



# Durham E-Theses

---

## *Learning algorithms for adaptive digital filtering*

Nambiar, Raghu

### How to cite:

---

Nambiar, Raghu (1993) *Learning algorithms for adaptive digital filtering*, Durham theses, Durham University. Available at Durham E-Theses Online: <http://etheses.dur.ac.uk/5544/>

### Use policy

---

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a [link](#) is made to the metadata record in Durham E-Theses
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the [full Durham E-Theses policy](#) for further details.

The copyright of this thesis rests with the author.  
No quotation from it should be published without  
his prior written consent and information derived  
from it should be acknowledged.

# Learning Algorithms for Adaptive Digital Filtering

Raghu Nambiar  
School of Engineering and Computer Science  
University of Durham

A thesis submitted in partial fulfilment of the require-  
ments of the Council of the University of Durham for  
the Degree of Doctor of Philosophy (Ph.D.).

January 1993



i 12 MAY 1993

## Abstract

In this thesis, we consider the problem of parameter optimisation in adaptive digital filtering. Adaptive digital filtering can be accomplished using both Finite Impulse Response (FIR) filters and Infinite Impulse Response Filters (IIR) filters. Adaptive FIR filtering algorithms are well established. However, the potential computational advantages of IIR filters has led to an increase in research on adaptive IIR filtering algorithms. These algorithms are studied in detail in this thesis and the limitations of current adaptive IIR filtering algorithms are identified. New approaches to adaptive IIR filtering using *intelligent learning algorithms* are proposed. These include Stochastic Learning Automata, Evolutionary Algorithms and Annealing Algorithms. Each of these techniques are used for the filtering problem and simulation results are presented showing the performance of the algorithms for adaptive IIR filtering. The relative merits and demerits of the different schemes are discussed. Two practical applications of adaptive IIR filtering are simulated and results of using the new adaptive strategies are presented. Other than the new approaches used, two new hybrid schemes are proposed based on concepts from genetic algorithms and annealing. It is shown with the help of simulation studies, that these hybrid schemes provide a superior performance to the exclusive use of any one scheme.

*TO MY PARENTS*

*To whom much more is owed  
than can be mentioned here.*

## Acknowledgments

It would be presumptuous to think that a thesis is the sole effort of a single individual. This is my sincere effort to thank all those who have, either directly or indirectly, helped during my stay at Durham and in my study.

- To my parents and sister for all the love and support.
- To my supervisor Prof. Mars - his wit and good cheer always made things more tractable, and for his infinite patience especially during the last stages.
- To Dr. Tang - for his thought provoking questions during the initial phase.
- To the British Council - for all the support especially financial, and in particular to Angie Stephenson - my program adviser at the British Council - she was a true friend.
- To Prof. Sengupta - all this would have not materialised if it had not been for his help.
- To Ritu, Bipul, Amit, Manju, Bipul, Rashmi, Rajeev, Nithya and John - for providing a touch of home in the cold climes of Britain.
- To Shyam Sunder - for being such an accommodating host during my visits to the United States.
- To John, Alan and David in the lab - for all the good times.
- To numerous friends at the halls of residence - none is mentioned by name lest I offend those who I have forgotten by oversight.
- To Sylvia - for all the help during the three years.
- To Jamie, Gemma, Neil and Trisha - for .... well, they wanted to be in the acknowledgments !!.

## Declaration

I hereby declare that this thesis is a record of work undertaken by myself, that it has not been the subject of any previous application for a degree, and that all sources of information have been duly acknowledged.

© Copyright 1993, Raghu Nambiar

The copyright of this thesis rests with the author. No quotation from it should be published without his written consent, and information derived from it should be acknowledged.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Why Adaptive Filtering ? . . . . .	1
1.2	Outline of Thesis . . . . .	4
<b>2</b>	<b>Adaptive Digital Filtering</b>	<b>8</b>
2.1	Introduction . . . . .	8
2.2	Adaptive FIR Filtering . . . . .	12
2.3	Adaptive IIR Filtering . . . . .	13
2.3.1	Introduction . . . . .	13
2.3.2	Different Formulations of Estimation Error . . . . .	14
2.3.3	Adaptive Algorithms . . . . .	16
2.4	Alternative Realizations . . . . .	19
2.4.1	Parallel Form . . . . .	20
2.4.2	Cascade Form . . . . .	21
2.4.3	Lattice Form . . . . .	22
2.5	Applications of Adaptive IIR Filtering . . . . .	22
2.5.1	Adaptive Noise Cancelling . . . . .	23
2.5.2	Adaptive Equalization . . . . .	24
2.6	Discussion . . . . .	25
<b>3</b>	<b>Stochastic Learning Automata</b>	<b>32</b>
3.1	Introduction . . . . .	32
3.2	Stochastic Learning Automata . . . . .	33
3.2.1	Stochastic Automata . . . . .	33

3.2.2	The Environment . . . . .	35
3.2.3	Norms of Behaviour . . . . .	37
3.3	Learning Algorithms . . . . .	38
3.3.1	Standard Learning Algorithms . . . . .	38
3.3.2	Discretised Learning Algorithms . . . . .	41
3.3.3	Estimator Algorithms . . . . .	42
3.3.4	S-Model Learning Schemes . . . . .	48
3.4	Interconnected Automata . . . . .	51
3.4.1	Hierarchical Learning Automata . . . . .	51
3.4.2	Automata Games . . . . .	52
3.5	Discussion . . . . .	53
<b>4</b>	<b>Adaptive Digital Filtering using Stochastic Learning Automata</b>	<b>56</b>
4.1	Introduction . . . . .	56
4.2	Simulation Configuration . . . . .	57
4.2.1	Introduction . . . . .	57
4.2.2	Using Stochastic Learning Automata . . . . .	57
4.2.3	Different Categories of Modeling . . . . .	60
4.3	Simulation Results . . . . .	62
4.3.1	Introduction . . . . .	62
4.3.2	Results using P-Model Learning Algorithms . . . . .	64
4.3.3	Results using S-Model Learning Algorithms . . . . .	66
4.3.4	Other Categories . . . . .	70
4.3.5	Automata Games and Hierarchical Schemes . . . . .	71
4.4	Conclusions . . . . .	72
<b>5</b>	<b>Genetic and Evolutionary Optimisation</b>	<b>90</b>
5.1	Introduction . . . . .	90
5.2	Genetic Algorithms . . . . .	94
5.2.1	Introduction . . . . .	94
5.2.2	Standard Genetic Operations . . . . .	97
5.2.3	Improved Genetic Operations . . . . .	100

5.2.4	Adaptive Extensions of Genetic Algorithms . . . . .	103
5.3	Evolutionary Strategies . . . . .	104
5.3.1	Introduction . . . . .	104
5.3.2	Standard Evolutionary Strategies . . . . .	105
5.3.3	Improved Evolutionary Strategies . . . . .	109
5.4	Evolutionary Programming . . . . .	111
5.4.1	Introduction . . . . .	111
5.4.2	Salient Features . . . . .	112
5.4.3	Adaptive Extensions to Evolutionary Programming . . . . .	113
5.5	Discussion . . . . .	114
<b>6</b>	<b>Adaptive Digital Filtering using Genetic and Evolutionary Optimi-</b>	
	<b>sation</b> . . . . .	<b>116</b>
6.1	Introduction . . . . .	116
6.2	Simulation Configuration . . . . .	117
6.2.1	Genetic Algorithms . . . . .	117
6.2.2	Evolutionary Strategies and Programming . . . . .	120
6.3	Simulation Results . . . . .	121
6.3.1	Genetic Algorithms . . . . .	121
6.3.2	Evolutionary Strategies . . . . .	130
6.3.3	Evolutionary Programming . . . . .	132
6.3.4	Applications using the Adaptive IIR Filter . . . . .	134
6.4	Conclusions . . . . .	137
<b>7</b>	<b>Simulated and Genetic Annealing</b> . . . . .	<b>171</b>
7.1	Introduction . . . . .	171
7.2	Simulated Annealing . . . . .	173
7.3	Fast Simulated Annealing . . . . .	176
7.4	Very Fast Simulated Reannealing . . . . .	177
7.5	Genetic Annealing . . . . .	178
7.5.1	Introduction . . . . .	178
7.5.2	Hybrid Scheme - I . . . . .	179

7.5.3	Hybrid Scheme - II . . . . .	181
7.6	Simulation Configuration and Results . . . . .	182
7.7	Conclusions . . . . .	184
<b>8</b>	<b>Conclusions and Further Work</b>	<b>199</b>
8.1	Conclusions . . . . .	199
8.2	Further Work . . . . .	201
8.2.1	Use of Genetic Algorithms in Non-stationary Environments . .	201
8.2.2	Parallel Implementation . . . . .	202
8.2.3	Genetic Algorithms and Neural Networks . . . . .	203
8.2.4	Theoretical Analysis using Natural Genetics . . . . .	203
8.2.5	Hybrid Schemes . . . . .	204
	<b>Appendix A</b>	<b>205</b>
	<b>Appendix B</b>	<b>207</b>
	<b>Bibliography</b>	<b>210</b>
	<b>Publications</b>	<b>222</b>

# List of Figures

1.1	Conventional and Adaptive Filtering Configurations . . . . .	6
1.2	Direct and Inverse System Modeling Configurations . . . . .	7
2.1	Digital Filter . . . . .	26
2.2	Adaptive Digital Filter . . . . .	26
2.3	System Identification Configuration . . . . .	27
2.4	Equation Error Formulation . . . . .	27
2.5	Equation Error Identifier . . . . .	28
2.6	Output Error Formulation . . . . .	28
2.7	Parallel Form Realization . . . . .	29
2.8	Lattice Form Realization . . . . .	29
2.9	Adaptive Noise Canceling Configuration . . . . .	30
2.10	Adaptive Equalization Configuration . . . . .	31
3.1	Stochastic Learning Automata . . . . .	54
3.2	Hierarchical Stochastic Learning Automata . . . . .	55
4.1	System Identification Configuration incorporating Stochastic Learning Automata . . . . .	74
4.2	Discretisation of the Parameter Space . . . . .	74
4.3	The New Scheme of Error Estimation . . . . .	75
4.4	Performance of Standard Learning Algorithms . . . . .	76
4.5	Performance of Discretised Learning Algorithms . . . . .	77
4.6	Performance of Estimator Learning Algorithms . . . . .	78
4.7	Performance of Pursuit Algorithms . . . . .	79

4.8	Performance of Discretised Pursuit Algorithms . . . . .	80
4.9	Performance of S-LRI Learning Algorithms (Old Normalisation) . . .	81
4.10	Performance of S-LRI Learning Algorithms (New Normalisation) . .	82
4.11	Performance of Estimator Learning Algorithms (S-Model) (Old Normalisation) . . . . .	83
4.12	Performance of Estimator Learning Algorithms (S-Model) (New Normalisation) . . . . .	84
4.13	Performance of Relative Reward Learning Algorithms (S-Model) (Old Normalisation) . . . . .	85
4.14	Performance of Relative Reward Learning Algorithms (S-Model) (Old Normalisation) . . . . .	86
4.15	Performance of Relative Reward Learning Algorithms (S-Model) (New Normalisation) . . . . .	87
4.16	Performance of Relative Reward Learning Algorithms (S-Model) (New Normalisation) . . . . .	88
4.17	Performance of P-Model Learning Algorithms (Category (IV) Model)	89
6.1	Comparison between Genetic and Random Search Algorithms . . . .	138
6.2	Comparison between Genetic and Random Search Algorithms . . . .	139
6.3	Comparison between Genetic and Stochastic Learning Automata Algorithms . . . . .	140
6.4	Different Order Filters . . . . .	141
6.5	Effect of Mutation . . . . .	142
6.6	Effect of Crossover . . . . .	143
6.7	Effect of Population Size . . . . .	144
6.8	Effect of Coding Schemes . . . . .	145
6.9	Effect of the Number of Bits . . . . .	146
6.10	Effect of New Crossover Schemes ( $p_m = 0.075$ ) . . . . .	147
6.11	Effect of New Crossover Schemes ( $p_m = 0.025$ ) . . . . .	148
6.12	Effect of Improved Selection Operations . . . . .	149
6.13	Effect of the Ranking Selection Scheme . . . . .	150

6.14	Effect of the Ranking Elitist Selection Scheme . . . . .	151
6.15	Effect of Measurement Noise . . . . .	152
6.16	Results using Self Adaptive Genetic Algorithm . . . . .	153
6.17	Effect of Standard Deviation in ESs . . . . .	154
6.18	Effect of the Number of Parents/Offspring . . . . .	155
6.19	Effect of Parents in Evolutionary Programming . . . . .	156
6.20	Effect of the Number of Competitions in EP . . . . .	157
6.21	Effect of the Number of Competitions in EP . . . . .	158
6.22	Adaptive Noise Canceling - Sum of Sinusoids . . . . .	159
6.23	Adaptive Noise Canceling - Square Wave . . . . .	160
6.24	Adaptive Noise Canceling - PRBS Input . . . . .	161
6.25	Adaptive Noise Canceling - PRBS Input . . . . .	162
6.26	Evolution of the Adaptive Noise Canceling . . . . .	163
6.27	Evolution of the Adaptive Noise Canceling . . . . .	164
6.28	Evolution of the Adaptive Noise Canceling . . . . .	165
6.29	Evolution of the Adaptive Noise Canceling . . . . .	166
6.30	Results from the Adaptive Equalisation Experiment . . . . .	167
6.31	Results from the Adaptive Equalisation Experiment . . . . .	168
6.32	Results from the Adaptive Equalisation Experiment . . . . .	169
6.33	Results from the Adaptive Equalisation Experiment . . . . .	170
7.1	Results using Classical Simulated Annealing . . . . .	186
7.2	Results using Fast Simulated Annealing . . . . .	187
7.3	Comparative Results using Classical and Fast Simulated Annealing (Decay Parameter = 0.9) . . . . .	188
7.4	Results using Hybrid Scheme - I (Decay Parameter = 100) . . . . .	189
7.5	Results using Hybrid Scheme - I (Decay Parameter = 100) . . . . .	190
7.6	Results using Hybrid Scheme - I (Decay Parameter = 50) . . . . .	191
7.7	Results using Hybrid Scheme - I (Decay Parameter = 50) . . . . .	192
7.8	Results using Hybrid Scheme - I (Decay Parameter = 15) . . . . .	193
7.9	Results using Hybrid Scheme - I (Decay Parameter = 15) . . . . .	194

7.10 Results using Hybrid Scheme - II ( $p_m = 0.075$ , Decay = 0.9) . . . . .	195
7.11 Results using Hybrid Scheme - II ( $p_m = 0.075$ , Decay = 0.7) . . . . .	196
7.12 Results using Hybrid Scheme - II ( $p_m = 0.025$ , Decay = 0.9) . . . . .	197
7.13 Results using Hybrid Scheme - II ( $p_m = 0.025$ , Decay = 0.7) . . . . .	198

# List of Abbreviations

ARMA	Auto Regressive Moving Average
DPA	Discretised Pursuit Algorithm
EP	Evolutionary Programming
ESs	Evolutionary Strategies
FIR	Finite Impulse Response
FSA	Fast Simulated Annealing
GAs	Genetic Algorithms
HARF	Hyperstable Adaptive Recursive Filter
IIR	Infinite Impulse Response
LMS	Least Mean Square
LRS	Linear Random Search
mGAs	Messy Genetic Algorithms
MSE	Mean Square Error
MSOE	Mean Square Output Error
PLR	PseudoLinear Regression
RLMS	Recursive Least Mean Square
RLS	Recursive Least Square
RPE	Recursive Prediction Error
SA	Stochastic Automaton
SHARF	Simple Hyperstable Adaptive Recursive Filter
SLA	Stochastic Learning Automata
SPR	Strictly Positive Real
VFSR	Very Fast Simulated Reannealing
VSSA	Variable Structure Stochastic Automaton

# Chapter 1

## Introduction

### 1.1 Why Adaptive Filtering ?

The term *filtering a signal* refers to processing the signal in such a manner, so as to extract relevant information from it. This could relate to enhancing certain desired components or on the other hand the removal of interfering noisy components. The earliest filters were usually of the analogue type. However the advent of digital electronics and the subsequent rapid developments in integrated circuit technology meant that digital filters were a cheaper and more reliable alternative to the conventional analogue filters. There are a number of advantages of digital filters over the analogue filters, these include easy modification of signal processing functions by means of software, higher order of precision and operational characteristics which remain stable over a wide range of conditions.

A digital filter operates with discrete samples of the input signal and is composed of adders, multipliers all implemented in digital logic. This results in a much better control over the accuracy of the operation than is possible in an analogue filter. In an analogue filter, tolerances in the components make it extremely difficult for a system designer to control the precision of the filter.

There are however many digital signal processing applications where the characteristics of a digital filter cannot be specified *a priori*. In such applications, the digital filter characteristics must be adaptable, so that the filter can adjust to different environments. This is achieved by using *adjustable coefficients* for the digital filter. Such

a filter is referred to as an *adaptive filter*. Conventional digital filtering operates in an open-loop fashion; the filter characteristics are fixed and there is no feedback from the output. Adaptive filters on the other hand function in a closed-loop fashion - the digital filter characteristics are modified by means of a feedback mechanism which monitors the output of the filter. The feedback mechanism uses an adaptive algorithm to modify the filter coefficients. The adaptive algorithm usually uses the input signal, the output signal and a reference signal to generate an *error signal* which is used in the feedback mechanism. This is illustrated in Figure [1.1] which shows both conventional and adaptive filter configuration.

Adaptive digital filtering can be achieved using either Finite Impulse Response (FIR) or Infinite Impulse Response (IIR) filters. In FIR filters, the output of the filter is a linear function of the delayed and current values of the input signal. These filters are well-behaved and are generally free of stability problems since as they possess only adjustable zeroes. However, to achieve a given degree of modeling accuracy, a high order FIR filter is required. This increases the computational load as the number of multiplications and additions are increased. The output of an IIR filter on the other hand is generated using a linear function of the delayed and current values of the input signal as well as *delayed values of the output signal*. Using an IIR filter results in a better model using a lesser number of coefficients than a FIR filter providing a similar performance. This is however countered by the fact that IIR filters possess adjustable poles as well as zeroes and thus are prone to stability problems caused by the migration of the poles during the adaptive process. More details of these issues are presented in Chapter 2.

The applications of adaptive filtering are many - the following table shows some important application areas:

Function	Applications
Equalisation	Telecommunications
Noise Cancelling	Medical Electronics, Aircraft cockpit communications
Multipath Compensation	Microwave Radio, TV ghost suppression
Stabilization	Space Applications
Modeling	Industrial control applications

Of the many configurations in which an adaptive digital filter may be used, two important configurations are the direct system modeling and the inverse system modeling configurations. They have been used in this thesis to simulate different applications using adaptive filters. In the direct system modeling configuration (Figure [1.2]), the adaptive filter produces an output signal  $\hat{y}(n)$ , which is an estimate of a desired response  $y(n)$ . In other words, the adaptive filter models the characteristics of the unknown filter. This configuration is used in applications such as adaptive noise cancellation. Inverse system modeling configuration, (Figure [1.2]), consists of the adaptive filter generating an output signal which is an estimate of the input signal  $x(n)$ . In such a configuration, the input signal is distorted by a process which is modeled by the unknown filter. The adaptive filter models the inverse of the unknown filter thereby restoring the degraded signal. This configuration has found use in applications such as adaptive equalisation. More details of both configurations are given in Chapter 2.

Thus the main motivation in studying adaptive digital filtering is that in real world applications, the characteristics of a system being modeled may be unknown and time varying. Using an adaptive filter makes it possible to model a large variety of systems under different operating conditions.

## 1.2 Outline of Thesis

The next chapter (Chapter 2) provides an in-depth review of adaptive digital filtering and especially concentrates on adaptive IIR filtering algorithms. Brief details of the different alternative realizations used in the simulation experiments are presented. The manner in which the stability issue of high order IIR filters was handled using these alternative realizations are discussed. Two applications of adaptive IIR filtering - adaptive noise cancellation and adaptive equalisation, are explained. These have been used as testbeds in the research to demonstrate the efficacy of the proposed new approaches to adaptive filtering which have been examined in this thesis.

Chapter 3 and 4 explain the theory and applications of using Stochastic Learning Automata algorithms (SLA) for adaptive IIR filtering. The basic theory and the learning algorithms are covered in Chapter 3. Both the P-Model and S-Model schemes are examined in detail. A new normalisation scheme for the S-Model algorithms is proposed and from the simulation results is shown to perform better than the standard S-Model normalisation schemes. A brief mention is made of the automata games approach and a scheme of hierarchical automata. The original reason for using the automata approach for the problem of adaptive filtering was that the technique had shown capability of global optimisation when searching a noisy, stochastic multimodal surface. The results of using the automata algorithms are presented in Chapter 4. The simulation configuration is explained as well as the manner in which an automaton is used to optimise the parameters for the adaptive filtering problem. The advantages and shortcomings of each learning scheme is detailed. An explanation is given why the S-Model learning algorithms performed poorly as compared to P-Model schemes. The chapter concludes with a discussion on the viability of Stochastic Learning Automata as a tool for adaptive digital filtering. Although the SLA algorithms provide a powerful set of results, their utility for adaptive filtering is limited, mainly due to the fact that the iterations required for convergence when adapting a high order filter is very large and impractical.

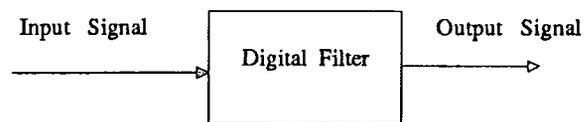
Thus a new approach, especially one in which dimensionality was not a hindering factor, was examined. This new scheme can be broadly classified as evolutionary op-

timisation, though three specific paradigms of evolutionary optimisation were examined. Chapter 5 presents a detailed overview of the technique of simulated evolution used as an optimisation tool. The different paradigms covered include genetic algorithms, evolutionary strategies and evolutionary programming. The basic algorithms are explained along with improved schemes which result in a better performance. Chapter 6 presents the use and results of the evolutionary optimisation schemes for adaptive IIR filtering, concentrating on the use of genetic algorithms. Two practical applications of adaptive IIR filtering - adaptive noise cancellation and adaptive equalisation, are simulated with the evolutionary strategy being used as the adaptive algorithm.

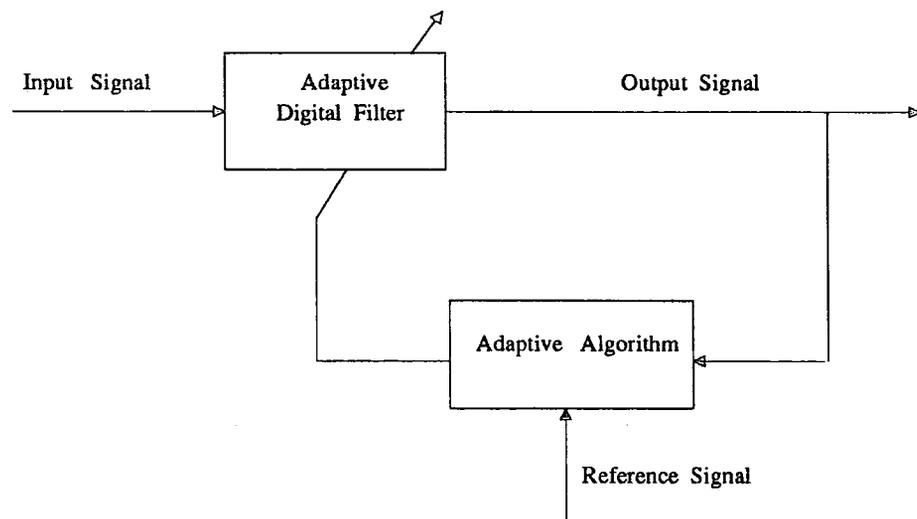
Some limitations of the evolutionary schemes were observed during the simulation studies. One of these, was the fact that there was no established stopping criterion which could be used to terminate further iterations. This led to an attempt, where the behaviour of evolutionary schemes was modified by incorporating concepts from other established optimisations algorithms. Specifically the optimisation strategy of simulated annealing was used.

Chapter 7 presents the theory and results obtained in using the simulated annealing approach for adaptive IIR filtering. Both the classical annealing approach and the more recent fast annealing approach are applied to the adaptive IIR filtering problem. Results obtained using the annealing approaches show that although the method was able to locate the exact global optimum, the time samples required for convergence was very large, thus reducing the practical use of the scheme. Two new schemes are proposed which combine concepts of *genetic algorithms and simulated annealing*. The motivation behind these schemes was to use the convergence speed of the evolutionary schemes and a stopping criterion derived from the annealing algorithm. Thus, these schemes present a stopping criterion for genetic algorithms which otherwise were stopped by unsatisfactory heuristic methods.

Chapter 8 presents the overall conclusions for the research. The main results of all the different approaches used for the filtering problems are compared. Finally a discussion is provided of promising areas for future research.

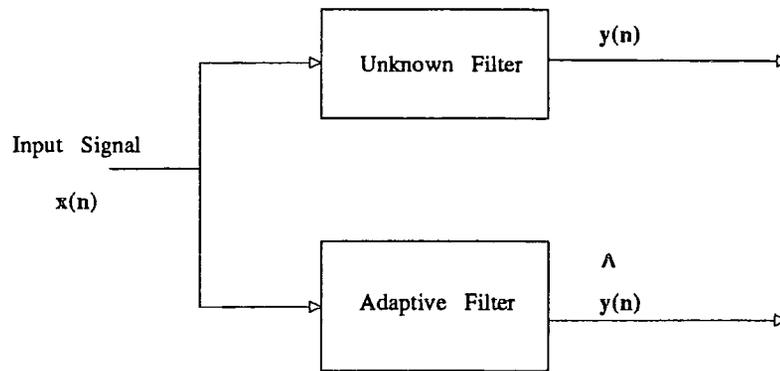


**Conventional Digital Filtering**

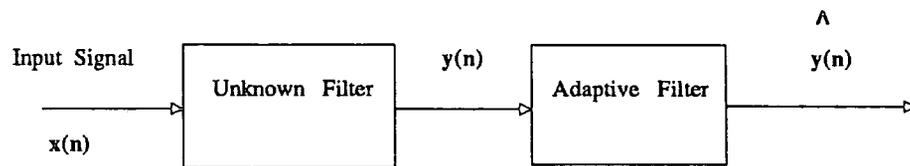


**Adaptive Digital Filtering**

Figure 1.1: Conventional and Adaptive Filtering Configurations



**Direct System Modeling**



**Inverse System Modeling**

Figure 1.2: Direct and Inverse System Modeling Configurations

## Chapter 2

# Adaptive Digital Filtering

### 2.1 Introduction

This chapter gives a broad overview of adaptive digital filtering concentrating more on adaptive IIR filtering. The interest and research in adaptive filtering can be gauged from the large number of books [TJL87, SD88, HM84, WS85, CG85, Ale86] which have been published on the subject. The basic direct form configuration is discussed along with the alternative realizations. Different error formulations used for adaptive IIR filtering and the limitations of the existing adaptive algorithms are detailed and discussed.

Digital filters have found extensive applications in many diverse areas of engineering such as communications, control, signal processing etc. [WS85, PM88]. The attractive feature of digital filters is their availability as dedicated signal processing hardware in the form of integrated circuits. A digital filter operates with discrete samples of the signal and is mainly composed of adders, multipliers and delays all implemented in digital logic. The main advantages of using digital filters are thermal stability, precision and adaptability.

The fundamental equation describing the input-output relationship of a general digital filter is given by

$$y(n) = \sum_{i=0}^M b_i \times x(n-i) + \sum_{j=1}^N a_j \times y(n-j) \quad (2.1)$$

where

$y(n)$	$\equiv$	Output sample at instant $n$
$x(n)$	$\equiv$	Input sample at instant $n$
$x(n-i)$	$\equiv$	Input sample delayed by $i$ time samples
$y(n-j)$	$\equiv$	Output sample delayed by $j$ time samples
$a_j$	$\equiv$	Feedback filter coefficients
$b_i$	$\equiv$	Feedforward filter coefficients

The equivalent block diagram is shown in Figure (2.1).

An equivalent form of Equation [2.1] is given below:

$$y(n) = B(n)x(n) + A(n)y(n) \quad (2.2)$$

where

$$B(n) = \sum_{i=0}^M b_i z^{-i}$$

$$A(n) = \sum_{j=1}^N a_j z^{-j}$$

where

$$z^{-1} \equiv \text{Unit delay operator}$$

$$\text{i.e. } x(n) z^{-1} = x(n-1)$$

As shown in Equation [2.1], the output  $y(n)$  can be regarded as an autoregressive moving average (ARMA) process driven by the input  $x(n)$ . The coefficients  $a_j$ ,  $b_i$  determine the characteristics of the filter.

Digital filters can be classified into two main groups:

- Finite Impulse Response (FIR) Filters
- Infinite Impulse Response (IIR) Filters

The equation describing an IIR filter is given by Equation [2.1], while the block diagram is as shown in Figure (2.1). The infinite nature of the impulse response of an IIR filter is because of the dependence of the output  $y(n)$  on previous output samples as shown in Equation [2.1]. As a result of this recursion, the stability of the filter is guaranteed only under certain conditions and forms an important issue in the analysis and design of adaptive IIR algorithms.

The output of an FIR filter is dependent only on the past and current input samples and is given by

$$y(n) = \sum_{i=0}^M b_i \times x(n - i) ; \quad (2.3)$$

This form can be obtained from Equation [2.1] by equating coefficients  $a_j$ 's to zero. Similarly the block diagram of a FIR filter can be obtained from Figure [2.1] by making the feedback coefficients  $a_j$ 's equal to zero.

The main advantage of an IIR filter over a FIR filter is that, as an IIR filter requires considerably fewer coefficients to model a system than an equivalent FIR filter, there is a significant saving in the computational overheads. For the same number of coefficients, an IIR filter can provide better performance. A desired frequency response can be better approximated by a filter possessing both poles and zeroes (IIR filter) than a filter having only zeroes (FIR filter). This is another significant advantage in using IIR filters in place of FIR filters.

An important feature of digital filters which has been mentioned before is that of adaptability. This property is significant when the operating environment of the filter is changing and the filter has to modify its behaviour in order to track the change. The filter which is used in such a situation is called an adaptive digital filter. In such a filter composed of either an IIR filter or a FIR filter, the coefficients  $a_j$  and  $b_i$  are variable and can be altered until the output satisfies a specified criteria. A block diagram of an adaptive digital filter is shown in Figure [2.2] [Shy89a]. It consists of the following :

- A FIR or IIR filter with adjustable coefficients  $\Theta(n)$ .
- An adaptive algorithm to adjust the coefficients so that the output  $y(n)$  ap-

proximates a desired response  $d(n)$ .

Thus the adaptive filtering problem can be succinctly expressed as: *Given  $x(n)$  and  $d(n)$ , the coefficients of the adaptive filter have to be chosen such that a performance measure based on the estimation error is minimised.* The estimation error  $e(n)$  (Figure [2.2]) is defined as

$$e(n) = d(n) - y(n) \quad (2.4)$$

A commonly used configuration in adaptive control is the system identification configuration in which an adaptive system is used to model an unknown system. This configuration is also frequently used in adaptive signal processing. Thus, the adaptive digital filtering problem using the system identification configuration (Figure [2.3]) is as follows: The input signal is applied both to the unknown system and the adaptive system. The unknown system output forms the desired response for the adaptive system, which uses the estimation error as defined in Equation [2.4] above to update its coefficients. In most applications there is the presence of additive measurement noise which is shown in (Figure [2.3]) by  $v(n)$ . In the system identification configuration, the desired response  $d(n)$ , is generated by the same input  $x(n)$  which drives the adaptive system. Thus, some characteristics of the signal  $d(n)$  may be known if the properties of the driving signal  $x(n)$  is known. The desired response need not always be generated in this manner and depends upon the application in which the adaptive system is used. Thus the adaptive filtering problem can be cast as an optimisation problem, where a suitable function of  $e(n)$  is to be minimised.

A commonly used criterion in adaptive filtering is to minimise the **Mean Square Output Error  $\Psi$**  which is defined as

$$\Psi = \mathbb{E}[e^2(n)] \quad (2.5)$$

where

$\mathbb{E}$  = Statistical Expectation Operator.

Recursive algorithms using this criteria are referred to as **Stochastic Gradient** algorithms [Shy89a]. Another criteria which has been used frequently minimizes the

sum of the squares of the estimation error  $e(n)$ , i.e.

$$\Phi = \sum e^2(n) \quad (2.6)$$

These algorithms are referred to as the **Recursive Least Squares** algorithms. Adaptive algorithms effectively search a performance surface defined by the criterion used. The optimum set of coefficients are then the coefficients corresponding to the global minimum on the performance surface.

## 2.2 Adaptive FIR Filtering

In adaptive FIR filtering using the system identification configuration (Figure [2.3]), the adaptive filter is of the FIR type. The estimation error  $e(n)$ , which is the difference between the desired response and the output of the adaptive filter is used in the criterion to update the filter coefficients. The criterion usually used for adaptation is the minimization of the **Mean Square Estimation Error** which is defined as

$$\Psi(b_i) = E[e^2(n)] \quad (2.7)$$

where  $b_i$ 's are the set of coefficients of the adaptive FIR filter. It has been proved that the function  $\Psi$  is a *quadratic unimodal function* of the adaptive filter coefficients [WS85]. Thus there exists an unique set of coefficients of the adaptive filter at which the error reaches the minimum value which is the global minimum. This facilitates the use of powerful gradient algorithms which can converge to the optimum set of coefficients rapidly. In particular a commonly used stochastic gradient algorithm is the **Least Mean Square (LMS)** algorithm first proposed in [WH60]. Complete details of the **LMS** algorithm are given in [WS85].

Currently FIR filters are more practical to use and are widely used in adaptive filtering. The main reason for this is that since FIR filters contain only adjustable zeroes, it is free from the stability problems associated with filters having both poles and zeroes (IIR Filters). However, interest in using IIR filters as the adaptive filter has been increasing, prompted mainly by the reduced computational demands when

using an IIR filter.

## 2.3 Adaptive IIR Filtering

### 2.3.1 Introduction

The non-recursive nature of the FIR filter results in a heavy computational load when using adaptive FIR filters. Modeling a system with an IIR filter can be achieved to a higher degree of precision using a much lower order filter than an equivalent FIR filter. For example, a fifth order IIR filter requiring nine multiplications and eight additions matches an unknown system as well as a 64<sup>th</sup> order FIR filter requiring 64 multiplications and 63 additions. This has led to exploring the possibility of using IIR filters as the adaptive element and as a consequence research into adaptive IIR filtering algorithms has been quite intensive in the past decade. Though the algorithms relating to adaptive IIR filtering are not as thoroughly analysed and developed as adaptive FIR filtering algorithms, they nevertheless form a substantial set of results. Work in adaptive IIR filtering algorithms have been carried out by various researchers [SEA76, Whi75, Fei76, PAS80a, Joh79, TLJ78, LTJ80]. The main work which has been carried out in adaptive IIR filtering has concentrated on the issues of global optimality, stability and the rate of convergence of the adaptive algorithms. New algorithms have been devised which solve some of the problems stated above but are usually constrained by a set of conditions. Two important review papers which present the current results in adaptive IIR filtering are [Joh84, Shy89a]. Using an IIR filter as the adaptive element in an adaptive scheme has the following implications [CG85]:

- Feedback in the filter structure itself allows a low order filter to have a long duration impulse response.
- The IIR filter structure is not stable for all choices of coefficients, thus stability forms an important aspect in the analysis.

- o Use of gradient algorithms result in increased computational complexity than is the the case with FIR filters.
- o Presence of the poles in the filter structure complicates the convergence analysis.

The adaptive IIR filtering problem has been approached in two ways, the difference being the manner in which the estimation error (Equation [2.4]) has been formulated. This is explained in the next section.

### 2.3.2 Different Formulations of Estimation Error

#### Equation Error Formulation

The equation error approach has been used in adaptive control where it is referred to as the series-parallel model. The **Equation Error** approach was proposed in [Men73] and has been used for adaptive filtering [Goo83]. In this formulation, the feedback coefficients of the IIR filter are updated in an all-zero, non-recursive form which are then copied to a second filter which is implemented in an all-pole form as shown in Figure (2.4) [Shy89a]. Essentially this formulation is of the adaptive FIR filter type where the FIR filter has two inputs. This can be seen in Figure (2.5) which shows the setup when the equation error formulation is used in the system identification configuration [LTJ80]. With reference to Figure (2.4), the defining equation for the equation error approach is given by

$$y_e(n) = \sum_{i=0}^M b_i \times x(n-i) + \sum_{j=1}^N a_j \times d(n-j) ; \quad (2.8)$$

From Equation [2.8], it can be seen that the output  $y_e(n)$  is obtained from *delayed samples of the input  $x(n)$  and the desired response  $d(n)$*  and not from the past output samples  $y_e(n)$ . Thus the output  $y_e(n)$  is a linear function of the coefficients  $(a_j, b_i)$ . Hence gradient calculations are simplified when using gradient-based algorithms. The equation error is given by

$$e_e(n) = d(n) - y_e(n) \quad (2.9)$$

as is shown in Figure (2.4). Expanding the above equation and using Equation [2.2], the equation error can be written as

$$\begin{aligned}
 e_e(n) &= d(n) - y_e(n) \\
 &= d(n) - [(A(n)d(n) + B(n)x(n)] \text{ (see footnote}^1\text{)} \\
 &= [d(n)(1 - A(n)) - [B(n)x(n)] \tag{2.10}
 \end{aligned}$$

Thus as  $e_e(n)$  is generated using the difference between two expressions/equations, it is referred to as the equation-error formulation. Since the equation error  $e_e(n)$  is a linear function of the filter coefficients, the Mean Square Output Error (Equation [2.5]) is a quadratic function of the filter coefficients with a single global minimum. Thus the performance of the equation error adaptive IIR filter is similar to the adaptive FIR filter especially with respect to the convergence and stability of the coefficient updates. However the limitation of the equation error approach is that in the presence of measurement noise which is invariably present (Figure [2.3]), the algorithm converges to a solution that is biased away from the true values. In a system identification context, this corresponds to incorrect estimates of coefficients  $\theta$  such that  $E[\theta(n)] = \theta_* + \text{bias}$  in the limit  $n \rightarrow \infty$  where  $\theta$  is the coefficient vector and  $\theta_*$  is the optimal set of coefficients of the adaptive filtering problem. It has been shown that this bias is eliminated if the measurement noise is zero. A numerical example regarding the effect of noise on the bias is given in [Shy89a].

### Output Error Formulation

This error formulation has also been used extensively in adaptive control and is referred to as the parallel model. The **Output Error** formulation is as shown in Figure [2.6] and is characterized by the recursive equation

$$y_o(n) = \sum_{i=0}^M b_i \times x(n-i) + \sum_{j=1}^N a_j \times y_o(n-j). \tag{2.11}$$

---

<sup>1</sup>Using the expression of  $y_e(n)$  from Fig. 2.4

The current output  $y_o(n)$  depends on the past output samples adding complexity to the adaptive algorithms. As shown in Figure [2.6], the output error is given by

$$e_o(n) = d(n) - y_o(n) \quad (2.12)$$

$e_o(n)$  is a nonlinear function of the filter coefficients. Thus the Mean Square Output Error need not be a quadratic function of the filter coefficients and can have multiple optima. This results in suboptimal performance when using gradient techniques as the algorithm could converge to a local optimum depending on the initial values of the coefficients. A specific numerical example is detailed in [JL77].

### 2.3.3 Adaptive Algorithms

This section presents a brief overview of adaptive IIR algorithms. The adaptive algorithms relating to adaptive IIR filtering are more involved and less complete than FIR filter adaptive algorithms. The two formulations of the estimation error explained above lead to adaptive algorithms with different characteristics. The equation error approach has been accepted widely as an alternative to the computationally intensive output error formulation but lead to biased estimates of the coefficient vector. However there exists an argument which suggests that the output error formulation is the *correct* approach as the adaptive filter is only operating on  $x(n)$  to generate  $y(n)$  which is the estimate of the desired response  $d(n)$ . On the other hand, the equation error approach uses the past values of the desired response  $d(n)$  as well as  $x(n)$  to estimate the current value of  $d(n)$ . The output error formulation has been adopted in all the simulation results presented.

A simplified form of an adaptive algorithm for IIR filters is as follows

$$\theta(\mathbf{n} + 1) = \theta(\mathbf{n}) - \mu(\mathbf{n})[\nabla_{\theta}\mathbf{J}(\theta(\mathbf{n}))] \quad (2.13)$$

where

$$\begin{aligned} \mu(\mathbf{n}) &\equiv \text{The parameter of the algorithm} \\ \nabla_{\theta}\mathbf{J}(\theta(\mathbf{n})) &\equiv \text{Gradient of the error function} \end{aligned}$$

The two popular classes of adaptive algorithms for IIR filtering are the Least Squares approach and Gradient Search algorithms. Least Square techniques use the input data samples recursively to minimize a least squares criterion. Detailed analysis of the least squares method is given in [Hay86]. Gradient based algorithms require the gradient at a point on the error surface to be measured, the next point searched being in the direction of the negative of the gradient. Two such algorithms are the Recursive Prediction Error (RPE) and the Recursive Least Mean Square (RLMS) algorithms [LS83, Shy89a]. These algorithms use an instantaneous values of the estimation error leading to noisy estimates of the gradient but result in asymptotically unbiased coefficients values. Another algorithm for adapting IIR filters is the Pseudolinear regression (PLR) algorithm which is a simpler version of the RPE algorithm derived by using an approximate expression for the gradient [Shy89a]. Development of fast algorithms for the gradient techniques have reduced much of the computational load.

The main problem with gradient techniques is suboptimal performance when dealing with multimodal error surfaces. The initial interest in adaptive IIR algorithms was sparked off by Feintuch in 1976 who suggested a simple algorithm for adapting IIR filter coefficients [Fei76]. This was a direct application of the FIR filter LMS algorithm on an IIR filter structure. However this algorithm was shown to converge to false minima by Johnson and Larimore [JL77] who also showed the Mean Square Output Error (MSOE) performance surface could be multimodal if the adaptive filter was of insufficient order with respect to the unknown system. This was later confirmed by Parikh and Ahmed [PA78] who showed the inability of the recursive LMS to identify a reduced order example proposed by them. Further work on adaptive IIR filters was carried out by Stearns [Ste81], who stated a unimodality conjecture for the system identification conditions. Söderström and Stoica [SS82] subsequently added to the set of conditions put forward by Stearns for an unimodal error surface. These conditions are as follows:

- The adaptive filter is of sufficient order to be able to model the unknown system.
- The input signal is white.

- o The order of the adaptive filter numerator exceeds that of the unknown system denominator.

The last condition was put forward by Söderström and Stoica. Fan and Jenkins [FJ86] proposed a new adaptive algorithm which has the characteristics of both the output error and equation error formulation. They used the system identification configuration and classified the error surfaces for such a configuration with a stationary stochastic setting into four cases depending on the order of the adaptive filter and the nature of the input excitation. These four cases are:

- o Class (I) : Sufficient Order Modeling - White Noise Input
- o Class (II) : Sufficient Order Modeling - Coloured Noise Input
- o Class (III) : Reduced Order Modeling - White Noise Input
- o Class (IV) : Reduced Order Modeling - Coloured Noise Input

It can be seen that both complexity and practical reality increase as we move down the above list. More recently extensive work has been done by Fan and Nayeri [FN89], wherein they proved Stearn's conjecture for first and second order filters even without Söderström and Stoica's additional constraint. They also showed that the MSOE error surface could be multimodal even when the adaptive filter was of sufficient order (Class (I)) or when the order is over estimated.

A different approach in designing adaptive IIR algorithms was based on the concept of *Hyperstability* and was detailed in [Joh79]. The resulting algorithm was referred to as the Hyperstable Adaptive Recursive Filter (HARF) algorithm. Hyperstability was a concept which was associated with the analysis of closed loop nonlinear time varying control systems [Pop73]. The algorithm had provable convergence properties but was computationally intensive especially for real time applications. This led to a simplified version of the algorithm referred to as the Simple Hyperstable Adaptive Recursive Filter (SHARF) algorithm [LTJ80]. The SHARF algorithm had convergence properties similar to HARF algorithm but under weaker conditions. A further constraint of this approach was that it relied on a *Strictly Positive Real (SPR)*

condition for global convergence. This condition effectively reduced the operating region of the adaptive filter by restricting the pole positions.

Random Search algorithms were another technique used to search performance surfaces. They made use of a random process to generate new points and made no assumptions about the nature of the error surfaces. This approach was used for FIR filtering [WM76], where the proposed linear random search (LRS) algorithm was compared to LMS. A whole chapter dedicated to different adaptive algorithms is given in [WS85].

All the adaptive algorithms detailed in this section use the direct form structure. A drawback with the direct form realization is the sensitivity of the structure to the quantization of the coefficients which would result in any implementation. Another shortcoming with the direct form approach is that the stability check involves additional computational overheads. As a result, alternative realizations which have been derived from the direct form configuration have been used extensively in all the simulation experiments conducted in this thesis and are detailed in the next section.

## 2.4 Alternative Realizations

The direct form realization of an IIR filter is as given by Equation [2.1] and is repeated here for ease of reference.

$$y(n) = \sum_{i=0}^M b_i \times x(n-i) + \sum_{j=1}^N a_j \times y(n-j)$$

Another possible way of characterizing the above class of systems is to use the transfer function approach. The transfer function for the above equation is given by

$$H(z) = \frac{\sum_{i=0}^M b_i z^{-i}}{1 - \sum_{j=1}^N a_j z^{-j}} \quad (2.14)$$

which is a ratio of two polynomials. From the above equation, the poles and zeroes of the system function  $H(z)$  can be obtained. The built-in feedback structure of the

IIR filter leads to problems of stability. This is especially true in the case of adaptive filters as during the adaptation one or more poles could move outside the unit circle in the  $z$ -plane resulting in an unstable filter. Thus adaptive algorithms need some form of stability check which may prove to be computationally expensive if it involves factorizing the denominator at each iteration. Another limitation of the direct form structures is the large sensitivities caused by the poles inadvertently slowing down the convergence rate. A way to resolve this problem is to decompose the direct form structure into alternative realizations like the parallel or cascaded forms which have lower coefficient sensitivities and arithmetic quantization effects. The parallel or cascaded realizations are composed of smaller order filters arranged in parallel or series which as a whole realize the transfer function given by Equation [2.14]. These realizations also allow easier implementation of the stability check.

A different alternative realization which does not directly follow from the direct form structure as given in Equation (2.1) is the lattice configuration. The advantage with the lattice configuration is that there exists a unique set of lattice coefficients for each direct form IIR filter. The stability check is also incorporated very easily in the adaptive lattice algorithms.

### 2.4.1 Parallel Form

A parallel form realization of an  $P^{th}$  order IIR filter can be obtained by performing a partial fraction expansion of  $H(z)$  as given in Equation [2.14]. This results in

$$H_p(z) = \sum_{i=1}^{P/2} H_i(z) \quad (2.15)$$

where

$$H_i(z) = \frac{b_{i0} + b_{i1}z^{-1} + b_{i2}z^{-2}}{1 + a_{i1}z^{-1} + a_{i2}z^{-2}} \quad (2.16)$$

The parallel form is usually composed of second order filters having the transfer function as given in Equation [2.16]. The use of second order sub-systems prevents the use of complex arithmetic as would be the case if first order filters were used. The stability check is incorporated by ensuring that the denominator coefficients of the

second order sub-system lie inside the stability triangle [Shy89a]. This realization is shown in Figure (2.7) when used in an adaptive filtering setup.

The instantaneous output error is the given by

$$e(n) = d(n) - \sum_{i=1}^{P/2} y_i(n) \quad (2.17)$$

Detailed analysis of the parallel form adaptive IIR filter is given in [NJ89, Shy89b]. In [Shy89b] a frequency domain implementation of the parallel form IIR filter is presented based on the discrete Fourier transform. The discussion includes a study of the MSOE surface and the convergence properties. In [NJ89], the different MSOE surfaces for alternative realizations like the parallel and cascade forms are examined and analysed. The main conclusions drawn from the analysis is that whenever a direct form IIR filter with a unimodal MSOE surface is transformed into an alternative realization using either a parallel or cascaded form, the MSE surface of the new structure may have additional stationary points which are either new equivalent minima or saddle points which are unstable solutions in the parameter space.

### 2.4.2 Cascade Form

The cascade form of Equation [2.14] is given by

$$H_c(z) = \prod_{i=1}^{P/2} H_i(z) \quad (2.18)$$

where  $H_i(z)$  is as given in Equation [2.16]. The analysis of the cascade form is similar to that of the parallel form and is given in [NJ89]. The computation of the gradient in the cascade form is more involved as the output of each section depends on the output of the previous sections. It has been shown that the cascade form has slower convergence rate than other realizations. A detailed analysis of the adaptive recursive filtering using the cascade form is presented in [TCC87].

### 2.4.3 Lattice Form

The lattice form has been used in adaptive signal processing for linear prediction and noise cancellation [Gri78, MV78]. Adaptive IIR filtering using the lattice form has been discussed in [Hor76, PAS80b]. A thorough exposition of the basic lattice structure is given in [CG85]. The main advantages of using the lattice structure are stability check by inspection, cascading of identical sections and good numerical round-off characteristics. The lattice form of a digital filter is entirely different from the forms which have been listed before. Each stage of a lattice structure is characterized by having a pair of input and output terminals. The lattice structure equivalent to a direct form filter given by Equation [2.14], is shown in Figure [2.8]. The algorithm to convert from a direct form filter to a lattice form is given in Appendix A.

An advantage over the parallel and cascaded form is the MSOE surface for the lattice configuration used in the adaptive filtering, does not possess any saddle points. Convergence properties of an adaptive lattice filter are similar to that obtained for a direct form filter [Shy87]. Some recent results regarding stable and efficient adaptive lattice algorithms are presented in [Reg92].

## 2.5 Applications of Adaptive IIR Filtering

To give a complete picture, the new approaches to adaptive IIR filtering have been tested in two important applications, both which use an adaptive IIR filter. These are adaptive noise cancelling and adaptive equalization. Adaptive noise cancelling as the term indicates, is used to remove the distortion from a signal which has been corrupted by extraneous noise sources and restore the signal to its original state. Previous work in these areas has been done with success using FIR filters, however the need for real time processing requires the use of IIR filters. Additive noise canceling has been used in a variety of engineering areas such as biomedical measurements and antenna beam-forming.

In modern telecommunications, the transmission of data over large distances is of vital importance. This is usually achieved using transmission lines or radio waves. Currently, digital transmission is becoming more prevalent, with the analogue

voice/data source being digitised at the source and then transmitted as a sequence of bits. At the receiver, these bits are then converted back to the analogue information. The main problem with this mode of transmission, is that during the transmission, the signals get corrupted and transformed. Corruption may occur due to addition of background thermal noise or impulse noise. Transformation usually occurs as a result of the finite bandwidth of the transmission channel and could be frequency translation or time dispersion. In a modem transmitter, a number of bits are encoded into symbols and transmitted. Due to the finite bandwidth of the transmission channels, the effect of each symbol extends beyond the time interval used to represent that symbol. The distortion caused by the resulting overlap is termed as intersymbol interference (ISI). Equalization is a broad term for techniques which overcomes this problem by compensating for them at the receiver end.

