matter; coming to a decision on what the facts are, where possible, and making recommendations, if appropriate. If this is not the case please let me know.

As ever this is quite a difficult matter - the panel and the Clerk of the Senate have an unenviable task! I do think the draft letter needs to be revised for the reasons set out below.

Generally I think the response to Kathleen Bolt ("KB") from the University should be in fairly "conciliatory" terms and that the panel should be seen to use her questions as a helpful way of re-expressing or adding to the report to provide greater clarity. To start with I will use the letter from KB as a point of reference and my comments on the points she makes (which I have numbered on a copy of her letter which I enclose) are as follows:-


- 1 KB claims the introduction is misleading. It is not necessary to agree with her on this but arguably every thing in the first paragraph with the exception of the first sentence is strictly not essential to the report and could be deleted.
- 2 What is the position here?
- 3 I would reply by confirming that in light of the interpretation KB has put on this, it is recognised that that use of the words "on the whole" was inappropriate and misleading given the panel's findings.

- 4 I think KB wants to know exactly why the panel felt successive heads of department failed to grasp the nettle of "adequate supervision" -because of resources...pressure of business or for some other reason? Was Dr Char asked what the consequences of this were or alternatively did she volunteer this information? I would respond by confirming the position.
- 5 You can see why KB is anxious to establish whether the panel formed a view on whether Dr Char told Dr Sharman about the proposal. You would ordinarily have expected her to do so. Equally though if it was lost what was to stop Dr Char getting a copy of the letter from its author? I anticipate KB would like to know whether the panel asked Dr Sharman if he was positive that he did not get a copy of the letter? Did he speak with Dr Char with this on a number of occasions? Did the panel make enquiries with the "senior staff" that Dr Char claimed she spoke with? Did these assist the panel? I think the response the University has prepared is the right way forward at this stage subject to clarification of the above questions.
- 6 Provided Dr Char's position to the Panel was that she was that she was never unwilling to use the UNIX facilities then I suggest the report should start by confirming this and by confirming that the facility was available to all. The report should probably confirm who gave evidence that she was unwilling to use the UNIX now that this seems to be relevant. KB's question is very much a loaded one and the issue is really whether the panel were satisfied that Dr Char had been given adequate facilities. I note the report does give its finding in this regard. Is it the case that there was a similarity of facilities and, if so, should the report be amended to show that this was a factor the panel took into account in reaching their decision on adequacy?
- 7 I think the response to this query should reflect carefully what Dr Char's position to the Panel was. In addition I would be tempted to expand the report accordingly to deal with this. I would refrain from inviting KB to respond by asking her the question about computing capabilities.
- 8 I think the point KB is trying to make is that the findings in the report state that there is no evidence when there is of course, Dr Char's evidence. However I feel that the general position is not irrelevant as KB perhaps seeks to suggest. It might be a factor one could legitimately look at when determining what was actually said. I think KB is looking to ascertain what the panel felt had been said to Dr Char about grants and I too would be interested to know what they found as it will help us understand their position. The report needs to be amended in my opinion.
- 9 The report states that the department should have acted more decisively and KB has extrapolated from this that the panel also found that there was inadequate supervision. I think it is important for the panel to set the record straight on the issue of adequate supervision. Dr Char is looking for compensation and from a purely legal perspective the main issues are therefore (1) whether there was a breach of contract and (2) if so, what the loss is. I assume the University has no proposals but I do think they should go into a bit more detail about why they feel this should be the case.

- 10 I note that Dr Char made no direct claim of racial harassment. Whilst Dr Char may have not used the word "discrimination" it is clearly KB's intention to now raise this as one of her arguments. What are the reasons for not allowing the racial harassment issue to be dealt with as part of the complaint procedure as KB suggests? Is it the case that the panel did not consider that the issue of her status as an overseas student or her ethnic origin was a relevant factor in what took place in this case, or did they just not consider these issues at all?
- 11 I think it is important for the University to explain the position in a bit more detail given KB's desire to know whether efforts were made to trace these people and if not why? Did Dr Char make anything of this at the oral hearings? Was the position explained to her then?
- 12 My reading of the report is that Dr Char received adequate supervision and facilities (see finding 9) but in light of KB's comments I wonder whether the report at section 2 and 5 needs to be clarified or expanded.
- 13 Again should the report be amended to deal with these points?

I think it would be helpful for the letter to KB to be redrafted in light of my above comments and would be delighted to look over the final version before it goes. I am on holiday on 16th 20th -28th and 30th-31st December. I am also off from 1st -3rd January 2000.

Yours sincerely


Mark A C Morton
Associate

ETHNIC MINORITIES LAW CENTRE

41 ST VINCENT PLACE

GLASGOW

G1 2ER

ATHLEEN BOLT
PRINCIPAL SOLICITOR

TEL: 0141 204 2888

FAX: 0141 204 2006

Our Ref: KB/AS/HKY21

8 July 1999

Professor R R Whitehead
 Clerk of Senate
 Senate Office
 University of Glasgow
 Glasgow

SENATE OFFICE

Rec'd - 9 JUL 1999

Dear Professor Whitehead

Dr K G Char - University Complaint

I refer to your letter of 28th April to Dr Char enclosing the Report of the Panel set up to investigate her complaint to the Senate and your request for her comments regarding the factual accuracy of the Report. Dr Char has prepared a summary of the points she wishes to raise and I enclose this document.