### 2.5.1 Adaptive Noise Cancelling

The simulation configuration to demonstrate the adaptive noise cancelling is taken from the paper by Larimore et. a. [LTJ80]. The setup is shown in Figure [2.9]. It is desired to estimate the signal  $s(n)$  which has been corrupted because of the additive *uncorrelated* noise process  $v1(n)$ . Thus the primary signal source denoted by  $z(n)$  is given by

$$z(n) = s(n) + v1(n) \quad (2.19)$$

To compensate for the noise  $v1(n)$ , usually a sensor is used which measures only the noise process as is shown at the top of Figure[2.9]. Thus a reference measurement,  $v2(n)$  is available, which is correlated to original noise process  $v1(n)$ . By means of proper filtering, the configuration in Figure [2.9] should be able to reduce the interference caused by the noise process and provide a good estimate of the signal  $s(n)$ . As is shown in Figure [2.9], the system identification configuration has been employed. This setup could be rearranged as an output error identifier as has been shown in [LTJ80]. Then, minimising the mean square output error, leads to the cancellation of the *correlated signals* which are present in  $y(n)$  and  $z(n)$ . Since, it is the noise component of two signals  $y(n)$  and  $z(n)$  which are correlated, it gets

cancelled, resulting in output error approaching the undistorted signal  $s(n)$ . This fact is of paramount importance, because if the original signal  $s(n)$  is in some manner correlated to the noise process  $v(n)$ , then the output error identifier would lead to the cancellation of the desired signal itself.

### 2.5.2 Adaptive Equalization

In modern digital communication, data is transmitted using analogue channels. As a result of the finite bandwidth of the channel, the transmitted signals are invariably distorted. Once such form of distortion is intersymbol interference caused as a result of time dispersion or multipath effects. To overcome the effects of this distortion, the received signals are passed thorough an equalizer which compensate for the distortion and recovers the original symbols which were transmitted. One widely used form for the equalizer has been the *linear transversal equalizer* which is in effect an FIR filter. It has been shown however that this kind of structure is not suitable for non-minimum phase channel compensation.

The system shown in Figure [2.10] is used for the experimental configuration. The input signal  $x(n)$  is modeled using an independent binary random sequence, the bits being represented by +1 and -1. The effect of the channel are modeled using a FIR filter with real coefficients. The output of this filter is given by

$$\begin{aligned} y(n) &= a_0x(n) + a_1x(n-1) + \dots + a_Mx(n-M) \\ &= \sum_{l=0}^M a_lx(n-l) \end{aligned} \quad (2.20)$$

where  $(a_0, \dots, a_M)$  are the coefficients of the FIR filter which models the transmission channel characteristics. The additive noise  $v(n)$  is of unity power and zero mean. Thus the signal which is presented to the equalizer is the noise corrupted signal  $\hat{y}$ . The function of the equalizer is to use the values of  $\hat{y}(n), \dots, \hat{y}(n-K)$  to produce the best estimate of  $x(n)$ , where  $K$  is the order of the equalizer. In most cases, because of the non-minimum phase characteristics of equalizer only a delayed estimate of the original sequence  $x(n)$  is obtained. More details of the implementation are given in

Chapter 6, where the evolutionary algorithm is used for adaptive equalization.

## 2.6 Discussion

This chapter presented an overview of adaptive digital filtering and in particular adaptive IIR filtering. Adaptive FIR filtering is a mature field with well analysed algorithms with respect to rate of convergence and optimality. However the area of adaptive IIR filtering is still evolving. The main limitations of the current adaptive IIR algorithms are either the computational complexity or the failure of the algorithm when dealing with multimodal error surfaces. A problem which arises when modeling high-order IIR filters is one of stability. Ensuring stability of the IIR filter kernel for all choices of filter coefficients is computationally expensive. Other adaptive techniques like random search algorithms have been used to solve this problem but have not given encouraging results. In the next chapter we present a different approach which is based on Stochastic Learning Automata. Stochastic Learning Automata are techniques which make use of probabilistic transitions and have been shown by simulations to exhibit global optimality.

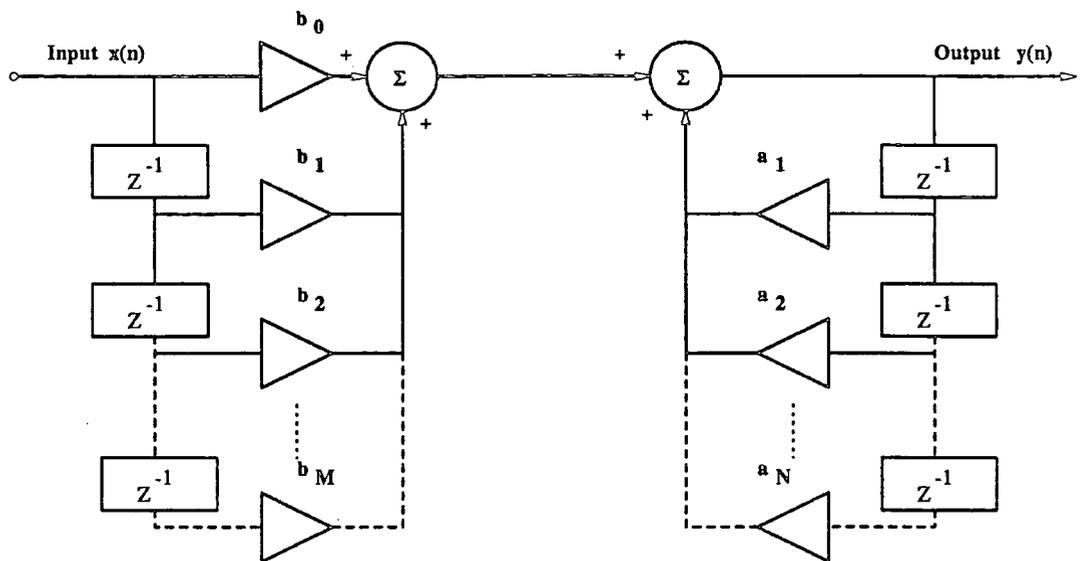


Figure 2.1: Digital Filter

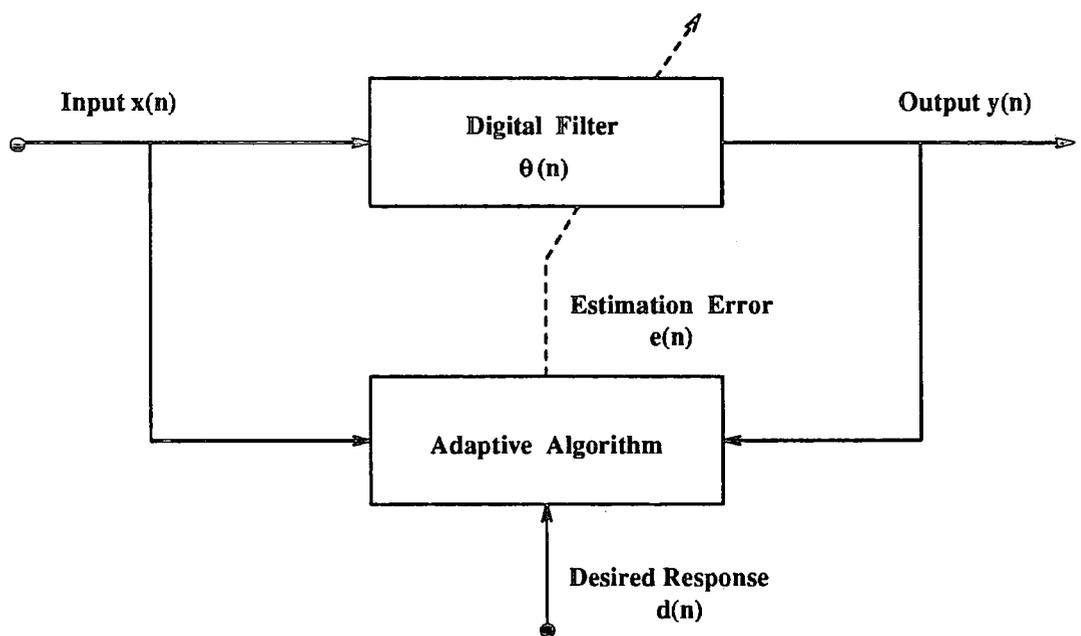


Figure 2.2: Adaptive Digital Filter

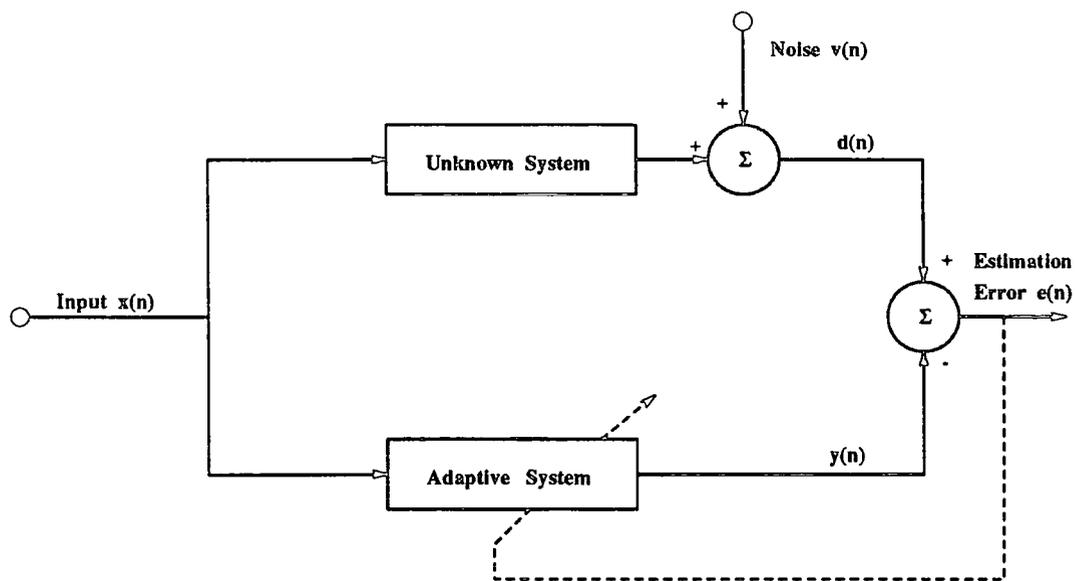


Figure 2.3: System Identification Configuration

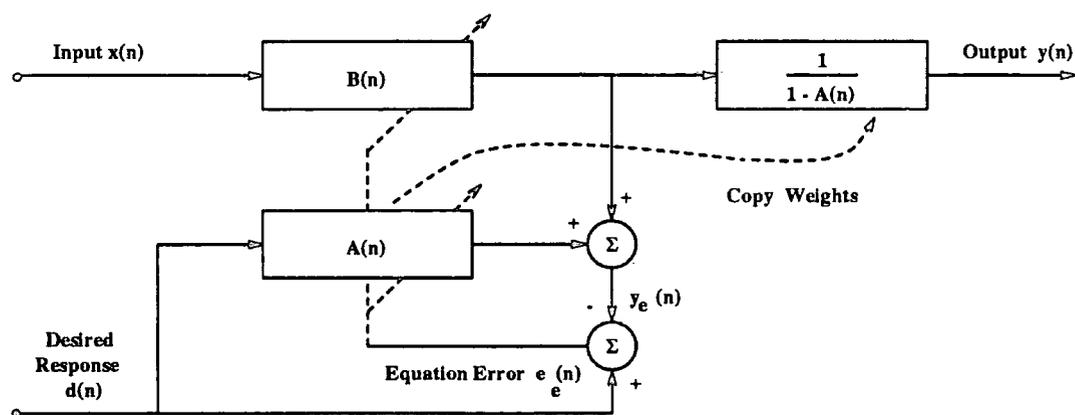


Figure 2.4: Equation Error Formulation

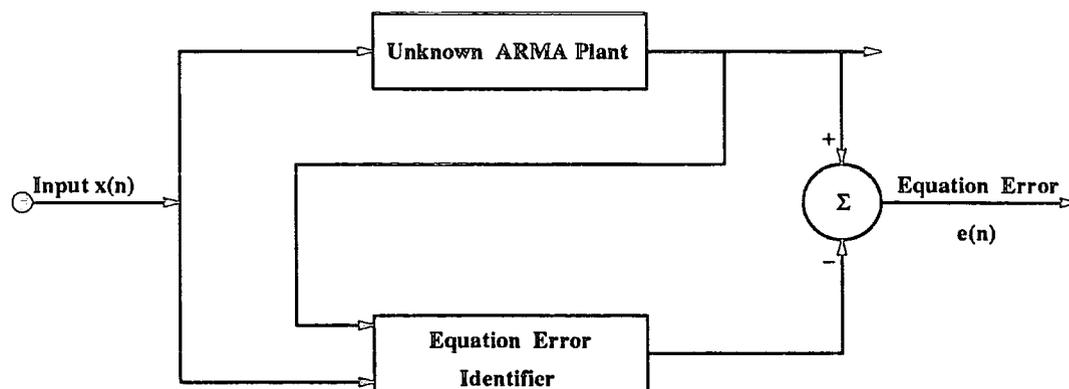


Figure 2.5: Equation Error Identifier

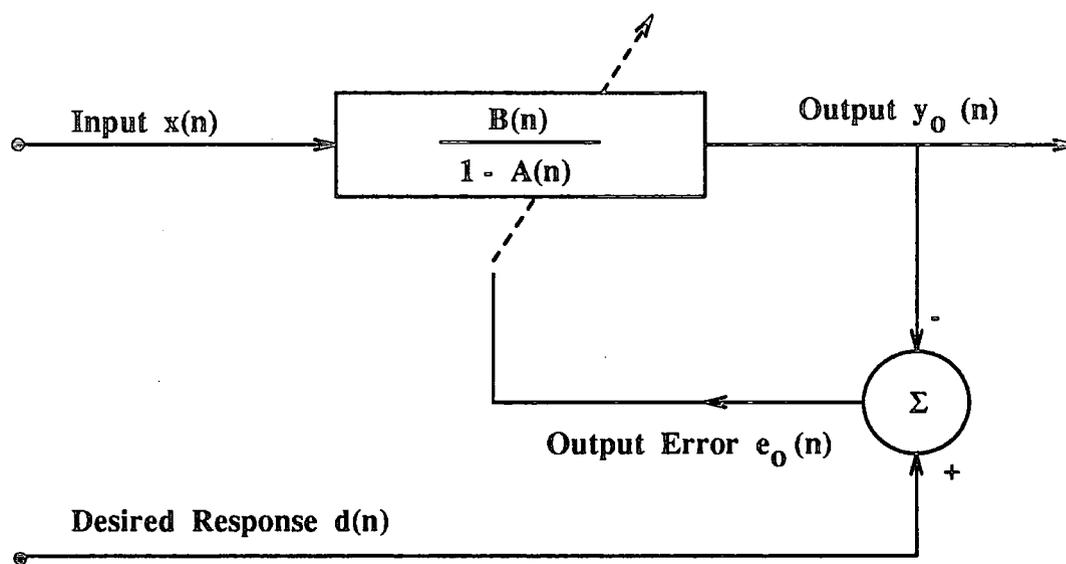


Figure 2.6: Output Error Formulation

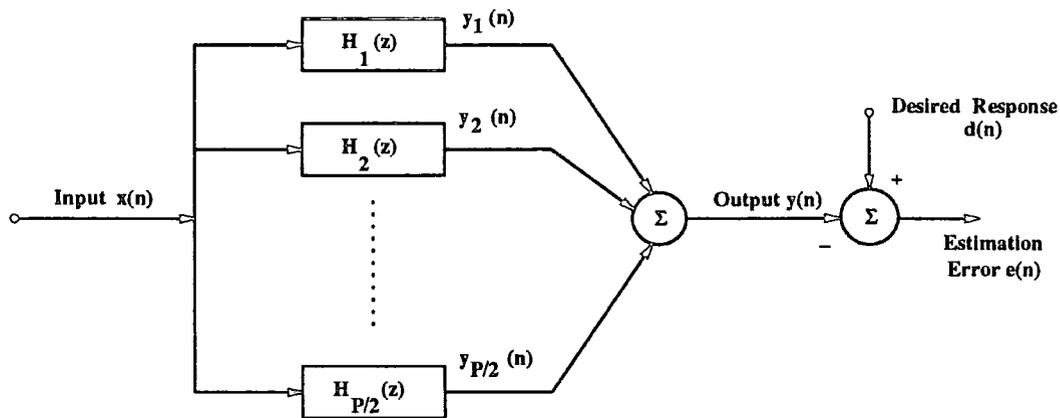


Figure 2.7: Parallel Form Realization

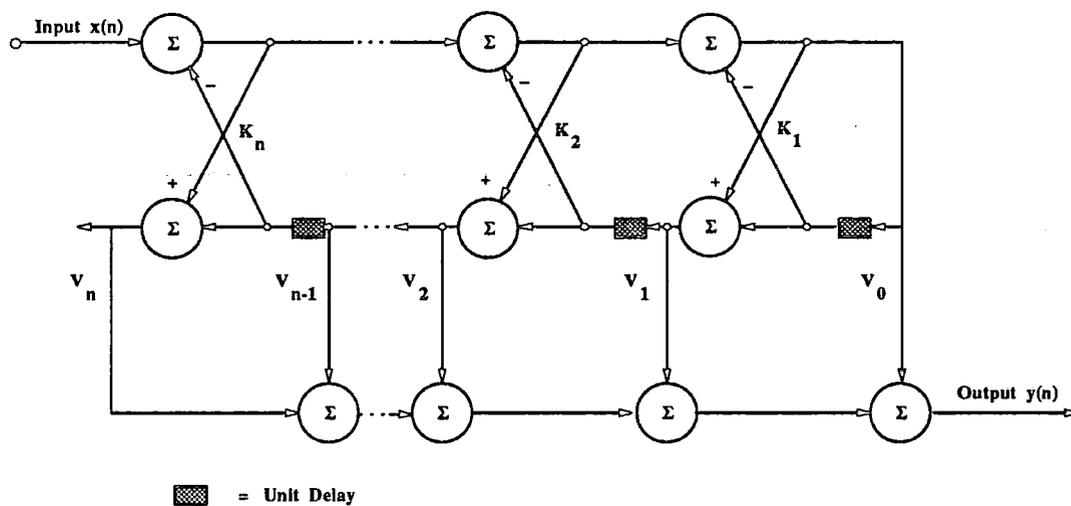
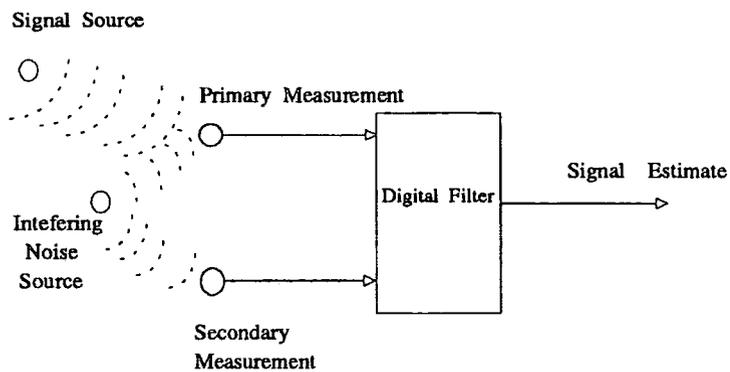
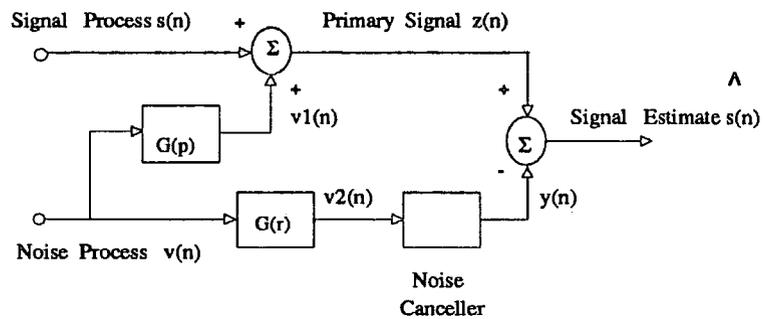


Figure 2.8: Lattice Form Realization



A) Physical Model



B) Lumped Model

Figure 2.9: Adaptive Noise Canceling Configuration

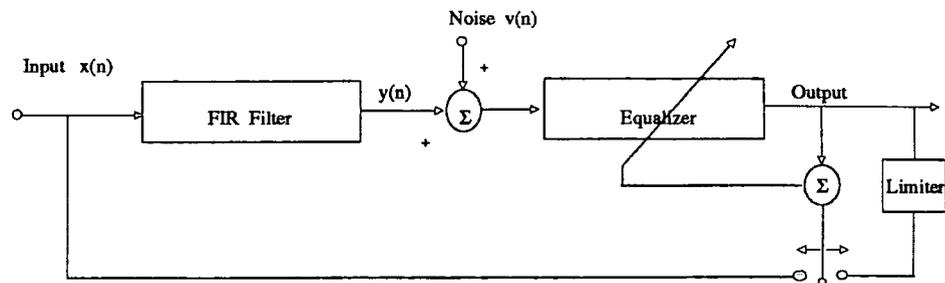


Figure 2.10: Adaptive Equalization Configuration

## Chapter 3

# Stochastic Learning Automata

### 3.1 Introduction

The process by which biological organisms *learn* has been a fascinating area of research for well over a century. The focus of research has been mainly two pronged - to understand the principles involved during the learning process of biological systems and to develop methodologies whereby these principles could be incorporated into machines. Learning can be regarded as a change brought about in a system performance as a result of past experience [NT89]. An important characteristic of a *learning system* is its ability to improve its performance with time. In a strictly mathematical context, the goal of a learning system can be said to be the optimization of a functional which may not be known completely. Thus an approach to this problem is to reduce the objective of the learning system to an optimization problem defined on a set of parameters and use established techniques to arrive at the optimal set of parameters. This chapter is concerned with the learning methods based on *Stochastic Learning Automata*.

The concept of *Stochastic Automata* was first introduced by the pioneering work of Tsetlin in the early 1960s in the Soviet Union who was interested in the modeling of the behaviour of biological systems [Tse62]. Subsequent research has considered the use of the learning paradigms in engineering systems. This has led to extensive work using automata as models of learning with applications in telephone routing, pattern recognition, object partitioning and adaptive control [NT74, Lak81, NT89,

OM88, SN69, FM66]. A *Learning Automata* could be regarded as an abstract object having a finite number of actions. It operates by selecting an action from a finite set of actions which is then evaluated by a random environment. The response from the environment is used by the automaton to select the next action. By this process, the automaton learns asymptotically to select the optimal action. The manner in which the automaton uses the response from the environment to select its next action is determined by the specific learning algorithm used. The next section gives details of the components of a *Stochastic Learning Automata*.

### 3.2 Stochastic Learning Automata

A Stochastic Learning Automaton (SLA) comprises of two main building blocks:

- A Stochastic Automaton with a finite number of actions and a Random environment with which the automaton interacts.
- The Learning Algorithms by which the automata *learns* the optimal action.

#### 3.2.1 Stochastic Automata

An Automaton can be regarded as a finite state machine. Mathematically it can be described by a quintuple

$$SA \equiv \{ \alpha, \beta, F, G, \phi \} \tag{3.1}$$

where

- $\alpha$      $\equiv$   $\{ \alpha_1, \alpha_2, \dots, \alpha_r \}$      $\equiv$  Set of Actions of the Automaton ;
- $\beta$      $\equiv$   $\{ \beta_1, \beta_2, \dots, \beta_r \}$      $\equiv$  Set of Inputs to the Automaton ;
- $F$      $\equiv$   $\phi \times \beta \longrightarrow \phi$      $\equiv$  Function which maps current state and input into next state ;
- $G$      $\equiv$   $\phi \longrightarrow \alpha$      $\equiv$  Output function mapping the current state into the next output ;
- $\phi(n)$   $\equiv$   $\{ \phi_1, \phi_2, \dots, \phi_k \}$      $\equiv$  Set of Internal states of the Automaton at time  $n$  ;

The set  $\alpha$  forms the output set of the automaton, the automaton selecting one of the  $r$  actions at each iteration. The input set  $\beta$  defines the input to the automaton and is explained in the next section. The mappings  $F$  and  $G$  transform the current state and input to the next output(action) chosen by the automaton. When the mappings  $F$  and  $G$  are deterministic, the automaton is referred to as *Deterministic Automaton*. In such a case, given the initial state and input, the next state and output are uniquely specified. When the mappings  $F$  and  $G$  are stochastic, the automaton is referred to as a *Stochastic Automaton*. In this case only probabilities associated with the next states and outputs are specified. Stochastic Automata can be further classified into *Fixed Structure* and *Variable Structure* automata. In a fixed structure stochastic automaton, the probabilities associated with the different actions are fixed, while in a variable structure stochastic automaton (VSSA) the probabilities are updated at each iteration  $n$ . The internal state of the automaton  $\phi$  is represented by the action probabilities of the actions of the automaton. For mathematical simplicity it is assumed that each internal state corresponds to an unique action. Thus the internal state of the automaton  $\phi$  is replaced by the action probability vector  $\mathbf{p}$  which is defined as

$$\mathbf{p}(n) \equiv \{p_1(n), p_2(n), \dots, p_r(n)\} \quad (3.2)$$

where

$$p_i(n) = Prob[\alpha(n) = \alpha_i] \quad (3.3)$$

and

$$\sum_{i=1}^r p_i(n) = 1; \forall n. \quad (3.4)$$

Defining the simplex

$$S = \left\{ p \mid p_i \geq 0, \sum_{i=1}^r p_i = 1 \right\} \quad (3.5)$$

we have  $\mathbf{p}(n) \in S ; \forall n$ . Initially all the action probabilities are set equal to one another, i.e.

$$p_i = 1/r \tag{3.6}$$

where  $r$  is the number of actions of the automaton.

### 3.2.2 The Environment

The random environment can be mathematically described by a triple

$$E \equiv \{\alpha, \beta, c\} \tag{3.7}$$

where

$$\begin{aligned} \alpha &\equiv \{ \alpha_1, \alpha_2, \dots, \alpha_r \} && \equiv \text{Set of inputs ;} \\ \beta &\equiv \{ \beta_1, \beta_2, \dots, \beta_r \} && \equiv \text{Set of outputs ;} \\ c &\equiv \{ c_1, c_2, \dots, c_r \} && \equiv \text{Set of penalty probabilities ;} \end{aligned}$$

The input of the environment is one of the  $r$  actions selected by the automaton. The output(response) of the environment to each action  $i$  is given by  $\beta_i$ . When  $\beta_i$  is a binary response, the environment is said to be the **P-Model** type. In such an environment,  $\beta_i(n) = 1$  is taken as a failure while  $\beta_i(n) = 0$  is taken as a success. This notation is purely due to convention. In the **Q-Model** environment,  $\beta_i(n)$  can take a finite number of values between  $[0,1]$ , while in the **S-Model**  $\beta_i(n)$  is a random variable between  $[0,1]$ , i.e.  $\beta_i(n) \in [0, 1]$ . The set  $c$  of penalty probabilities characterize the environment and is defined as

$$c_i = Prob[\beta(n) = 1 \mid \alpha(n) = \alpha_i]; \quad i = \{1, 2, \dots, r\} \tag{3.8}$$

i.e. the probability that the action  $\alpha_i$  would result in an unfavourable response from the environment. The values of  $c_i$  are unknown and it is assumed that  $\{ c_i \}$  has a unique minimum. The environment could also alternatively be characterized by a set of reward probabilities which would represent the probability that a particular

action elicits a favourable response from the environment. When dealing with stationary environments, the penalty probabilities are constant, while in a non-stationary environment the penalty probabilities vary with time.

The connection of the Stochastic Automata and the Environment in a feedback arrangement as shown in Figure (3.1), together with the *Learning Algorithms*, form the Stochastic Learning Automata. Thus a Stochastic Learning Automata can be formally described by a quintuple

$$SLA = \{ \alpha, \beta, p, T, c \} \tag{3.9}$$

where

- |   |   |
|---|---|
| $\alpha \equiv \{ \alpha_1, \alpha_2, \dots, \alpha_r \}$ | $\equiv$ Set of outputs of Automaton /<br>Set of inputs to the Environment.     |
| $\beta \equiv \{ \beta_1, \beta_2, \dots, \beta_r \}$     | $\equiv$ Set of inputs to the Automaton /<br>Set of outputs of the Environment. |
| $p \equiv \{ p_1, p_2, \dots, p_r \}$                     | $\equiv$ The probability vector.  |
| $T \equiv p(n+1) = T[\alpha(n), \beta(n), p(n)]$          | $\equiv$ The learning algorithm.  |
| $c \equiv \{ c_1, c_2, \dots, c_r \}$                     | $\equiv$ Set of penalty probabilities<br>defining the Environment.              |

As stated before, for mathematical ease, every internal state of the automaton corresponds with an unique action (output) of the automaton. Thus the function  $G$  (Equation [3.1]) reduces to an identity mapping. The function  $F$  (Equation [3.1]) of the stochastic automaton is replaced by the learning algorithm  $T$ , which determines the next action of the automaton. The learning algorithms are of vital significance to the operation of the SLA and are examined in detail in a subsequent section (section [3.3]).

### 3.2.3 Norms of Behaviour

To quantify the performance of the SLA, certain measures have been defined which determines the effectiveness of the automaton and enables the comparison of different learning schemes [NT89]. A *pure-chance Automaton* is defined as one in which every action is equally likely to be picked. Thus an automaton which is said to *learn* must perform better than the *pure-chance automaton*.

As stated before, the random stationary environment is represented with penalty probabilities  $\{c_1, c_2, \dots, c_r\}$ , where  $c_i$  is the penalty probability corresponding to action  $\alpha_i$ . A quantity  $M(\mathbf{n})$  is defined as the average penalty received by the automaton for a given action probability vector and is given by

$$\begin{aligned} M(\mathbf{n}) &= E[\beta(\mathbf{n}) \mid \mathbf{p}(\mathbf{n})] \\ &= \sum_{i=1}^r c_i p_i(\mathbf{n}) \end{aligned} \quad (3.10)$$

For a pure-chance automaton, the average penalty  $M(\mathbf{n})$  is a constant  $M_o$  and is given by

$$M_o = \frac{1}{r} \sum_{i=1}^r c_i \quad (3.11)$$

For an automaton to perform better, its average penalty must be less than  $M_o$  at least asymptotically. Since  $M(\mathbf{n})$  is a random variable, the expected value of  $M(\mathbf{n})$ , i.e.  $E[M(\mathbf{n})]$ , is compared with  $M_o$ . Thus we have the following definitions:

- **Definition A:** A learning automata is said to be *expedient* if

$$Lt_{n \rightarrow \infty} E[M(\mathbf{n})] < M_o \quad (3.12)$$

- **Definition B:** A learning automata is said to be *optimal* if

$$Lt_{n \rightarrow \infty} E[M(\mathbf{n})] = c_l \quad (3.13)$$

where  $c_l = \min_i \{c_i\}$ . While optimality is a desirable feature in a stationary en-

vironment, in a practical situation a sub-optimal performance may be required<sup>1</sup>  
 Thus we have

- **Definition C:** A learning automata is said to be  $\epsilon$ - *optimal* if

$$Lt_{n \rightarrow \infty} E[M(n)] < c_l + \epsilon \tag{3.14}$$

is realized for any arbitrary  $\epsilon > 0$ .

- **Definition D:** A learning automata is said to be *absolutely expedient* [LT73] if

$$E[M(n + 1) | p(n)] < M(n) \tag{3.15}$$

$\forall n, \forall p_i(n) \in (0, 1)$  and for all possible sets  $\{c_i\}(i = 1, 2, \dots, r)$ .

*Expediency* merely demonstrates that the SLA performs better than a pure chance automata and thus a more desirable behaviour would be *optimality*. Optimality ensures that the optimal action is chosen by the automaton asymptotically and is desirable in a stationary environment. But in a practical situation, the environment is usually non-stationary and an  $\epsilon$ -*optimal* behavior is preferred as previously.

The type and performance of a SLA is characterized by the learning algorithm used. The next section reviews the various learning schemes which have been studied in the literature.

### 3.3 Learning Algorithms

#### 3.3.1 Standard Learning Algorithms

As shown in Equation [3.9], the learning algorithm **T** can be represented by

$$p(n + 1) = T[p(n), \alpha(n), \beta(n)] \tag{3.16}$$

---

<sup>1</sup>In a practical situation the environment is usually non-stationary and therefore the optimal action may change with time. A sub-optimal learning algorithm may be more suitable since the algorithm does not get locked into any particular state.

If operator  $\mathcal{T}$  is linear, the reinforcement (learning) algorithm is said to be linear, otherwise it is referred to as a non-linear scheme. The fundamental idea behind all learning algorithms are as follows: If the SLA selects action  $\alpha_i$  at iteration  $n$  and obtains a favourable response from the environment, the action probability  $p_i(n)$  is increased while the action probabilities of the other actions are decreased. For an unfavourable response,  $p_i(n)$  is decreased, while the other action probabilities are increased. Thus we have

o Favourable Response

$$\begin{aligned} p_j(n+1) &= p_j(n) - f_j[p_j(n)] ; \forall j; j \neq i \\ p_i(n+1) &= p_i(n) + \sum_{\substack{j=1 \\ j \neq i}}^r f_j[p_j(n)] \end{aligned} \quad (3.17)$$

o Unfavourable Response

$$\begin{aligned} p_j(n+1) &= p_j(n) + g_j[p_j(n)] ; \forall j; j \neq i \\ p_i(n+1) &= p_i(n) - \sum_{\substack{j=1 \\ j \neq i}}^r g_j[p_j(n)] \end{aligned} \quad (3.18)$$

The functions  $f_j$  and  $g_j$  are referred to as the reward and penalty functions respectively and are assumed to be non-negative functions. The above equations preserve the validity of Equation [3.4]. Linear learning algorithms have been studied extensively as they are mathematically more tractable. For a linear reinforcement algorithm with multiple actions, the functions  $f_j$  and  $g_j$  are given by [NT89]

$$f_j[p_j(n)] = ap_j(n) ; 0 < a < 1 ; \quad (3.19)$$

$$g_j[p_j(n)] = \frac{b}{r-1} - bp_j(n) ; 0 \leq b < 1 ; \quad (3.20)$$

where

$r \equiv$  Number of actions of the automaton

$a \equiv$  Reward Parameter

$b \equiv$  Penalty Parameter

Learning algorithms with different characteristics are obtained based on the relative values of the learning parameters  $a$  and  $b$ . Thus we have

- $L_{RP}$  Scheme: When  $a$  and  $b$  are equal to each other, we obtain the **Linear Reward Penalty ( $L_{RP}$ ) Scheme**.
- $L_{R\epsilon P}$  Scheme: When  $b$  is an order of magnitude less than  $a$ , the resulting learning scheme is called the **Linear Reward epsilon Penalty ( $L_{R\epsilon P}$ ) Scheme**.
- $L_{RI}$  Scheme: When the penalty parameter  $b$  is equal to zero, the scheme is referred to as the **Linear Reward Inaction ( $L_{RI}$ ) Scheme**.

Using the Equations [3.19] and [3.20] for the functions  $f_j$  and  $g_j$ , the general form of a learning algorithm is as follows: If at iteration  $n$  action  $\alpha_i$  is chosen, then at iteration  $(n + 1)$  we have

- **Favourable Response from Environment**

$$p_i(n + 1) = p_i(n) + a[1 - p_i(n)] ; \tag{3.21}$$

$$p_j(n + 1) = (1 - a)p_j(n) ; \forall j ; j \neq i \tag{3.22}$$

- **Unfavourable Response from the Environment**

$$p_i(n + 1) = (1 - b)p_i(n) ; \tag{3.23}$$

$$p_j(n + 1) = \frac{b}{r - 1} + (1 - b)p_j(n) ; \forall j ; j \neq i \tag{3.24}$$

The above equations give the general rule for the updating of the action probabilities. If in the above equations ( $a = b$ ) the  $L_{RP}$  scheme is obtained, while ( $b = 0$ ) results in the  $L_{RI}$  scheme. The  $L_{RP}$  scheme leads to expedient behaviour of the automaton, while both  $L_{RI}$  and  $L_{R\epsilon P}$  schemes result in  $\epsilon$ -optimal behaviour. Non-linear updating schemes have been pursued by researchers [VN70, LT72b, LT73], but gave no appreciable improvement over the linear updating schemes.

A crucial factor which limits applications involving SLA is their slow rate of convergence. This factor becomes more pronounced when the number of actions increase and the SLA has to update more action probabilities at each iteration. The next few sections present some new approaches which have been devised with the aim of improving the rate of convergence of the basic learning algorithm detailed above.

### 3.3.2 Discretised Learning Algorithms

Discretised Learning Algorithms are based on discretising the action probabilities and was first proposed in [TO79]. Such automata are discretised versions of their continuous counterparts. Discretisation involves restricting the values of the action probabilities to discrete values in the interval  $[0,1]$ . The discretisation is termed linear if the allowable values in  $[0,1]$  are equally spaced, otherwise it is called non-linear. The idea behind discretising the action probabilities is to allow the action probabilities to approach the limiting value of unity directly, rather than approach it asymptotically as is the case with the continuous algorithms. Thus the speed of convergence of the learning algorithm should increase significantly.

Another advantage of using discretisation is the minimization on the requirements on the system random number generator where the algorithm is applied. This fact is important as any implementations of SLA make use of random number generators. As a result of the finite precision of a computer system, only a finite number of values in the interval  $[0,1]$  can be obtained. Thus the precision of the continuous algorithm is limited by the random number generator of the system on which the algorithm is implemented. Theoretical results involving discretised automata were proved in [OH84, OC88]. For a two action automaton with actions  $\alpha_1$  and  $\alpha_2$ , the probability update equations are as follows: Suppose action  $\alpha_1$  was chosen at iteration  $n$ . Then

#### • Favourable Response

$$\begin{aligned} p_1(n+1) &= \text{Min}\{p_1(n) + \Delta, 1 - \Delta\} \\ p_2(n+1) &= \text{Max}\{p_2(n) - \Delta, \Delta\} \end{aligned} \tag{3.25}$$

o Unfavourable Response

$$\begin{aligned} p_1(n+1) &= \text{Max}\{p_1(n) - \Delta, \Delta\} \\ p_2(n+1) &= \text{Min}\{p_2(n) + \Delta, 1 - \Delta\} \end{aligned} \quad (3.26)$$

The parameter  $\Delta$  is referred to as the step-size and is given by

$$\Delta = \frac{1}{(rN)} \quad (3.27)$$

where  $r$  is the number of actions of the automata and  $N$  is the resolution parameter which forms the learning parameter of the discretised algorithm. The resolution parameter  $N$  determines the speed and accuracy of convergence of the algorithm. The *Max* and *Min* functions ensure the probabilities satisfy  $0 \leq p_i(n) \leq 1$  and also ensures the automaton does not have any absorbing states by preventing any of the action probabilities converging to '0' or '1'. Theoretical results regarding the convergence of the discretised algorithm are available only for the 2-action case, though it is conjectured that the results also hold for the multi-action case [OC88].

### 3.3.3 Estimator Algorithms

In the standard learning algorithms, the environment characterised by the penalty probability vector was assumed to be unknown. An improvement in the basic learning scheme could be to determine the characteristics of the environment as the learning proceeds. Estimator algorithms work precisely on this principle and maintain an estimate of the penalty probabilities as the learning proceeds. This added information is used when updating the action probabilities. The first instance of using the idea of estimating the penalty probabilities of the environment using Bayesian techniques was proposed in [LT72a]. But the main thrust of the approach has been carried out by Thathachar and Sastry in [TS85, TS86].

Nonestimator algorithms update the action probability vector solely based on the response from the environment. Thus if an action results in a favourable response from the environment, the probability of choosing that action is increased. Estimator

algorithms on the other hand maintain a running estimate of the probability of reward (penalty) for each action. When an action obtains a favourable response from the environment, the estimator algorithm updates the estimate of reward for that action. Then, the change in the action probability for that action is based on both the feedback from the environment and the running estimates of the reward probabilities. Thus in a estimator algorithm it is possible for the probability of an action to be decreased even when it has obtained a favourable response from the environment.

In nonestimator algorithms, the action probability vector  $\mathbf{p}$  is defined as the internal state of the automaton (Equation [3.2]). Estimator algorithms on the other hand also use the estimates of reward for each action, and thus the internal state of the automaton is generalized to  $\mathbf{Q}(\mathbf{n})$  where

$$\mathbf{Q}(\mathbf{n}) = \{\mathbf{p}(\mathbf{n}), \mathbf{d}(\mathbf{n})\} \quad (3.28)$$

where

$$\mathbf{d}(\mathbf{n}) = [\hat{d}_1(\mathbf{n}), \hat{d}_2(\mathbf{n}), \dots, \hat{d}_r(\mathbf{n})] \quad (3.29)$$

and  $\hat{d}_i$  is the estimate of reward for the  $i^{\text{th}}$  action. The SLA is now represented as

$$SLA = \{\alpha, \beta, \mathbf{p}, \mathbf{d}, T, \mathbf{c}\} \quad (3.30)$$

where the different components are as stated in Equation [3.9].

The estimate  $\hat{d}_i$  for each action is given by

$$\hat{d}_i(\mathbf{n}) = \frac{M_i(\mathbf{n})}{Z_i(\mathbf{n})} \quad (3.31)$$

where

$M_i(\mathbf{n}) \equiv$  Number of times action  $i$  has been rewarded.

$Z_i(\mathbf{n}) \equiv$  Number of times action  $i$  has been chosen.

Using the above equations the updating rules for the estimator algorithms are as

follows: Suppose at iteration  $n$  action  $i$  was chosen. Then

$$\begin{aligned}
 p_i(n+1) &= p_i(n) + \lambda \sum_{j \neq i} [f(\hat{d}_i(n) - \hat{d}_j(n))] \\
 &\quad \left[ S_{ij}(n)p_j(n) + S_{ji}(n) \frac{p_i(n)}{r-1} (1-p_j(n)) \right] \\
 p_j(n+1) &= p_j(n) - \lambda [f(\hat{d}_i(n) - \hat{d}_j(n))] \\
 &\quad \left[ S_{ij}(n)p_j(n) + S_{ji}(n) \frac{p_i(n)}{r-1} (1-p_j(n)) \right] ; \forall j ; j \neq i \quad (3.32)
 \end{aligned}$$

and

$$\begin{aligned}
 M_i(n+1) &= M_i(n) + \beta_i(n) \\
 M_j(n+1) &= M_j(n) \\
 Z_i(n+1) &= Z_i(n) + 1 \\
 Z_j(n+1) &= Z_j(n) \\
 \hat{d}_l(n+1) &= \frac{M_l(n+1)}{Z_l(n+1)} ; 1 \leq l \leq r \quad (3.33)
 \end{aligned}$$

where

$$\begin{aligned}
 S_{ij}(n) &= 1, \quad \text{if } \hat{d}_i(n) > \hat{d}_j(n) \\
 &= 0, \quad \text{if } \hat{d}_i(n) \leq \hat{d}_j(n) \quad (3.34)
 \end{aligned}$$

$0 < \lambda < 1$  is the learning parameter and  $f$  is a monotonic increasing function.

In the estimator algorithm, the change in probability of an action  $i$  depends on the sign of  $[\hat{d}_i(n) - \hat{d}_j(n)]$ . Thus if action  $i$  is selected, then the updating for action  $j$  ( $j \neq i$ ) is as follows: If  $[\hat{d}_i(n) > \hat{d}_j(n)]$ , then an amount proportional to  $p_j(n)$  is subtracted from  $p_j(n)$ ; if  $[\hat{d}_i(n) \leq \hat{d}_j(n)]$ , then an amount proportional to  $(p_i(n)/(r-1))(1-p_j(n))$  is added to  $p_j$ . This asymmetry ensures that the action probability vector remains in the simplex  $S$  (Equation [3.5]).

The existing learning algorithms for learning automata can be broadly classified into two groups: *Ergodic* and *Absolutely expedient*. Ergodic learning algorithms result in the optimal action probability vector converging in distribution independent of the

initial action probability distribution. In non-stationary environments, if the optimal action changes with time, an ergodic SLA can track the change. Absolutely expedient learning schemes on the other hand possess absorbing barriers. If an automaton enters an absorbing barrier then it is locked into that state for all time. Thus convergence to one of these absorbing states can be proved. Since all extremities of the simplex  $S$  (Equation [3.5]) are absorbing states, there exist a finite probability of convergence to the wrong action and thus the algorithm is  $\epsilon$ -optimal. Estimator algorithms however use the enhanced definition of the state (Equation [3.28]) and use this extra information in the updating algorithms. This ensures with a large probability that the unit vector corresponding to the optimal action forms the only absorbing barrier. Thus convergence to the optimal action in probability is established [TS85].

### Pursuit Algorithms

Pursuit algorithms are a subset of the estimator algorithms and were first proposed by Thathatchar and Sastry in [TS86]. They have been used in learning of Boolean functions [MT89]. Pursuit Algorithms retain all the characteristics of estimator algorithms but yield much simpler expressions for updating the action probabilities. They are characterized by the fact the action probability vector *pursues* the optimal action. Thus whenever the automaton is rewarded by the environment, the action which has at that instant, the largest estimate of reward, has its action probability increased. The update equations for the pursuit algorithm are as follows: Suppose action  $i$  was chosen at iteration  $n$ . Then

#### • Favourable Response

$$\mathbf{p}(n+1) = (1 - \lambda)\mathbf{p}(n) + \lambda e_m \quad (3.35)$$

#### • Unfavourable Response

$$\mathbf{p}(n+1) = \mathbf{p}(n) \quad (3.36)$$

and

$$\begin{aligned}
 M_i(n+1) &= M_i(n) + \beta_i(n) \\
 Z_i(n+1) &= Z_i(n) + 1 \\
 M_j(n+1) &= M_j(n) ; \forall j \neq i \\
 Z_j(n+1) &= Z_j(n) ; \forall j \neq i \\
 \hat{d}_l(n+1) &= \frac{M_l(n+1)}{Z_l(n+1)} ; 1 \leq l \leq r
 \end{aligned} \tag{3.37}$$

where

$$\begin{aligned}
 \lambda &\equiv \text{The learning parameter} ; 0 < \lambda < 1 \\
 m &\equiv \text{Index of the maximal component of } \mathbf{d}(\mathbf{n}) \\
 \mathbf{e}_m &\equiv \text{Unit r-vector with '1' in its } m^{\text{th}} \text{ coordinate} \\
 M_i(n) &\equiv \text{Number of times action } i \text{ has been rewarded} \\
 Z_i(n) &\equiv \text{Number of times action } i \text{ has been selected} \\
 \hat{d}_i(n) &\equiv \text{Estimate of the reward probability of action } i
 \end{aligned} \tag{3.38}$$

Essentially the algorithm operates by , multiplying all the action probability by the factor  $(1 - \lambda)$  in case of a favourable response. Then the probability of the action that has the largest estimate of reward ( $\hat{d}_i$ ) is increased by  $\lambda$ . This ensures that the probability measure of Equation [3.4] is satisfied. While the  $L_{RI}$  algorithm moves the action probability vector in the direction of the most recently rewarded action, the pursuit algorithm moves the action probability vector in the direction of the action possessing the largest estimate of reward. Theoretical results regarding the convergence of the algorithm are presented in [TS86], where it is shown that the pursuit algorithm is  $\epsilon$ -optimal.

### Discretised Pursuit Algorithms

Discretised pursuit algorithms (DPA) are constructed similarly to their continuous counterparts except that the action probability changes in discrete steps [OL90]. As in the case of the discretised  $L_{RI}$ , the action probabilities are decreased by subtracting from it the value of  $\Delta$  which is the smallest step size. The parameter for the algorithm is referred to as the resolution parameter  $k$ . Thus the update equations for the DPA areas follows: Suppose action  $m$  has the largest estimate of reward at iteration  $n$

#### ◦ Favourable Response

$$\begin{aligned} p_j(n+1) &= \text{Max}\{p_j(n) - \Delta, 0\} ; j \neq m \\ p_m(n+1) &= 1 - \sum_{j \neq m} p_j(n+1) ; \end{aligned} \quad (3.39)$$

#### ◦ Unfavourable Response

$$p_j(n+1) = p_j(n) ; \forall j \quad (3.40)$$

The updating of the estimate vector  $\mathbf{d}(n)$  is done in the same manner as in the continuous case i.e. Equation [3.37]. The parameter  $\Delta$  is given by  $\Delta = 1/(rN)$  where  $r$  is the number of actions and  $N$  is the resolution parameter. As a result of the discretisation, the action probabilities need to be stored only as integer values  $k_i$ , from which the action probabilities at any instant can be calculated as  $k_i\Delta$ . The  $\epsilon$ -optimality of the scheme has been proven in [OL90].

All the learning algorithms which have been detailed above assume a P-Model environment providing a binary response of *success* or *failure*. In the real world this may be a gross simplification and a better scheme would be an environment that provides a continuous response to decide the quality of the action chosen by the automaton. Such an environment is provided by the S-Model and the next section presents learning schemes which operate in such an environment.

### 3.3.4 S-Model Learning Schemes

S-Model environments provide a response which is a random variable lying between  $[0,1]$ . Thus the output of the environment (input to the automaton) is modified to

$$\beta(n) \equiv \{\beta_1, \beta_2, \dots, \beta_r\} \equiv \beta_i \in [0, 1] \quad ; \quad \forall i \quad (3.41)$$

Since the response from the environment in the case of the S-Model is a random variable between  $[0,1]$ , application of the S-Model to learning system problems require the *a priori* knowledge of the lower and upper bounds of the performance indices in order to scale the responses to lie between  $[0,1]$ . Expedient performance using the S-Model was shown in [LT76]. In [VN73], the authors derive an optimal nonlinear algorithm for a two action automaton using the S-Model. In the same paper, a scheme based on the  $\epsilon$ -optimal  $L_{RI}$  (P-Model) scheme was proposed for the multi-action case. An  $\epsilon$ -optimal scheme for the multi-action case was also reported in [Mas73].

#### $S - L_{RI}$ Scheme

In the P-Model environment, the penalty probabilities defined the environment. For each action  $\alpha_i$ , the environment responds with a random value  $[\beta_i(n) | \alpha_i]$  which also forms the input to the automaton. For a P-Model, the response  $\beta_i(n)$  was '1'(penalty) with probability  $c_i$  and '0'(reward) with probability  $(1 - c_i)$ . For the S-Model, the environment is defined as

$$E \equiv \{\alpha, \beta, s\} \quad (3.42)$$

where

$$s(n) \equiv \{s_1, s_2, \dots, s_r\} \quad ; \quad s_i \triangleq E\{\beta_i(n) | \alpha_i\} \quad ; \quad \forall i$$

i.e.  $s_i$  is the mean value of the response  $\beta_i$  for action  $\alpha_i$ .  $s_i$ 's are referred to as the *penalty strengths*. The updating rule for the  $S - L_{RI}$  scheme is as follows: Suppose at iteration  $n$ , action  $\alpha_i$  was chosen and the response from the environment was  $s$ ,

then

$$\begin{aligned} p_i(n+1) &= p_i(n) + a(1-s)(1-p_i(n)); \\ p_j(n+1) &= p_j(n) - a(1-s)p_j(n); \quad \forall j; j \neq i \end{aligned} \quad (3.43)$$

where  $0 < a < 1$  is the learning parameter. The detailed manner in which learning algorithms operate in a S-Model are presented in Chapter 4.

### S-Model Estimator Schemes

In the P-Model environment, the binary responses from the environment were used to update the estimate of reward probabilities for each action  $\alpha_i$  for the estimator learning algorithm (Equation [3.33]). For the S-Model case, the response  $s_i$  itself is used as an estimate of response for each action. The updating equations remain the same as given before in Equation [3.32].

In the next section an alternative S-Model learning algorithm is detailed in which the relative magnitude between the rewards of actions are used to update the action probabilities.

### Relative Reward Strength Learning Algorithms

The relative reward strength algorithms were proposed by Simha and Kurose in [SK89]. The automaton in this scheme operate in a S-Model environment but maintains and uses the most recently obtained reward for each action until that action is selected again. It is similar to the estimator algorithms which used the estimate of the reward probability in updating the action probabilities.

In this scheme the definition of the SLA (Equation [3.9]) is expanded to include a most recent reward vector  $\mathbf{s}(\mathbf{n})$ . The notation  $s_i(n)$  is used to denote the most recent response for the action  $i$  at iteration  $n$ . Thus if the action chosen at the  $n^{\text{th}}$  step was the  $i^{\text{th}}$  action and the response from the environment is denoted by  $r$ ,  $s_i(n) = r$ . The update algorithms take into account the relative reward of all actions, i.e the entire vector  $\mathbf{s}(\mathbf{n})$ . This scheme is similar to the estimator algorithms in that it uses the past response from the environment to update the probabilities. However the important

difference is that the estimator algorithms use the entire past response to form the estimate of the reward probability, while the relative reward algorithm uses only the most recent reward obtained for an action in the updating algorithm. The update equations for the scheme are as follows: Suppose action  $m$  has the largest reward, i.e,  $s_m(n) > s_i(n) ; \forall i; i \neq m$ . Then

$$p_i(n + 1) = p_i(n) + a_n \Delta p_i(n) ; \forall i \tag{3.44}$$

where  $a_n$  is the learning parameter. Thus the update equation is specified by the expression for each  $\Delta p_i(n)$  which is given by

$$\begin{aligned} \Delta p_i(n) &= [(s_i(n) - s_m(n))] ; \forall i ; i \in A_1(n); i \neq m \\ \Delta p_m(n) &= - \left( \sum_{i \in A_1(n), i \neq m} \Delta p_i(n) \right) \end{aligned} \tag{3.45}$$

whereas  $\forall i \ni A_1(n), \Delta p_i(n) = 0$ . The set  $A_1$  is defined as

$$A_1(n) = \{i \mid p_i(n) + a_n(s_i(n) - s_m(n)) > q_{min}\}$$

and is a form of constraint condition. The quantity  $q_{min}$  which is a small positive quantity and the set  $A_1$  is used to ensure that the algorithm retains the ability to track a non-stationary environment, i.e., it does not get locked into a particular state.

The previous sections detailed learning algorithms which improved the speed of convergence of the standard algorithms. However the basic structure of a single automaton has limitations and this is most pronounced when the number of actions of the automaton is large. When this happens the time taken to converge increases drastically and the practical use of the automaton is reduced. The next section explains how single automaton can be connected together to form structures which perform better than a large state single automaton.

## 3.4 Interconnected Automata

The previous sections detailed the standard learning algorithms used in the updating of action probabilities and also presented some new learning schemes which resulted in faster rates of convergence. However to overcome the basic limitations of the large state single automata, a useful strategy would be to connect single automaton into teams of automata to determine whether the collective structure is better at solving complex problems. From a control point of view, the practical use of the automaton is when a single automaton can be used as a building block to build more complex systems. Two such structures will be examined in the subsequent sections: Hierarchical systems of automata and Games of automata.

### 3.4.1 Hierarchical Learning Automata

Research in systems of hierarchical learning automata have been explored in [TR81, MK84, NT89]. A hierarchical system of learning automata is arranged in a tree structure, with a single automaton with  $r$  actions at the first level, each action of which is connected to a automaton at the second level having  $r$  actions and so forth depending on the number of levels there are in the hierarchy. The actions correspond to the leaf nodes (lowest level nodes of the tree) of the hierarchical structure interact with the environment. The response of the environment is then used to update the different automaton along the path upward to the root automaton.

The operation of the hierarchical system is as follows: Initially, the automaton at the first level selects one of the  $r$  actions. This action then triggers the automaton at the second level which selects one of the  $r$  actions which is available to it. This process continues until a leaf node is selected which forms the action selected by the automaton to interact with the environment. This general structure is shown in Figure (3.2). It is assumed that every automaton in the hierarchy has  $r$  actions though this not necessarily so. The response from the environment is then used to update the actions probabilities of all the automaton which were used to arrive at the action selected. Complete details of the updating algorithms are given in [NT89].

The advantage of using the hierarchical structure is that the number of proba-

bility updatings are significantly reduced especially when the number of actions of the automaton are large. This can be illustrated as follows: Suppose the number of actions of the automaton is  $N$ . If a single automaton is used, the number of probability updatings per iteration would be  $N$ . But if the structure used was that of a hierarchical automata arranged in form of a binary tree with two actions available to each automaton, then, if  $N = 2^k$ , the number of probability updatings is equal to only  $k$ . This reduction is significant when the number of actions  $N$  is large.

### 3.4.2 Automata Games

Game theory has had important ramifications in social and economic problems where conflict of interest between the decision makers play an important part in the final analysis. A game is said to be played between players when each player chooses an action and elicits a response from the environment. The players may or may not have complete information regarding the number of other players, the options available to them etc. . A player bases the next move depending on the response obtained from the environment.

The concept of automaton games was first suggested by Krylov and Tsetlin in [KT63] and subsequent work has been carried out by Chandrashekar and Shen [CS69], Viswanathan and Narendra [VN74] and Lakshmiarahan and Narendra [LN81, LN82]. In automata games, a number of automata operate in an environment without the complete knowledge about the each other. Each automaton may have different number of actions and learning rules. A general mathematical formulation of automata games can be given as follows: Let  $N$  automaton  $\{A^1, A^2, \dots, A^N\}$  take part in a game of automata. A typical automaton  $A^j$  can be described by the quintuple  $\{ \alpha^j, \beta^j, \phi^j, F^j, G^j \}$  where

$$\begin{aligned}
 \alpha^j &= \{ \alpha_1^j, \alpha_2^j, \dots, \alpha_{r_j}^j \} \equiv \text{Set of Actions} \\
 \beta^j &= \{ \beta_1^j, \beta_2^j, \dots, \beta_{r_j}^j \} \equiv \text{Set of Inputs} \\
 \phi^j &= \{ \phi_1^j, \phi_2^j, \dots, \phi_{r_j}^j \} \equiv \text{Set of Internal States} \\
 F^j, G^j &= \text{Updating rule for the automaton}
 \end{aligned} \tag{3.46}$$

A play  $\alpha(n)$  is defined as a set of strategies which are chosen by the team of automaton at iteration  $n$  and is given by

$$\alpha(n) = \{\alpha^1(n), \alpha^2(n), \dots, \alpha^N(n)\} \quad (3.47)$$

The outcome of the play  $\alpha(n)$  is given by  $\beta(n)$  which is defined as

$$\beta(n) = \{\beta^1(n), \beta^2(n), \dots, \beta^N(n)\} \quad (3.48)$$

The  $N$  automata are said to participate in a game if the probability of the outcome  $\beta(n)$  depends on the play  $\alpha(n)$ .

Further details of automata games, the learning algorithms and convergence results used are given in [NT89]. The details of a team of co-operative game playing automata using the pursuit algorithm is given in [MT89]. The games approach presents a method of using the single automaton in complex structures to get enhanced performance than that would be obtained when using a single automaton.

### 3.5 Discussion

In this chapter a general review of Stochastic Learning Automata was presented. The basic block structure of a SLA was explained along with the standard learning algorithms. Limitations of the standard algorithms were then presented. Improved learning algorithms which have been proposed in the literature were detailed subsequently, emphasis being given to highlight the differences between the standard and improved schemes. Both the P-Model and S-Model learning environments were discussed and compared. Use of the basic structure of a single automaton in more complex structures have been indicated. In the next chapter, we present the method and results of using the SLA approach in adaptive digital filtering.

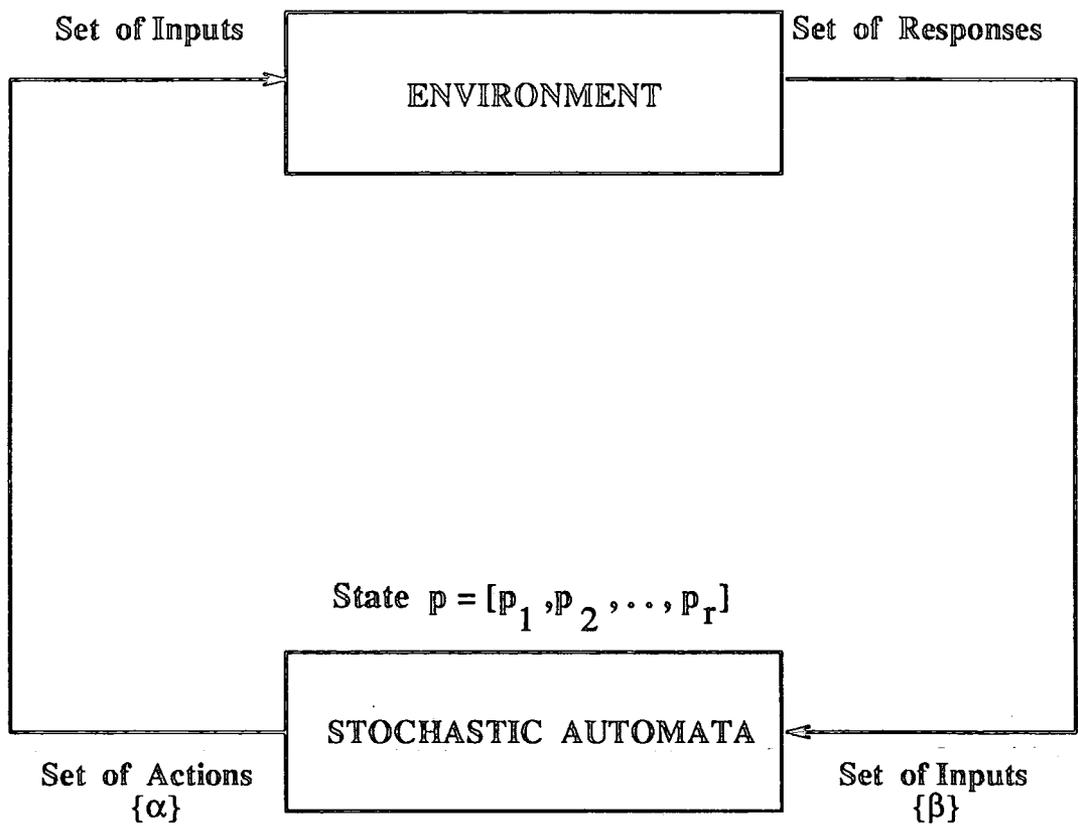


Figure 3.1: Stochastic Learning Automata

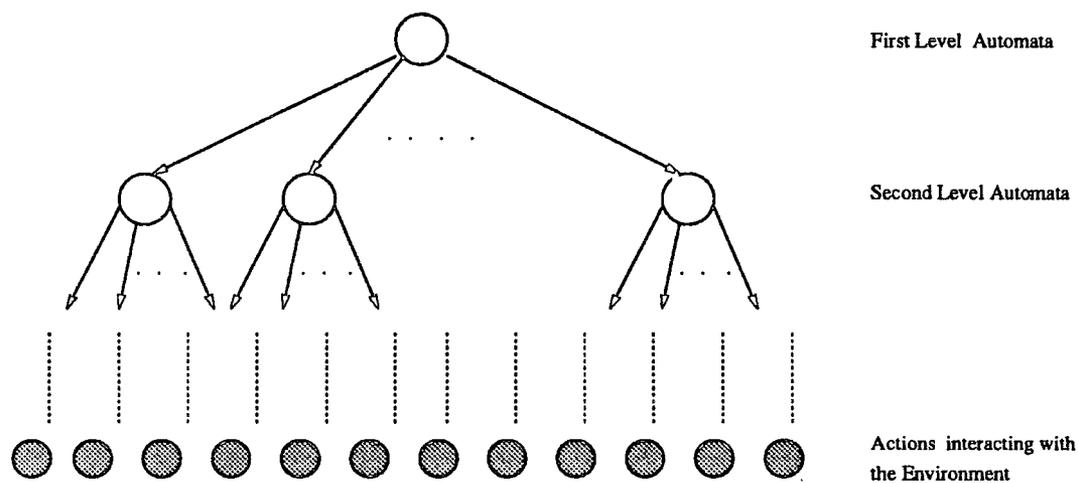


Figure 3.2: Hierarchical Stochastic Learning Automata

## Chapter 4

# Adaptive Digital Filtering using Stochastic Learning Automata

### 4.1 Introduction

This chapter presents results obtained using *Stochastic Learning Automata* as the adaptive technique for adaptive filtering. As detailed in Chapter 2, adaptive filtering may be classified into adaptive FIR filtering and adaptive IIR filtering. The algorithms relating to adaptive FIR filtering are well established and currently are extensively used in various applications. On the other hand, adaptive IIR algorithms are still an active area of research and are in the process of establishing themselves as a viable alternative in certain applications to adaptive FIR algorithms. The main problems associated with adaptive IIR filtering algorithms are problems of stability and existence of error functions which may be multimodal with respect to the filter parameters. Although the past couple of decades has seen extensive research [Whi75, SEA76, PA78, Joh79, TLJ78, LTJ80, FJ86, FN89], the above problems have not yet been completely resolved. One of the new approaches that has been suggested for adaptive IIR filtering is that of Stochastic Learning Automata the details of which were presented in Chapter 3.

The initial work of applying the SLA approach to adaptive IIR filtering was carried out by Tang and Mars [TP89, TP91]. Extensive simulations were performed

using the standard learning algorithms. Hybrid schemes were proposed which combined Recursive Least Mean Square gradient techniques with the SLA approach. The automaton games approach was also investigated as a possible solution to the problem of dimensionality when adapting high-order IIR filters. In this chapter detailed results using the SLA approach to adaptive filtering are presented. In particular the improved learning algorithms which were detailed in Chapter 3 have been used for the adaptive filtering case and the results obtained are compared with the results obtained using the standard learning algorithms. The S-Model environment learning algorithms are looked at in detail and the results compared with that obtained using the P-Model environment.

In the next section details of the simulation configuration are given.

## 4.2 Simulation Configuration

### 4.2.1 Introduction

To use the different learning algorithms which were presented in Chapter 3, the system identification configuration was employed, where an adaptive filter is used to model an unknown system as shown in Figure [4.1]. The output error formulation detailed in Chapter three was used to form the estimation error  $e(n)$ . The equation error approach was not used as it resulted in biased estimates of the filter parameters. Another reason for using the output error formulation is that it gave good approximation when applied to reduced order models [SS82] which were encountered when modeling a system by an insufficient order adaptive filter.

### 4.2.2 Using Stochastic Learning Automata

The main motivation in using the Stochastic Learning Automata as an adaptation algorithm for adaptive filtering was to use its capabilities of global optimisation when dealing with multimodal error surfaces [SN69]. As was detailed in Chapter 3, the error surfaces for adaptive IIR filters could be multimodal. Using Stochastic Learning Automata as the adaptation technique, the search for the optimum is carried out in

probability space rather than in parameter space as is the case with other adaptation algorithms. In the standard gradient methods, the new operating point lies within a neighbourhood distance of the previous point. This is not the case for adaptation algorithms based on stochastic principles, as the new operating point is determined by a probability function and is thus not constrained to be near the previous operating point. This gives the algorithm the ability to locate the global optimum.

In using Stochastic Learning Automata in the adaptive filtering context, the output set of actions of the automaton are made to correspond to a set of filter coefficients. Each output action of the automaton is thus related to a specific combination of filter coefficients. Since the number of actions of the automaton is finite, this would involve the discretisation of the parameter space into a number of hyperspaces. Thus the error surface is partitioned into a number of hyperspaces, the total number of hyperspaces being equal to the total number of actions of the automaton. The dimension of each hyperspace would be equal to the number of filter parameters. In this case the task of the automaton would be then to asymptotically choose that action corresponding to the set of filter coefficients which results in the minimum error. This is clarified by presenting an example: Suppose the number of filter parameters were three, i.e.  $[a, b, c]$  and the number of actions of the automaton were  $N$ . Then the actions of the automaton can be described as follows:

$$\begin{aligned} \text{Action 1} &\equiv [a_1, b_1, c_1] \\ \text{Action 2} &\equiv [a_2, b_2, c_2] \\ \text{Action 3} &\equiv [a_3, b_3, c_3] \\ &\vdots \\ \text{Action } N &\equiv [a_N, b_N, c_N] \end{aligned}$$

Thus choosing action 3 would result in choosing the parameters  $[a_3, b_3, c_3]$  for the filter coefficients. This concept of discretising the parameter space is illustrated in Figure [4.2] where the adaptive filter is a second order filter with filter parameters  $[a, b]$ .

A block diagram of an adaptive filter incorporating a Stochastic Learning Automaton in a system identification configuration is shown in Figure [4.1]. As shown

in Figure [4.1], the operating environment of the automaton was the environment of the adaptive filter. The response from the environment for a particular action was the short term average of the instantaneous squared error obtained with the coefficients represented by that action. To obtain the short term average, a rectangular window was used the length of which was seen to play a significant role in the rate of convergence. The optimum size was obtained after extensive simulations with different window lengths. Thus the short time average of the instantaneous square error, henceforth referred to as the Mean Square Output Error (MSOE), was used by the environment to decide whether the action chosen was to be penalized or rewarded. This assumed that the environment was of the P-Model type. The procedure for deciding this was presented in [SN69], where the global minimum of a multimodal, stochastic noisy error surface was determined using a learning automaton. For the S-Model, the Mean Square Output Error was used directly to decide whether the action chosen was optimum. Further details and results using the S-Model environment are presented in a subsequent section.

As was detailed in Chapter 2, the three conditions put forward by Stearns [Ste81] and Söderström and Stoica [SS82] for a unimodal error surface were

- The adaptive filter is of sufficient order to model the unknown system
- The input signal is white
- The order of the adaptive filter numerator exceeds that of the unknown system denominator

Further work has been recently been carried out by Fan and Nayeri [FN89], wherein they have proved the first two conditions for first and second order filters without the third condition. They have also shown that the error surface could be multimodal even in the case of sufficient order modeling or when the order of the adaptive filter is overestimated. In practice, sufficient order modeling is quite difficult to achieve as the order of the system being modeled is usually not known. Thus in most practical cases, the modeling filter may be of an order less than that of the unknown filter resulting in a multimodal error surface. *Thus the important point regarding adaptive IIR filtering*

is that the error surface may be multimodal and the adaptation algorithm must be able to locate the global optimum. The paper by Fan and Jenkins [FJ86] proposed a new algorithm for adaptive IIR filtering and also presented four different cases for the system identification configuration (Figure [4.1]) wherein the error surface could be multimodal. The four categories based on the order of the adaptive filter and the nature of the input excitation are

- Sufficient Order Modeling - White Noise Input
- Sufficient Order Modeling - Coloured Noise Input
- Reduced Order Modeling - White Noise Input
- Reduced Order Modeling - Coloured Noise Input

The four cases detailed above form the backbone of the simulation experiments which have been carried out using stochastic learning automata as the adaptation technique. For each of the above cases a suitable simulation experiment is constructed the details of which are presented in the next section.

### 4.2.3 Different Categories of Modeling

#### I) Sufficient Order Modeling - White Input

This was first illustrated as a counterexample to Stearns conjectures [Ste81] by Fan and Nayeri [FN89], where it was shown that for an adaptive IIR filter of order greater than two, the error surface may be multimodal even for sufficient order modeling with white noise input. The transfer functions of the unknown system and adaptive filter for the example chosen were

$$\begin{aligned} H(z^{-1}) &= \frac{d_0}{1 - 2.4z^{-1} + 1.91z^{-2} - 0.504z^{-3}} \\ H_a(z^{-1}) &= \frac{b}{1 + a_1z^{-1} + a_2z^{-2} + a_3z^{-3}} \end{aligned} \quad (4.1)$$

A fundamental problem in adaptive IIR filtering is to maintain stability of the adaptive filter during adaptation. Thus the partitioning of the parameter space formed by

$a_1, a_2, a_3$  may result in a unstable filter configuration during adaptation. To overcome this problem, the denominator of the adaptive filter was factorised into a product of a second order and first order filter as given below

$$H_a(z^{-1}) = \frac{b}{(1 - (p_1 + p_2)z^{-1} + p_1p_2z^{-2})(1 - p_3z^{-1})} \quad (4.2)$$

where  $p_1, p_2$  and  $p_3$  are the poles of the system. By constraining the poles of the filter to lie inside the unit circle in the  $z$ -plane, the stability of the adaptive filter can be assured during adaptation. The global minimum of the configuration is located at  $(b, a_1, a_2, a_3) = (1.0, -2.4, 1.91, -0.504)$  for which the corresponding poles are  $p_1 = 0.7$ ,  $p_2 = 0.8$ , and  $p_3 = 0.9$ . The numerator coefficients were set to 1.0 in the simulations.

## II) Sufficient Order Modeling - Coloured Input

The example for this case was first presented in [Sod75] and was also used by Fan and Jenkins [FJ86]. The transfer functions of the unknown system and modeling filter are

$$\begin{aligned} H(z^{-1}) &= \frac{1}{(1 - 1.4z^{-1} + 0.49z^{-2})} \\ H_a(z^{-1}) &= \frac{b}{(1 + a_1z^{-1} + a_2z^{-2})} \end{aligned} \quad (4.3)$$

To colour the input, white noise was filtered through a FIR filter having transfer function  $(1 - 0.7z^{-1})^2(1 + 0.7z^{-1})^2$ . This colouration gave rise to a multimodal error surface with the global optimum located at  $(b, a_1, a_2) = (1, -1.4, 0.49)$ .

## III) Reduced Order Modeling - White Input

The example for this case was first proposed by Larimore and Johnson in [JL77] in which a second order system was modeled by a first order filter. The transfer functions of the filters involved were

$$H(z^{-1}) = \frac{0.05 - 0.4z^{-1}}{1.0 - 1.1314z^{-1} + 0.25z^{-2}}$$

$$H_a(z^{-1}) = \frac{b}{1 - az^{-1}} \quad (4.4)$$

The insufficient degree of freedom in the adaptive filter resulted in a bimodal error surface (Figure 9 in [JL77]). This example has also been extensively used by other researchers in testing new adaptive algorithms. The global minimum is located at  $([b,a] = [-0.3, 0.8])$  with error value  $\approx 0.3$ .

### Reduced Order Modeling - Coloured Input

The example for this case is an extension of the example used for the second case given above. The relevant transfer functions are

$$\begin{aligned} H(z^{-1}) &= \frac{1}{(1 - 0.6z^{-1})^3} \\ H_a(z^{-1}) &= \frac{b}{1 + a_1z^{-1} + a_2z^{-2}} \end{aligned} \quad (4.5)$$

The colouring FIR filter transfer function is changed to  $(1 - 0.6z^{-2})(1 + 0.6z^{-2})$  resulting in a multimodal error surface as shown in Figure 9 in [FJ86].

In the next section the performance of the different learning algorithms for the four cases listed above are examined and compared.

## 4.3 Simulation Results

### 4.3.1 Introduction

As discussed previously, the main motivation in using Stochastic Learning Automata for adaptive filtering has been its ability to distinguish the global optimum from local optima. Each of the learning algorithms detailed in Chapter 3 had some defining parameters. To check the effect of the parameters on the learning process, simulations with a range of parameter values were performed. Of the four categories which have been detailed, categories three and four deal with situations which are more complex and practical. Sufficient order modeling (Category (I) and (II)) is not commonly realised in practical situations as it would assume some knowledge of the unknown

system. Thus the simulation experiments were carried out using the reduced order modeling using both white and coloured input excitation.

To determine whether a particular action chosen by the automaton was to be rewarded or penalized, a short term average of the instantaneous square error (MSOE) was used. Thus for example, if at iteration  $N$ , action 4 had been selected by the automaton, then the MSOE  $e_4$  obtained using the filter coefficients represented by action 4 was used. This scheme does not use the fact that action 4 could have also been selected a few times before iteration  $N$ . An improved scheme would be to average the MSOE obtained for a particular action every time the action was chosen. To clarify this point consider the case where at iteration  $N$  action  $i$  was chosen resulting in a MSOE of  $e_i(N)$ . Suppose action  $i$  had been chosen once before at iteration  $K$ , and had resulted in a MSOE of  $e_i(K)$ . The new scheme would then use the previous value of MSOE along with the current value of MSOE and the resulting MSOE is given by

$$e_i(N) = \frac{e_i(N) + e_i(K)}{2}$$

The main advantage in the new scheme is that the short term average is not restricted by the window length but is also determined by the number of times the particular action is chosen. This effectively increases the window length by a factor equal to the number of times a particular action is selected. The effect of using this scheme is shown in Figure [4.3] for two different values of the learning parameter and shows a faster rate of convergence. The results in Figure [4.3] with the label  $NE$  refers to result obtained without using the new error estimation scheme. Thus the results pertaining to labels  $Lri(I)$  and  $Lri(I)-NE$  are obtained using the  $L_{RI}$  with and without the new error scheme, for the same values of the learning parameter. The variance of the MSOE using the new scheme also is seen to be reduced as the effective window length is now increased. The example used to illustrate the new error was reduced order model of category three. In all the subsequent simulation results, this scheme of determining the MSOE has been used.

In the next section the results obtained operating in the P-Model environment are presented.

### 4.3.2 Results using P-Model Learning Algorithms

#### Standard Learning Algorithms

All the results presented in this section use the example given in category (III) which used a first order IIR filter to model a second order IIR filter. This configuration results in a bimodal error surface with a local minimum corresponding to a error value of 0.9 and a global minimum corresponding to a error value of 0.3. The two parameters  $a, b$  were discretised into ten discrete values resulting in the automaton having 100 actions. Each action corresponded to a particular set of coefficient values for  $a, b$ . The results obtained using the standard learning algorithms are shown in Figure [4.4]. The learning parameters used for the different schemes are as follows:

$L_{RP}(I)$	$\equiv$	Rew. Par. = 0.1	Pen. Par. = 0.1
$L_{RP}(II)$	$\equiv$	Rew. Par. = 0.2	Pen. Par. = 0.2
$L_{RcP}(I)$	$\equiv$	Rew. Par. = 0.01	Pen. Par. = 0.001
$L_{RI}(I)$	$\equiv$	Rew. Par. = 0.01	
$L_{RI}(II)$	$\equiv$	Rew. Par. = 0.005	

These results were originally presented in [TP89] and have been repeated here for the sake of completeness. All the algorithms were able to locate the global optimum point. Of the standard algorithms the  $L_{RI}$  gave the fastest rate of convergence i.e about 50,000 time samples were required for the algorithm to locate the optimal set of coefficients. The  $L_{RP}$  algorithms had a slower rate of convergence (60-180,000 time samples), the main reason for this being the increased value of the penalty parameter which did not allow the algorithm to settle into a particular state rapidly. The value of the window length used to obtain the MSOE was 50. The results shown are ensemble average of 25 runs of the simulation experiment. It was noticed that the learning parameter played an important role in the rate of convergence and accuracy of the algorithm. Large values of the learning parameter resulted in faster convergence but at the expense of possible convergence to a non-optimum point while small values of the parameter resulted in an increased convergence time.

### Discretised Learning Algorithms

The results obtained using the discretised learning algorithms are presented in Figure [4.5]. The values of the defining parameter (resolution parameter) used in the algorithm are 1000, 5000 and 10,000. It can be seen that decreasing the value of the resolution parameter (increasing the learning rate) too much results in convergence to a non-optimal action (Parameter Value = 1000), while increasing it (decreasing the learning rate) results in slower convergence (Parameter Value = 10000). The main reason for this result is the discretisation of the probability space now results in the action probability vector moving towards an absorbing state more rapidly than that obtained using the standard learning algorithm. The rate of approaching an absorbing state is dependent on the resolution parameter. Too large a value of the learning rate results in the algorithm getting locked up in a non-optimal state. Comparing Figures [4.4] and [4.5], as expected, the discretised algorithms are seen to result in faster convergence as compared to the standard learning algorithms.

### Estimator Algorithms

Estimator algorithms were devised to increase the rate of convergence of the standard learning algorithms and results using this approach are shown in Figure [4.6]. As can be seen from Figure [4.6], the estimator algorithm shows faster convergence as compared to the standard learning algorithms and are comparable to the results obtained using the discretised  $L_{RI}$  algorithm. The values of the learning parameter used for this simulation are 0.005, 0.01 and 0.05. The algorithm successfully located the global minimum as can be seen from the final error value at the end of the simulation run.

### Pursuit Algorithms

Pursuit algorithms as explained in Chapter 3 are a subset of the estimator algorithms possessing much less computational complexity. The results of using these for adaptive filtering are shown in Figure [4.7]. The rate of convergence is comparable to that obtained using the estimator algorithms though the computational time required was

much less. The learning parameter values for the algorithm were 0.0075, 0.01 and 0.05.

### Discretised Pursuit Algorithms

Discretised pursuit algorithms are the discretised version of the continuous pursuit algorithms, the results of which are presented in Figure [4.8]. From the results it can be seen these algorithms give the best performance in the terms of the rate of convergence (25,000 time samples) and are able to locate the global minimum. The value of the resolution parameter in Figure [4.8] were 1000, 5000 and 10,000 . As shown in Figure (4.8) (Parameter Value = 1000)), increasing the learning rate too much results in premature convergence and a non-optimal performance.

### Discussion

The important aspect of all the learning schemes detailed above is that all of them were able to locate the global minimum when searching a bimodal error surface. The standard learning algorithms took about 180,000 time samples ( $L_{RP}$ ) to 50,000 time samples ( $L_{RI}$ ) for locating the optimal set of coefficients. Though this is large when compared to results obtained using gradient schemes like the (LMS), the ability to locate the global optimum validates the utility of this approach. The main motivation for using the improved learning algorithms was to reduce the number of time samples required for convergence. All the new schemes were able to locate the global optimum using a significantly less number of time samples. The value of the learning parameter was found to play a crucial role in determining the accuracy and rate of convergence of the respective algorithms. The next section presents the results when the S-Model environment is used.

#### 4.3.3 Results using S-Model Learning Algorithms

##### Introduction

The S-Model environment is intuitively better suited for modeling the environment in which the adaptive filter operates as every action generated a response lying between

[0,1] rather than the binary value generated in a P-Model environment. Thus actions resulting in a response closer to 1 were the more optimal actions. To normalize the response from the environment to lie between [0,1], the maximum and minimum values of the responses should be known *a priori*. In a practical case this is usually not known and so the adaptive process garners this knowledge as the process evolves. This is achieved as follows: At any iteration  $k$ , the current value of the response is chosen as the minimum value if it is less than the previous minimum value ( $e_{min}$ ), and as the maximum value if it is greater than the previous maximum value ( $e_{max}$ ). If the current response lies between the maximum and minimum values, then both the limits are not changed. Thus we have,

$$\begin{aligned} e_{min}(k) &= \begin{cases} e(k) & \text{if } e(k) < e_{min}(k) \\ e_{min}(k-1) & \text{otherwise} \end{cases} \\ e_{max}(k) &= \begin{cases} e(k) & \text{if } e(k) > e_{max}(k) \\ e_{max}(k-1) & \text{otherwise} \end{cases} \end{aligned} \quad (4.6)$$

The normalisation is then achieved by using the equation

$$s_i(k) = \frac{e_{max}(k) - e_i(k)}{(e_{max}(k) - e_{min}(k))} \quad (4.7)$$

where  $s_i(k)$  is the normalised response from the environment for action  $i$  at the  $k^{th}$  iteration and  $e(k)$  is the unnormalised response. The above scheme of normalisation was proposed in [VN73].

As will be shown later on in subsequent sections, this method of normalisation did not result in very fast convergence and sometimes the convergence time was extremely large resulting in limited practical use. The reason for this is found to be the normalisation scheme given by Equation [4.7] and the nature of the error surface which is generated by the simulation experiment. The error surface which is bimodal, is found to have a large maximum value. The normalisation scheme scaled the error values from the environment *linearly* between 0 and 1. As a result of this linear scaling and the large maximum value, points on the error surface which are close to the global minimum are assigned responses close to 1. This corresponded to a number

of actions of the automaton being assigned response values close to 1. As a result, the algorithm was unable to locate the global optimum rapidly. To resolve this problem, a new normalisation scheme employing a nonlinear scaling function was used. The new scheme used the following equation:

$$s_i = \exp(-(e(k) - e_{min}(k))^2) \quad (4.8)$$

This scheme assigned response values near '1' only to the actions which resulted in a error value very close to the minimum value determined until then thus enabling the learning algorithm to distinguish between the actions. In all the subsequent algorithms operating in a S-Model environment, both the normalisation procedures are used and results compared.

#### S-Model Standard Algorithms

Figures [4.9,4.10] shown the convergence results obtained using the  $S - L_{RI}$  Algorithms for the adaptive filtering using the old and new normalisation schemes. The old normalisation scheme is unable to find the optimum point even after 20,00,000 time samples which makes the practical use of the algorithm extremely limited. On increasing the value of the learning parameter, there is an increase in the speed of convergence, but the algorithm is still unable to locate the optimal filter coefficients. Figure [4.10] shows the result of using the new normalisation scheme and exhibits satisfactory location of the optimal set of filter coefficients as indicated by the error level to which the algorithm converges. This again was achieved only after about 17,00,000 iterations resulting in limited practical use. Increasing the value of the learning parameter resulted in faster convergence at the expense of accuracy. Thus surprisingly, the S-Model  $L_{RI}$  learning algorithm resulted in a poorer performance than the P-Model learning schemes. A possible reason for this behaviour is given later in this chapter.

### S-Model Estimator Algorithms

The results of using the S-Model estimator algorithms are shown in Figures [4.11,4.12]. As was the case with the  $S - L_{RI}$  learning algorithm, the old normalisation (Equation [4.7], Figure [4.11]) was not able to locate the global optimum even after a large number of time samples. Using the new normalisation, the algorithm was able to locate the global optimum (Figure [4.12]), the time samples required for convergence being less than that for the  $S - L_{RI}$  algorithms. Too high a value of the learning parameter resulted in inaccurate results while too low a value increased the number of time samples required for convergence.

### Relative Reward Schemes

Figures [4.13,4.14,4.15,4.16] show the results of using the relative reward learning algorithms. Figures [4.13,4.15] show the result of using the old and new normalisation schemes when using *small values of the learning parameter*. It can be seen that the new normalisation performs better resulting in faster convergence. Figures [4.14,4.16] also present the results in using the old and new normalisation schemes but for *larger values of the learning parameter*. In this case it can be seen that the old normalisation scheme performs better leading to faster convergence. To explain this anomaly, reference is made to the defining equation of the relative reward scheme (Equation [3.44]) where  $\Delta p_i(n)$  is determined by the difference in value between the responses of action  $i$  and the action which currently resulted in the maximum response. The new normalisation scheme weights the responses non-linearly and thus the value of  $\Delta p_i(n)$  mentioned above is large. This in combination with a large value of the learning parameter resulted in impermissible values for the probability of an action, i.e.  $p_i(n) > 1.0$  or  $p_i(n) < 0.0$ . In such a case the algorithm does not update the action probabilities and thus the learning rate of the algorithm drops. Thus for large values of the learning parameter the old normalisation scheme gives faster and more accurate convergence.

### Discussion

Of the S-Model learning algorithms which were attempted, the relative reward algorithm gave the best results (convergence in about 600,000-700,000 time samples). The other algorithms, though able to locate the global optimum, did so only after a large number of time samples. Thus the practical use of these algorithms in the adaptive filtering context are limited. The new normalisation scheme resulted in faster convergence than the old scheme. One reason why the S-Model schemes performed poorly when compared to the P-Model schemes is that in the S-Model scheme *every action* resulted in a response lying between  $[0,1]$  which was used in updating the probability of that action being chosen in the next iteration. In the P-Model scheme as the responses were binary, the action probabilities were updated faster. This is clarified using an example: Suppose action  $i$  was the optimal action and both the S-Model and P-Model schemes gave a response of  $1$  when action  $i$  was selected. If in the next iteration, action  $k$  (non-optimal) was selected, the P-Model scheme would result in a response  $0$  while the S-Model scheme would result in a response which is a *finite value less than 1*. Thus in the S-Model case, the action probability of action  $k$  would increase by an amount proportional to the response it obtained. This would result in the probabilities of the other actions *being reduced* in order to keep the probability vector in the unit simplex. In the P-Model  $L_{RI}$  scheme this will not happen as when actions result in a  $0$  response, no updating is performed. The net result of this argument is that in a P-Model scheme the learning is faster than that obtained in a S-Model environment. This also explains the success of the new normalisation scheme which effectively drives the S-Model environment asymptotically towards a P-Model environment using Equation [4.8].

#### 4.3.4 Other Categories

The simulations in the last section concentrated on the adaptive filter model given in Category (III) which was involved with reduced order modeling with the input signal being white. This case was taken to be the most general setting for an adaptive filtering algorithm as has been explained before. Further simulations were also

carried out using the model given in Category (IV) which concerned reduced order modeling with coloured input. The transfer functions for this category were as given by Equations [4.5] and the resulting error surface was multimodal as shown in Figure 9 in [FJ86]. Figure [4.17] shows the results obtained using the different P-Model learning algorithms for the model in Category (IV). The discretised algorithms are seen to give the fastest rate of convergence with the discretised pursuit algorithm being slightly faster. All the algorithms were able to locate the global optimum. The S-Model learning algorithms were not tested on this model as they had shown limitations when tested on the model given in Category (III). Results using standard learning algorithms on the models in Category (I) and (II) have been presented in [TP91] where it was shown all the algorithms were able to locate the global optimum. The new algorithms were not specifically tested on these models. It is assumed that the discretised algorithms would perform better and result in faster convergence as has been noticed from the results which have been obtained.

#### 4.3.5 Automata Games and Hierarchical Schemes

The primary disadvantage of using the Stochastic Learning Automata approach in adaptive filtering was the increased computational time when the number of parameters of the filter was large. This arose because of the discretisation of the parameter space. For example, if the adaptive filter had three parameters each being discretised into ten sections, the resulting automaton would have 1000 actions. Updating the probabilities of an automaton having a large number actions increases the computational time and thus limits the practical use to which the approach can be put to. Two different approaches had been proposed to overcome this problem - hierarchical automata and automata games. Simulation results using these approaches are given in [TP91] where a novel hybrid technique using the standard Recursive Least Mean Square (RLMS) algorithm and stochastic learning automata was proposed. The hybrid technique proposed used the RLMS algorithm to update the numerator coefficients of the adaptive IIR filter, while the SLA approach was used to adapt the denominator coefficients. This made use of the fact that the error function for an

adaptive recursive filter in a system identification configuration was quadratic with respect to the numerator coefficients. The ability of the automata approach to determine the global optimum was used to determine the denominator coefficients. Tang and Mars also used the games approach to adapt the denominator coefficients and have shown through simulation experiments that this approach was able to locate the global optimum. However the main drawback with the automata games approach was that theoretical results regarding global optimality are not available. Thus using the games approach could result in a non-optimal performance.

Hierarchical systems of stochastic learning automata are another method to obtain faster convergence with respect to the computational time. Although the number of iterations are the same as that obtained with a single automaton, the time required for a single iteration is reduced as the number of probability updatings are reduced in a hierarchical scheme as was explained in the Chapter 3. Results obtained using the hierarchical scheme are given in [TP91] and show that the automata games approach and the hybrid scheme gave faster convergence than the hierarchical scheme.

## 4.4 Conclusions

This chapter presented the results in using Stochastic Learning Automata as an adaptation technique for adaptive digital filters. The specific case examined was that of adaptive IIR filtering. The main motivation for using the SLA approach was its ability to locate the global optimum when searching a multimodal performance function. This was tested using adaptive IIR filtering as a testbed and the results presented show that the technique was able to locate the global optimum. Results using the new and improved learning schemes were also presented and resulted in the reduction in the number of iterations required for convergence. The S-Model environment was also examined and a possible reason as to why S-Model learning algorithms did not perform as well as P-Model algorithms has been explained. A possible use of the SLA approach is to use the technique as a first level search whereby the section containing the global optimum is determined by the automaton. Thereafter established gradient algorithms could be used to reach the precise global optimum.

The main drawback with the SLA approach is the increased computational time required for convergence when the number of actions of the automaton is large. Thus when adapting high order filters, the SLA approach would result in a slow rate of convergence as increasing the order of the filter increases the number of parameters which would lead to a large number of actions for the automaton. This is the classical problem of high dimensionality which inhibit most adaptive schemes. Automata games have been proposed to overcome this drawback, but lack of strong theoretical results regarding the global optimality of such an approach renders this idea impractical. The next chapter presents a technique based on genetic and evolutionary optimisation. The primary advantage of this approach is the ease with which the dimensionality issue is handled.

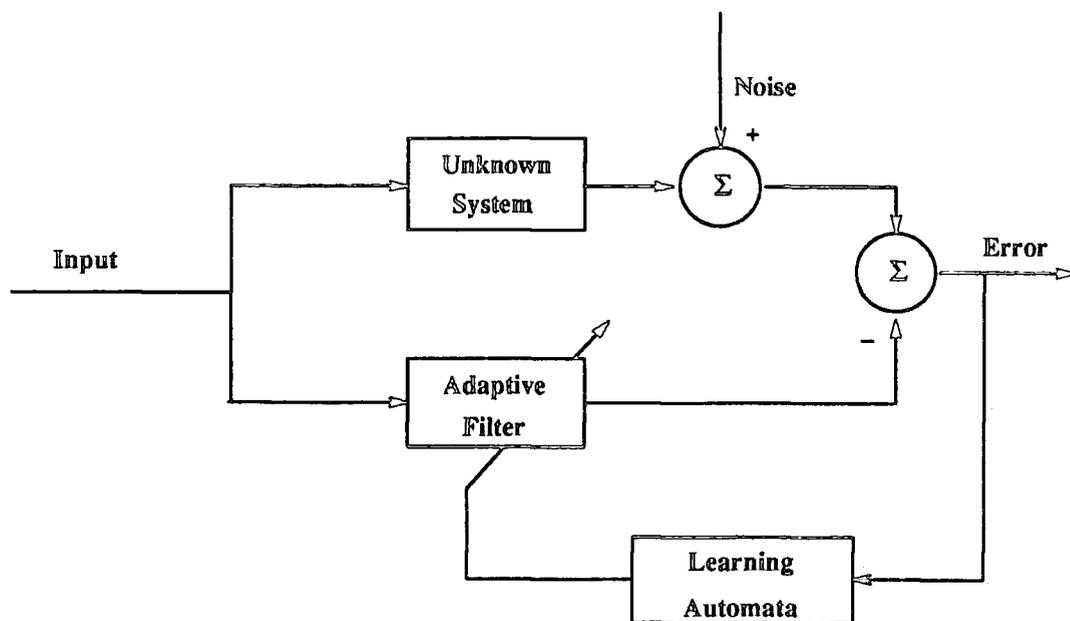


Figure 4.1: System Identification Configuration incorporating Stochastic Learning Automata