I note that following consideration of Dr. Char's comments a final Report will go to the Senate itself. Given your findings on a number of issues I would ask that the following points be addressed by yourselves and/or by the Senate.

① Firstly it is my opinion that your introduction is slightly misleading. Dr Char was advised by the Principal to take the complaint about which she had written to him to the Senate Appeals procedure. As you are aware there was no complaints procedure in place at that time and her complaint was not against an academic matter, given that she had been awarded her PhD. There appears therefore to have been no appropriate remedy available at that time and she merely followed the advice of the Principal in approaching the Senate Clerk who after considering her case advised/allowed her to proceed by way of your draft complaints procedures. Indeed one of the difficulties which Dr Char appears to have encountered is the lack of any appropriate procedures either at Departmental or Senate level for dealing with her complaints at a much earlier stage. This point arises in your findings and is discussed again below.

SC

ETHNIC MINORITIES LAW CENTRE (a company limited by guarantee) registered in Scotland: 134099

Registered Office: 41 St Vincent Place, Glasgow, G1 2ER

ETHNIC MINORITIES LAW CENTRE IS A CHARITY

Its legal work is undertaken by the independent legal practice of Kathleen Bolt, Principal Solicitor at the above address.

- ② Secondly your Report fails to mention Dr Char's request for other Departmental members and others to be present at the third meeting of the Panel and your reasons for refusing this request.

I would like to raise some points arising from the findings:-

- ③ 1. You say that 'on the whole' Dr Char was dealt with honourably by members of the Department. This suggest exceptions - what are these?
- ④ 2. You say that successive Heads of Department 'failed to fully grasp the nettle' What does this mean and what were the consequences for Dr Char? Further you think that a 'pastoral' supervisor might have alerted the Department to the breakdown between Dr Char and Dr Sharman. Whilst this may well have assisted, it seems that this breakdown should nonetheless have been apparently obvious to the Departmental Heads.
- ⑤ 4. Is there any reason to disbelieve Dr Char that she showed Dr Sharman the proposal from BT and that he failed to do the necessary work to allow her to process this matter?
- ⑥ 5. As stated in Dr Char's summary she denies that she was unwilling to use the UNIX facilities. These findings also do not address the issues of whether the facilities available to Dr Char were the same or similar to those available to others in the Department at her level.
- ⑦ 7. There seems to be some confusion here. Dr Char is adamant that she did not write a GP kernel count and did not want to write one. She advises that this was beyond her computing capabilities and that to have written such a count would have taken an enormous amount of time which she could not have allocated to her PhD. She advises that the software was available on the Adam Fraser Kernel and her intention was to use genetic programming to write an engineering application and to combine neural networks with it which she ultimately did. She advises that she repeatedly told the panel that she did not write a GP kernel.
- ⑧ 8. Your findings seem to state the general position rather than address the specifics of Dr Char's position. Whilst it may be the case that a Head has no authority to propose more than consideration of a position, that does not deal with the issue of whether on this occasion the impression of being able to promise more was given.

SC

Conclusion. You refer to 'accepted norms'. What are these, and upon what are they based?

9 Given your findings at number 1 and 2, what remedy do you propose for Dr Char in relation to this matter. I note your recommendations regarding the workings of the Department, however this will be of little consequence to Dr Char herself. The failure of the Department to consider her complaints timeously or seriously has clearly exacerbated the problems she was facing, not least the difficulties of establishing events at this stage.

10 Secondly, Dr Char has a number of times, in correspondence and in her detailed complaint, made reference to the fact that she believes she was discriminated against i.e. treated differently or less favourably by virtue of her ethnic origin and status as an overseas student. Her main concerns have been failure to change her supervisor, the lack of facilities and indeed the complete deprivation of suitable facilities for a considerable period of time and the failure of the department/university to investigate her complaints about harassment and other incidents despite repeated requests, all of which she believes may have been discriminatory. At no point is this issue addressed. Whilst I understand that there is a separate code relating to racial harassment, the issues arising for Dr Char appear to relate as much to unfavourable or differential treatment as to racial harassment as such and therefore I see no reason why they cannot be dealt with as part of the complaints procedure. Further no one has suggested to her that she pursue the alternative remedy of racial harassment.

11 Thirdly Dr Char has given the names and contacts for a number of other students who she believes also had difficulties with Dr Sharman. It would appear that no efforts have been made to contact these other students and investigate the matter. It would certainly put a different perspective on matters if it was established that Dr Char was not the only student who felt that she had been mistreated by Dr Sharman and may well change the panel's view.

12 Fourthly the crux of her complaint is really why the Department did not change her supervisor and provide appropriate facilities at a much earlier stage. Once again the panel does not appear to have resolved the failure of the Department to do either of these.

13 Finally the question arises as to why Dr Char was assigned a supervisor with no background in neural networks or artificial intelligence. If there was no such supervisor available within the Department then this begs the question as to why she was invited to complete her PhD. within the department in the first place.

SC

We hope that the panel can address these various issues in re-discussing the report and look forward to your comments in relation to these matters.

Sincerely yours
on behalf of EMLC

Kathleen Bolt

KB Kathleen Bolt
PRINCIPAL SOLICITOR