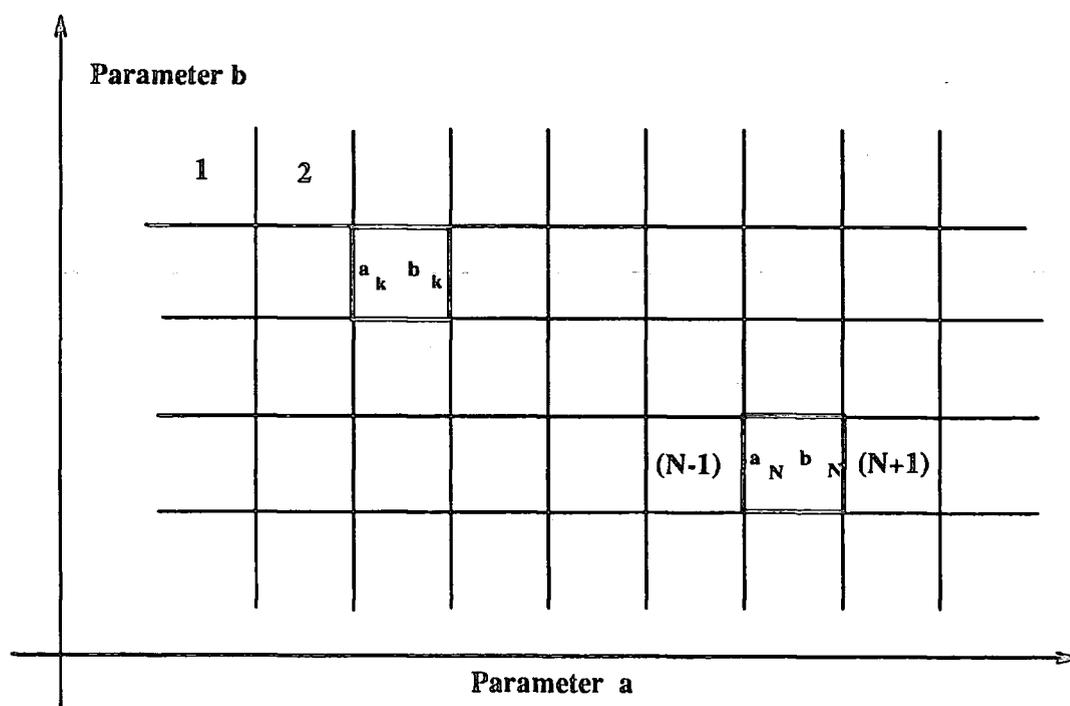


Figure 4.2: Discretisation of the Parameter Space

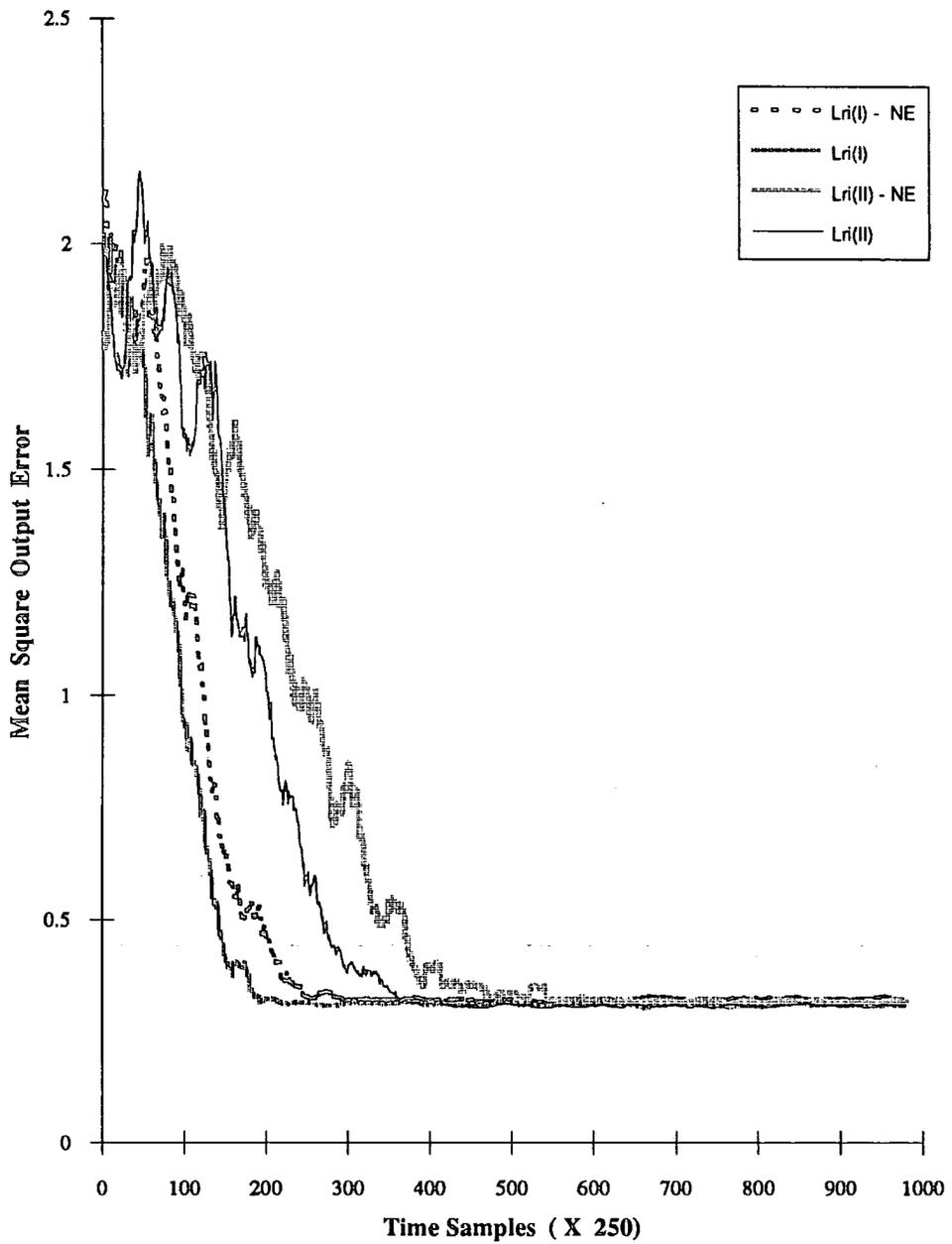


Figure 4.3: The New Scheme of Error Estimation

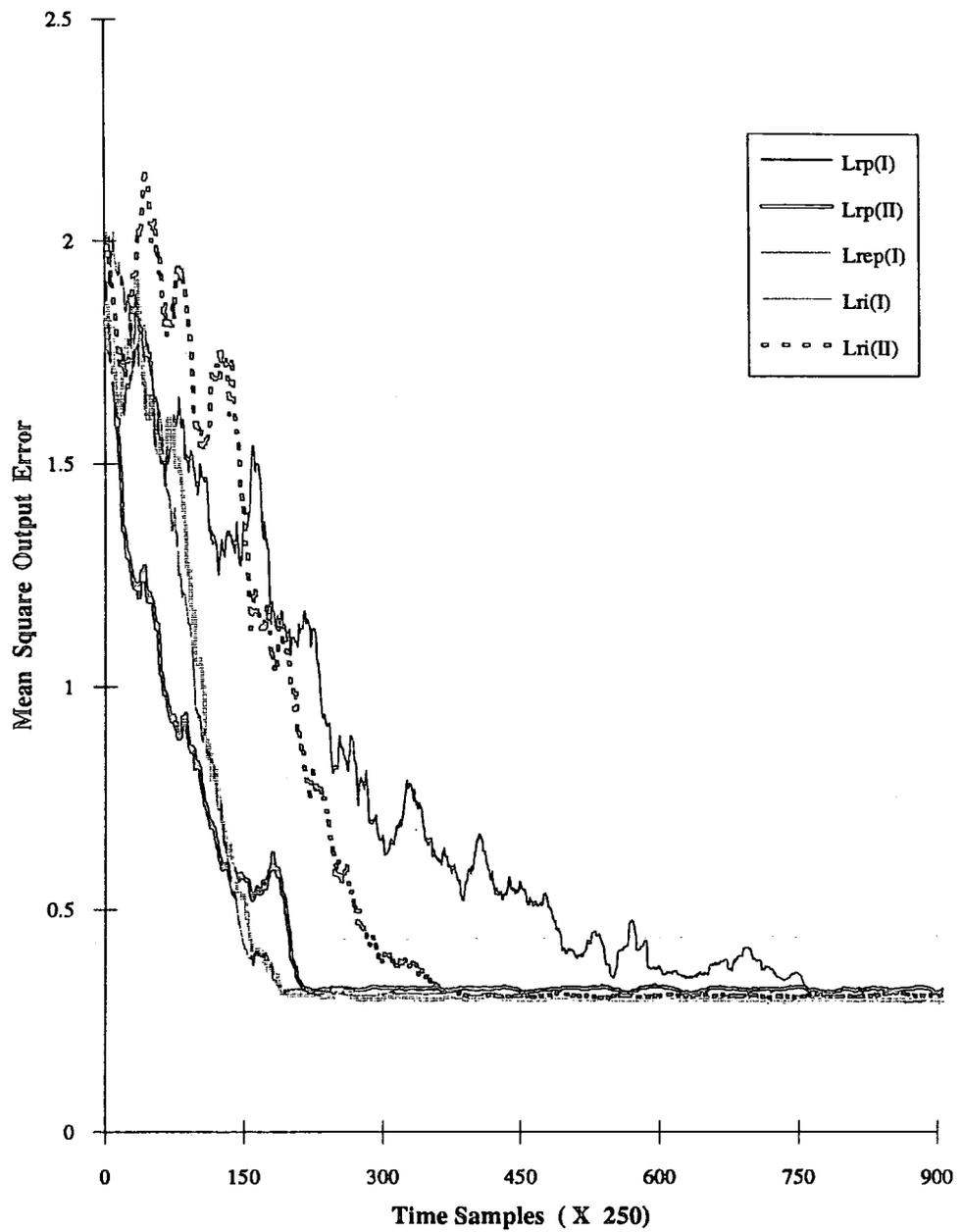


Figure 4.4: Performance of Standard Learning Algorithms

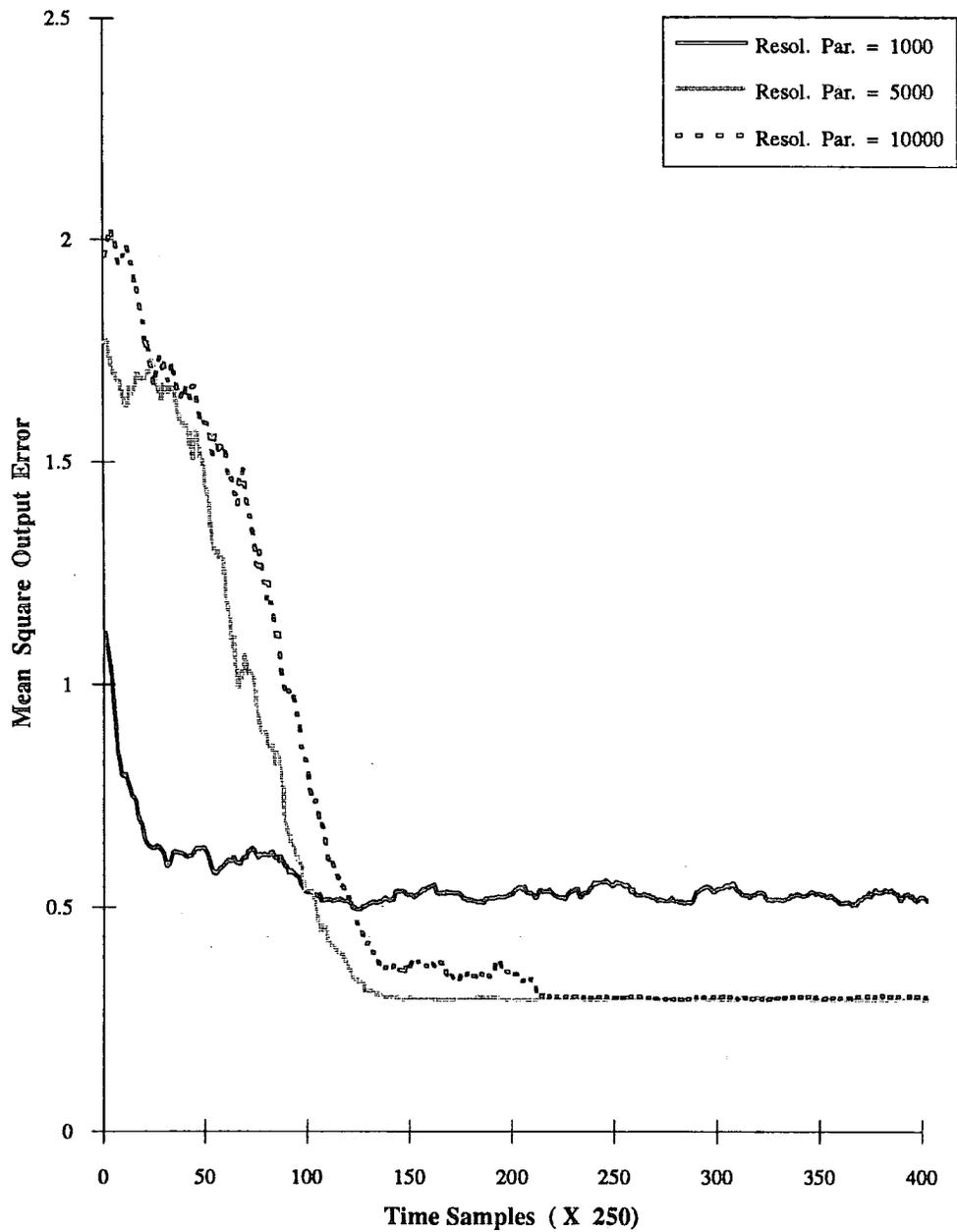


Figure 4.5: Performance of Discretised Learning Algorithms

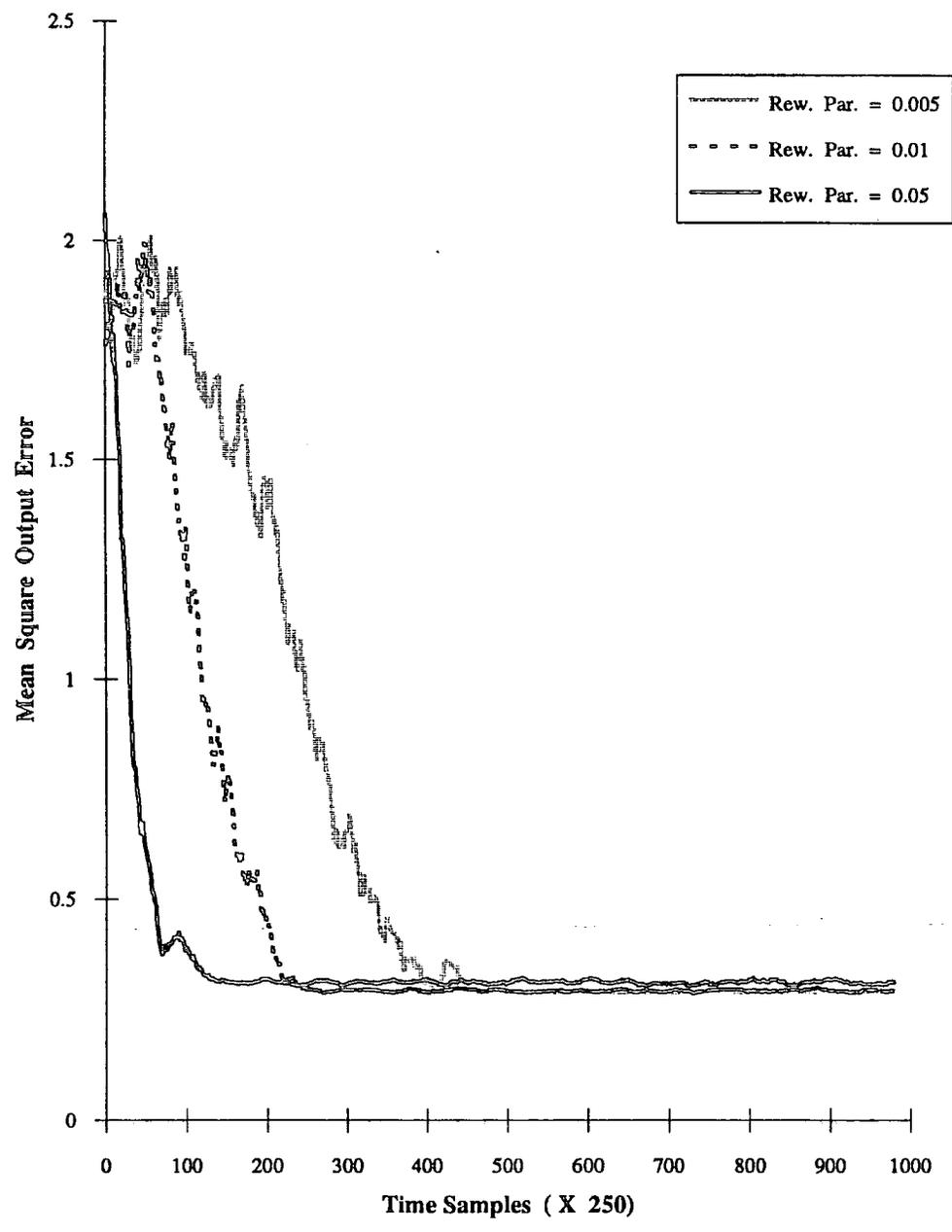


Figure 4.6: Performance of Estimator Learning Algorithms

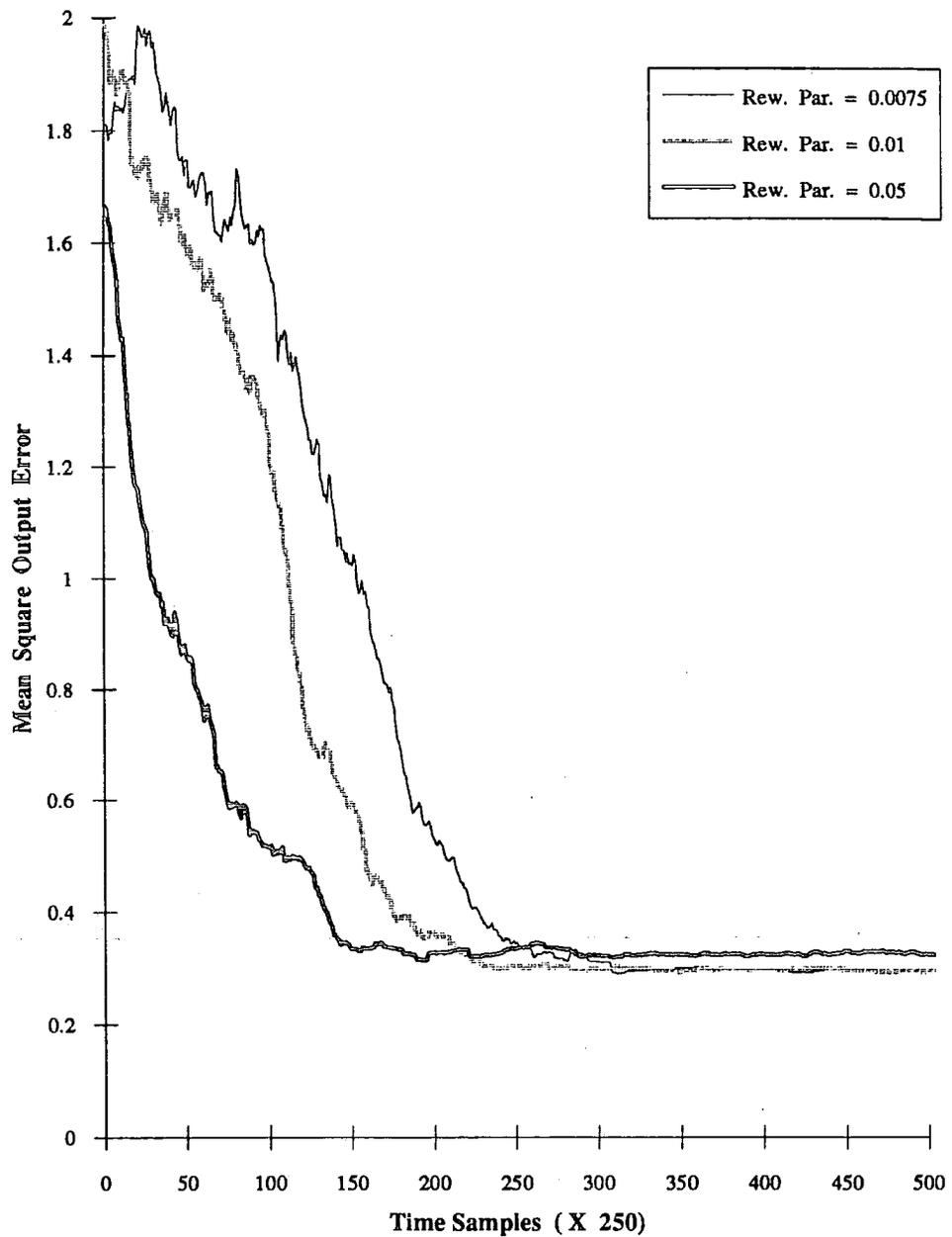


Figure 4.7: Performance of Pursuit Algorithms

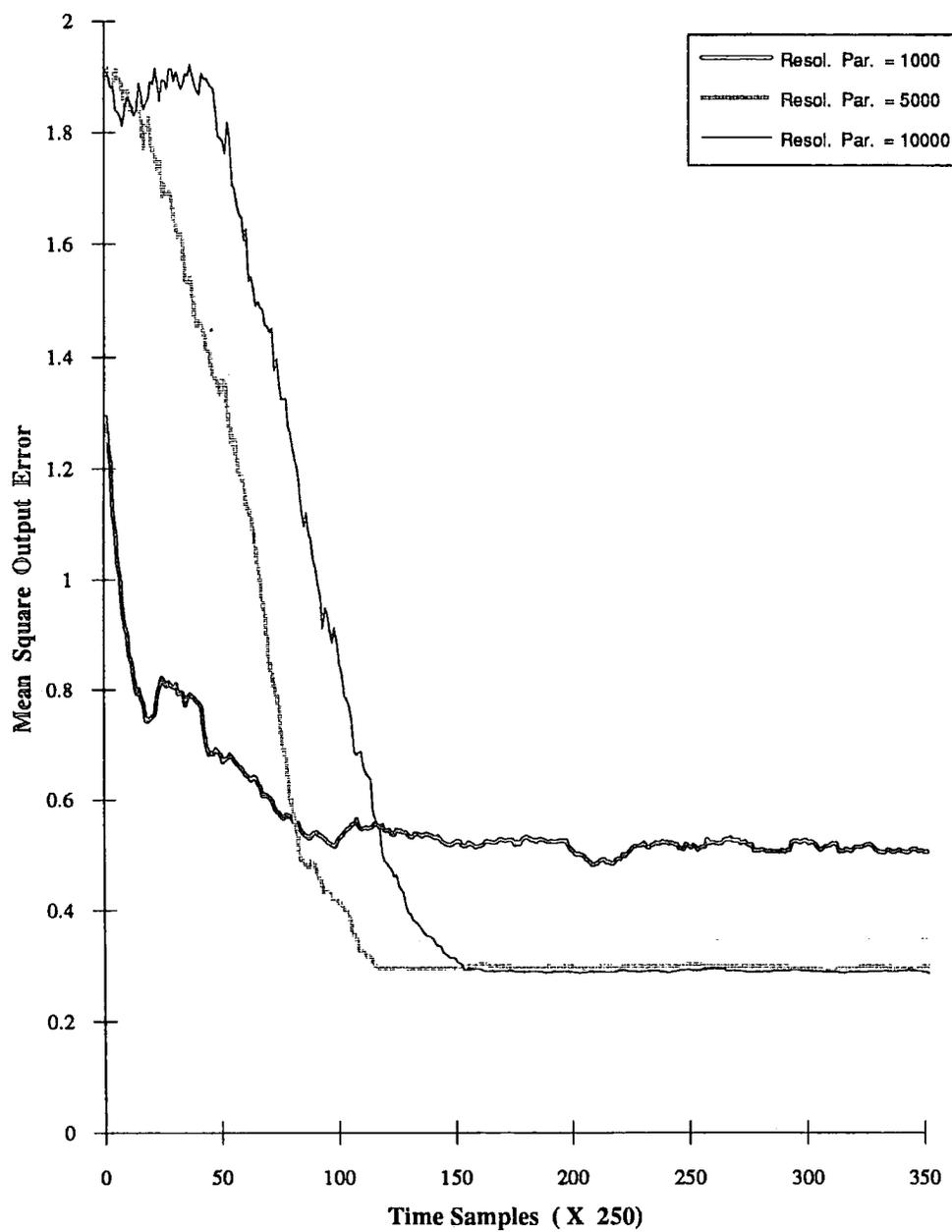


Figure 4.8: Performance of Discretised Pursuit Algorithms

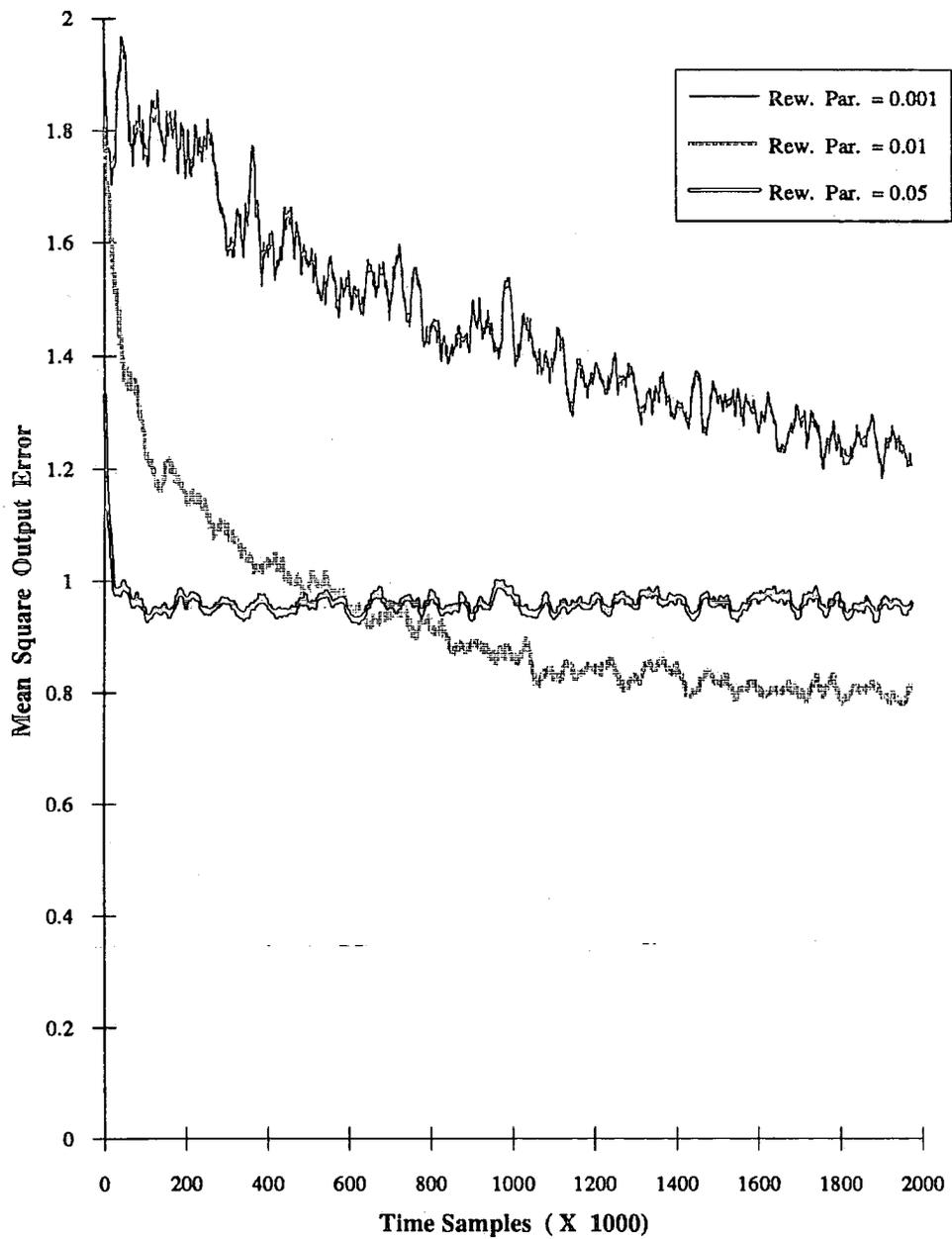


Figure 4.9: Performance of S-LRI Learning Algorithms (Old Normalisation)

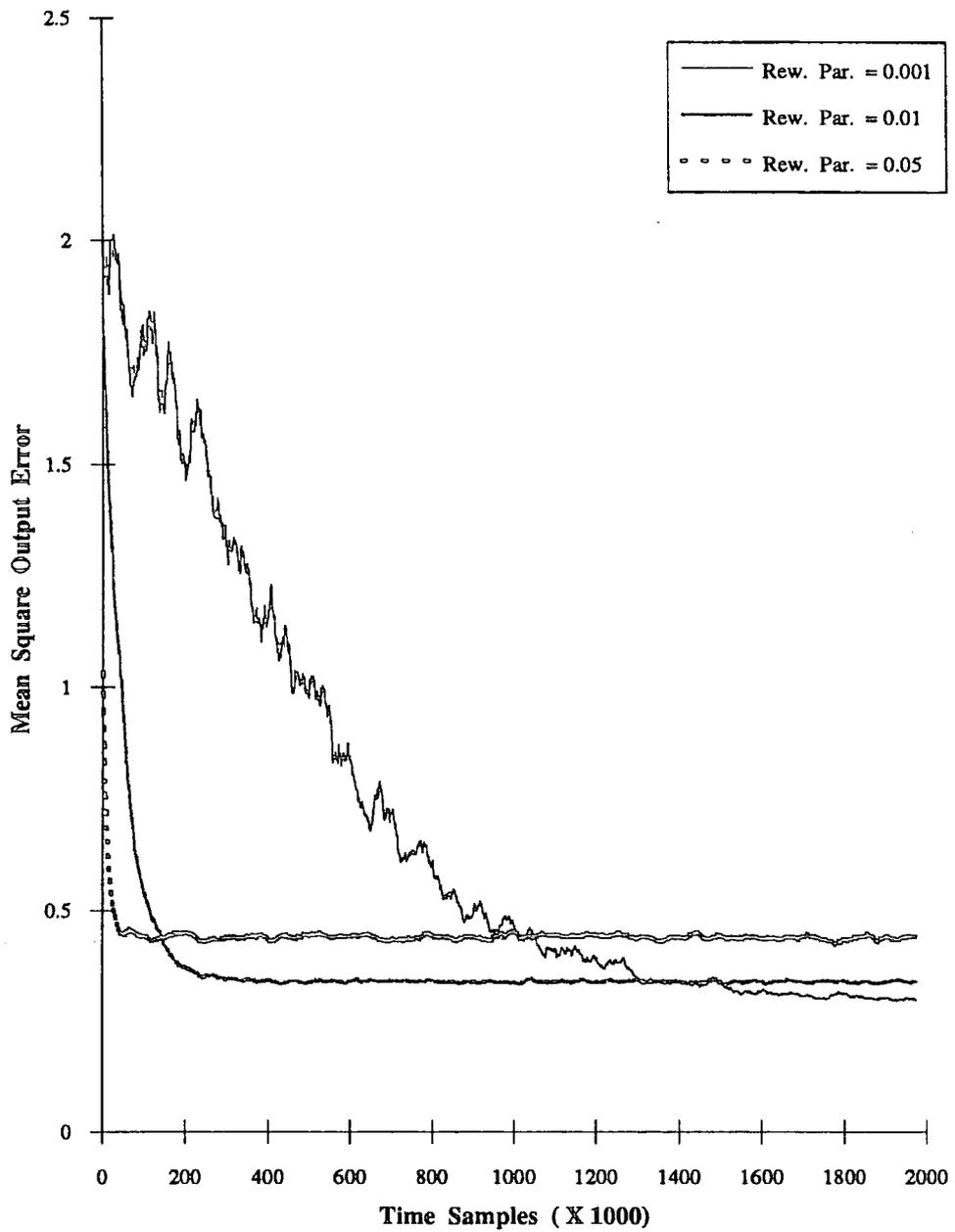


Figure 4.10: Performance of S-LRI Learning Algorithms (New Normalisation)

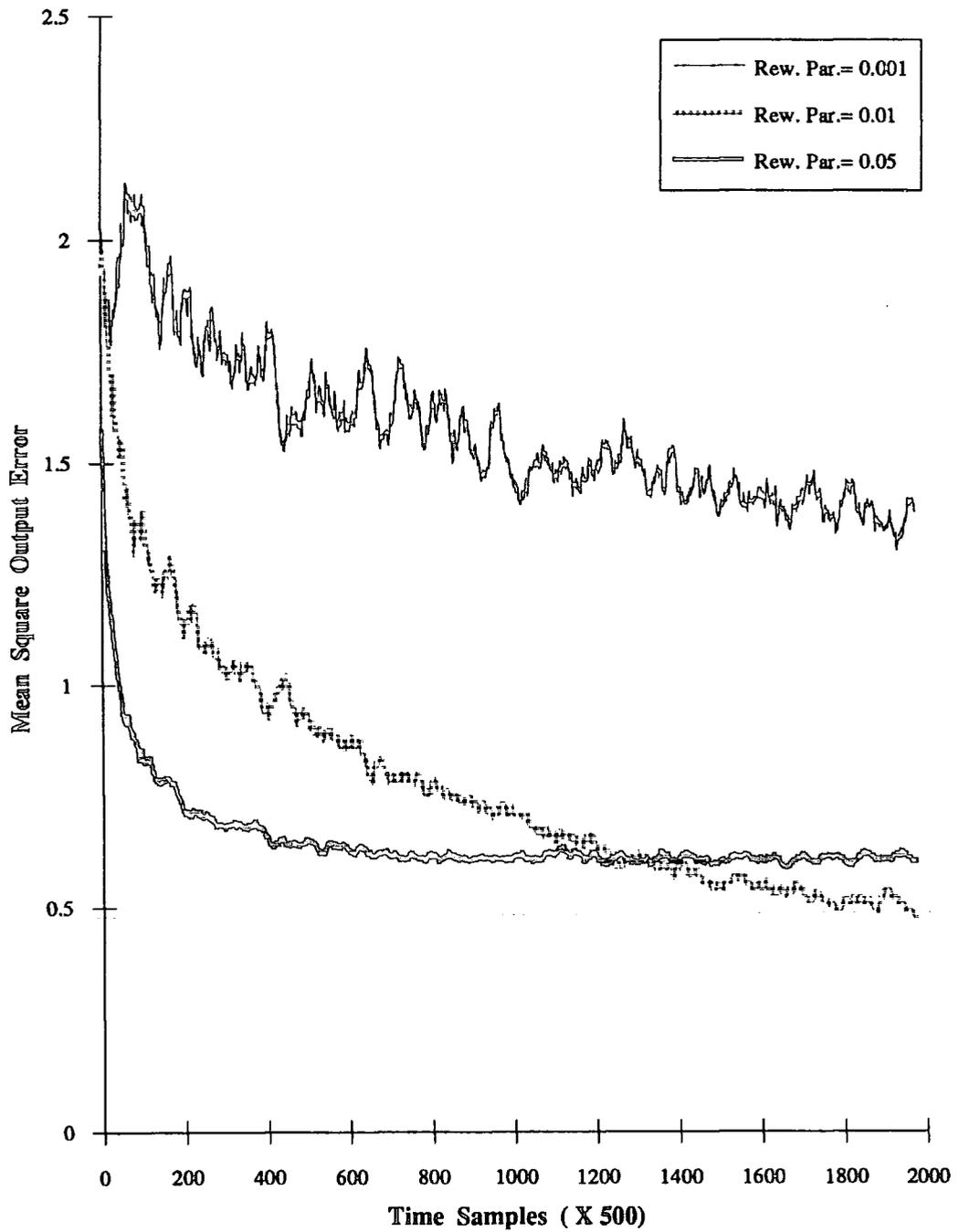


Figure 4.11: Performance of Estimator Learning Algorithms (S-Model) (Old Normalisation)

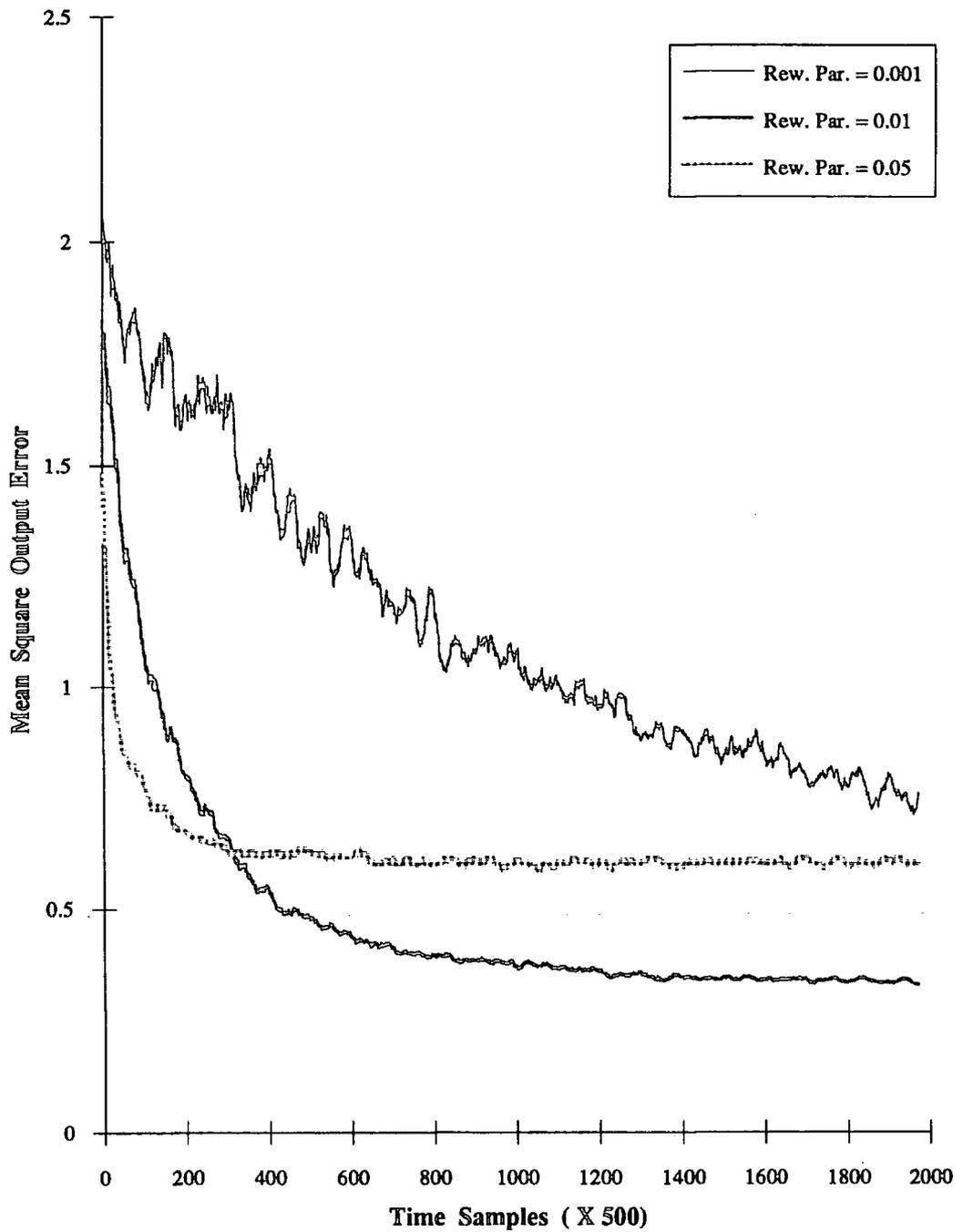


Figure 4.12: Performance of Estimator Learning Algorithms (S-Model) (New Normalisation)

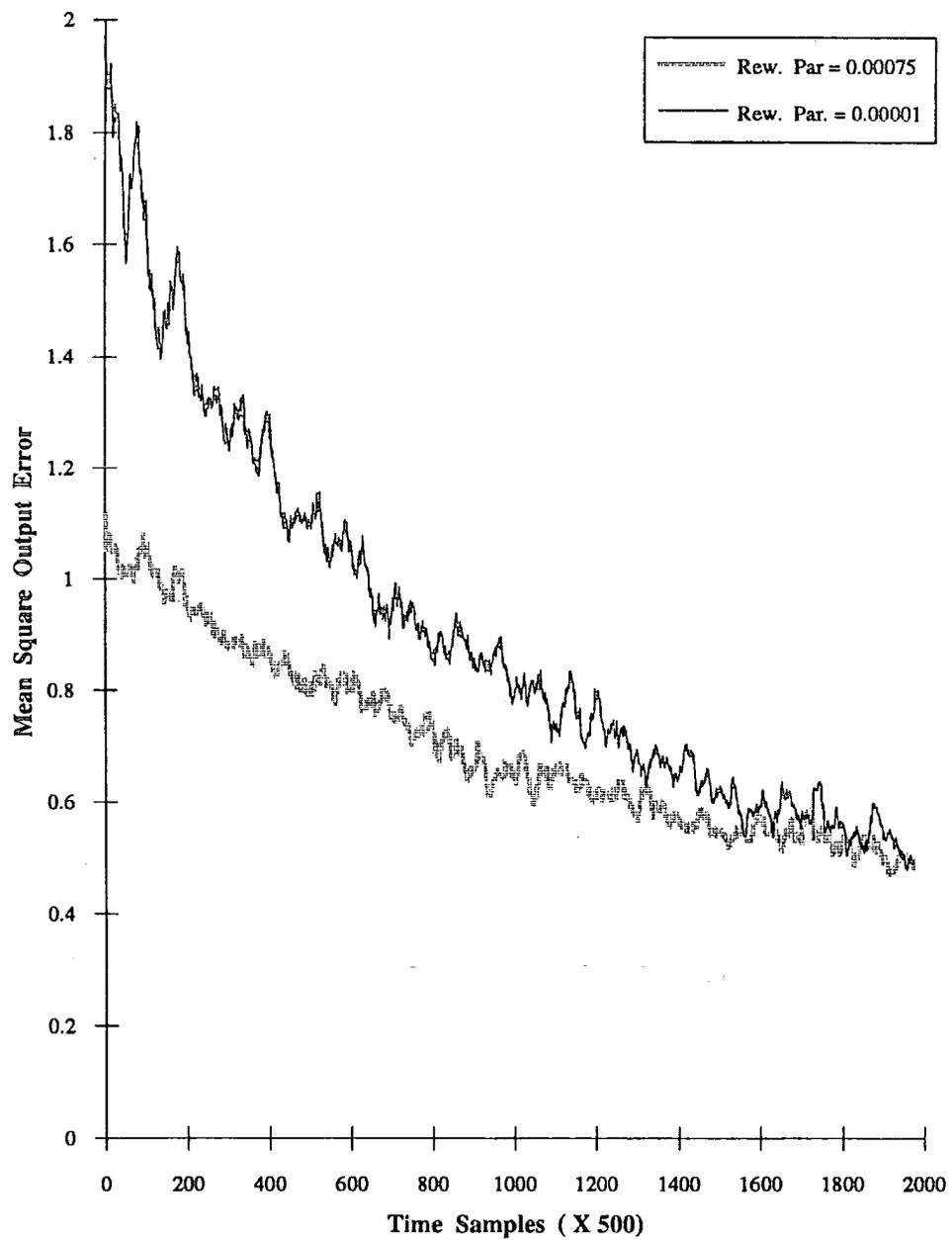


Figure 4.13: Performance of Relative Reward Learning Algorithms (S-Model) (Old Normalisation)

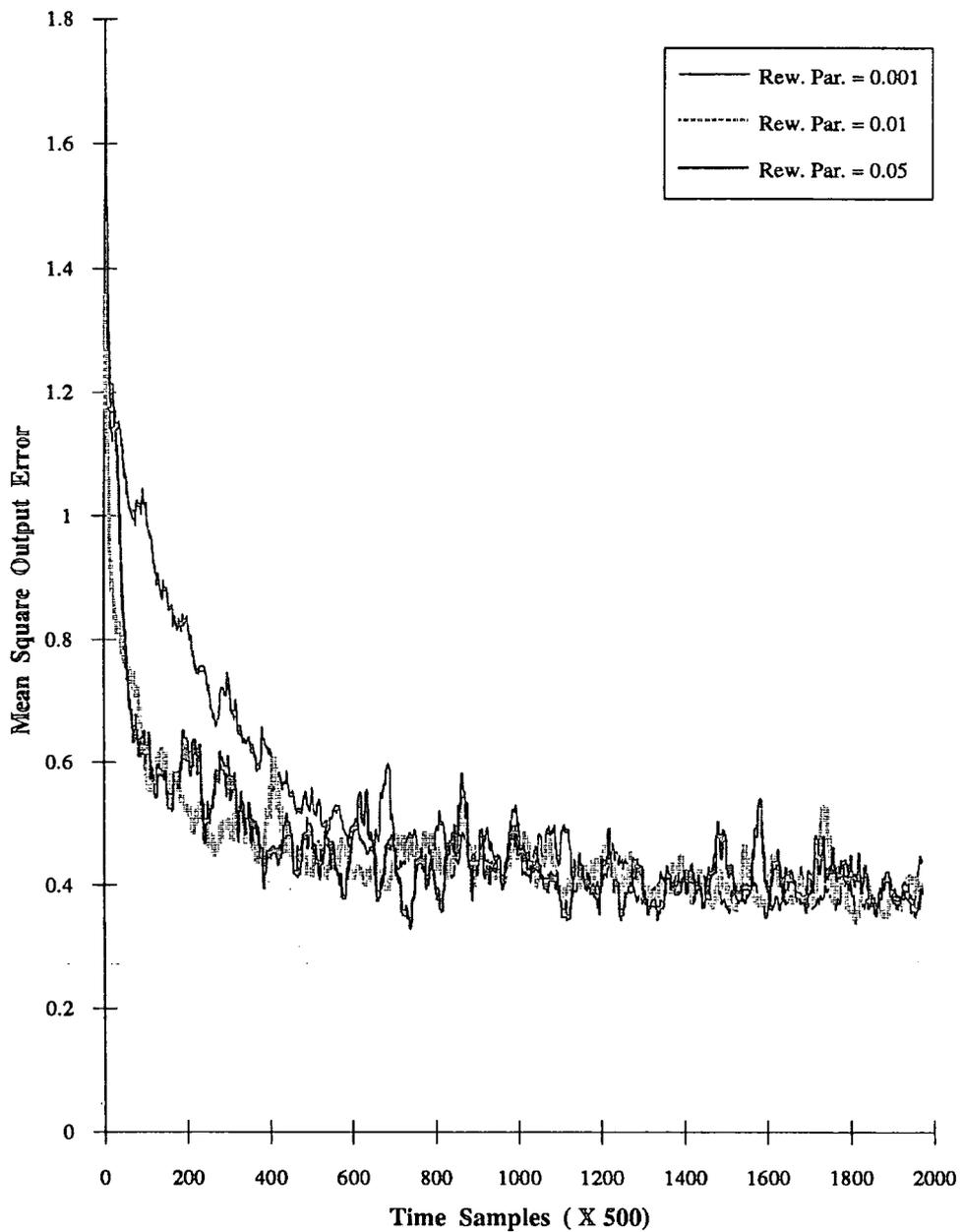


Figure 4.14: Performance of Relative Reward Learning Algorithms (S-Model) (Old Normalisation)

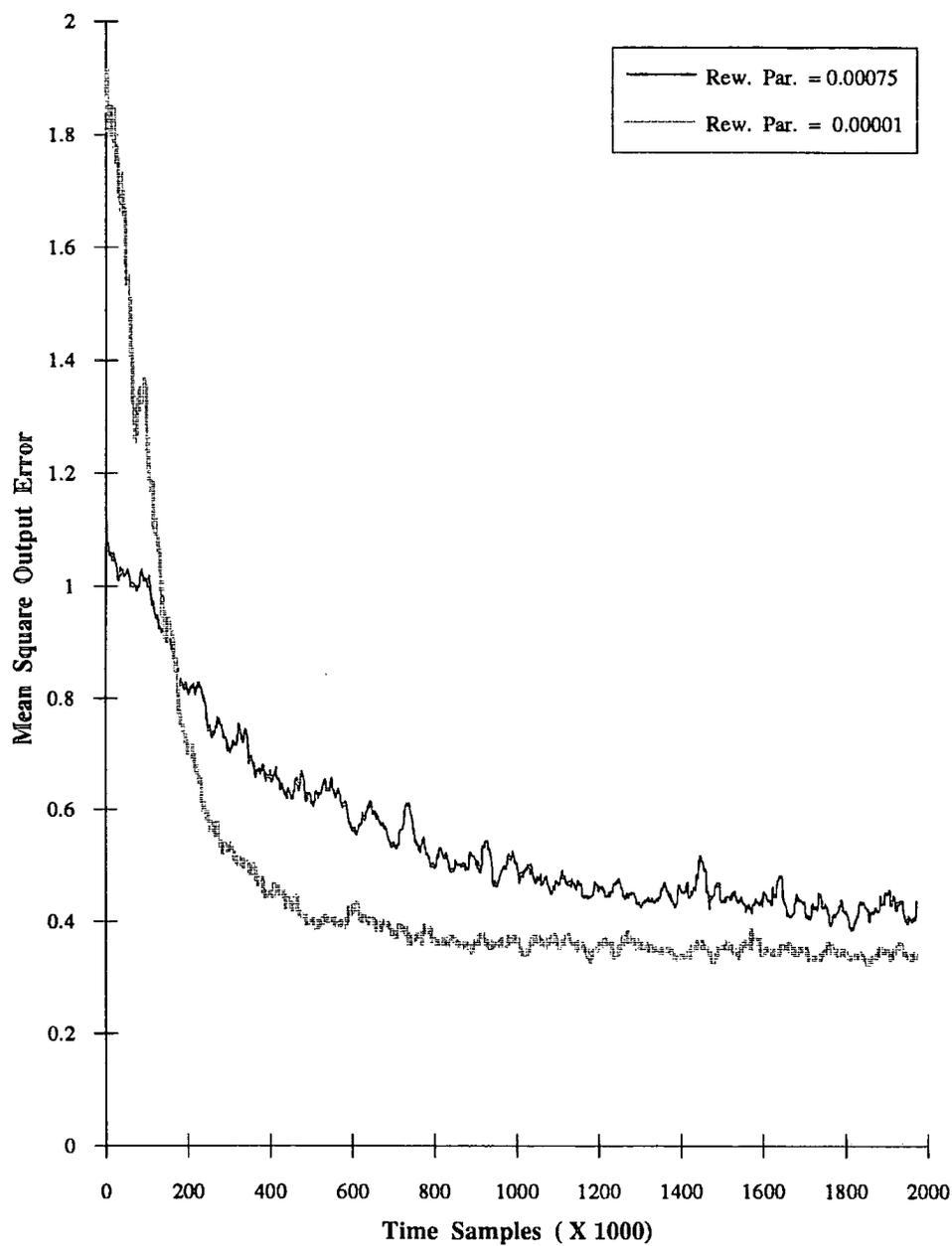


Figure 4.15: Performance of Relative Reward Learning Algorithms (S-Model) (New Normalisation)

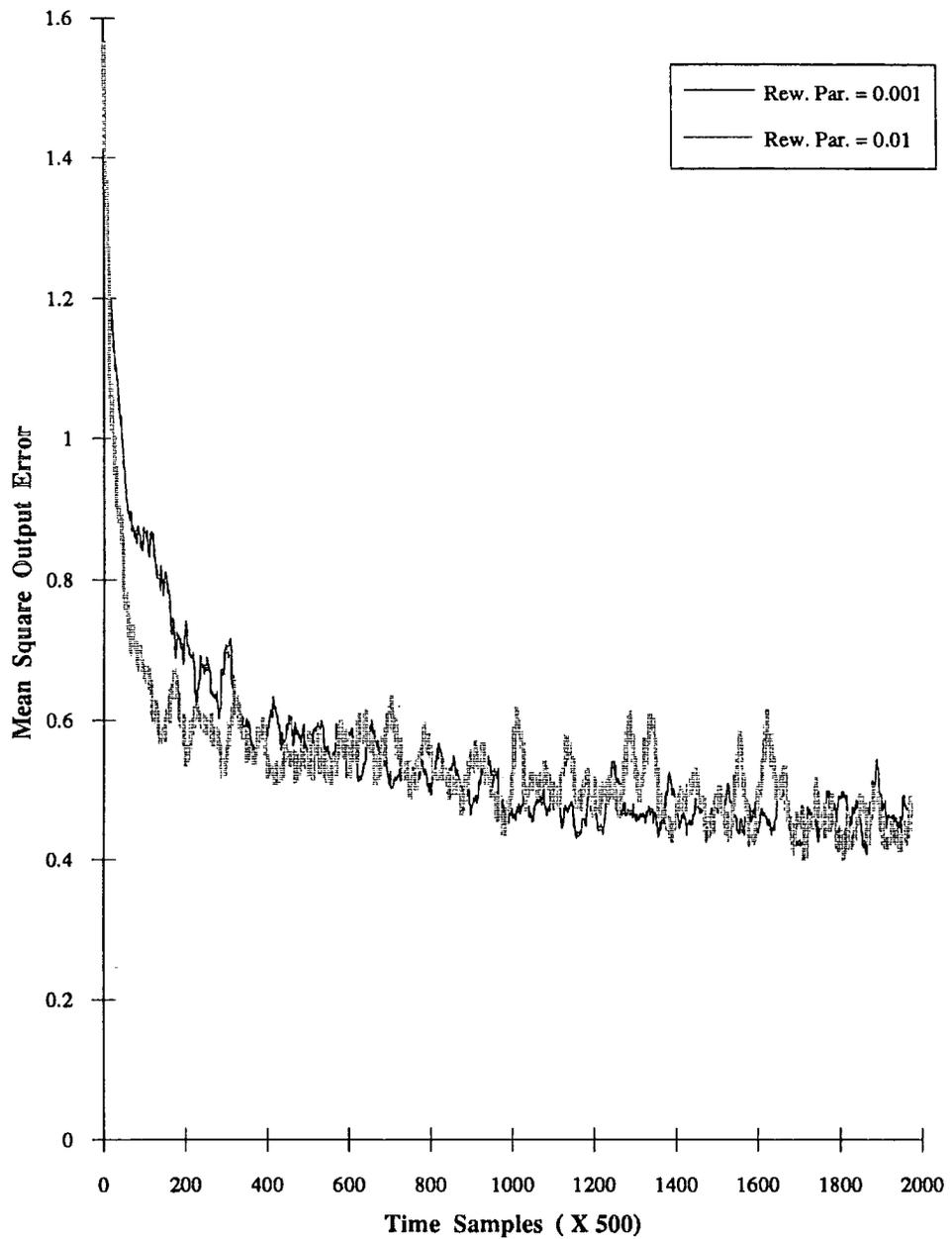


Figure 4.16: Performance of Relative Reward Learning Algorithms (S-Model) (New Normalisation)

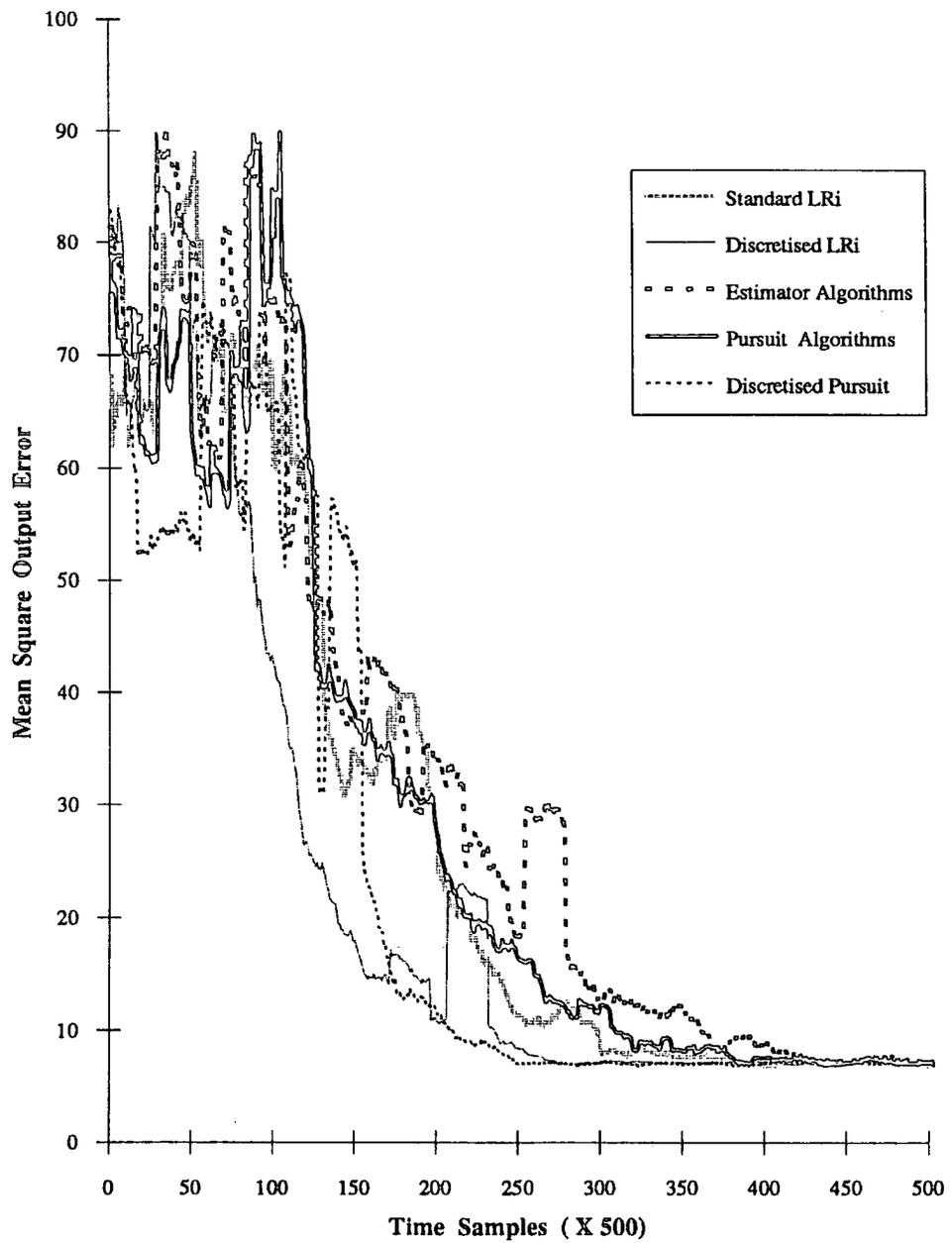


Figure 4.17: Performance of P-Model Learning Algorithms (Category (IV) Model)

# Chapter 5

## Genetic and Evolutionary Optimisation

### 5.1 Introduction

The process of evolution over many thousands of years has been a vitally important feature of the biological organisms which are presently found on earth. It has been used as a sort of *filtering process*, whereby organisms which are better adapted to the ever changing external environment survive, while organisms not so adaptable perish. This process of selective survival was initially recorded by the works of Charles Darwin and Alfred Russel, who referred to the process as *survival of the fittest*. Thus, the process of evolution could be viewed as a mechanism of optimisation whereby organisms being *optimised* are better equipped to survive in a variable environment. This led to the idea that evolution as seen in nature could be used as an optimisation tool as an alternative method to the standard optimisation strategies [FOW66, Hol75]. Subsequent research into the use of natural evolution as an optimisation technique has been intensive and has led to it being established as an important technique of optimisation called *Simulated Evolution*.

Traditional optimisation methods can be broadly classified into three categories [Gol89]:

- o Calculus based schemes

- Enumerative Schemes
- Random Search Schemes

*Calculus based search schemes* are based on using the gradient of the objective function and are the multidimensional generalization of finding the extrema of a function. As they use the concept of neighbourhood, their scope is local to the neighbourhood around the current search point and presences of local optima in the current area of search would result in the algorithm getting stuck in such an optima. Thus they are best used in a limited problem domain especially when dealing with real world problems. *Enumerative schemes* are very simple to implement as they involve in looking at every point in the search space to determine which is the best. However, the scheme results in enormous computational overheads as the size of search space increases. *Random search techniques* have been popular, but in the long run perform no better than enumerative schemes. A different approach to optimisation is to use *randomised techniques* which use random choice to guide the search algorithm through the parameter space. Two of the optimisation techniques which use such randomised techniques are *Simulated Evolution* and *Simulated Annealing*. Details of Simulated Annealing are presented in a subsequent chapter.

Simulated Evolution simulates a simplified version of the process of natural evolution on the computer. It is an effective numerical optimisation technique which is based on stochastic principles, thus making it extremely robust. The applications of the technique have been varied and include designing and training of neural networks, automatic control of nonlinear systems and optimal routing in telecommunications. Research in simulated evolution has progressed mainly on three fronts:

- Genetic Algorithms
- Evolutionary Strategies
- Evolutionary Programming

At the core of all three approaches lies the concept of a *population*, which has been derived from natural evolution. A population consists of a collection of structures which

in the case of simulated evolution represents possible solutions to the optimisation problem. In natural evolution these structures would correspond to the chromosomes found in all biological organisms and which determine the characteristics of the organism. Each structure is assigned a fitness value which determines the progress of the structure in subsequent generations, as structures with a large fitness value would tend to survive over an increased number of generations. These structures then undergo genetic operations which modify existing structures and generate new ones. The operations are of paramount importance to the method as they determine how new structures are formed from existing ones. This process is repeated to generate the members of subsequent generations. As the process works on the principle of survival of the fittest, structures which represent more optimal solutions and possess large fitness value, survive and propagate through the generations. Although the fundamental concepts of all three methodologies are derived from natural evolution, there exist significant differences between them which make each approach have different properties. The basic principles of evolutionary optimisation can be stated in an algorithmic form as follows:

### Evolutionary Optimisation

1. Initialise a population of structures.
2. Evaluate each structure and assign them a fitness value.
3. Create new structures by mating existing structures
4. Evaluate new structures and insert them into the existing population to form the next population.
5. Go to Step 3 if time limit is not exceeded.

As simulated evolution uses terminology that has been derived from natural evolution, these are clarified in the table given below:

Natural Evolution	Simulated Evolution
Chromosome	String
Gene	Feature/Character
Allele	Feature Value
Locus	Position on string
Genotype	Coded form of Parameters
Phenotype	Actual Parameter Set

Thus if a binary string is used as the chromosomal representation of a solution in a simulated evolution experiment, each position on the binary string would correspond to being a *locus*. A *gene* would then correspond to either a single or a group of bit locations. An *allele* would be the possible values the gene may have at each locus - the use of a binary string forces the allele values to be either a zero or an one.

Genetic Algorithms were devised by John Holland at the University of Michigan in the early seventies and was detailed in his pioneering work *Adaptation in Natural and Artificial Systems* ([Hol75]). Subsequently, research in genetic algorithms has experienced an exponential growth with applications in telecommunications, aircraft design, neural network architecture, control of gas pipeline transmission, seismic applications and jet turbine design. Evolutionary Programming and Evolutionary Strategies are two paradigms of simulated evolution which are very similar in structure and operation. Evolutionary Programming had its origins in the sixties based on the work of Fogel [FOW66] which concentrated on using simulated evolution as a tool for evolving artificial intelligence. Thereafter the scheme has been used in a number of diverse applications including underwater acoustics [Fog91a], robot path planning [MP90] and system identification [Fog91b]. The introductory work in Evolutionary Strategies was carried out in Germany at the University of Berlin by Rechenberg [Rec73] and further developed by Schwefel [Sch75]. Although evolutionary strategies are conceptually similar to evolutionary programming techniques, there are subtle but important differences between the schemes. The next sections present the detailed working of all the three paradigms highlighting both the similarities and differences.

## 5.2 Genetic Algorithms

### 5.2.1 Introduction

Ever since the evolutionary theory of biological change was accepted, the mechanics of evolution has attracted research interest. John Holland at the University of Michigan was interested in using the ideas from natural evolution to devise a technique to solve difficult optimisation problems. He called this method *Genetic Algorithms* as the principle of the method was based on ideas from genetics. Subsequent to Hollands work [Hol75], research activity in the area of genetic algorithms has been extensive and the method has found applications in a variety of engineering problems [Gol89, Dav91].

A genetic algorithm (GA) can be represented by a 8-tuple as follows:

$$\text{GA} = (\mathbf{P}^0, \lambda, l, f, s, c, m, i) \quad (5.1)$$

where

$\mathbf{P}^0$	$\equiv$	$(a_1^0, \dots, a_\lambda^0)$	Initial Population
$\lambda$	$\in$	$\mathbf{N}(\text{Set of Integers})$	Population Size
$l$	$\in$	$\mathbf{N}(\text{Set of Integers})$	Length of each string
$f$	:		Fitness/Objective Function
$s$	:		Selection Operator
$c$	:		Crossover Operator
$m$	:		Mutation Operator
$i$	:		Inversion Operator

The initial population  $\mathbf{P}^0$  is created by randomly generating  $\lambda$  binary strings, each binary string being a coded form of the parameters of the optimisation process. The multi-parameter case is handled by concatenating the string representations of all the parameters. This process is elaborated in a subsequent section. The parameter  $\lambda$  is the size of the population and is an important parameter of the genetic algorithm.

The length  $l$  of each binary string determines the precision with which the actual parameters have been coded.

An important concept which arises as a result of using binary strings is that of a *Schema*. A Schema is a similarity template which describes a subset of strings with similarities at certain strings positions. For example, suppose a binary string representation is defined using six bits. Then

$$\begin{array}{cccccc} 0 & * & * & 1 & * & * \\ 1 & 1 & * & * & 0 & * \end{array}$$

are two examples of schemata defined on the string. The  $*$  refers to a don't-care condition and can be either a 1 or a 0. Thus schemata are defined to be elements of  $\{0, 1, *\}^l$  where  $l$  is the length of the binary string. Two important properties of a schema are its *order* and *defining length*. The *order* of a schema  $H$  denoted by  $O(H)$  is the number of fixed positions (in a binary coding, the number of 1's and 0's). The *defining length* denoted by  $\delta(H)$  is the distance between the first and last specific string position. For example the schema

$$* 1 * * * 0 1$$

has an order of 3 and a defining length of 5 (i.e.  $7 - 2$ ).

Genetic algorithms obtain most of their exploratory power by the sampling and distribution of schemata during the creation of new generations. It has been proved [Hol75, Gol89] that if a genetic algorithm operates with a population size of  $\lambda$ , then the number of schemata processed during a single generation is  $O(\lambda^3)$ . This effect is known as *Implicit Parallelism*. The concept of schemata also strengthens the case for the binary coding scheme, as it has been shown [Gol89] that maximum number of schemata is processed when a binary coding is employed.

The fitness function  $f$  assigns a real value to each string which determines the survivability of a particular string in subsequent generations. A large fitness function results in a particular string surviving in subsequent generations either as itself or

as offspring which have been created from it. This concept is made clear when the genetic operation of selection is explained.

A genetic algorithm operates on a population of string structures each of which represent a possible solution to the problem under consideration. An important difference between genetic algorithms and the other evolutionary optimisation algorithms is that genetic algorithms operate on the *genotypic* representation while the evolutionary strategies and evolutionary programming methods operate on a *phenotypic* representation. This distinction means that genetic algorithms operate on a coded form of the actual parameter space while the other evolutionary schemes operate on the actual parameters themselves. Thus in genetic algorithms, the coding scheme used to represent the parameters is of significant importance. Though Holland [Hol75] stated that the binary coding is the optimal coding scheme, subsequent work has shown that this need not be so [Dav91]. The main argument against using a binary coding is it unnecessarily constrains the problem.

Using a binary coding would entail each parameter being coded as a  $l$  bit string. The number of bits  $l$  dictates the precision of the coding process as a larger number of bits would represent a parameter more precisely. Depending on the function to be optimised, a parameter value may be constrained to lie between certain limits. This constraint satisfaction is elegantly handled in genetic algorithms by using a linear mapping, which maps each binary coded form of a parameter to a particular parameter value. This is explained using the following example: Suppose a parameter is constrained to lie between the limits of  $P_{min}$  and  $P_{max}$ , and is coded using a binary string of  $l$  bits. Then the coded form would have  $2^l$  discrete values and the linear mapping would then map the values  $\{0, \dots, 2^l - 1\}$  of the binary string to real values lying between  $P_{min}$  and  $P_{max}$ . An important feature of evolutionary optimisers is the natural way the problem of dimensionality is handled [Gol89]. The problem of dimensionality plagues most current optimisation schemes which break down on problems of moderate size and complexity. The genetic algorithm deals with the dimensionality problem as follows: Each parameter of the process is as usual coded using a binary string. *The binary codings of all the parameters are then concatenated to form a larger string which forms the chromosomal representation to be used in a*

as offspring which have been created from it. This concept is made clear when the genetic operation of selection is explained.

A genetic algorithm operates on a population of string structures each of which represent a possible solution to the problem under consideration. An important difference between genetic algorithms and the other evolutionary optimisation algorithms is that genetic algorithms operate on the *genotypic* representation while the evolutionary strategies and evolutionary programming methods operate on a *phenotypic* representation. This distinction means that genetic algorithms operate on a coded form of the actual parameter space while the other evolutionary schemes operate on the actual parameters themselves. Thus in genetic algorithms, the coding scheme used to represent the parameters is of significant importance. Though Holland [Hol75] stated that the binary coding is the optimal coding scheme, subsequent work has shown that this need not be so [Dav91]. The main argument against using a binary coding is it unnecessarily constrains the problem.

Using a binary coding would entail each parameter being coded as a  $l$  bit string. The number of bits  $l$  dictates the precision of the coding process as a larger number of bits would represent a parameter more precisely. Depending on the function to be optimised, a parameter value may be constrained to lie between certain limits. This constraint satisfaction is elegantly handled in genetic algorithms by using a linear mapping, which maps each binary coded form of a parameter to a particular parameter value. This is explained using the following example: Suppose a parameter is constrained to lie between the limits of  $P_{min}$  and  $P_{max}$ , and is coded using a binary string of  $l$  bits. Then the coded form would have  $2^l$  discrete values and the linear mapping would then map the values  $\{0, \dots, 2^l - 1\}$  of the binary string to real values lying between  $P_{min}$  and  $P_{max}$ . An important feature of evolutionary optimisers is the natural way the problem of dimensionality is handled [Gol89]. The problem of dimensionality plagues most current optimisation schemes which break down on problems of moderate size and complexity. The genetic algorithm deals with the dimensionality problem as follows: Each parameter of the process is as usual coded using a binary string. *The binary codings of all the parameters are then concatenated to form a larger string which forms the chromosomal representation to be used in a*

*population*. To assign a fitness value to each string in the population, the strings are decoded to form the actual parameters of the objective function. The function value then obtained using these parameters in the objective function are used as the fitness value of that string. In some instances, the raw function value itself is not used as the fitness measure, instead a modified value of the raw function value is employed. Thereafter the strings of each population undergo the standard genetic operations of selection, crossover and mutation to generate the strings of the new population. These operations are explained in the next section.

### 5.2.2 Standard Genetic Operations

There have been differences in the literature as to which operations constitute standard genetic operations. The genetic operations presented in this section are as given by Holland in [Hol75] and Goldberg in [Gol89]. These set of operations have also been used by DeJong [DeJ75] where it is referred to as plan R1 (reproductive plan 1).

#### Selection Operation

The selection operation decides which of the strings in a population are selected for further genetic operations. Each string  $i$  of a population is assigned a fitness value  $f_i$ . The fitness value  $f_i$ 's are used to assign a probability value  $p_i$  to each string. The probability value  $p_i$  assigned to a string is calculated as

$$p_i = \frac{f_i}{\sum_{l=1}^{\lambda} f_l} \quad (5.2)$$

Thus, from the above equation it can be seen that strings with a large fitness value have a large value of probability of selection. Using the probability distribution defined by Equation [5.2], strings are selected for further genetic operations. This scheme of selection is referred to by researchers by various names like stochastic sampling with replacement [Gol89] and proportional selection [Hol75].

### Crossover Operation

The crossover operation as stated by Holland, gives the genetic algorithm most of its exploratory power. The parameters defining the crossover operation are the probability of crossover ( $p_c$ ) and the crossover point. The crossover operator works as follows:

- From a population, two strings are drawn at random.
- If the crossover probability is satisfied, a crossover point is selected at random so as to lie between the defining length of a string, i.e.  $x \in \{1, \dots, l-1\}$  ;  $x \equiv$  crossover point.
- The sub-string to the left of the first string and to the right of the second string are swapped to create a new string. A similar operation is performed with the two remaining substrings. Thus two new strings are generated from the parent string.

The operation is illustrated by means of a example given below:

#### Before Crossover

```
0 0 1 1 | 0 1 1
1 1 1 0 | 1 1 0
```

#### After Crossover

```
0 0 1 1 | 1 1 0
1 1 1 0 | 0 1 1
```

The usual value used for the crossover probability ( $p_c$ ) lies between  $0.6 \approx 0.8$ . According to Holland, the crossover operation is responsible for combining short high performing schemata which in tandem generate strings with a larger fitness value.

However, it is also likely that the offspring generated may be worse than the parent strings. The crossover operation as given in [Hol75] used the one-point crossover operator given above. Current research has shown [Sys89, DS91] that increasing the number of crossover points leads to better performance of the genetic algorithm. Simulation studies carried out in this thesis suggest that this indeed is true.

### Mutation Operation

In genetic algorithms mutation is usually assigned a secondary role. It is primarily used as a background operator to guard against total premature loss of an allele at a particular locus which effectively results in the search space being reduced. Use of the crossover operation by itself would not recover this loss. The mutation operator allows for this by changing the bit value at each locus with a certain probability. Thus every locus on the binary string has a finite probability of assuming either a value of '0' or '1'. The probability of this change is the defining parameter of the operation and is referred to as the probability of mutation ( $p_m$ ) and is assigned a very small value ( $\approx 0.001$ ). The operation is explained below with an example:

#### Before Mutation

**0** **0** 1 1 0 1 1

#### After Mutation

1 **0** 1 1 0 **0** 1

The bit values which have been affected by the mutation process are shown in bold. Holland had envisaged a secondary role for the mutation operator, as too large a value of the mutation probability would result in breaking up of optimal schemata, thus reducing the efficiency of the method. But this view has been challenged by subsequent research and now a greater emphasis is given to the mutation operator. Indeed, the evolutionary strategies and evolutionary programming approaches to simulated evolution use mutation as a primary operator.

### Inversion Operation

Holland had also included with the above operators a fourth operator which formed part of the genetic operations by which a new string could be formed from the parent strings. This was the inversion operator which operates on a single chromosome. The inversion operator inverts the order of the bit values between two randomly selected points on the parent string. Though this operation has been observed in nature, it has not been used commonly in genetic algorithms as it adds to the computational complexity of the process. Some details of the inversion operator is presented in [Gol89]. This operator has not been used in the genetic algorithm simulation experiments conducted in this thesis.

The genetic operations detailed above form the backbone of a genetic algorithm. Thus the operation of a genetic algorithm would proceed as follows: The initial population of  $\lambda$  strings are generated randomly and a fitness value assigned to each string. Using the fitness values, a probability measure is calculated for each string. Using this probability distribution, two strings are drawn from the population. These two strings then undergo the crossover operation if the crossover probability ( $p_c$ ) is satisfied. Thereafter each of the newly generated strings undergo the mutation operation resulting in two new strings which forms a part of the new population. This sequence is repeated till there are  $\lambda$  strings in the new population. The process is then repeated to create new generations. In the next section, we present improvements to the basic techniques discussed above.

### 5.2.3 Improved Genetic Operations

Some of the problems using the standard genetic operators were slow rate of convergence and premature convergence to non-optimal locations even when optimising simple unimodal surfaces. To overcome these deficiencies, a host of improvements have been suggested by various researchers. A few of these techniques are reviewed in the next sections.

### Alternate Coding Schemes

In genetic algorithms, the effect of a single bit mutation at the genotype level was not easily noticeable at the phenotypic level and depended mainly on the coding scheme used. Using the binary coding scheme a single mutation caused a change which depended on the location of the bit. An improved coding scheme which alleviates this problem is the Gray coding in which adjacent phenotypic values differ by a single bit (Hamming distance of 1). This scheme yields better performance in parameter optimisation problems and has been noted by Hollstein [Hol71] and more recently by Caruana and Schaffer [CS88]. Another coding scheme which has been suggested is to use the real parameters themselves - i.e. the genetic algorithm in this case operates on a phenotypic level. This scheme has been used in some of the real world applications presented in [Dav91].

### Alternative Selection Schemes

A number of alternative selection schemes have been listed in [Gol89]. These include

- Deterministic Sampling
- Remainder stochastic sampling without replacement
- Remainder stochastic sampling with replacement
- Stochastic sampling without sampling
- Stochastic Tournament

Complete details of the above schemes are given in [Gol89]. It has been shown by simulations that the stochastic remainder selection schemes results in a superior performance as compared to the other schemes.

A basic technique which has been employed to improve the performance of the standard genetic algorithms is to *scale* the objective function. A common problem experienced using the standard GA is the presence of a superindividual<sup>1</sup> in a population, which results in loss in diversity in subsequent generations as this individual

---

<sup>1</sup>A string with a large fitness value compared to the other strings in the population



dominates and multiplies rapidly. This can be avoided by scaling back the objective function to prevent the population being dominated by a few individual strings. Scaling the objective function also helps in stretching the objective function at the final stages of a run thereby introducing more competition between the member strings. The different scaling schemes which have been used include linear scaling, sigma truncation, and power law scaling ([Gol89, HB92]). Power law scaling involved using a specified power of the raw fitness value as the scaled fitness and has been used in this thesis. This scheme was suggested by Gillies [Gil85] and detailed in [Gol89].

### Alternative Crossover Schemes

The main argument favouring the use of the one-point crossover is the initial formal analysis conducted by Holland who showed that optimal allocation of high performance schemata was possible, when the disruptive effects of the genetic operations are minimised. This was one reason why the mutation probability is kept at a low value. The only other operator which introduced disruption in the allocation of schemata was the crossover operator. Since the crossover probability is kept at a large value, the disruptive effects are minimised when the number of crossover points are kept at a low value. Thus the number of crossover points is usually kept low, i.e. 1 or 2. However recent research [Sys89, DS90, DS91] have shown that a higher number of crossover points is beneficial to the search process. This led to the formulation of the n-point crossover operation and the uniform crossover operator.

Uniform crossover involves swapping the alleles of the two parents with probability 0.5. This involves on a average  $(L/2)$  crossover points for a string of length  $L$ . Spears and DeJong [DS91] have shown that a parameterised uniform crossover scheme gives better results as compared to standard single point crossover, especially when the population size is small. Parameterised uniform crossover involves making the probability of swapping a parameter of the operation. Thus parameterised uniform crossover with a parameter value of 0.5 reduces to the standard uniform crossover operation. An immediate advantage of the parameterised uniform crossover operation is that the only defining parameter of the crossover operation is now the probability of swapping. It has been shown in [DS91] that lowering the value of this probability

results in the crossover operation having less disruptive effects than is the case with the 2-point crossover.

The above section detailed some improved schemes over the standard genetic operations. Goldberg [Gol89] presents details of more complex operators such as dominance, diploidy, intrachromosomal duplication, deletion, translocation, segregation, niche exploitation and speciation. Dominance and diploidy play an important role in the case of non-stationary environments as they present a method of implementing long term population memory.

#### 5.2.4 Adaptive Extensions of Genetic Algorithms

One of the interesting areas where current research in GAs is active, is in developing techniques whereby the parameters of the Genetic Algorithm can themselves learn to attain the optimal values as is required by the particular optimisation problem. The important parameters which define a Genetic Algorithm are the population size, the crossover probability and the mutation probability. This problem was recognised early on by DeJong [DeJ80] who had suggested that the rate of mutation itself undergo adaptation in parallel with the exploration of the parameter space. He suggested the addition of an extra sequence of bits on the chromosome which would code the rate of mutation. These extra bits would undergo genetic modifications via the selection and other genetic operators in the same manner as the other bits of the string.

Another approach which was used by Grefenstette [Gre86] involved using a meta-level Genetic Algorithm which controlled the values of the parameters of a genetic algorithm which was involved in the main search process. The values for the parameters of the meta-level genetic algorithm were set to the values obtained by DeJong in [DeJ75] which was defined as the standard genetic algorithm. The contribution of this work was to show that while it was possible to obtain optimal parameter values for a GA, the algorithm showed good performance over a range of parameter values, thus illustrating the robustness of the scheme.

A new approach to this problem has been a new class of genetic algorithms known as *messy Genetic Algorithms (mGA)*. These have been proposed by Goldberg and

colleagues in [GDK89, GDK90]. The main differences between mGAs and standard Genetic Algorithms are as follows:

- mGAs use *variable length codes* that may be overspecified or underspecified with respect to the problem being solved.
- mGAs use *cut* and *splice* operators instead of the fixed length crossover operations.
- mGAs divide the evolutionary process into two phases: an initial phase which contain building blocks of all specified lengths, and a juxtaposition phase where by means of the cut and splice operators, the population is enriched leading to the globally optimal strings.
- mGAs use competitive templates to accentuate salient building blocks.

Simulation studies have shown that the mGAs always locate the globally optimal strings. More details of mGAs are given in [GDK89, GDK90].

The next section looks at the paradigm of Evolutionary Strategies and compares the scheme to Genetic Algorithms.

## 5.3 Evolutionary Strategies

### 5.3.1 Introduction

Evolutionary Strategies (ESs) are another optimisation technique which are based on principles of natural evolution. The basic concepts of the algorithm are very similar to that of genetic algorithms [HB92]. The algorithm operates on a population of string structures, each of which represents a solution to the optimisation problem. Each string then undergoes genetic modifications which result in a new string which then form part of a new population. Multi-parameter cases are handled in the same way as is done in GAs by concatenating the string representations all the parameters of the optimisation process. As for the genetic algorithms, the guiding principle of evolutionary strategies is *survival of the fittest*. Thus strings which represent near

optimal solutions to the optimisation problem survive for future generations leading to more optimal solutions.

The initial work in ESs was carried out at the Technical University of Berlin in the early sixties where it was for experimental optimisation problems like shape optimisation of a bent pipe, and optimisation of a PID regulator [Rec73]. Subsequent work included applications in numerical optimisation and binary parameter optimisation. The different ESs which have been developed so far are presented in the next sections. Extensive work involving ESs have also been carried out at the University of Dortmund where a through comparison between GAs and ESs has been reported [HB92].

### 5.3.2 Standard Evolutionary Strategies

#### (1+1)-ES

The (1+1)-ES was the earliest and simplest of the ESs which were devised. There was no real concept of a population as the algorithm operated with single parent string (real-valued vector) which produced an offspring by adding normally distributed random numbers to the parent vector. The single parent string was composed of the  $n$  parameter values. Associated with each parameter  $x_i$ , was the standard deviation value  $\sigma_i$  which decided the size of the neighbourhood of the search process for that parameter when creating the offspring string. The better of both individuals was then used as the parent of the subsequent generation. As was mentioned before, an important difference between GAs and ESs is the fact that GAs operate on a genotypic level (coding of the real parameters), while ESs operate at the phenotypic level using the parameter values themselves as genetic material. The descendent was created by a mutation process which is applied to all  $n$  components of the parent vector. This is accomplished by using normally distributed random numbers as follows:

$$x_i(k+1) = x_i(k) + N_{0,\sigma_i} ; i = 1, \dots, n \quad (5.3)$$

where

$x_i(k)$   $\equiv$  The value of the parameter  $x_i$  at time  $k$

$N_{0,\sigma_i}$   $\equiv$  Gaussian distributed random number with zero mean  
and standard deviation  $\sigma_i$ .

A selection operator then selects the fitter of the two vectors to become the parent of the next generation. The standard deviations  $\sigma_i$ s usually remain constant over the generations and have the same value for all the parameters in case of multi-parameter optimisation. However, Rechenberg ([Rec73]) has provided a rule-of-thumb to adapt the  $\sigma_i$ s dynamically. This was termed the 1/5 success rule which stated:

*The ratio of successful mutations to all mutations should be 1/5.  
If it is greater, then the variance  $\sigma$  is increased; if it is less,  
decrease the mutation variance.*

The derivation of this rule is given in [HB92]. It is to be noted that all the  $\sigma_i$ s are changed at the same time and not individually. Thus the (1+1)-ES had two main genetic operators - selection and mutation.

### $(\mu+1)$ - ES

As can be seen, the (1+1)-ES did not have any real notation of a population as it operated only on a single string at a time. It could be looked upon as a probabilistic gradient search technique using randomised techniques. In some respects it is like another popular search technique which is based on analogues from nature - namely the technique of *simulated annealing*. But in simulated annealing, the selection of the next point is done probabilistically, while in the (1+1)-ES it is achieved using a deterministic process.

Thus, to introduce the concept of population, the  $(\mu + 1)$ -ES was devised by Rechenberg ([Rec73]), wherein  $\mu > 1$  parents participated in the formation of a single offspring. As a result of the  $\mu$  parents, a recombination operator which imitates sexual reproduction was introduced. The recombination operator functions by selecting two strings randomly from the  $\mu$  parent strings. All the  $\mu$  strings have an equal probability

of selection. Suppose the two parent strings are represented by

$$\text{Parent A} \equiv x_1, \sigma_{x1}, x_2, \sigma_{x2}, \dots, x_n, \sigma_{xn}$$

$$\text{Parent B} \equiv y_1, \sigma_{y1}, y_2, \sigma_{y2}, \dots, y_n, \sigma_{yn}$$

Then the offspring C resulting from the recombination operation, is composed of  $\{z_1, \sigma_{z1}, z_2, \sigma_{z2}, \dots, z_n, \sigma_{zn}\}$  where  $z_n$  and  $\sigma_{zn}$  are given by

$$\begin{aligned} z_n &= \begin{cases} x_n & \text{if } X \leq 0.5 \\ y_n & \text{if } X > 0.5 \end{cases} \\ \sigma_{zn} &= \begin{cases} \sigma_{xn} & \text{if } X \leq 0.5 \\ \sigma_{yn} & \text{if } X > 0.5 \end{cases} \end{aligned} \quad (5.4)$$

where  $X$  is a uniform random value between 0 and 1. After the recombination operator, the offspring undergoes the mutation operation similar to that used in the (1+1)-ES i.e. Equation [5.3]. The selection operation is then used to remove the least fit individual - be it the offspring or one of the parents, from the  $(\mu + 1)$  individuals. Although each parameter  $x_i$  had its own standard deviation value  $\sigma_{xi}$ , these were fixed at the initialisation of the algorithm. The only change in the standard deviations values were as result of the recombination operation. Thus there was no self adaptive strategy in the  $\mu + 1$ -ESs.

#### $(\mu + \lambda)$ -ES and $(\mu, \lambda)$ -ES

The new variations of the ESs presented in this section were introduced by Schwefel ([Sch81]) for two important reasons: To make use of parallel computers and to provide a mechanism of self adaptation by adapting strategic parameters like the standard deviations during the evolution process itself. Schwefel viewed the  $\sigma$ s as a part of the genetic material which underwent the genetic operations of selection, recombination and mutation. Those individuals with better performing strategy parameters were expected to perform better than the other individuals. Thus the main difference from the ESs discussed earlier is the use of a larger number of offspring ( $\lambda > \mu$ ) and

the use of adaptive standard deviations for the mutation process. Thus, from the nomenclature it can be inferred that in  $(\mu, \lambda)$ -ES,  $\mu$  parents genetically combine to form  $\lambda$  children which are again reduced to  $\mu$  parents for the next generation. In the  $(\mu + \lambda)$  variation of the ES, both the  $\mu$  parents and  $\lambda$  children are used in the selection process to select the  $\mu$  parents of the next generation. The  $(\mu + \lambda)$  scheme can result in sub-optimal performance especially if the environment is noisy and non-stationary. The reason for this is a string with a large fitness value would tend to propagate through many generations, as in the  $(\mu + \lambda)$ -ES even the parent strings are considered for the selection process. As the  $(\mu, \lambda)$  - ES is used subsequently in this thesis for simulations in adaptive filtering, a formal description of the  $(\mu, \lambda)$  - ES is presented. A  $(\mu, \lambda)$ -ES may be mathematically described by the 8-tuple

$$(\mu, \lambda) - \text{ES} = \mathbf{P}^0, \mu, \lambda, \mathbf{f}, \mathbf{s}, \mathbf{r}, \mathbf{m}, \Delta\sigma \quad (5.5)$$

where

- $\mathbf{P}^0 \equiv$  Initial Population
- $\mu \equiv$  Number of Parents
- $\lambda \equiv$  Number of Offspring
- $\mathbf{f} \equiv$  Fitness/Objective function
- $\mathbf{s} \equiv$  The Selection Operator
- $\mathbf{r} \equiv$  The Recombination Operator
- $\mathbf{m} \equiv$  The Mutation operator
- $\Delta\sigma \equiv$  Step-size meta control

The fitness function  $\mathbf{f}$ , as before, assigns a fitness value to each string in the population. From the  $\mu$  strings which represent the parent strings,  $\lambda$  offspring are generated by using the recombination and mutation operator. The recombination operator generates a single string from two parent strings by the process explained before. Thereafter, the mutation operator operates on the new string to generate the final form of the offspring. The important difference in this scheme is that the

standard deviations  $\sigma$ s themselves undergo genetic operations and are not controlled by a meta-level rule like the 1/5 success rule. Thus, if  $x$  and  $\sigma_x$  are a parameter and the associated standard deviation, then the new values of  $x$  and  $\sigma_x$  are given by

$$\begin{aligned}\sigma_x(k+1) &= \sigma_x(k)N_{0,\Delta\sigma} \\ x(k+1) &= x(k) + N_{0,\sigma_x(k+1)}\end{aligned}\tag{5.6}$$

where  $N_{0,\Delta\sigma}$  is a Gaussian process with mean 0 and standard deviation  $\Delta\sigma$ . Thus mutation works both on the parameter value  $x$  and on the standard deviation  $\sigma_x$ . The step-size meta control  $\Delta\sigma$  has a constant value assigned to it at the beginning of the run. After the  $\lambda$  offspring strings are generated, the selection operator  $s$  selects the  $\mu$  strings having the largest fitness values which form the parents for the next generation.

The main differences between GAs and ESs arise either directly or indirectly from the representations used by the algorithm. As ESs are working with a phenotypic level, they use much more knowledge about the application domain including that of parameter boundaries. This is not the case with GAs which as a result of the coded form of the parameters are not aware of the parameter boundaries. Although the genetic operators are similar in concept in both GAs and ESs, the role they play are different. In GAs, the primary search operator is the crossover operation and serves to enlarge the search space. In ESs mutation is the main tool for exploration while in GAs the mutation operation is only used as a background operator to recover lost alleles. In the next section, advanced extensions of the ESs detailed above are presented.

### 5.3.3 Improved Evolutionary Strategies

#### Generalised Selection

The ESs detailed above used only a ranking scheme in order to select the parents of the next generation. Thus the absolute value of the fitness assigned to each string was not of importance as the fitness value was used only to rank the strings. A different

scheme of selection, which was used for GAs, was proportional selection, where for each string a probability value calculated from the fitness value assigned to it. This probability distribution was then used in the selection process. This scheme has been used for ESs and details of this scheme and some improved selection schemes for ESs are presented in [HB92].

### Improved Recombination Operators

The recombination operator as detailed in Equation [5.4] was a simple operation which chose a parameter value from either parents with equal probability. This recombination operator was referred to as the discrete recombination operator. Some modifications to this simple recombination operator were suggested by Schwefel [Sch81] and are

- **Intermediate:** In this type of recombination, the average value of the parameters from the parents was used as the parameter value of the offspring, i.e.

$$x_{new} = \frac{x_a + x_b}{2} ; x_a, x_b \equiv \text{Parent strings}$$

- **Global and Discrete:** In the global discrete recombination scheme, for each parameter value in a string, one of the two parent strings is chosen anew from the population. This results in a higher mixing of genetic material than the simple recombination operator of Equation [5.4].
- **Global and Intermediate:** This operator is similar to the intermediate recombination operator explained above except that it follows a global scheme, where for each parameter, one of the two parents is chosen anew from the population as in the global discrete case.

Using these operators, it was found that for object variables the discrete recombination operator gave best results, while for strategy parameters the intermediate scheme performed better [HB92].

### Correlated Mutations

In ESs the mutation operator is the main search operator, performing a hill-climbing operation when considered in conjunction with the selection operator. Each parameter of a string has its dedicated standard deviation, which can be looked upon as dictating the step-size for the search. However, this scheme establishes the preferred direction of search only along the axes of the coordinate system. The optimum search direction is dictated by the gradient of the search surface and need not be aligned along the coordinate axes. This can be achieved by chance only when suitable mutations are correlated. This concept was used by Schwefel [Sch81] who extended the mutation operator to handle correlated mutations. Complete details of this procedure are presented in [Sch81, HB92].

From the previous sections it can be seen that both GAs and ESs are very similar in basic concepts. The main differences arise in the genetic representation used and in the genetic operators used to generate new populations. In the next section the simulated evolution paradigm of Evolutionary Programming is explained.

## 5.4 Evolutionary Programming

### 5.4.1 Introduction

Evolutionary Programming represents one of the earliest attempts at using concepts from natural evolution for solving problems of optimisation. The initial work was done by Fogel et. al. in the late sixties [Fog62, FOW66], where simulated evolution was used to evolve artificial intelligence. Thereafter the method did not receive adequate support and in some instances was even labeled incorrect. Thus the interest in the approach did not resuscitate till the work of Holland in the early seventies in genetic algorithms. Recently there has been renewed interest in the method prompted by the work of David Fogel [Fog91b] and others. The Evolutionary Optimisation paradigm is very similar to the Evolutionary Strategies which were at the same time being investigated in Germany. It is rather fascinating to note there had been no copious exchange of information between the two schools in the United States and

Germany, with the result that a lot of effort has been duplicated. Both methods use a phenotypic representation of the parameters and rely on mutation as the primary search operator. The next section presents the salient features of the Evolutionary Programming approach.

### 5.4.2 Salient Features

The salient operations of the Evolutionary Programming paradigm are as follows:

- The initial population is generated randomly as in the case of ESs by selecting  $m$  strings, where each string  $s_i$  was composed of the  $k$  parameters of the optimisation problem. Each parameter value is selected to be a random value lying between the limits defined for that parameter.
- Each string  $s_i$  is assigned a fitness value  $\phi(s_i)$  which may be a complex function of the true fitness of  $s_i$  or the raw fitness value of  $s_i$  itself.
- Using each  $s_i, i = 1, \dots, m$ , a new string  $s_{i+m}$  is generated as follows

$$s_{i+m} = s_i + N_{0, \phi(s_i)} ; \quad (5.7)$$

where  $N_{0, \phi(s_i)}$  represents a Gaussian random variable with mean 0 and *variance*  $\phi(s_i)$ . This step represents a significant difference from the ESs where the standard deviations of the mutation process are a part of the genetic material and undergo genetic modifications during the adaptation, while from the above equation it can be seen that in the case of EP, the fitness value assigned to a parent is used as the standard deviation for generating new members. Usually the raw fitness value is not used for the standard deviation, instead a function of the raw fitness value is used.

- The new strings are then assigned a fitness value as in step two.
- For each string  $s_i$  ( $i = 1, \dots, 2m$ ), a rank  $w_i$  is assigned. The rank  $w_i$  is calculated as follows: Each string is made to compete against a fixed number of strings from the population. If the string has a fitness value less than the string

against which it is competing, then it is assigned a value of 1. The rank  $w_i$  of the string is then the total number of ones it has obtained during the competition process. Thus strings which are optimal would receive a large value for the rank. This process is explained below with help of equations:

$$\begin{aligned}
 w_i &= \sum_{l=1}^R w_i^* \\
 w_i^* &= 1 \text{ if } \phi(s_i) \leq \phi(s_r) \\
 &= 0, \text{ otherwise}
 \end{aligned} \tag{5.8}$$

where  $r$  is random integer selected lying between 1 and  $2m$ , and  $R$  is the number of competing strings.

- o Using  $w_i$ 's, the strings are ranked in the descending order of the ranks. The first  $m$  strings along with the corresponding fitness  $\phi(s_i)$  are then selected to form the next generation.

The main differences between the ESs and EP approaches are seen to be in the manner of the selection and the use of the fitness value as the standard deviation for mutation for a particular string. A important difference is the lack of any kind of crossover/recombination operator. Fogel et. al. emphasizes this point [FFA91] by stating that macromutations like the crossover and inversion operator are not required for successful adaptation. This is a radical departure from Hollands belief that the crossover operation was primarily responsible for the exploratory nature of the the genetic algorithm.

### 5.4.3 Adaptive Extensions to Evolutionary Programming

As in other simulated evolution techniques, the EP paradigm has a number of learning parameters such as the amount of mutational noise, the severity of the mutation operator etc.. The optimal values of these parameters are dependent on the particular optimisation problem, and the values obtained for a particular problem may not be suitable for another problem. Thus there is a necessity to automate the selection of values for the learning parameters. This was achieved in ESs and to some extent in

GAs, by including the strategic parameters as part of the genetic material which underwent genetic modifications. A similar approach is advocated for the case of EP by Fogel in [FFA91]. This was labeled as the meta-level evolutionary programming and consisted of attaching a perturbation variable to each parameter of the optimisation problem. This perturbation variable was then used as the standard deviation to mutate the parameter value. The perturbation values of the offspring were themselves modified by the addition of a Gaussian random variable of mean zero and standard deviation equal to the perturbation value of the parent. It can be seen that the meta-evolutionary EP technique is similar to the  $(\mu, \lambda)$ -ES with respect to the manner in which the strategic parameters are adapted.

## 5.5 Discussion

The previous sections discussed in detail the three paradigms of simulated evolution, namely genetic algorithms, evolutionary strategies and evolutionary programming. It can be seen that the basic principle of all the three methods is essentially the same and based on the principle of *survival of the fittest*. The concept of a population is of significant importance and forms the main functional unit in all three methodologies. Interest in research involving evolutionary strategies and evolutionary programming has only recently increased, though the method was first formulated in the late sixties. Genetic algorithms, on the other hand, have been an active area of research for couple of decades, though applications using genetic algorithms in engineering problems has been recent. Theoretical results regarding genetic algorithms are more mature. The evolutionary strategy and evolutionary programming methodologies are very similar to each other. Both rely on the mutation operation as the main search technique. As these techniques use the real parameter values themselves as the genetic material, the quality of the solution obtained is also more accurate and precise. All three techniques could be stated to operate using two main principles:

- The concept of a population which comprises of a set of solutions.

- A perturbation mechanism which perturbs the current set of solutions to generate new solutions.

The next chapter applies the techniques of evolutionary optimisation to the problem of adaptive IIR filtering. The simulation configuration is described along with discussion on the results obtained using the different algorithms stated in this chapter. It is shown how these techniques are not stymied by the problems of multimodal error surfaces and dimensionality associated with high order adaptive IIR filtering.

## Chapter 6

# Adaptive Digital Filtering using Genetic and Evolutionary Optimisation

### 6.1 Introduction

In this chapter, the methodologies and results obtained using the evolutionary optimisation schemes for the adaptive IIR filtering case are presented. This represents a novel approach to adaptive IIR filtering. The effect of varying parameter values and improved schemes of evolutionary algorithms are also tested using the adaptive filtering paradigm. As we have seen, the two main problems with current adaptive IIR filtering algorithms are the inability to locate the global optimum in the presence of multimodal error surfaces and the problem of dimensionality when adapting high order filters. From the simulation studies presented in this chapter, it is shown that the evolutionary optimisation schemes are able to overcome these problems.

The global optimality capability of genetic algorithms for adaptive IIR filtering were initially demonstrated by Etter in [EHC82]. Analysis regarding the global optimality of evolutionary strategies and evolutionary programming have been given in [HB92, Fog91b]. Previous work using genetic algorithms for adaptive filtering has been in the design of FIR filters [Suc91], where the genetic algorithm was used to

select from a basic set of filter templates so as to construct a new filter.

## 6.2 Simulation Configuration

To utilize the evolutionary schemes for adaptive IIR filtering, the system identification (Figure [2.3]) has been used. The unknown system in the configuration is an  $n^{th}$  order IIR filter whose coefficients are assumed to be unknown. The modeling system is also an IIR filter but whose coefficients are changed by the adaptive algorithm. Both reduced order and sufficient order modeling experiments have been carried out. The adaptive IIR filter is said to have identified the system when the estimation error  $e(n)$  reduces to zero or a minimum value. The input excitation used was white noise with unity power. The effect of measurement noise was simulated by adding white noise at varying power levels as indicated in Figure [2.3].

### 6.2.1 Genetic Algorithms

The main functional unit in evolutionary optimisation schemes as seen before is a population of string structures. For the particular case of adaptive filtering, each string structure represents a combination of the filter coefficients of the adaptive filter. Depending on the evolutionary scheme being used, the string structure is either a coded form of the parameters (genotype) or the actual parameter values themselves (phenotype). The genetic algorithms use a genotypic representation of the actual parameters. In the simulation experiments conducted in this thesis, a binary coding has been employed to obtain the genotypic representation for the genetic algorithms. Other coding schemes have used and the results obtained are presented.

The number of bits used to code a parameter determines the resolution of the parameter and could result in a situation wherein the error value does not reach the minimum value of zero as a result of the discretisation. Each coefficient of the adaptive IIR filter is coded using a binary string of  $l$  bits whereby a coefficient can take  $2^l$  distinct values. As the binary string of  $l$  bits can take values lying between 0 and  $(2^l - 1)$ , a mapping procedure is used to decode the unsigned integer linearly from  $(0, 2^l - 1)$  to a specified interval  $(P_{min}, P_{max})$ . This interval  $(P_{min}, P_{max})$  is significant

with respect to the stability of the modeling filter. The precision of the coded form is thus given by

$$\pi = \frac{P_{max} - P_{min}}{2^l - 1} \quad (6.1)$$

To use the evolutionary schemes for multiparameter optimisation, the coded parameter values or the actual parameter values themselves are all concatenated to form a larger string structure which then forms one member of the population. This is illustrated below:

#### Multiparameter Coding (10 Parameters)

$$0 - 1 \overset{P_1}{-} 0 - 0 | 0 - 0 \overset{P_2}{-} 1 - 1 | \dots \dots | 1 - 1 \overset{P_9}{-} 1 - 1 | 0 - 0 \overset{P_{10}}{-} 0 - 0$$

In the case of multiparameter optimisation, each parameter can be coded using a different number of bits, however the number of bits used to code a parameter is usually kept constant for all the parameters. The  $P_{min}$  and  $P_{max}$  values can also be different for different parameters. Thus each string structure in a population represents a particular combination of parameters of the adaptive filter.

To assign a fitness value to each string structure, the string is decoded into the constituent parameters. The error signal obtained using these parameters as coefficients of the adaptive filter is then used as the fitness measure for the string. Instead of the instantaneous error signal, a value averaged over a rectangular window is used. As all the signals used in the simulation experiments are stochastic in nature, the use of a window results in a better estimate of error for a particular set of coefficients. The length of the window used depends on the impulse response of the filter and plays an important role in the accuracy and rate of convergence of the algorithm. The raw error value itself was not used as the fitness measure - instead a modified value of the raw error was used. This modification was done in two ways:

- o Firstly, instead of the raw error value, a scaled value of the error signal was used. It has been reported that [Gol89] scaling the raw fitness values improves the performance of the genetic algorithm. In particular the power law scaling rule ([Gol89]) was used whereby the scaled error value was some specified power

of the raw error signal. In the simulation experiments a value of 4 was used as the power. It was noticed that larger values of power ( i.e.  $> 4$  ) led to premature convergence while lower values increased the iterations needed for convergence.

- o The second modification was to use an inverting function in order to convert the maximisation problem to a minimisation problem. Thus the actual fitness value  $f_i$  which was assigned to a string structure  $i$  was given by

$$f_i = \frac{1}{e_i^4} \quad (6.2)$$

where

$e \equiv$  Mean Square Output Error obtained for the string  $i$

Thus the use of genetic algorithms as the adaptive algorithm was carried out as follows: At the start of the algorithm, a population of  $\lambda$  binary strings were randomly generated, where  $\lambda$  was the population size. The length of each binary string was equal to the number of bits used to code a coefficient times the number of coefficients of the filter. Each string in the population was decoded into a set of coefficients of the filter. Using these coefficients in the adaptive filter, the error signal obtained was modified as given above and used as the fitness measure for the string. Thereafter the genetic operations of selection, crossover and mutation were carried out on the members of the population and the next generation was created. For each generation, the minimum error and the average error over all the members of the generation was recorded.

To overcome the problems of instability when adapting a high order IIR filter, alternative configurations were used. These were the cascade form, the parallel form and the lattice form. Brief details of these configurations were given in Chapter 2. For the cascade and parallel form, the subsystem which was used as a basic unit was

a second order all pole IIR filter having the transfer function

$$H(z^{-1}) = \frac{1}{1 - a_1 z^{-1} - a_2 z^{-2}} \quad (6.3)$$

The main motivation in using these forms was that the stability check could be incorporated into the adaptive algorithm by restricting the values of the coefficients to lie within the *stability triangle* as explained in Chapter 2. The decomposition could also have been made using first order sub-systems, but this would entail using complex coefficients for the filter parameters. For simulation experiments using the lattice configuration, a suitable order lattice form was selected. The coefficients of the lattice form were coded as binary strings and formed the string structure of a population. Thereafter the procedure adopted was similar to the one adopted for the parallel and cascade form. The main advantage using the lattice configuration was that the stability check was very simply incorporated in the adaptive algorithm by restricting the coefficients to have a magnitude of less than or equal to 1.

### 6.2.2 Evolutionary Strategies and Programming

The simulation configuration for the evolutionary strategies and evolutionary programming methodologies were very similar to the one adopted for genetic algorithms. The main difference was that as the evolutionary strategies and evolutionary programming used a phenotypic representation, no coding procedure was necessary to convert the actual parameter values to a genotypic representation. During the mutation process which was the main mode of search for both the algorithms, the parameter values were generated so as to always lie inside the stability triangle. For adapting high order filters, the alternative realizations used in the case of the genetic algorithms were used.

## 6.3 Simulation Results

### 6.3.1 Genetic Algorithms

In the simulation trials using the genetic algorithm, each coefficient was coded using a binary string of 14 bits. This resulted in each coefficient being discretised into 16384 discrete values between the limits imposed by the stability criterion. The effect of varying the number of bits are presented later on. In all the results which are presented, the minimum error obtained for each generation is shown plotted against the number of generations. Adaptation was stopped after 200 hundred generations. A window length of 100 was used to obtain the average instantaneous error. This is shown in the results on the x-axis as a multiplication factor of 100 indicating the actual number of time samples of the input signal which were needed for convergence. It was assumed that all the members of a population were evaluated in parallel, though the actual simulation experiments proceeded down the set of string structures which made up a population. All the simulation runs show the average results obtained after twenty simulation runs of the experiment.

For adapting high order IIR filters, alternative realizations such as the parallel, cascade and lattice forms were used. Of these configurations, the parallel form gave the best results. The cascade form was tested out in the early simulation experiments but resulted in a very large time of convergence. This was found to be caused by the cascade structure itself as the numerical and quantisation error propagated and multiplied through the structure. The main reason for the success of the parallel form was the fact that because of the decomposition of the direct form realization into a parallel form, *multiple global minimas* were created all of which were equivalent to each other. This was the result of the different ways the poles could be rearranged in the second order sub-systems. However, this resulted in the error surface for such a configuration to have a different characteristics ([NJ89]). It was shown in [NJ89] that if a direct form IIR filter was modeled using an alternative realization, the resulting error surface may have additional optimas, which may be equivalent global minimas or saddle points which are unstable solutions in parameter space. As the genetic algorithm is a stochastic technique, there is enough jitter provided in the algorithm

itself, which drives the algorithm away from the regions containing the unstable saddle locations. However a gradient algorithm could get stuck at such a point without reaching the global optimum if there is no noise present. From the simulation results it can be seen that genetic algorithms were able to locate the global optimum.

To demonstrate the genetic algorithm performs better than a pure random search algorithm, simulations experiments were carried out in which a population of string structures were selected randomly at each iteration. There was no genetic operations performed on the population. The minimum error of the population was recorded and the results obtained are presented in Figure [6.1] and [6.2] which were obtained using different order filters. It can be seen that the genetic algorithm *learns* and performs better than a pure random search algorithm. Convergence to the optimal set of coefficients was confirmed by checking the final set of coefficients which the algorithm determined.

### Reduced Order Modeling

This simulation experiment was devised to check whether the genetic algorithm approach was able to locate the global optimum when the error surface was multimodal. The experiment involved identifying a second order IIR filter using a first order model. This reduced order modeling resulted in a bimodal error surface and was first used in [LTJ80]. The unknown system was modeled using the second order model given by

$$H(z^{-1}) = \frac{0.01 - 0.4z^{-1}}{1.0 - 1.314z^{-1} + 0.25z^{-2}} \quad (6.4)$$

while the modeling was done by a first order IIR filter with the transfer function

$$H(z^{-1}) = \frac{b}{1 - az^{-1}} \quad (6.5)$$

Using the above model, it has been analytically proved that the two minimas have error values 0.3 (global minima) and 0.9 (local minima) ([JL77]). The result using this model and the genetic algorithms as the adaptive strategy is shown in Figure [6.3], where the genetic algorithm approach is compared to the Stochastic Learning

Automata (SLA) approach. From the error values obtained by the end of the simulation, it can be seen that the algorithm was able to locate the global minimum. The same result also shows that in comparison with the automata approach, genetic algorithms provide a faster rate of convergence. The model used in this experiment is different from that used by Etter in [EHC82], where a different example was used to demonstrate the property of global optimisation.

### High Order Filters

To check the capability of the genetic algorithm to adapt high order IIR filters, filters up to the order of ten were modeled in the simulation experiments. The transfer function of the different order filters are as given:

#### Fourth Order Model

$$H(z^{-1}) = \frac{2.0 - 2.8z^{-1} + 1.5z^{-2}}{1.0 - 2.8z^{-1} + 3.42z^{-2} - 2.04z^{-3} + 0.54z^{-4}} \quad (6.6)$$

#### Sixth Order Model

$$H(z^{-1}) = \frac{3.0 - 4.5822z^{-1} + 2.956z^{-2} - 0.58436z^{-3} + 0.168012z^{-4}}{1 - (2.2911z^{-1} - 1.729314z^{-2} - 0.364717z^{-3} + 1.281337z^{-4} - 0.73702899z^{-5} + 0.12988048z^{-6})} \quad (6.7)$$

#### Tenth Order Model

$$H(z^{-1}) = \frac{5.0 - 7.624z^{-1} + 8.577z^{-2} - 7.7029z^{-3} + 8.7961z^{-4} - 6.193z^{-5} + 5.484z^{-6} - 3.8431z^{-7} + 2.0182z^{-8}}{1 - (1.906z^{-1} - 1.52z^{-2} + 0.8279z^{-3} - 1.9478z^{-4} + 2.541z^{-5} - 1.5255z^{-6} + 0.52511z^{-7} - 0.79528z^{-8} + 0.77202z^{-9} - 0.31692z^{-10})} \quad (6.8)$$

The results for the different order filters are given in Figure [6.4]. Although the tenth order model is seen to take a larger number of generations to converge, the algorithm located the optimal set of coefficients for all the different order filters. For all the different order filters, the initial convergence with reference to the number of iterations is very rapid. This is a property of genetic algorithms in that they rapidly find regions of near optimal solutions. The high order filters were modeled using the parallel form configuration using the appropriate number of second order sub-systems. Thus the tenth order IIR filter was modeled using a parallel bank of five second order

sub-systems.

### Variation of Parameter Values

The main parameters of a genetic algorithm are the population size, the crossover probability and the probability of mutation. The effect of these parameters are shown in Figures [6.5,6.6,6.7].

Figure [6.5] shows results when the mutation probability is varied. As can be seen, very large and very small values of mutation probability results in non-optimal performance. Large values of the mutation probability ( $p_m = 0.2$ ) reduces the genetic algorithm to a random search routine with no learning process and thus the algorithm is unable to converge to the optimal solution. With very small values of mutation probability ( $p_m = 0.001$ ), the algorithm does not have sufficient exploratory power and thus converges prematurely to sub-optimal solutions. Both these effects are demonstrated in Figure [6.5].

The effect of the crossover probability is shown in Figure [6.6]. The results show that the crossover probability does not play as important a role as the probability of mutation. With larger values of crossover probability, the initial rate of convergence is faster, though the number of iterations need to locate the global set of coefficients remain unaltered. This result has been documented by different researchers who have stated that the crossover operation is not necessary for an extensive search in evolutionary algorithms. This is in contradiction to Hollands original hypothesis who envisaged the crossover operator as the main operator in genetic algorithms responsible for the exploratory search, while mutation was used only as a secondary operator to recover lost alleles.

Results showing the effect of the population size are given in Figure [6.7]. With small population sizes, the selective pressures on the population members are not sufficient enough, thus the algorithm is unable to locate the optimal set of coefficients. With an increased population size, the algorithm locates the optimal set of parameters, though this is achieved at an increased computational time.

### Effect of Coding Schemes

Holland has proposed the use of binary coding to obtain the genotypic representation of the actual parameters. It had been proved that the number of schemata which are processed in parallel attains a maximum value when the cardinality of the alphabet being used for the coding process is minimum [Hol75]. Thus binary coding should result in the optimum performance. However, this concept has also been questioned recently by researchers. In particular, Davis in [Dav91] lists a number of practical applications of genetic algorithms, none of which use the binary coding scheme. The success of the evolutionary algorithms, the results of which are presented later on, show that perhaps the use of a genotypic coding itself is redundant. Two other coding schemes were used and the results are presented in Figure [6.8]. The use of the gray coding enabled the algorithm to locate the optimal state with greater accuracy, as adjacent coefficients using a gray coding only differed by a single bit value. This enabled the algorithm to locate the optimal set of coefficients from near optimal solutions without a large number of bit changes. The variance of the error is also seen to have reduced using a gray coding instead of the binary coding. Both these codings however used the binary alphabet. The other coding employed was real coding - in actuality no coding was really used, instead the actual parameter values were themselves used as the genetic material. This is similar to the evolutionary algorithms except that the mutation operation is handled differently. Using the real coding, mutation was handled by using a uniform distribution centered around the current operating point. If the new point was outside the limits used for the stability criterion, the mutation operation was carried out again. It can be seen from the results in Figure [6.8] that the gray coding gave a better performance. The main reason for the poor performance of the real coding was that a uniform distribution was used in the mutation process to generate new strings. This could result in excessive mutation noise, resulting in the algorithm not being able to locate the optimal coefficients rapidly.

### Effect of the Bit Length

Figure [6.9] presents the results obtained using different number of bits to code a parameter value. Though there is no significant increase in the rate of convergence, using a larger number of bits enabled the algorithm to obtain a more accurate result. As was stated previously, the use of a coding scheme to obtain the genotypic representation forces the parameters to take discrete values. The number of bits used for the coding determines the resolution of the parameters. This can be seen in Figure [6.9] where using the four bit coding, the algorithm converges to a higher value of error even when it has located the optimal set of coefficients.

### Different Crossover Schemes

There have been a number of crossover schemes cited in the literature devised to improve on the original single point crossover scheme used by Holland. Some of these schemes were used for the adaptive IIR filtering simulation experiments and the results obtained are presented in Figures [6.10] and [6.11]. The four crossover schemes which were implemented were *One Point Crossover*, *Two Point Crossover*, *Uniform Crossover* and *Multiple Crossover*. The One Point Crossover operation was the standard single point operation proposed by Holland. In the Two Point operation, two crossover points were used, while the Multiple Point Crossover operation used a separate crossover point for *each parameter*. Each crossover point was constrained to lie between the limits defined for that parameter. Thus in the multiple crossover operation, the number of crossover points was equal to the number of parameters. The Uniform Crossover operation has been explained in Section [5.2.3].

The two sets results in Figures [6.10] and [6.11] are generated for two differing values of the mutation probability. Figure [6.10] shows the result for a mutation probability of 0.075. In this case the multiple point and single point crossover schemes give better results, while the uniform crossover schemes results in non-optimal solutions. The reason for this is the fact when coupled with the relatively large value of mutation, the uniform crossover scheme results in extensive disruption of the schematas. Thus the propagation of schemata with above average performance is reduced as they get

broken up. On the other hand when using a lower value of mutation ( $p_m = 0.025$ ), the uniform crossover scheme results in a reduced value of error as is shown in Figure [6.11]. However, with a low value of mutation, the algorithm was not able to locate the optimal set of coefficients. This again gives credence to the theory that mutation is an important operation and perhaps should be used as a primary operator in simulated evolutionary algorithms.

### Different Selection Schemes

The proportional selection (stochastic sampling with replacement technique) used for the genetic algorithm sometimes led to premature convergence. Thus new schemes of selection which have been mentioned in Chapter 5 were used in the simulation experiments. The results using these different schemes are presented in Figures [6.12, 6.13, 6.14]. The two selection schemes other than proportional selection which were used were the *ranking scheme* and *remainder stochastic sampling with replacements*. The *remainder stochastic sampling with replacements* has been labeled in Figure [6.12] as the *Deterministic Scheme*.

In the remainder stochastic sampling with replacement scheme, the probability of contribution for each string is calculated as in the proportional selection scheme. Then the expected number of individuals for each string was calculated as the product of the probability value for that string and the size of the population, rounded off to the nearest integer. If the total number of individuals thus created was less than the population size  $\lambda$ , the fractional parts of the expected number values were then used in a roulette wheel selection procedure to fill the remaining slots in the population. In the ranking scheme, out of a population size of  $\lambda$  members, the  $M$  best were selected to form the members of the next generation. The value of  $\lambda$  was fixed at 50 (population size), while the value of  $M$  was varied between 6 and 25 as shown in the results.

The comparative results between the three different selection schemes are presented in Figure [6.12]. Of the three, the remainder selection scheme is seen to give the better performance. The proportional selection scheme is prone to two sources of error ([Gol89]) - firstly only an estimate of the schema average is obtained using sequential finite sampling; secondly the selection scheme itself is a high variance process

with a large degree of scatter. This is to some extent reduced using the remainder stochastic sampling with replacement.

In the ranking scheme, no importance is given to the actual fitness value - the fitness value is used just in order to rank the strings. But interesting results are observed when the number of strings used to generate the next population are varied. The idea to change the number of parents has been adopted from the evolutionary strategies and has not been used before in genetic algorithms. The results are shown in Figures [6.13] and [6.14]. It can be seen that as the number of strings used to generate the offspring strings reduce, the performance of the algorithm improves. Figure [6.14] shows the result obtained using the ranking scheme but with an elitist strategy. In such a scheme, the best string structure of each generation is always carried over to be a member of the next generation. Using the elitist scheme along with the ranking selection procedure, the algorithm is able to locate optimal set of coefficients with a greater degree of accuracy (Figure [6.14]) as can be determined from the final error values which are obtained. From these results, it can be gathered that the proportional selection scheme can result in inaccurate convergence states and improved selection schemes are necessary to overcome this problem.

### Effect of Measurement Noise

The performance of the genetic algorithm with the presence of measurement noise is presented in Figure [6.15]. The measurement noise was added as shown in Figure [2.3]. Thus at convergence, the error value should reduce to the added noise level. From Figure [6.15], it can be seen that for low values of signal to noise power ratio (input signal power is unity), the algorithm is able to locate the optimal set of coefficients, though at very low signal to noise power ratio (noise power = 100), the noise dominates and the algorithm is unable to locate the correct set of coefficients. At large values of signal to noise power ratio (noise power = 0.01), the noise introduced by the discretisation of the coefficients prevents the algorithm from reaching the noise floor, even though it has located the optimal coefficients.

### Adaptive Extensions to Genetic Algorithms

As was detailed in Chapter 5, the real power of genetic algorithms is obtained when the optimum values of the strategic parameters are learnt online during the adaptation process. This would make the genetic optimisation scheme a completely general and robust scheme, the parameters of which would learn to adapt by themselves depending on the problem being optimised. Some initial work was carried out using the adaptive filtering paradigm. The values for mutation and crossover probability were coded as a binary string of 14 bits and included as part of the genetic material. This entailed using the additional two sets of 14 bits being attached to the binary coded forms of the parameters. One set of 14 bits decoded to the mutation probability, while the other set of 14 bits decoded to the crossover probability. Thus, when optimising a set of six parameters, the length of each string in the population was now 112 bits long. The compound string constructed as given above, underwent genetic operations in the usual manner. After the parameters had been decoded from the binary strings, the value of mutation and crossover probability are calculated. As these values are now different for each string, the following procedure was adopted. After the selection process, two parent strings were chosen to undergo the genetic operations of mutation and crossover as in the standard genetic algorithm. The value of mutation and crossover probability was obtained for each string by decoding the set of bits which represented these values. Then the average of the two values obtained for each string was used as the value for both the strings.

The result of using such a scheme are presented in Figure [6.16]. It can be seen that the scheme was able to locate the optimum set of coefficients at the same rate as standard genetic algorithms. It was noticed from the simulation results, that the mutation rate was driven towards a low value as the algorithm proceeded. This had the effect of driving all members of a population to converge to a single string structure. This result also can be observed in Figure [6.16] where the average error in a generation is seen to reduce and approach the minimum error of the generation. The advantage of the scheme was that the only parameter to be user controlled in this scheme was the population size. More research in this area of adaptive genetic

algorithms certainly seems to be justified.

### Discussion

Results obtained using genetic algorithms as the adaptive strategy for adaptive IIR filtering has been presented in the above sections. The method was able to overcome the twin problems of multimodal error surfaces and dimensionality when adapting high order IIR filters. Improved schemes which have been tested result in a better performance as compared to the standard genetic algorithm. The main observation from the above simulation results is that the mutation operator is of significant importance and is mainly responsible for the explorative abilities of the algorithm. Another important observation has been the fact that with a large value of mutation, the crossover operation has much reduced significance as shown in the results regarding the different crossover schemes. In the ranking schemes it has been shown that if the number of parent strings are sufficiently small (ratio between the number of parents and offspring is large), the selective pressures are increased leading to improved results.

#### 6.3.2 Evolutionary Strategies

The evolutionary strategy used a phenotypic representation of the parameters - thus the actual parameter values themselves were used to create the genetic representation which formed the members of a population. As stated before, the main search operation in evolutionary strategies was the mutation operation. The mutation operation was performed by adding a Gaussian distributed random variable centered around the current operating point and with variance determined by the adaptive process itself. Thus in the  $(\mu, \lambda)$ -ESs, there were three parameters which were varied. These were the number of parents  $\mu$ , the number of offspring  $\lambda$  and the initial variance of mutation process. As a result of incorporating the standard deviations of each parameter into the genetic material, the evolutionary strategy is capable of learning the optimal values of the standard deviation online. This is accomplished by adapting the standard deviation values themselves by use of a Gaussian process as has been

explained in Chapter 5. It is the value of the standard deviation of this Gaussian process which is varied in the simulation experiments. The crossover operation which was used for all the simulation experiments was the discrete recombination operation which has been explained in Chapter 5.

### Variation of the Standard Deviation

The results obtained by varying the standard deviation as explained above, are given in Figure [6.17]. It can be seen that with very small values of the standard deviation ( $\sigma = 0.001$ ), the algorithm is unable to locate the optimal coefficients in a reasonable number of iterations. However with large values of the standard deviation ( $\sigma = 0.1$ ), even though the initial rate of convergence is rapid, the algorithm gets locked into a non-optimal state. Thus it can be inferred that the initial value of the standard deviation plays an important role in the accuracy and the rate of convergence of the algorithm. For all the simulation experiments using the evolutionary strategies presented henceforth, a value of 0.01 was used for the standard deviation.

### Variation of the $\mu$ and $\lambda$

The effect of using different number of parents and children in the  $(\mu, \lambda)$ -ES are shown in Figure [6.18]. The important result is when the number of parents equal the number of the children as shown for the case of ( $\mu = \lambda = 50$ ). In this case the minimum error in a generation *increases* at first. The reason for this is the lack of any selective pressures in the adaptive process resulting in the search process degenerating into a random search algorithm. As the ratio between the number of offspring and parents increases, the algorithm results in a better performance. The optimal value for this ratio arrived at by Hoffmeister and Bäck in [HB92] was six. This can be seen from the results presented in Figure[6.18]. Further simulation experiments conducted in this thesis using the evolutionary strategy, used six parents which generated forty offspring strings.

## Discussion

As the evolutionary strategies use a phenotypic representation, they are operating with the real parameter values and thus do not suffer from the discretisation problems of the genetic algorithms. However for the same reason, the hardware implementation of evolutionary strategies has to be performed in a different manner from that proposed for genetic algorithms. Use of the binary coding for the genetic algorithms meant that the method could perhaps be implemented using standard digital logic circuits. This is not possible with the evolutionary strategies, however these methods could be processed on vector computers as most of the operations are performed in parallel and using real arithmetic. The value of the standard deviation used in the Gaussian process responsible for the mutation process was found to have significant effect on the algorithm, with large values making the algorithm behave in random fashion and too small values resulting in premature convergence. The ratio between the number of parents and children was also an important criterion for optimal convergence.

### 6.3.3 Evolutionary Programming

As seen from the descriptions presented in Chapter 5, both the evolutionary strategy and evolutionary programming methodologies are very similar. Both use a phenotypic representation and rely on mutation as the significant operation responsible for the search process. However, the role of crossover is largely insignificant and in the case of evolutionary programming is not employed at all. The main differences between the two schemes are the manner of the selection operation and the way in which the strategic parameters are varied during the adaptive process. In the evolutionary strategies, the varying of strategic parameters is accomplished by using a Gaussian process which perturbs the current value of the standard deviation of the mutation process. In evolutionary programming, the error value obtained for each string structure is itself used as the variance for that string structure. This explains one reason why the crossover/recombination operation has not been used in the evolutionary programming methodologies.

Two sets of simulation results are presented for the evolutionary programming

paradigm - the first results show the effect of varying the number of parents while the second set of results show the effect of varying the number of strings taking part in the competition against one another to assign ranks to each string structure.

### Variation of the Number of Parents

The evolutionary paradigm of simulated evolution functions by selecting  $\lambda$  parents which then produce  $\lambda$  offspring using the mutation process. The selection process then selects the  $\lambda$  best strings from this population of  $2\lambda$  strings to form the next generation. The effect of varying the value of  $\lambda$  is shown in Figure [6.19]. It can be seen from the final error values obtained at the end of the simulation run, that with small population sizes the selective pressures are not strong enough to drive the algorithm to locate the optimal set of coefficients.

### Variation of the Number of Competitions

The selection process in the evolutionary programming paradigm assigns a rank to each of the strings formed as the intermediate population. The ranks are assigned as follows: Each string in  $2\lambda$  strings of the intermediate population is made to compete against a certain number of strings of the population. Based on the competition, the string is assigned a rank. Details of how the strings compete against each other are given in Chapter 5. In this simulation experiment, the number of competitions for a particular string is varied and the results are shown in Figure [6.20] and [6.21]. Though the effect of the changing the number of competitions is negligible, small values of competition result in higher values of error as can be seen in Figure [6.21] which shows the same result shown in Figure [6.20] but at an higher resolution. However too large a value for the number of competitions does not result in a better performance - on the other hand increases the computational time.

### Discussion

Evolutionary programming and the evolutionary strategies are very similar with respect to the performance of the algorithms for the adaptive filtering problem. The number of iterations needed to locate the optimal set of coefficients are also roughly

the same. The main differences are in the manner in which the basic genetic operations are carried out. Intuitively, the use of the error value obtained for a string as the variation of the mutation process as is the case in the evolutionary programming paradigm looks promising. When the search process has located the optimal string, the error value for that string decreases to a very low value. Thus, use of the error value as the variance ensures that further disruption of that string does not occur. On the other hand, the standard deviation is included as part of the genetic material in evolutionary strategies. Since each parameter of the adaptive process has its own standard deviation value, the length of the string structure is now doubled. However, this results in better control of the strategic parameters, as each parameter is modified based on the standard deviation value assigned to it. This also allows the possible inclusion of the crossover operation as part of the algorithm.

#### 6.3.4 Applications using the Adaptive IIR Filter

As was explained in Chapter 2, two important applications which use adaptive filtering are adaptive noise canceling and adaptive equalization. These two applications were simulated on the computer and the evolutionary strategy was used as an adaptive algorithm. The main reason for using this strategy was the fact that the evolutionary strategy used real parameters as the genetic material and thus was able to locate the exact set of optimal coefficients. The genetic algorithm on the other hand would entail discretisation of the parameters and the subsequent loss of accuracy. It is however envisaged that genetic algorithms and evolutionary programming methods would also result in final results similar to that obtained using the evolutionary strategy.

##### Adaptive Noise Cancelling

The simulation configuration for the adaptive noise canceling experiment was as given in Figure [2.9]. The details of the procedure was explained in Chapter 2. From the lumped model of the adaptive noise canceling setup shown in Figure [2.9], when the

transfer function of the noise canceller is given by

$$H(z^{-1}) = \frac{G(p)}{G(r)} \quad (6.9)$$

the model has reached its optimum value and the signal estimate  $\hat{s}(k)$  would then be an exact estimate of the original signal  $s(k)$ . In the simulation experiment conducted, the filter transmission path  $G(p)$  was modeled using a sixth order IIR filter while  $G(r)$  was equal to unity (see Figure [2.9]). The modeling filter, which was the noise canceler, was modeled using an adaptive IIR filter using a parallel configuration of second order filters. The noise process was simulated using white noise with unity power. Three different signals were used to simulate the signal  $s(k)$  undergoing the distortion. These were a sum of sinusoids, a square wave and a pseudo-random binary sequence (prbs). The result of the noise filtering experiment using the sum of sinusoids is given in Figure [6.22]. As can be seen, the adaptive algorithm was able to remove the effect of the distortion and restore the noisy signal to its original state. This can also be observed when using the square wave signal (Figure [6.23]) and the prbs signal (Figure [6.24, 6.25]). When the signal to noise power is low, the adaptive algorithm is not able to remove the distortion completely (Figure [6.24], however when the noise power is reduced, the restoration is more complete (Figure [6.25]).

Figures [6.26 - 6.29] show snapshots of the evolution of the cleaning process taken at different generations using the sum of sinusoids as the test signal. At the beginning the algorithm is still searching for the optimal set of coefficients and thus the output signal is still noisy. By the time 100 generations have evolved, the algorithm has succeeded in locating the optimal coefficients with the distortion being greatly reduced as shown in Figure [6.29].

### Adaptive Equalization

For the adaptive equalisation simulation, reference is made to Figure [??]. As the output error configuration was used, a desired response was required to adapt the equalizer. Usually in adaptive equalization, the desired response is not available as the receiver is some distance away from the transmitter. This is overcome using the

following method - initially a known sequence of bits are transmitted. Since this sequence is available at the receiver end, the equalizer can be adapted using this sequence as the desired response  $d(n)$ . Thereafter a scheme, which was originally devised by Lucky [Luc66], who proposed the use of the equalizer output itself as the desired response after passing it thorough a limiter, is used. Thus the desired  $d(n)$  response, generated by this scheme was given by

$$d(n) = \begin{cases} +1 & \text{if } \hat{y}(n) \leq 0 \\ -1 & \text{if } \hat{y}(n) > 0 \end{cases} \quad (6.10)$$

The channel distortion was modeled using an sixth order FIR filter. The additive noise is simulated by adding uniform white noise with zero mean and varying power levels. The equalizer is modeled using an sixth order IIR realized as a parallel bank of three second order filters. The desired response was obtained in the following manner - for the first 50 generations of the adaptive process, the actual bit sequence represented by  $x(n)$  was used to adapt the equalizer. Thereafter the scheme devised by Lucky and explained before was used. In other words after 50 generations, the quantified output of the equalizer  $\hat{y}(n)$  itself was used as the desired response  $d(n)$  (Equation [6.10]).

The results of the equalisation experiment are shown in Figures [6.30 - 6.33]. Figures [6.30,6.31] show the final result after the equalizer has been adapted, for different time sequences of the same input signal with no additive noise. The equalizer is able to reconstruct the original signal with a unit delay. This delay is because the FIR filter modeling the transmission channel is of non-minimum phase. Figures [6.32,6.33] show the result of the same experiment when the FIR filter output is corrupted using additive noise. The noise signal used for the distortion had a power of 0.01. The equalizer is able to reconstruct the bit sequence even with presence of additive measurement noise.

## 6.4 Conclusions

This chapter presented the results obtained using the evolutionary optimisation algorithms for the adaptive IIR filtering problem. An important aspect of these results is that they have also provided an example of using the evolutionary optimisation schemes for a practical problem, rather than optimising artificially created functions.

The evolutionary optimisation approach was able to tackle the main problem of multimodal performance surfaces, prevalent with adaptive IIR filters, using alternative realizations. Of the alternative realizations used, the parallel form gave the best results. Although, the cascade form resulted in convergence to the optimal coefficients, the number of time samples for convergence was very large. The main reason for this was the propagation of errors through the cascade structure. The lattice configuration was also used in the early simulation experiments. However, for each direct form realization, there exists a unique set of lattice coefficients. Thus, to locate these optimal coefficients took a large number of iterations. The success of the parallel form can be attributed mainly to the creation of *multiple global optima* whenever a direct form structure is decomposed into a parallel realization. The adaptive algorithm, was thus able to converge to one of these multiple global optima rapidly.

The study has also revealed the shortcomings of the genetic algorithms and has also confirmed the recent conjecture by researchers, that the important search operation in genetic and other evolutionary schemes is the mutation operation. Thus two important concepts in evolutionary optimisation schemes which have been confirmed by the simulations are presented. The first is that of a parallel set of solutions as realised by a population, and secondly new solutions are generated from the current solutions by perturbing the current solutions. This seems to be the core of all evolutionary optimisation schemes. However, a limitation of all the evolutionary schemes is the lack of any stopping criterion whereby further iterations of the algorithm may be avoided. It was thought combining concepts from simulated annealing along with evolutionary schemes would result in such a criterion. This idea is fully explored in the next chapter which presents results obtained using annealing and new hybrid algorithms.

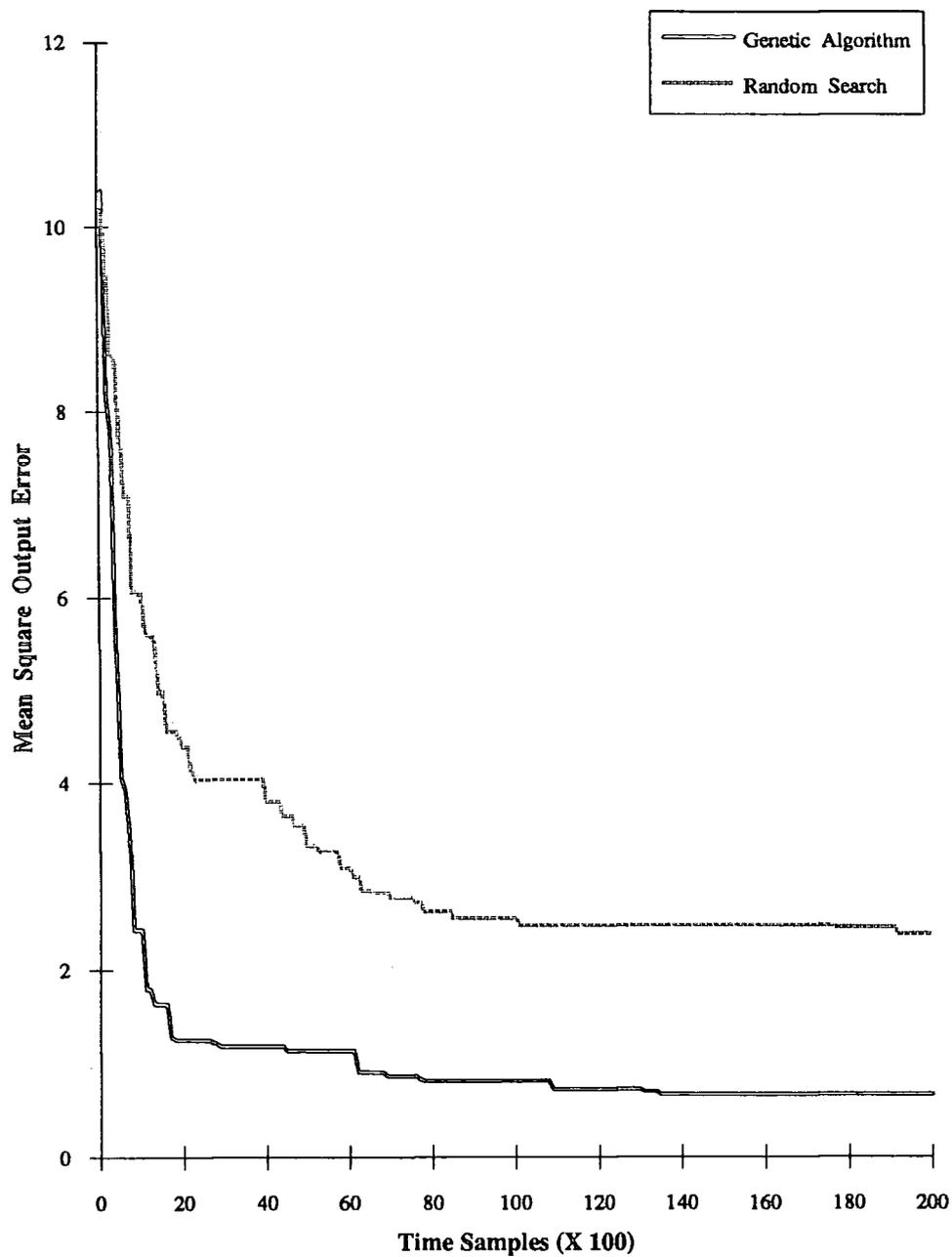


Figure 6.1: Comparison between Genetic and Random Search Algorithms

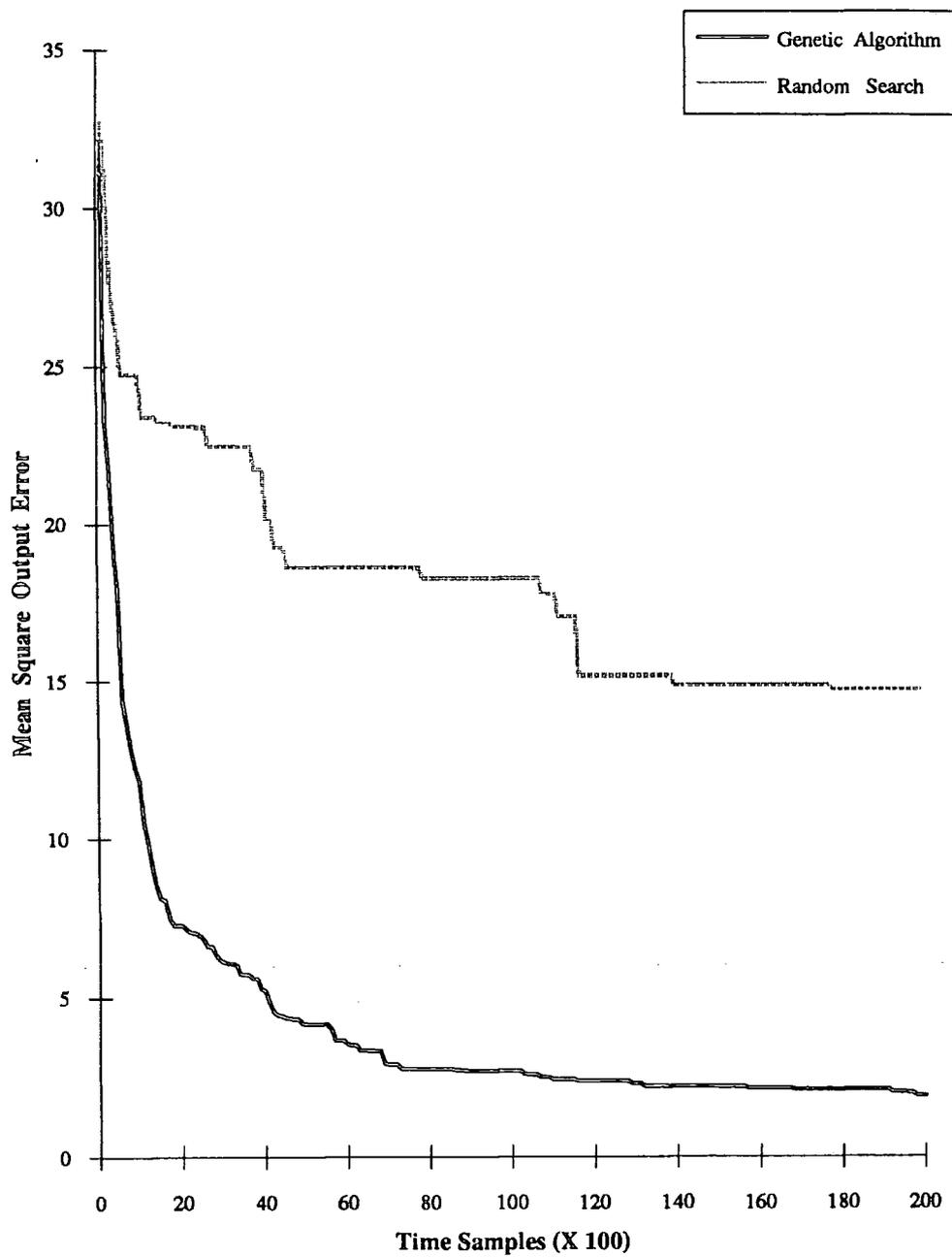


Figure 6.2: Comparison between Genetic and Random Search Algorithms

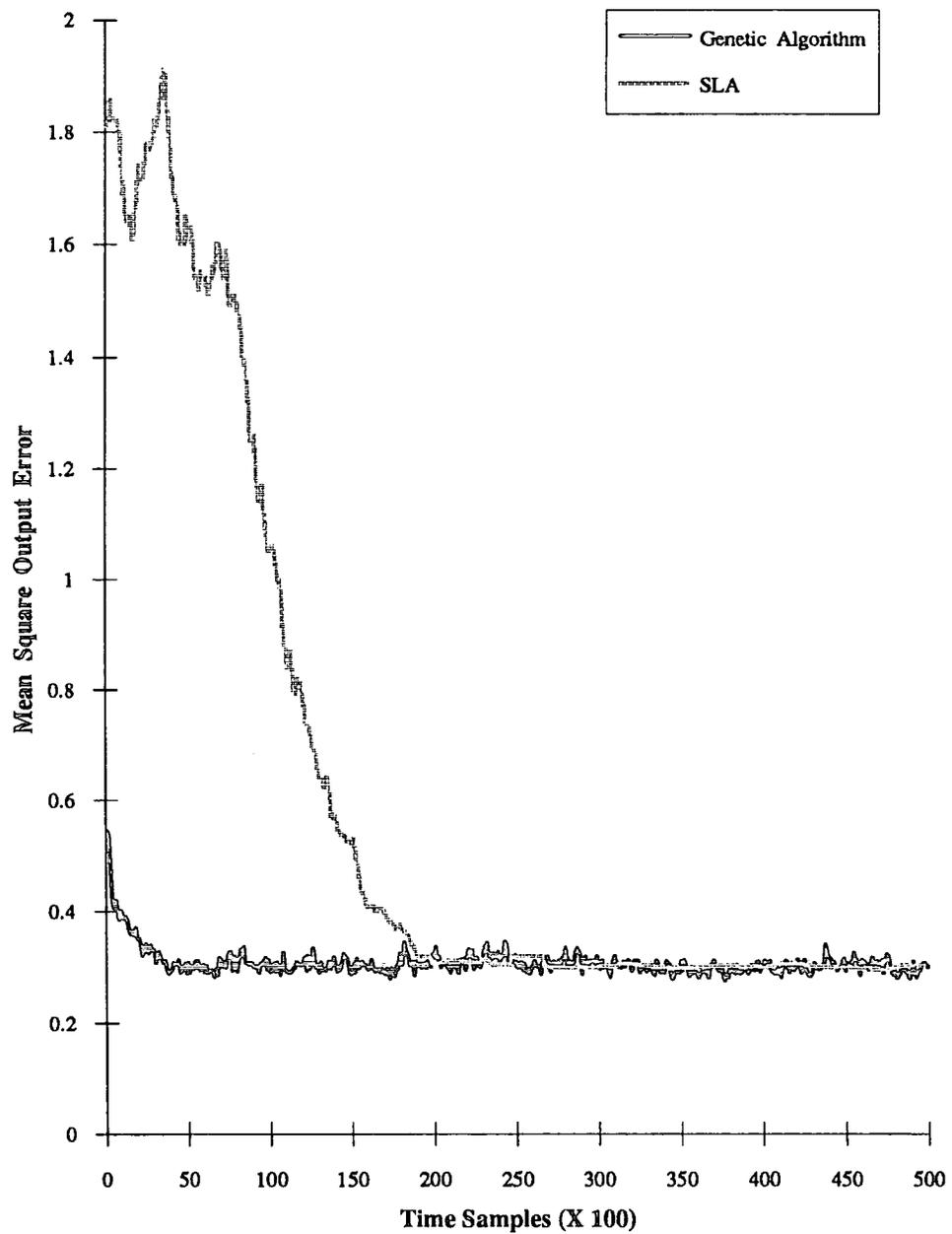


Figure 6.3: Comparison between Genetic and Stochastic Learning Automata Algorithms

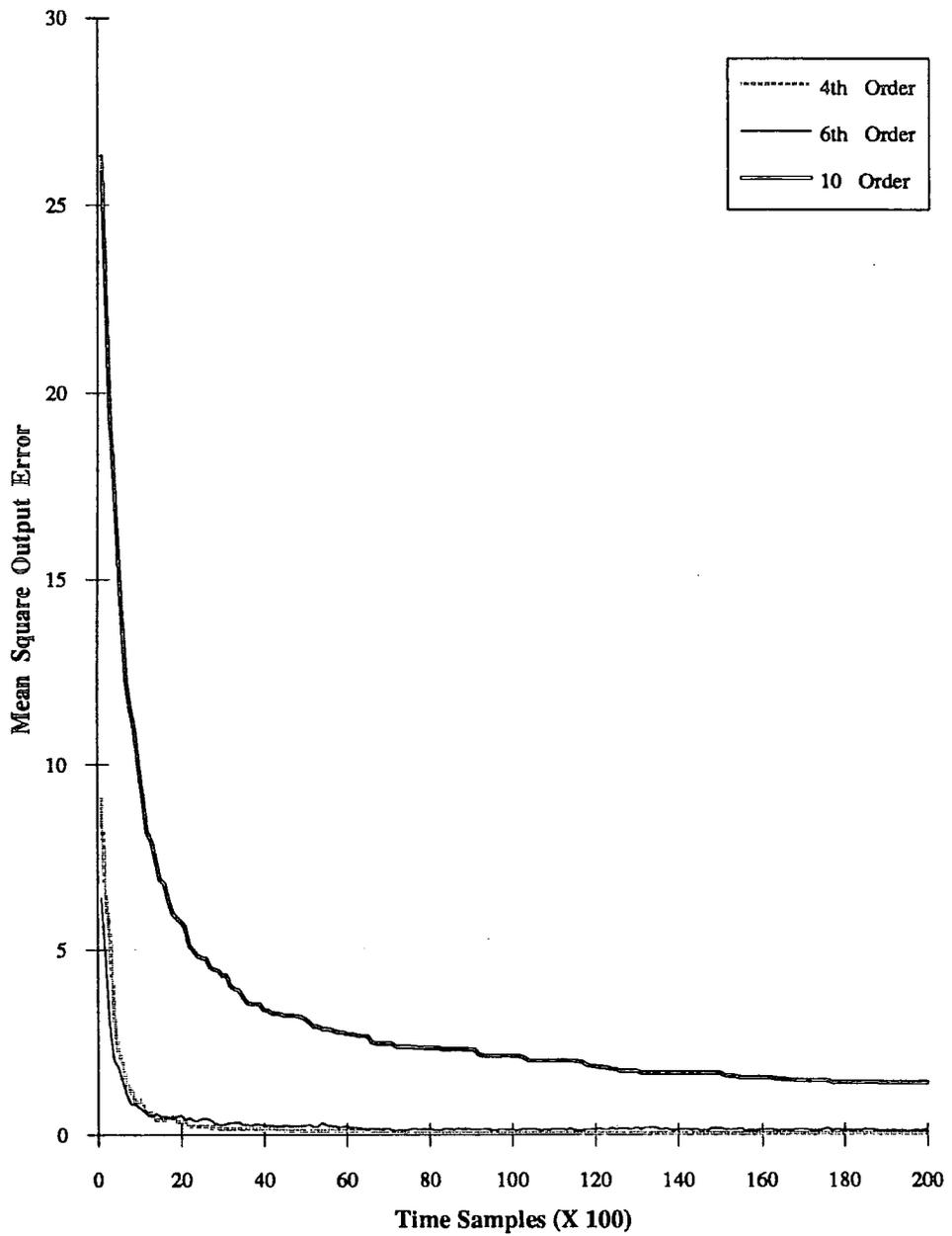


Figure 6.4: Different Order Filters

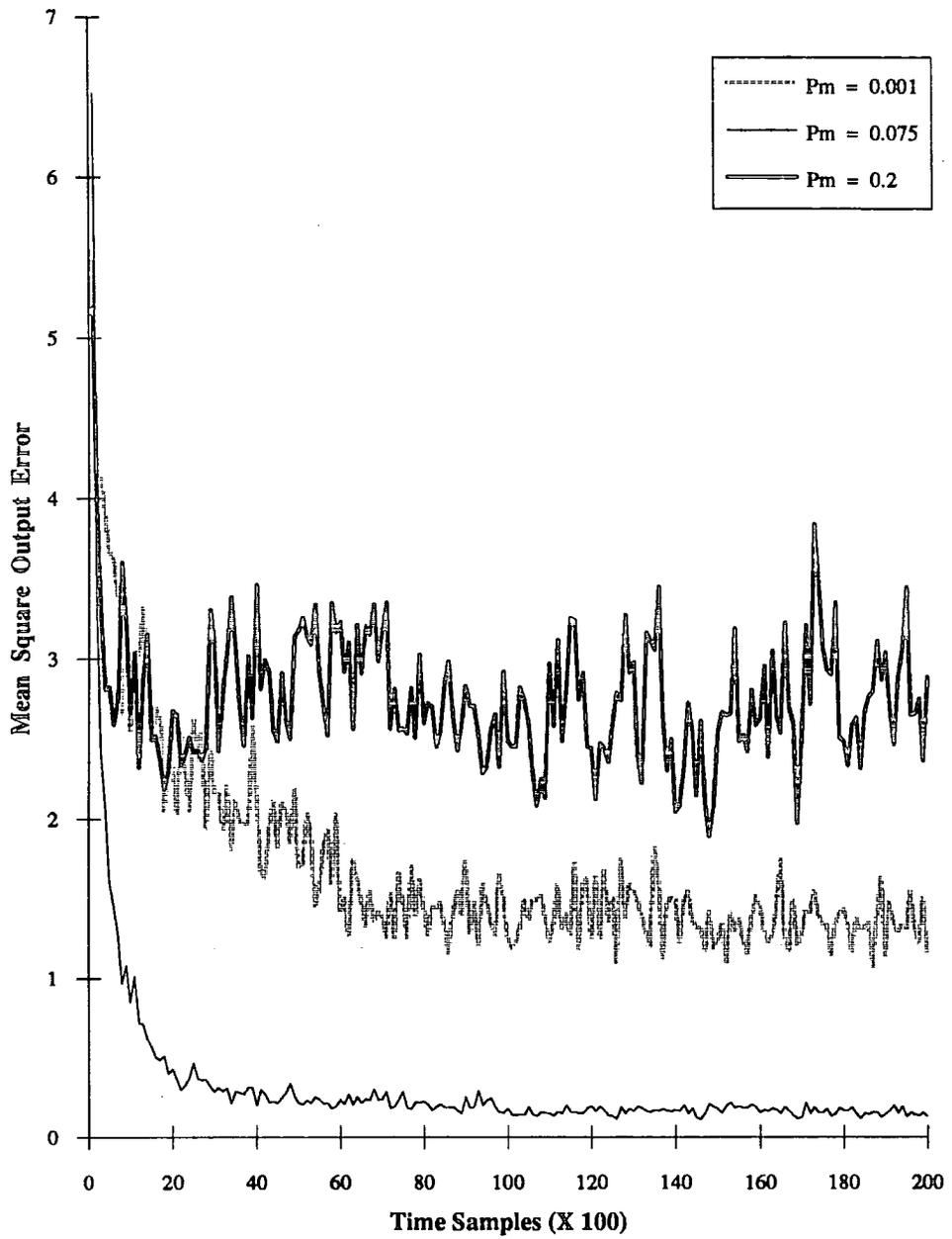


Figure 6.5: Effect of Mutation

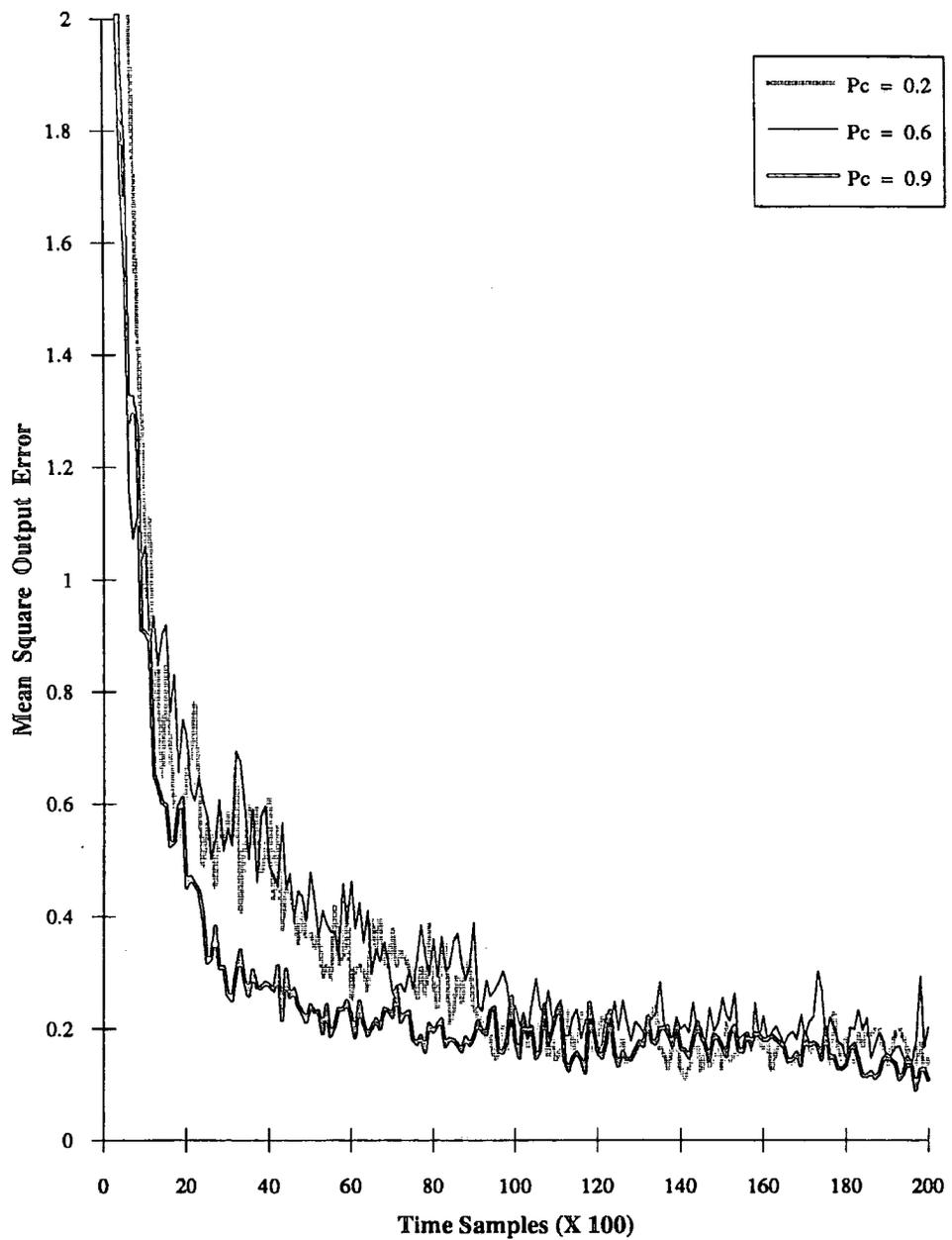


Figure 6.6: Effect of Crossover

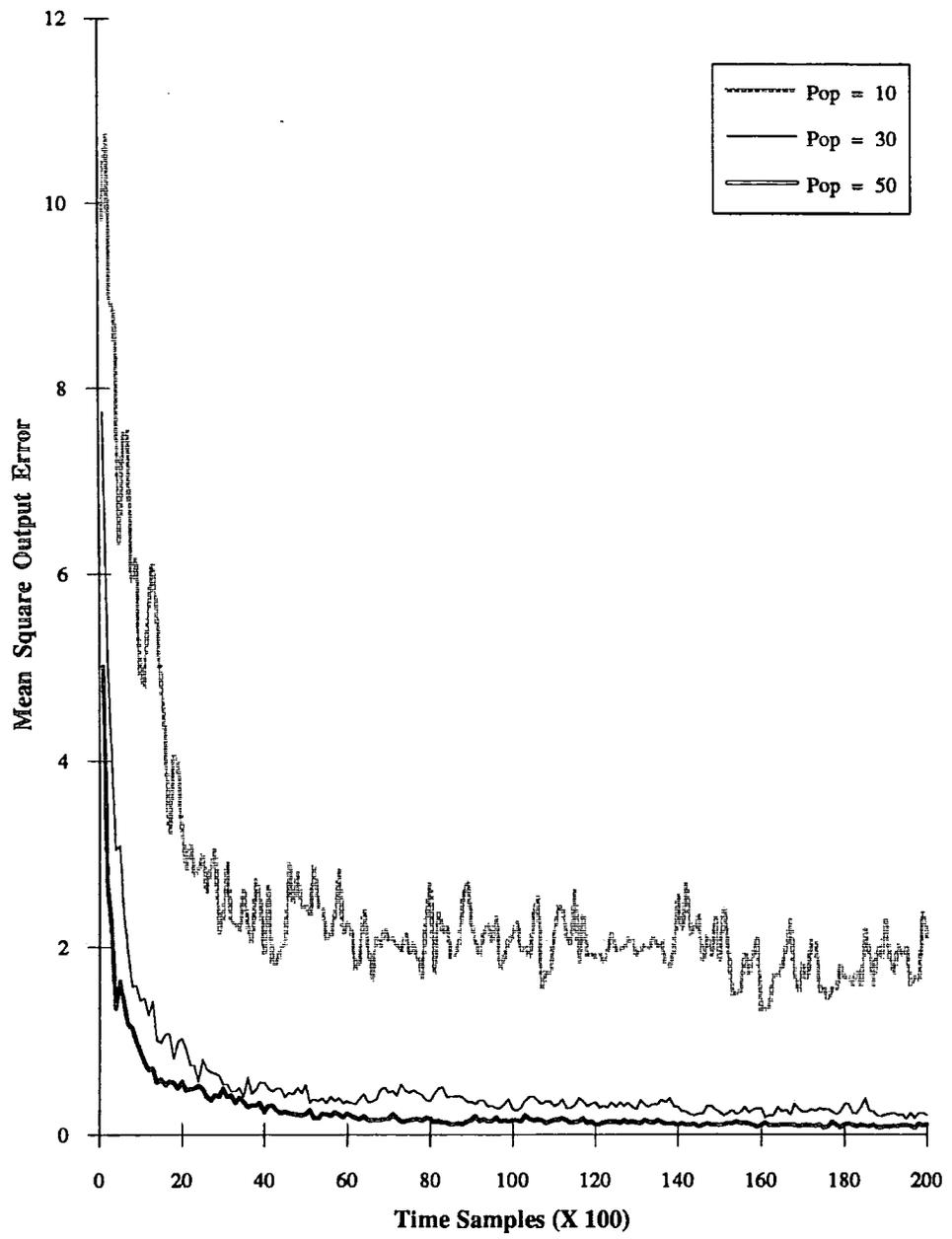


Figure 6.7: Effect of Population Size

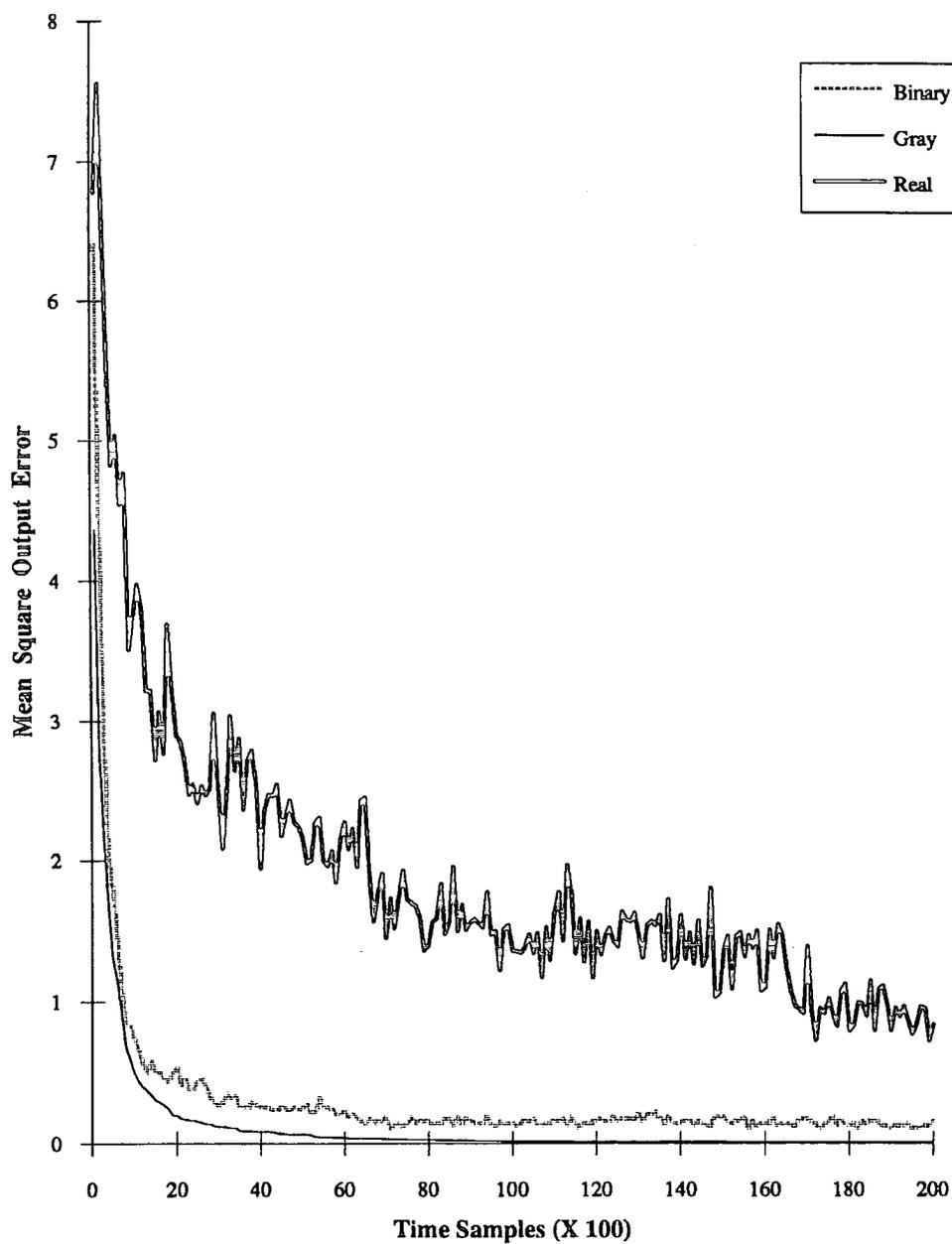


Figure 6.8: Effect of Coding Schemes

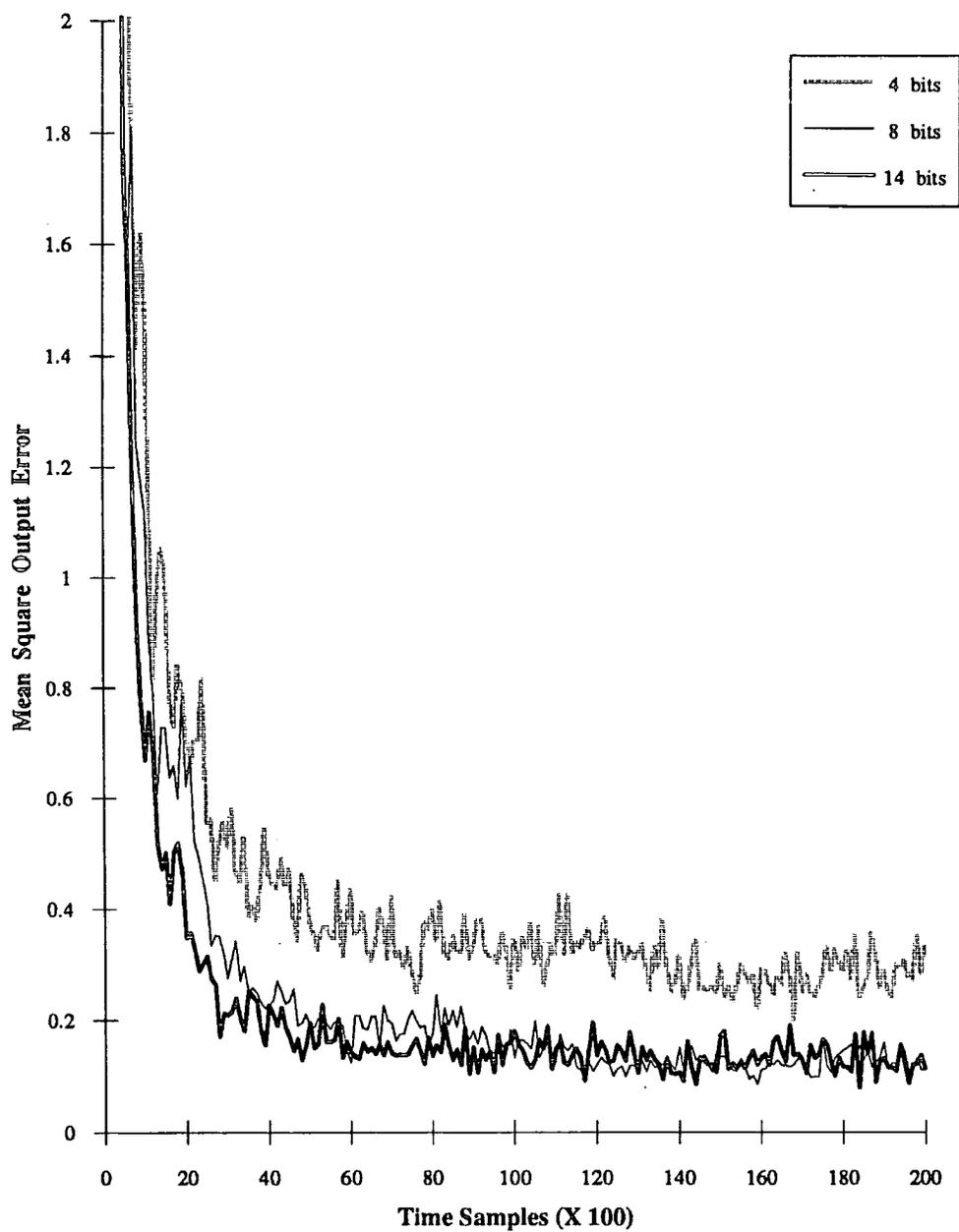
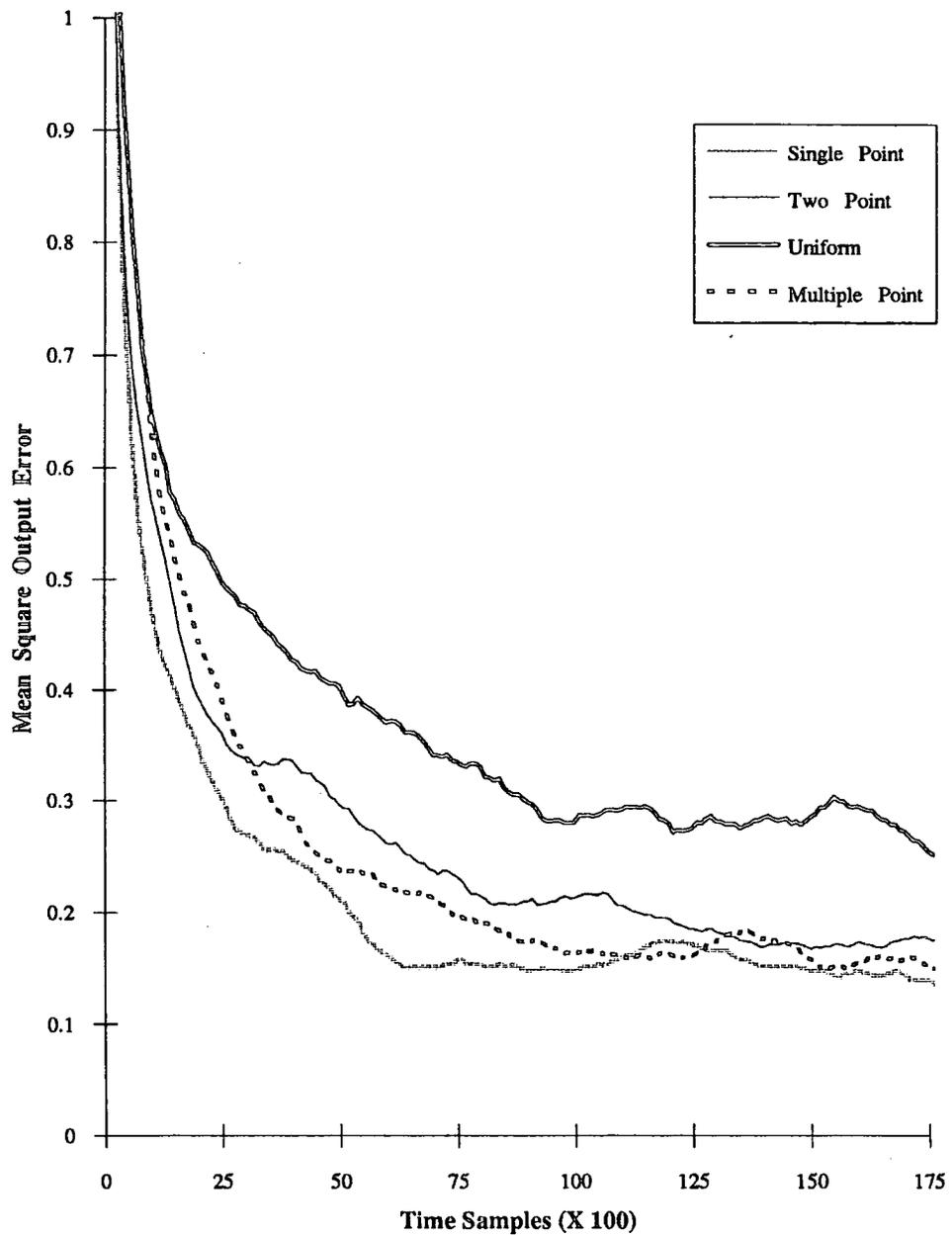


Figure 6.9: Effect of the Number of Bits

Figure 6.10: Effect of New Crossover Schemes ( $p_m = 0.075$ )

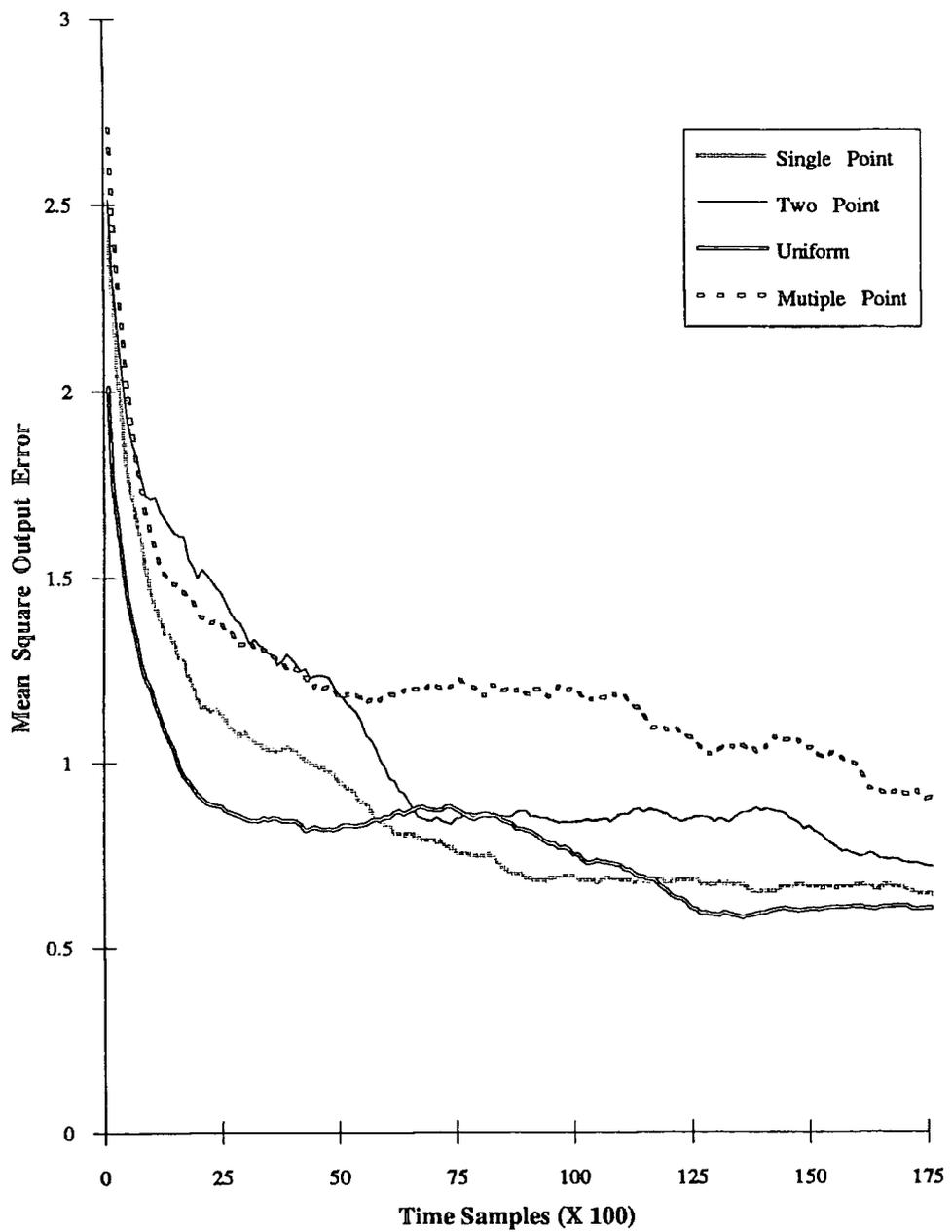


Figure 6.11: Effect of New Crossover Schemes ( $p_m = 0.025$ )

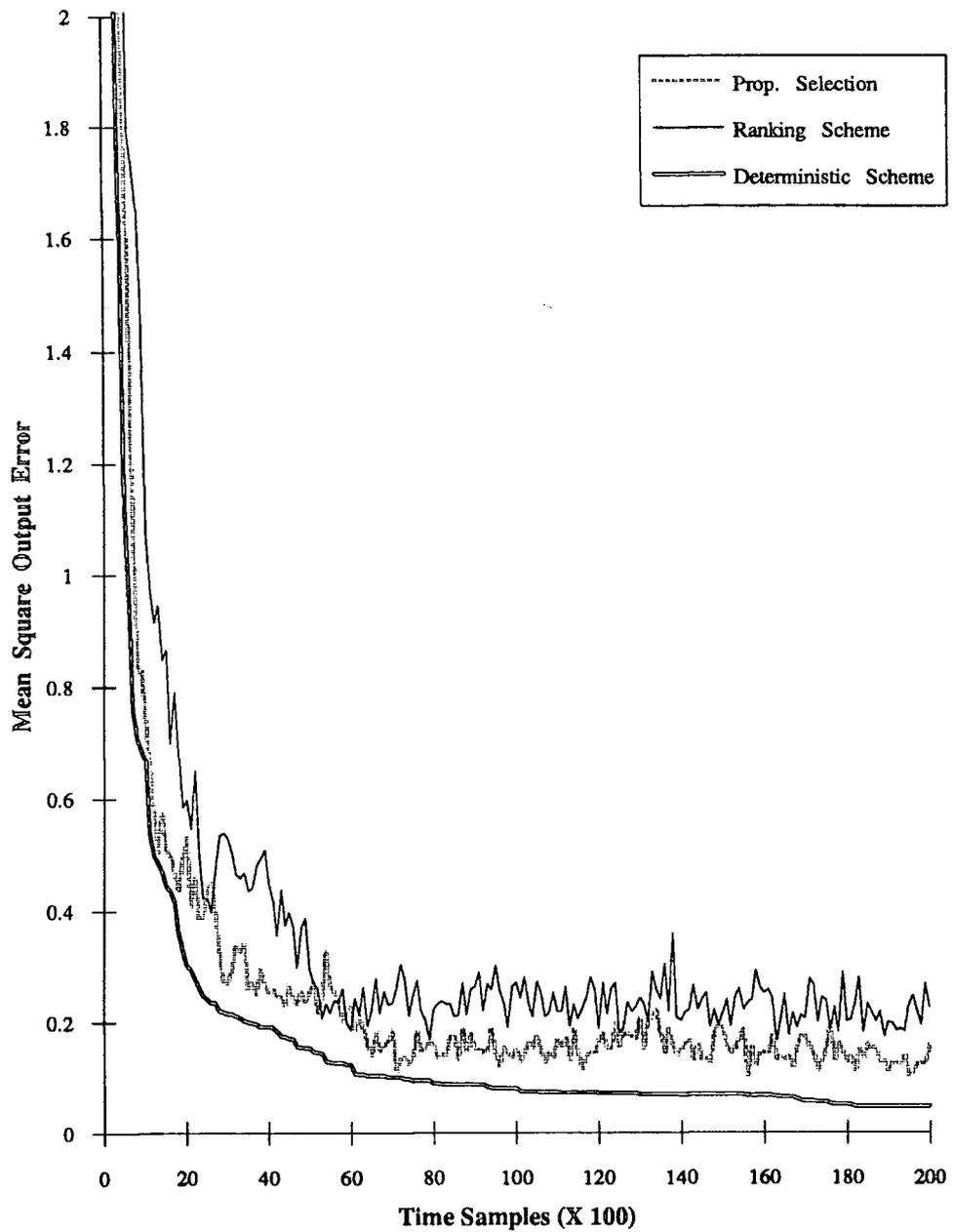


Figure 6.12: Effect of Improved Selection Operations

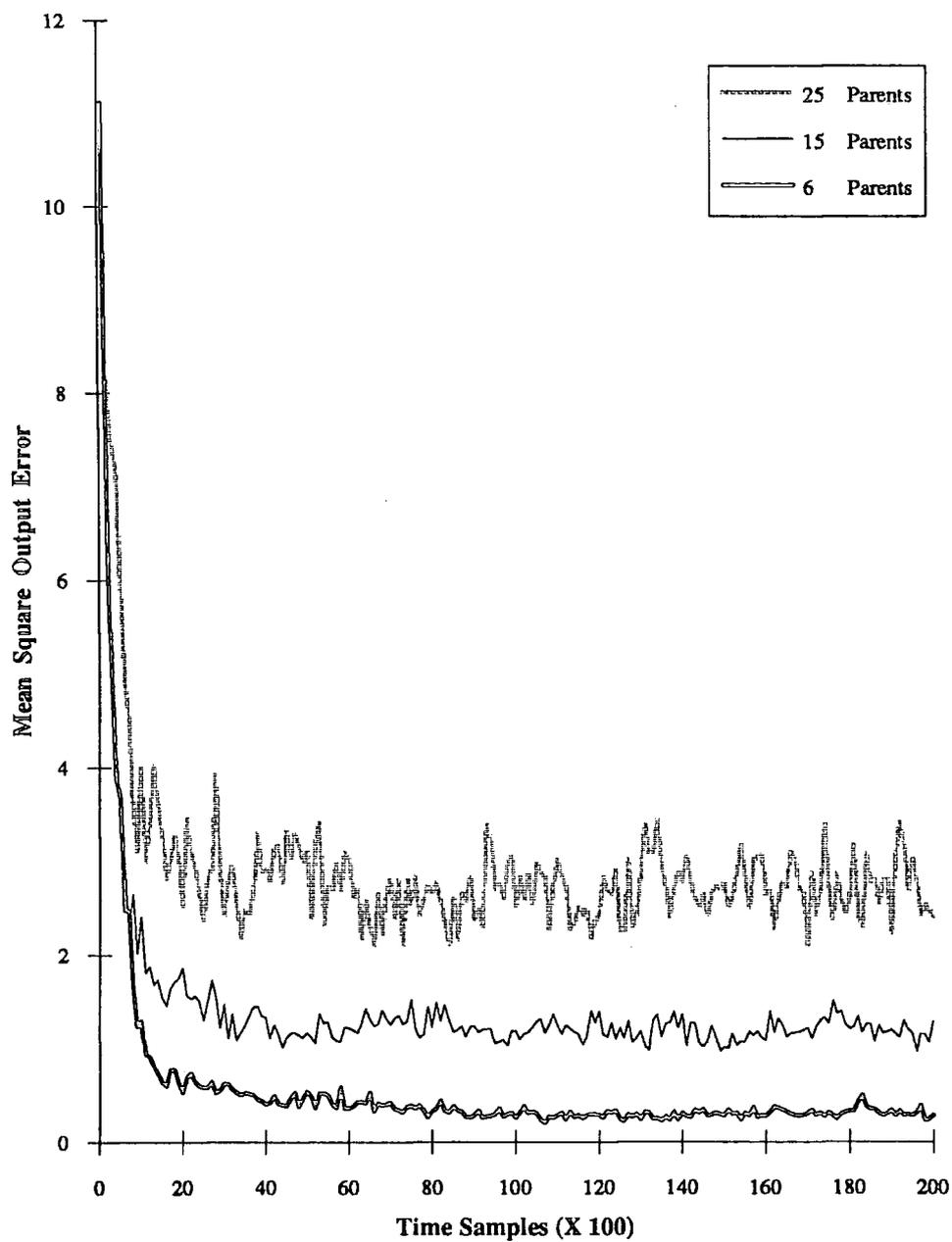


Figure 6.13: Effect of the Ranking Selection Scheme

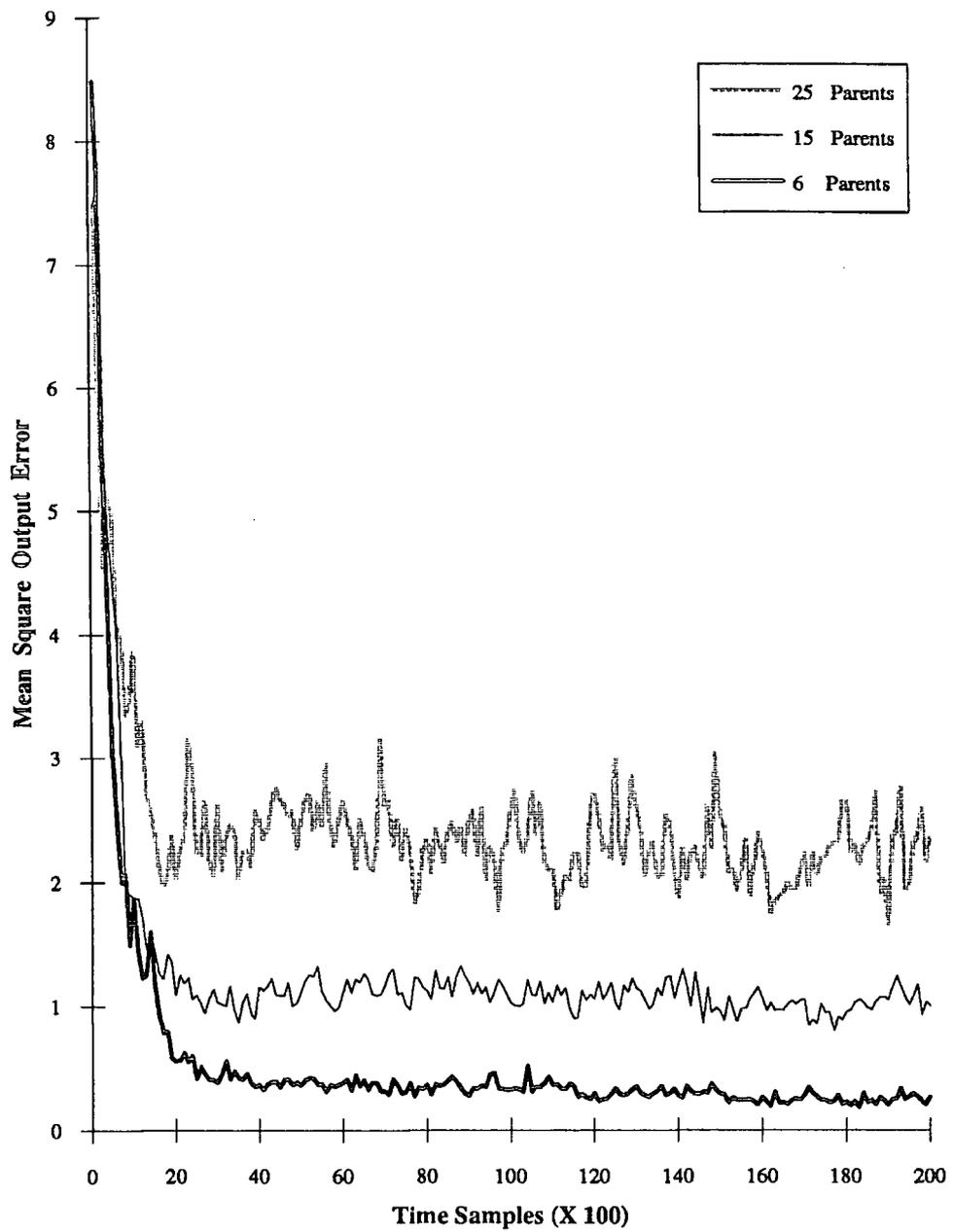


Figure 6.14: Effect of the Ranking Elitist Selection Scheme

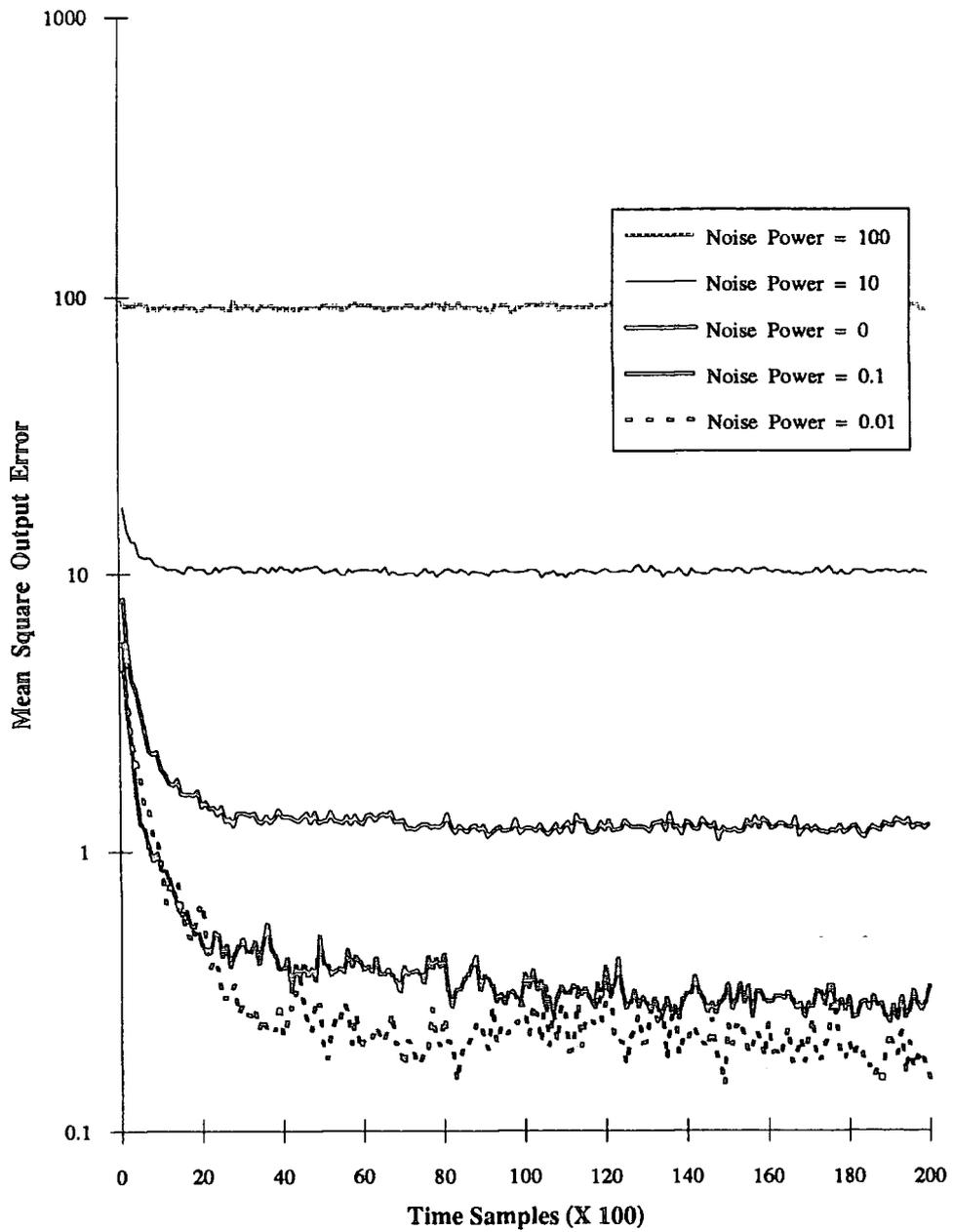


Figure 6.15: Effect of Measurement Noise

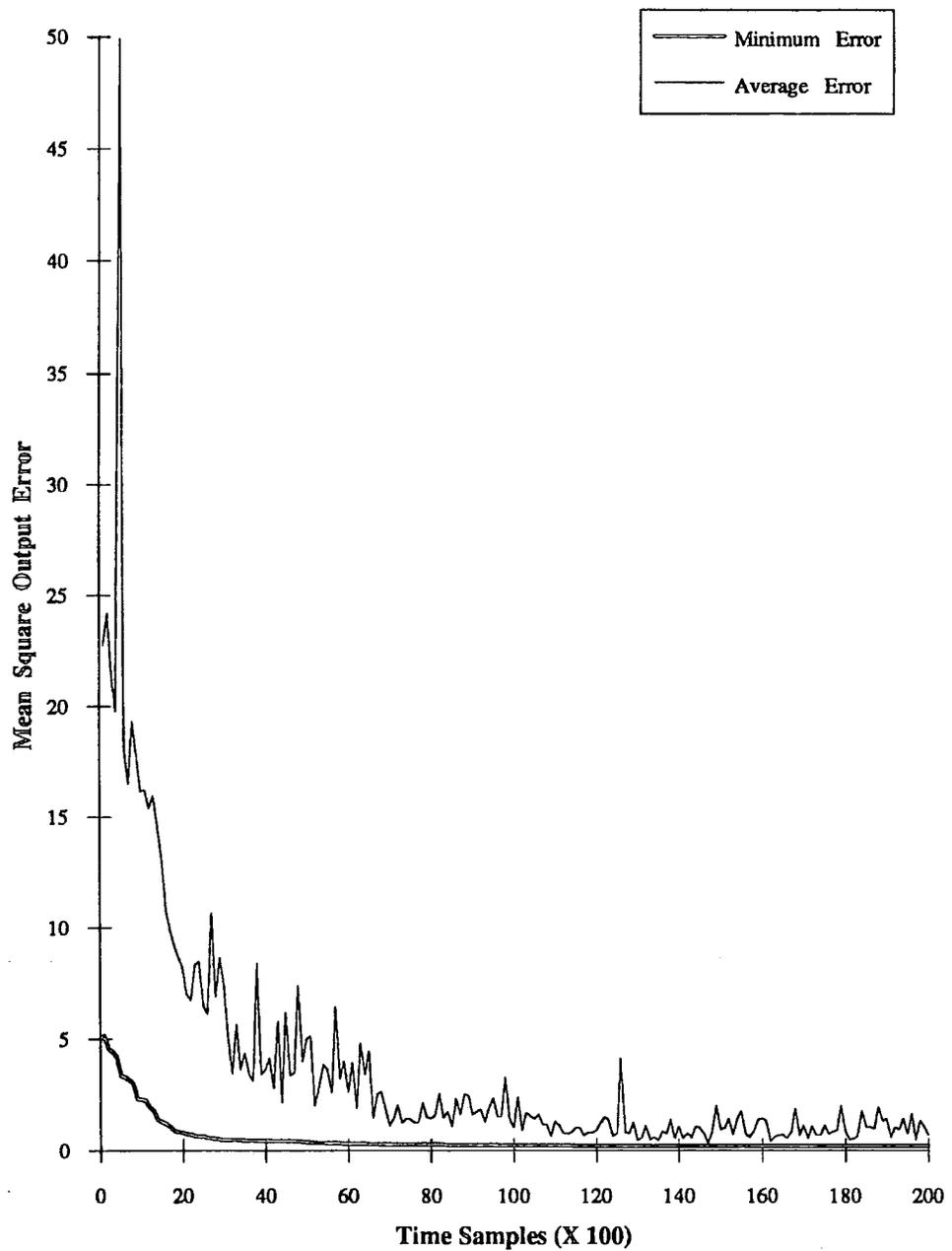


Figure 6.16: Results using Self Adaptive Genetic Algorithm

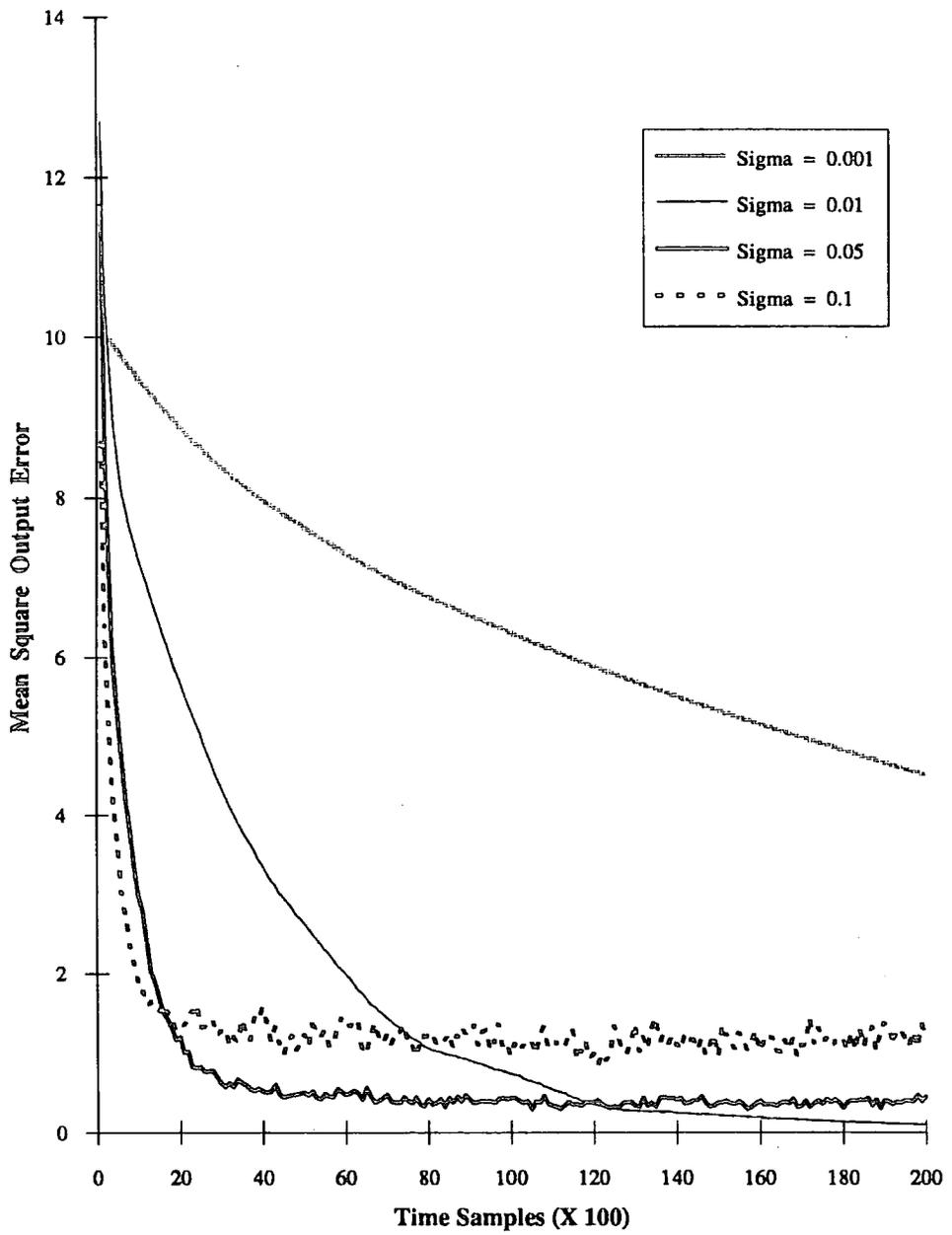


Figure 6.17: Effect of Standard Deviation in ESs

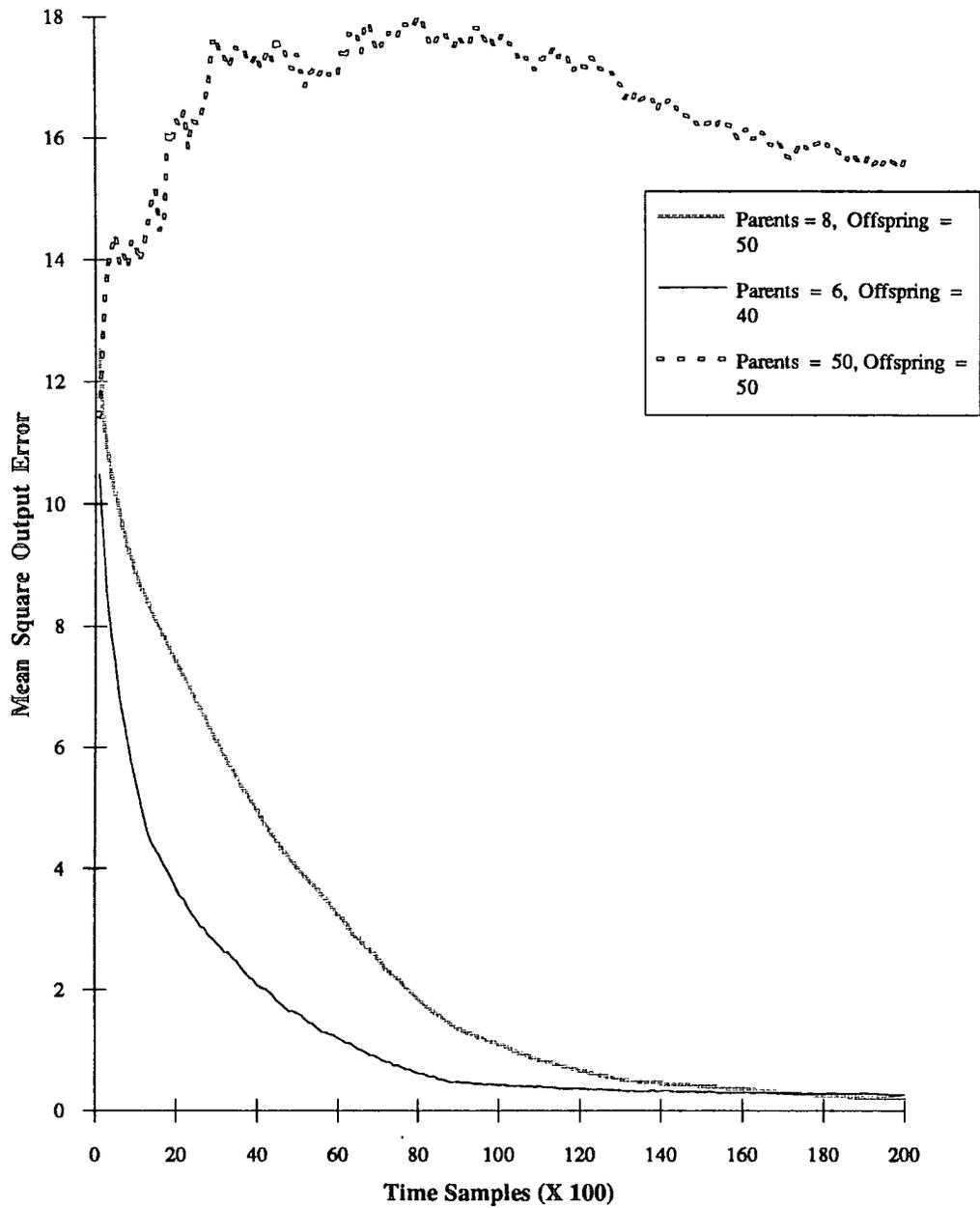


Figure 6.18: Effect of the Number of Parents/Offspring

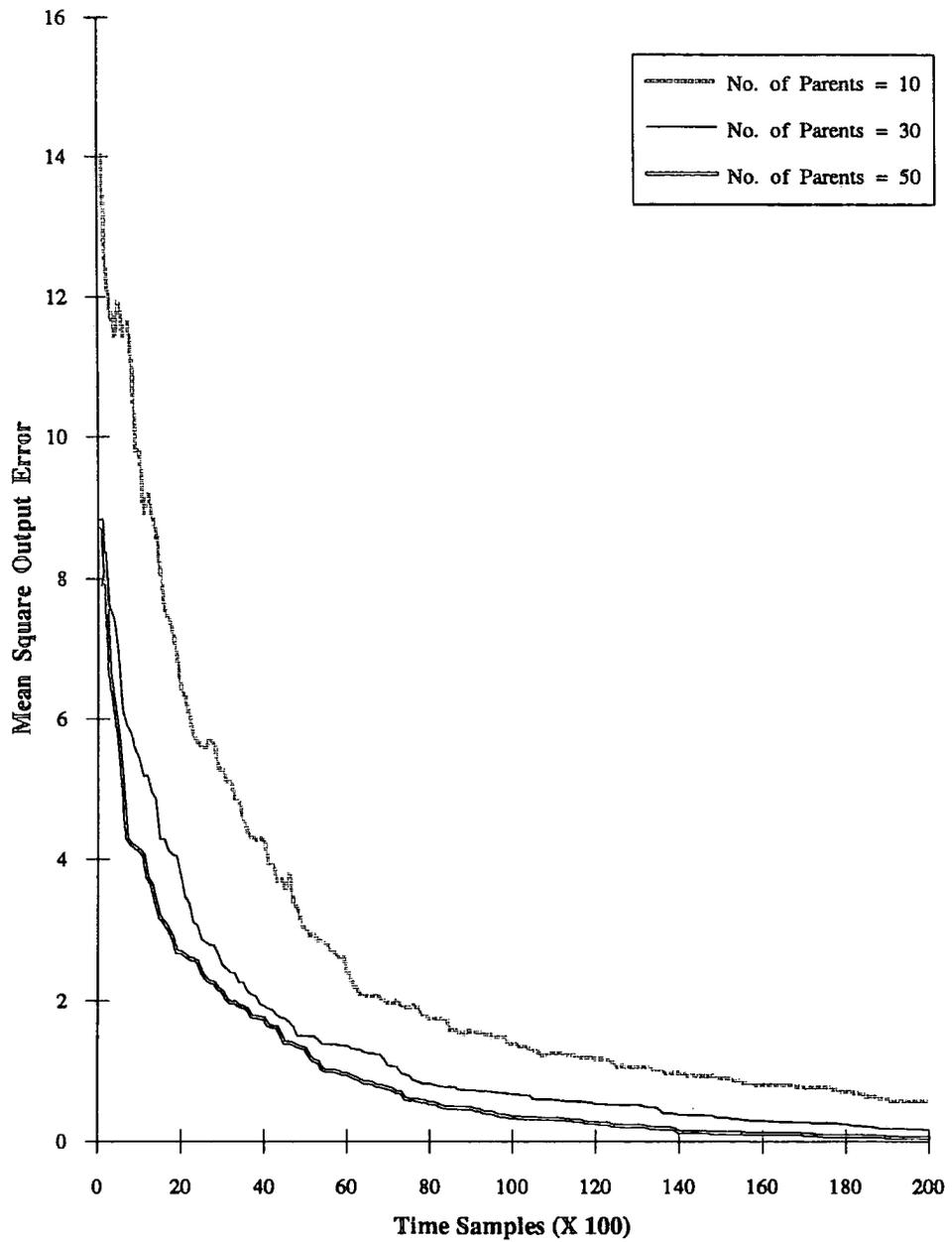


Figure 6.19: Effect of Parents in Evolutionary Programming

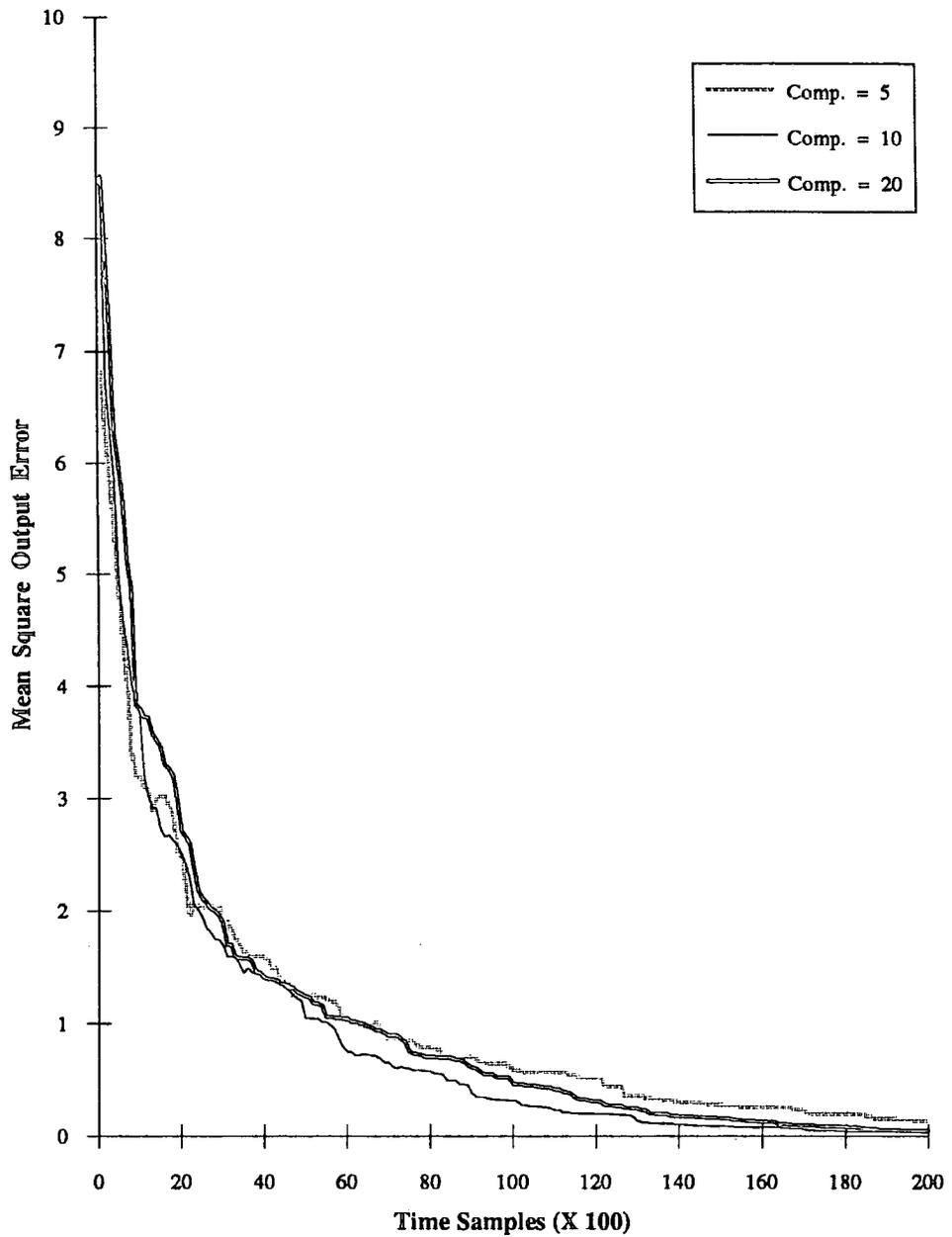


Figure 6.20: Effect of the Number of Competitions in EP

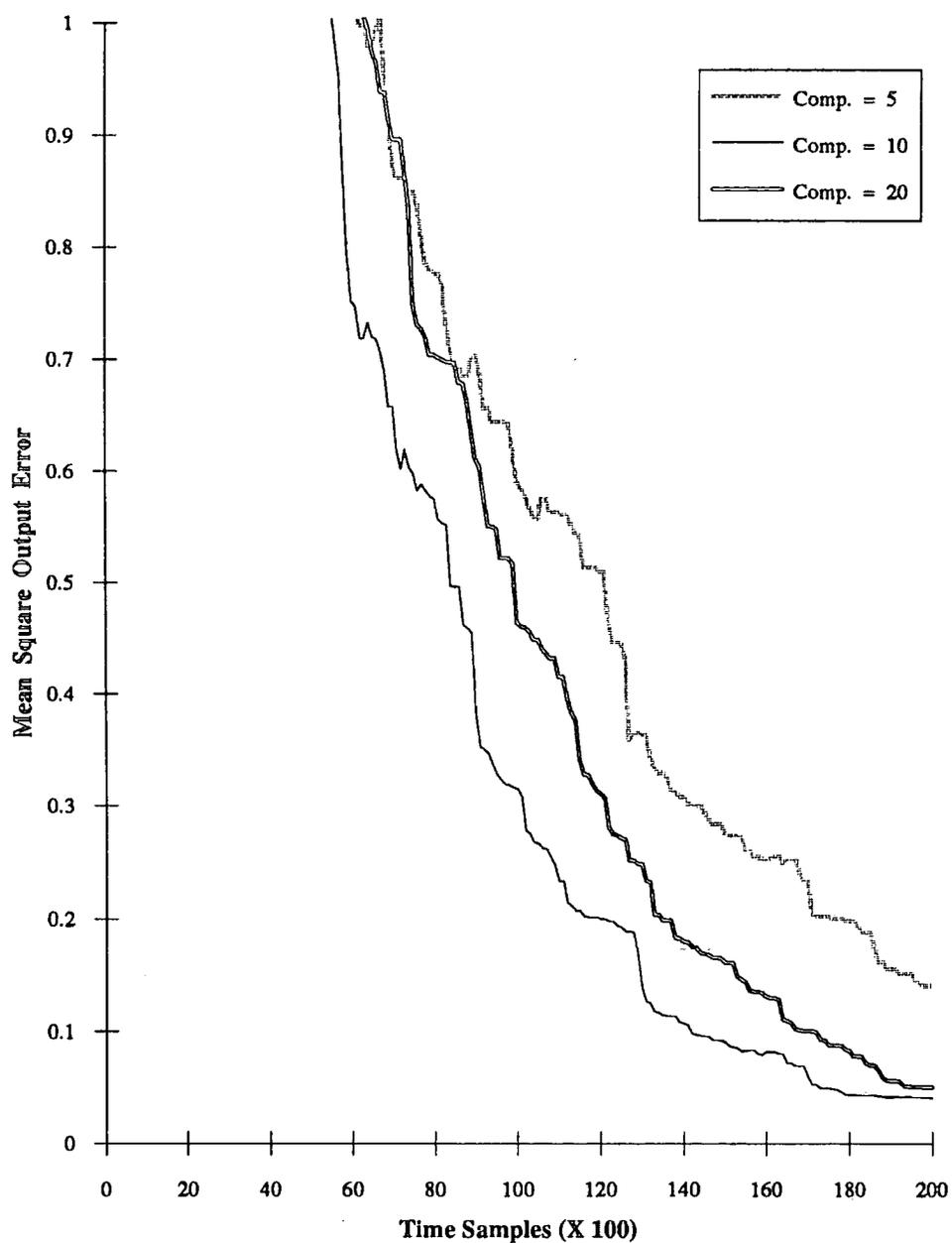


Figure 6.21: Effect of the Number of Competitions in EP

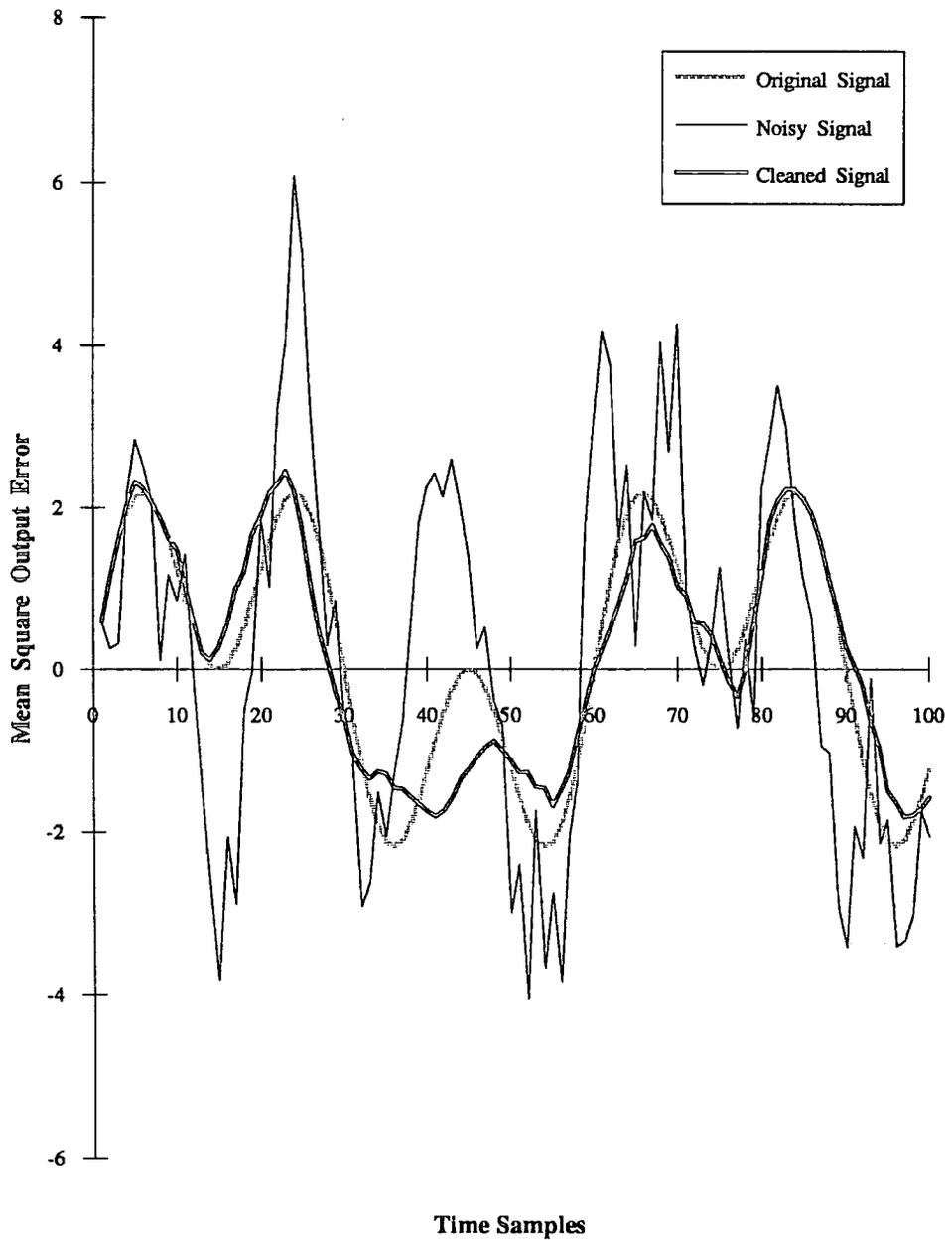


Figure 6.22: Adaptive Noise Canceling - Sum of Sinusoids

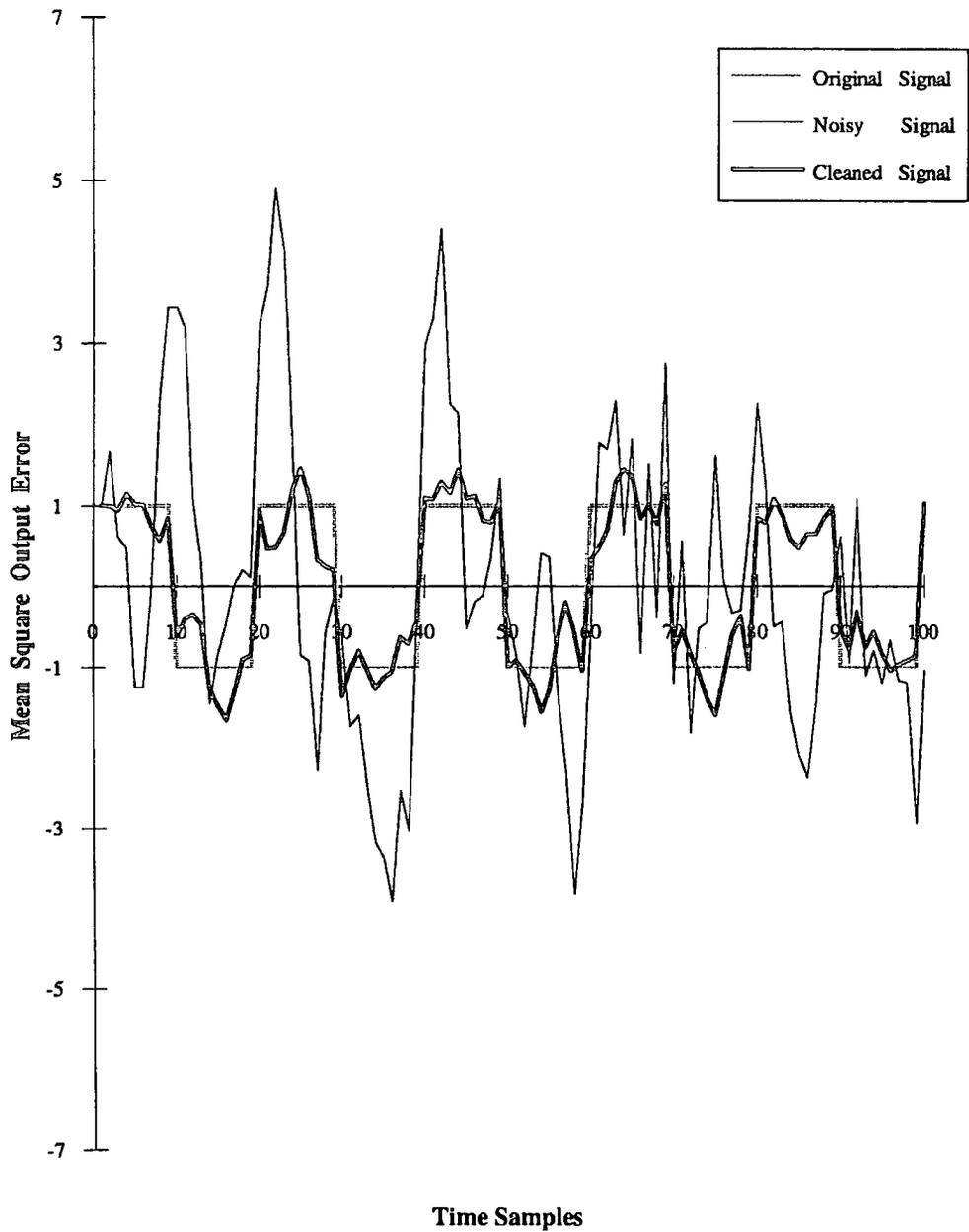


Figure 6.23: Adaptive Noise Canceling - Square Wave

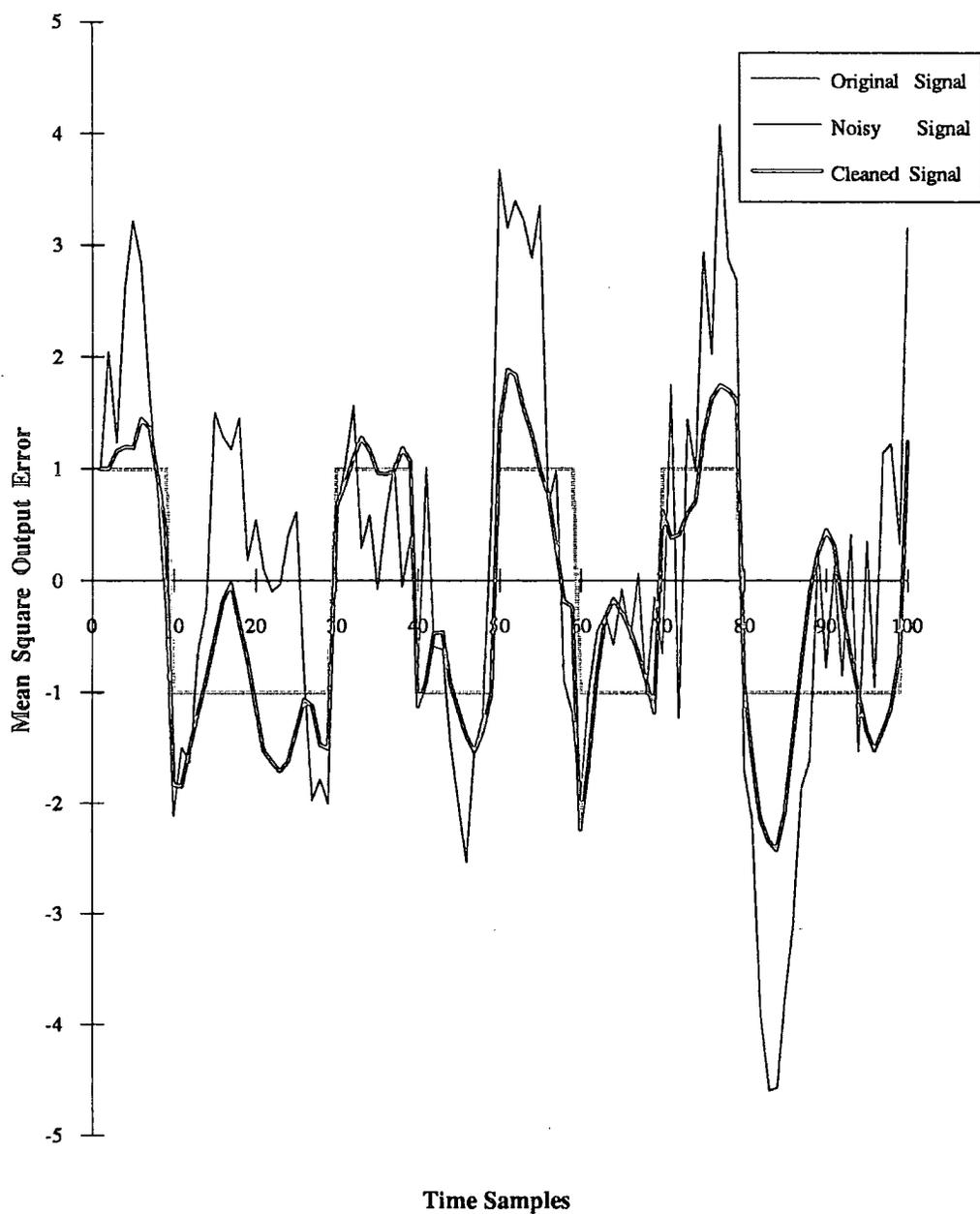


Figure 6.24: Adaptive Noise Canceling - PRBS Input

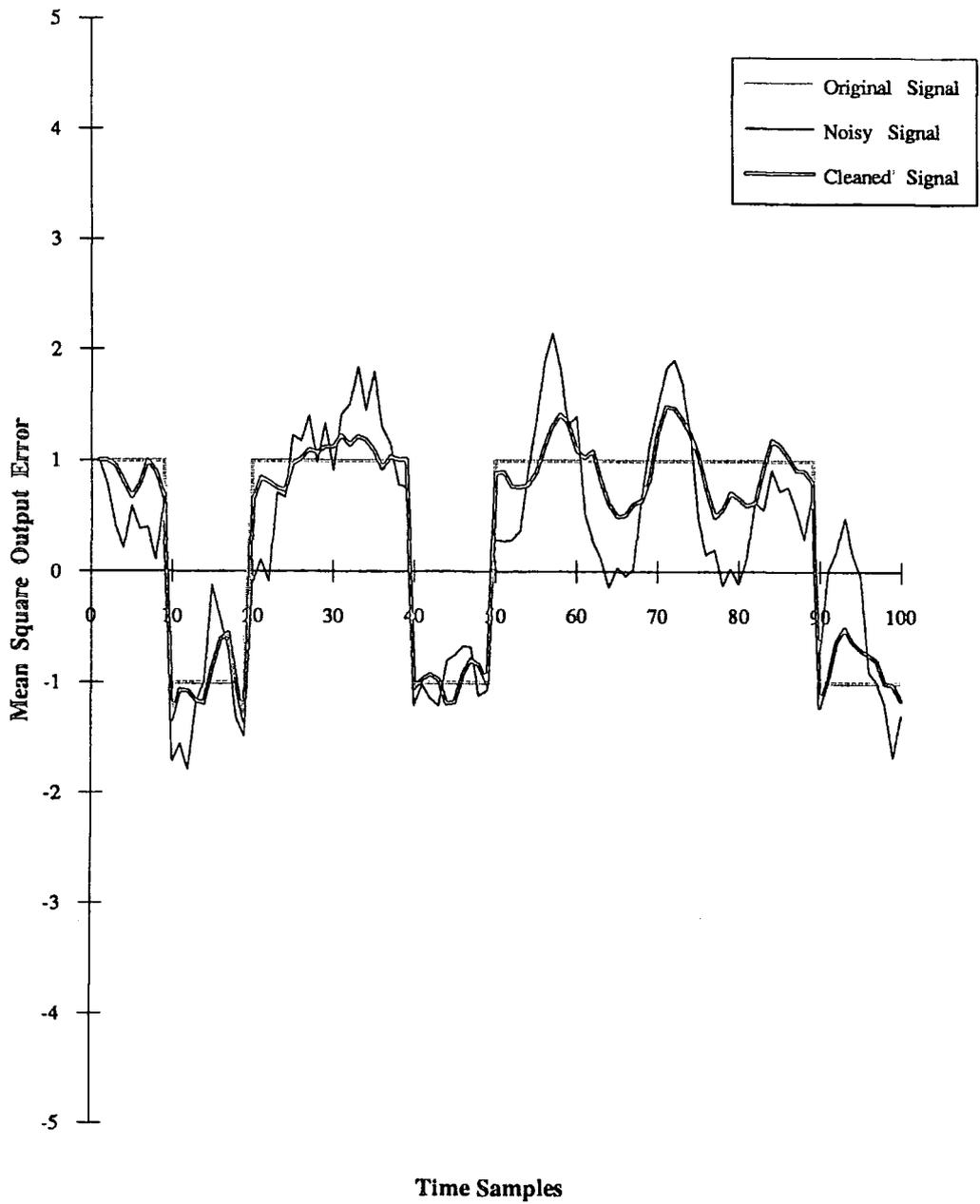


Figure 6.25: Adaptive Noise Canceling - PRBS Input

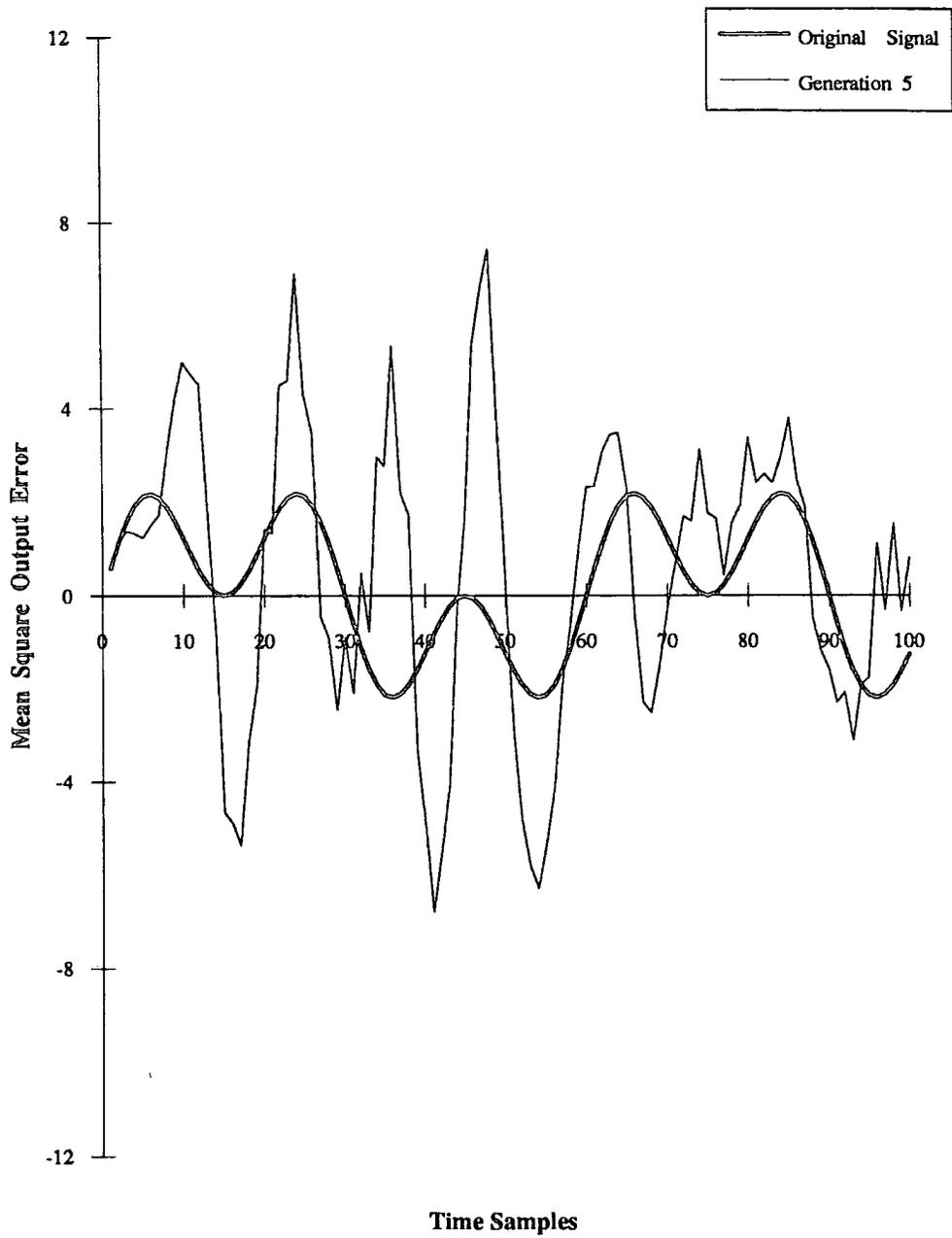


Figure 6.26: Evolution of the Adaptive Noise Canceling

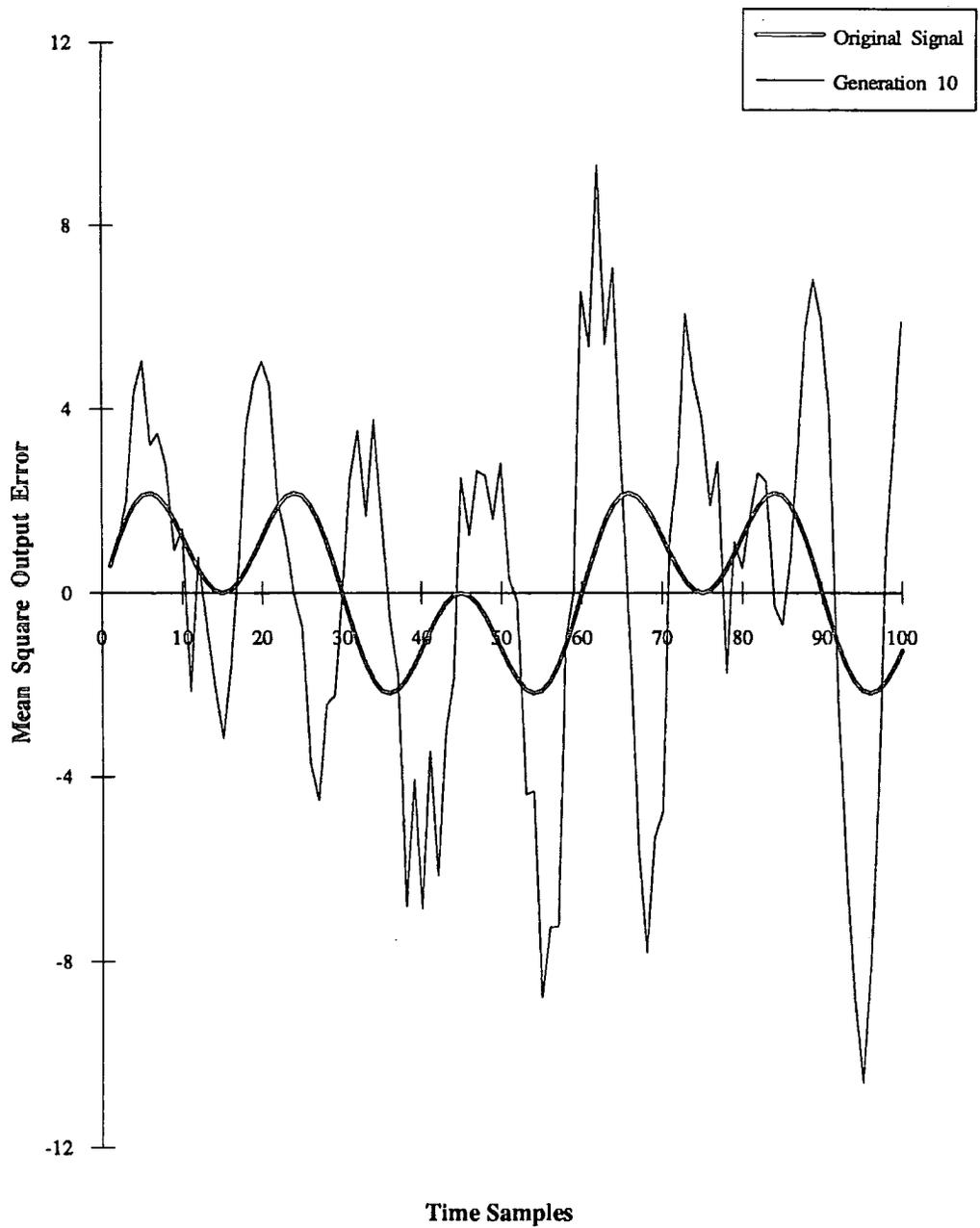


Figure 6.27: Evolution of the Adaptive Noise Canceling

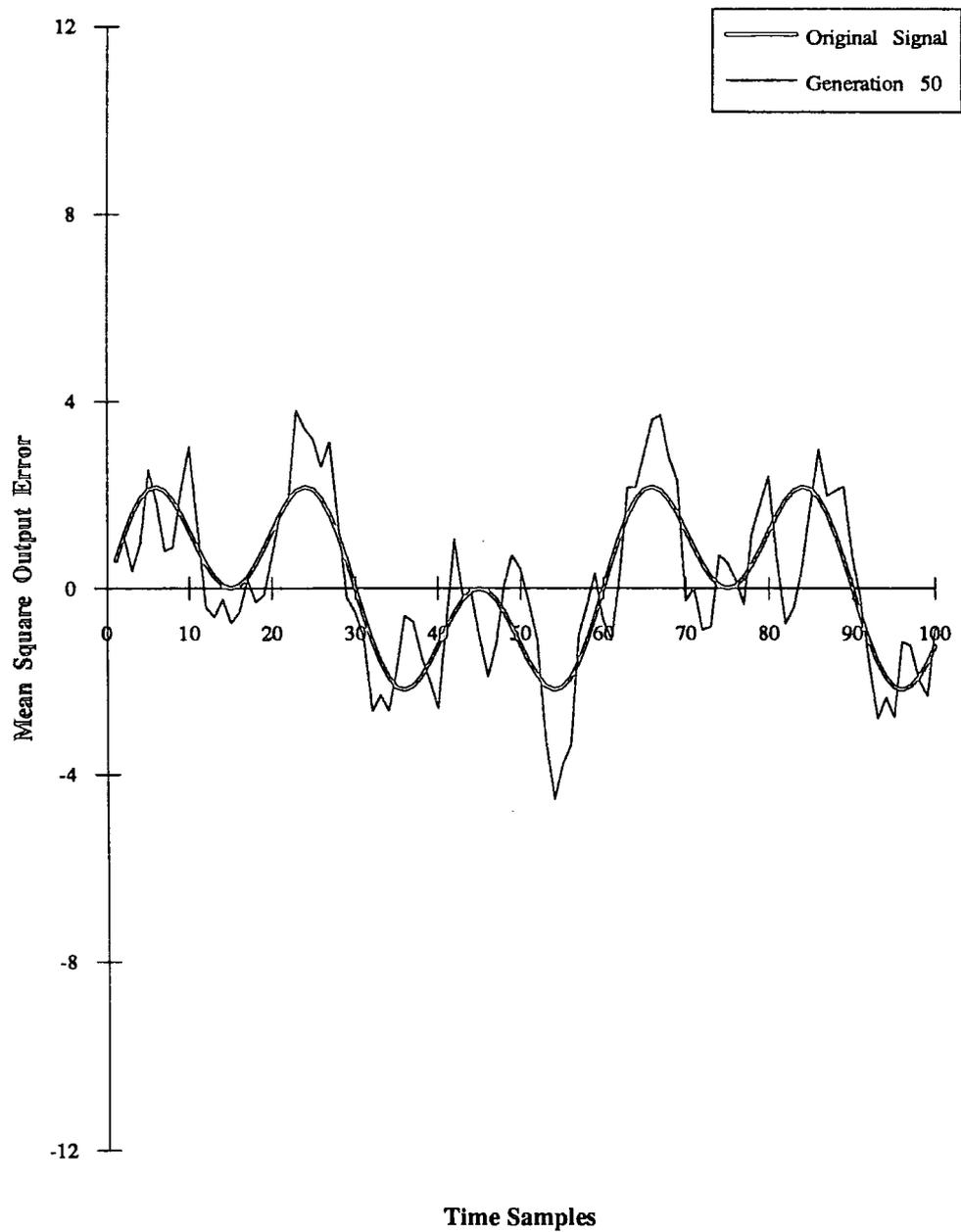


Figure 6.28: Evolution of the Adaptive Noise Canceling

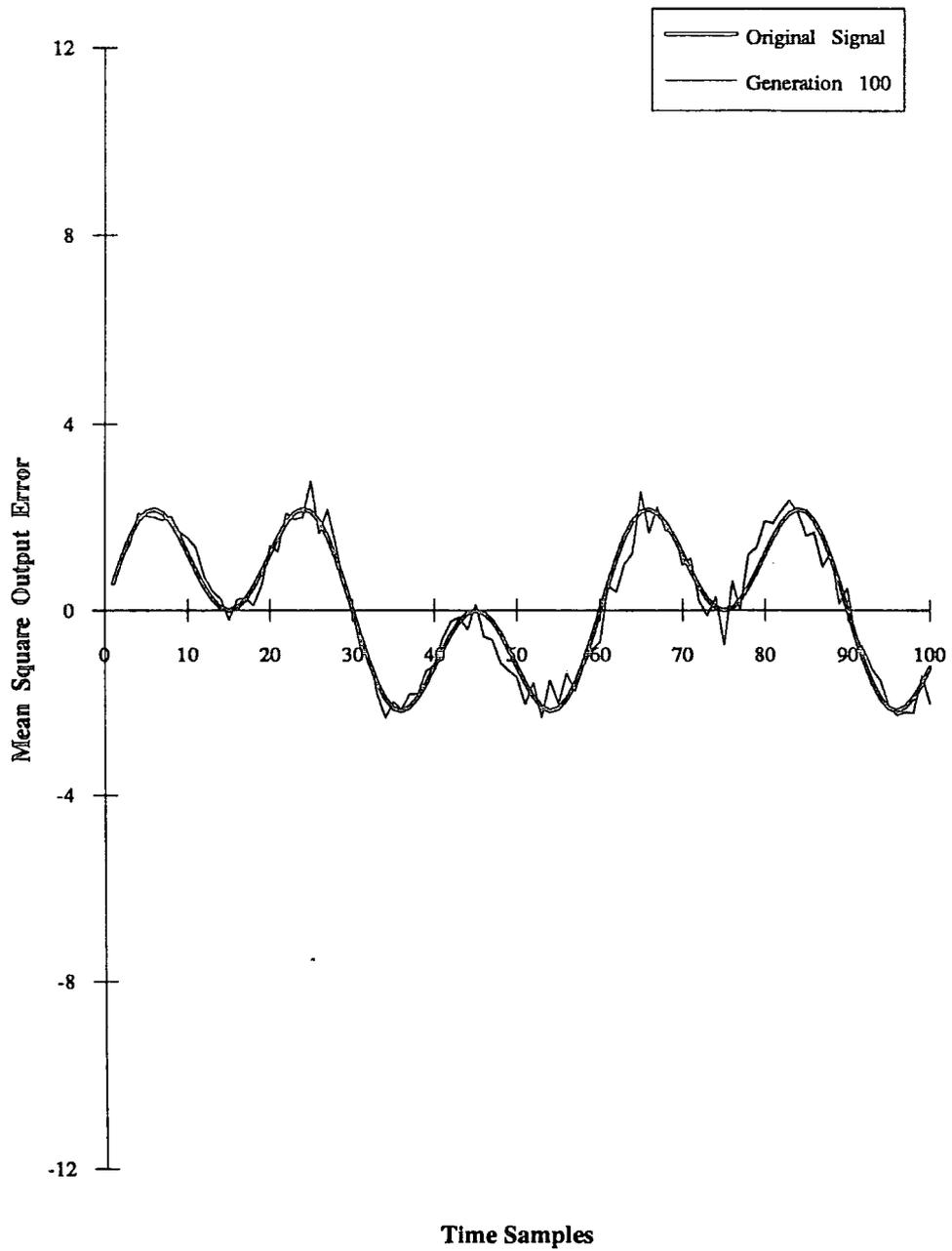


Figure 6.29: Evolution of the Adaptive Noise Canceling

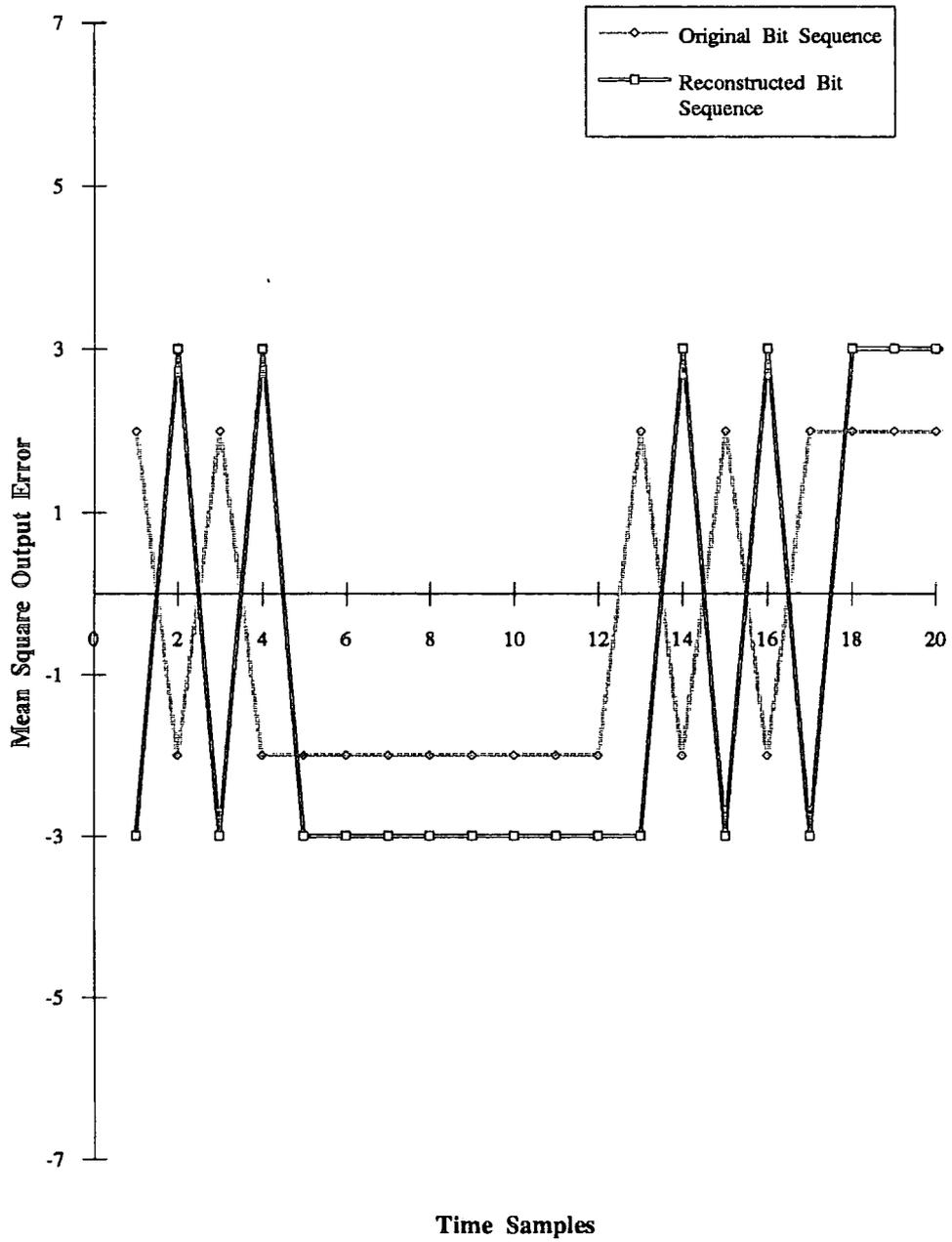


Figure 6.30: Results from the Adaptive Equalisation Experiment

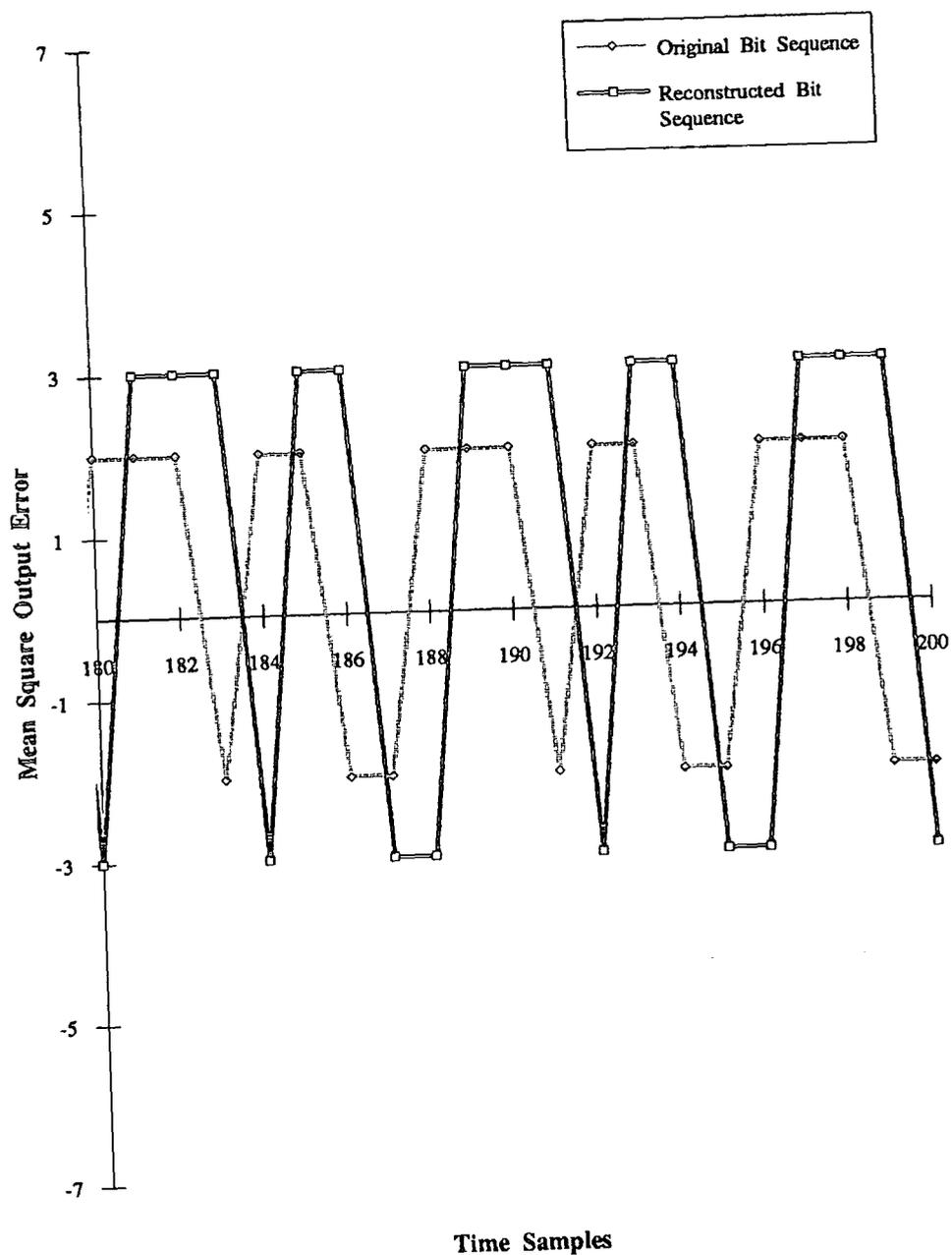


Figure 6.31: Results from the Adaptive Equalisation Experiment

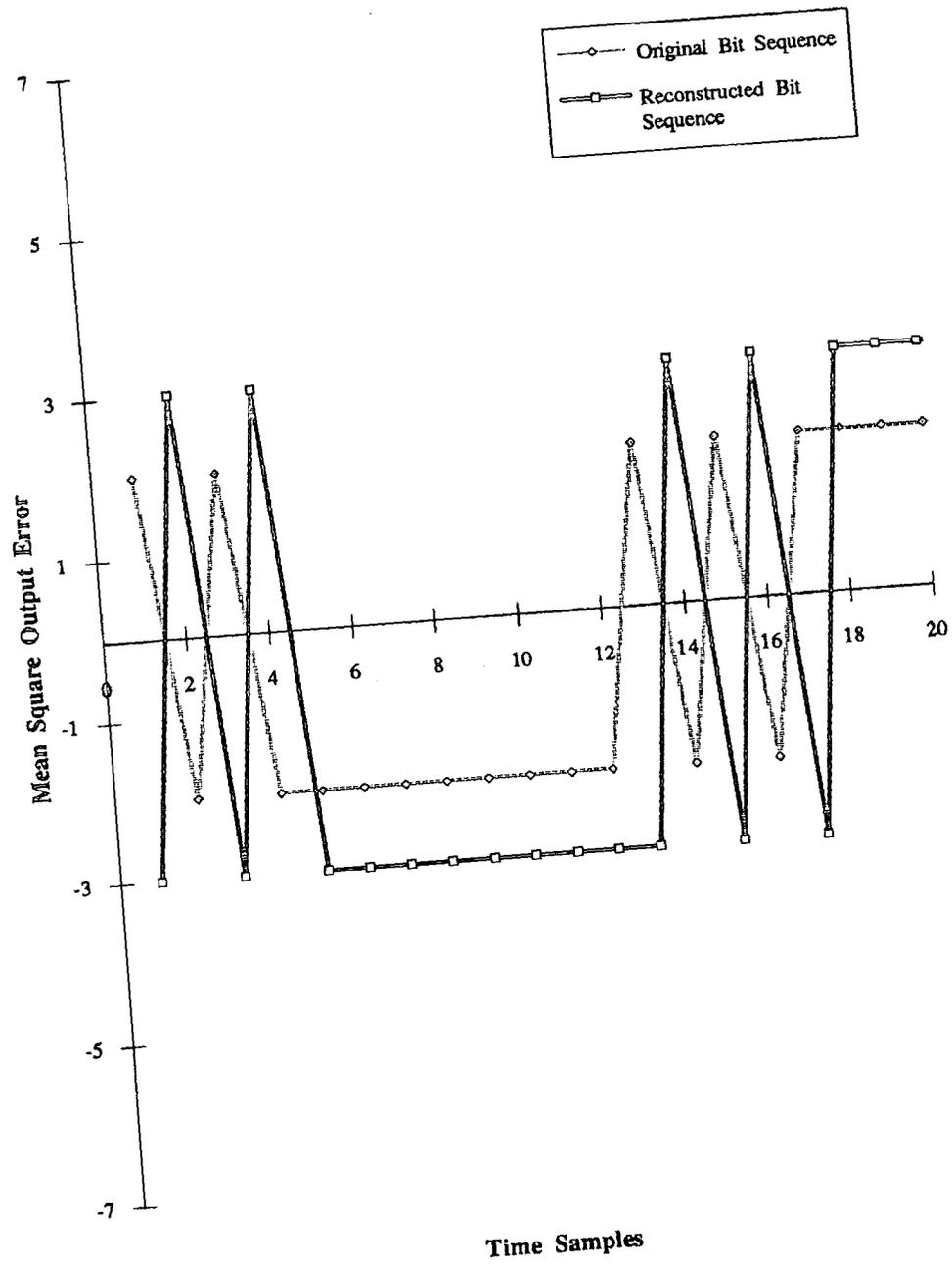


Figure 6.32: Results from the Adaptive Equalisation Experiment

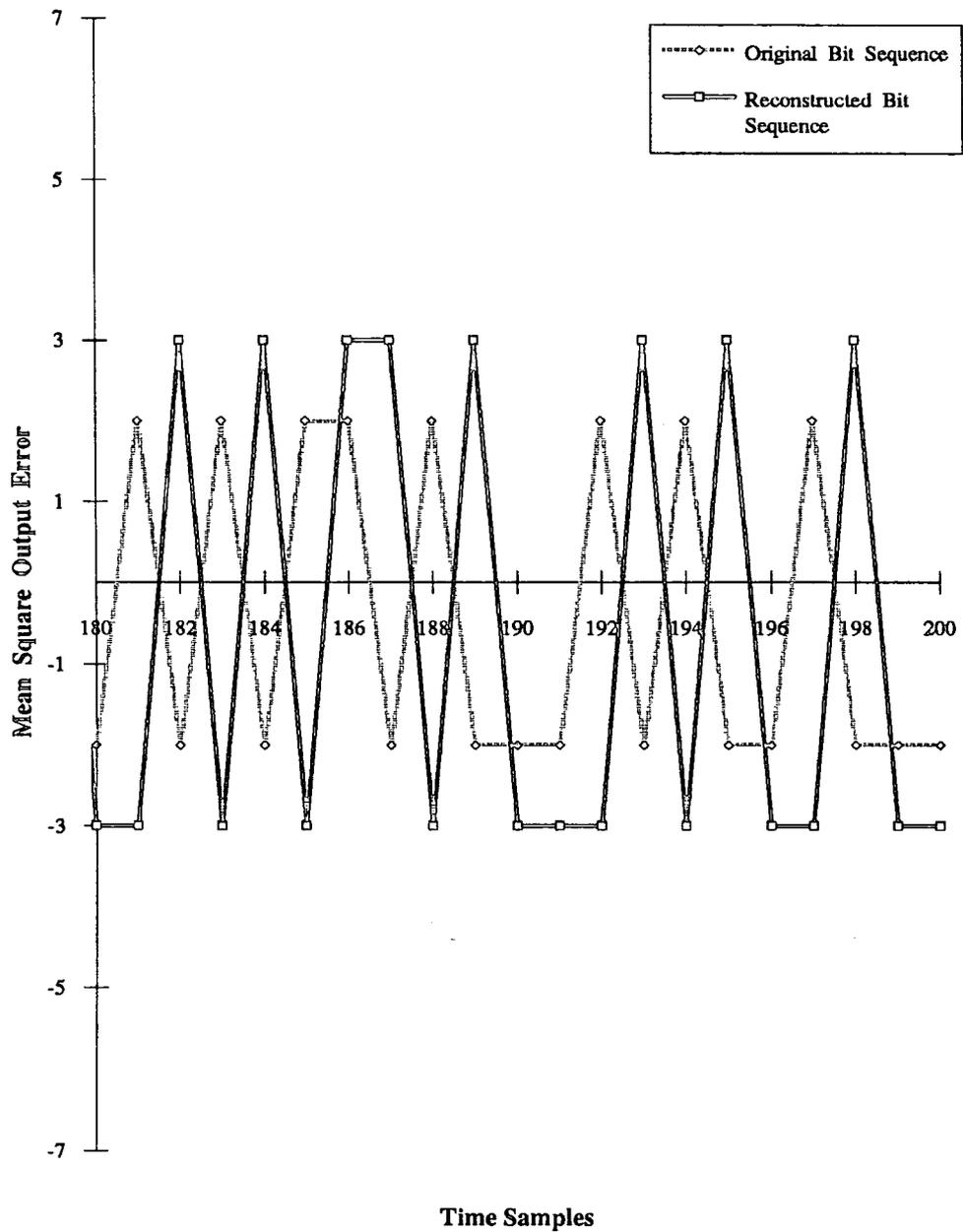


Figure 6.33: Results from the Adaptive Equalisation Experiment

## Chapter 7

# Simulated and Genetic Annealing

### 7.1 Introduction

Combinatorial optimisation problems, especially those which involve a large state space, are extremely difficult to optimise as the computational time increases exponentially with the number of object variables. These problems can only be solved approximately in polynomial time. Thus in such situations, approximation algorithms are used, with which one can arrive at a reasonable approximation to the optimal solution in an acceptable amount of computational time. One such approach is to use an iterative improvement algorithm with a large number of different initial starting points. Iterative improvement algorithms are also known as neighbourhood search algorithms and work by generating a new configuration from a current configuration. The new configuration is selected to be in a neighbourhood around the current configuration. If the new configuration is *better* than the current configuration, then it replaces the current configuration, else another neighbouring configuration is selected. The algorithm terminates when a configuration yields no neighbouring configuration which are better than the current configuration. Convergence to the optimal solution can then be obtained in a reasonable amount of time with such an approach, when a large number of initial configurations are used. Simulated annealing is another such approximation algorithm for combinatorial optimisation and is based on aspects of both *iterative improvement* and *randomisation techniques* [vLA87] enabling the method to be extremely robust.

The inspiration for simulated annealing in its original form [KGV83, Cer85], has been drawn from physics, where an analogy was drawn between slow cooling of a heated solid and the problem of minimizing the cost function of a combinatorial optimisation problem. In thermodynamics, *annealing* refers to the process in which a solid is heated up to a maximum value whereby the solid changes into the liquid phase with all the molecules in a state of random motion. The heated solid is then cooled slowly by reducing the temperature gradually. If the cooling is performed sufficiently slowly and the maximum temperature was reached during the heating phase, all the particles would settle into a minimum energy ground state of a corresponding lattice. At each value of temperature, the solid is allowed to reach *thermal equilibrium*. As the temperature approaches the limiting value of zero, the substance will settle into the minimum energy states corresponding to the low energy crystalline state. If the process of cooling is too rapid, then the solid may settle into a metastable state corresponding to an amorphous structure. These principles have been adopted in the optimisation technique of *simulated annealing*. The precise details of the approach are presented in a subsequent section. The method of simulated annealing is also known by different names such as *Monte Carlo annealing*, *statistical cooling*, *probabilistic hill climbing*, *stochastic relaxation* or *probabilistic exchange algorithm*.

The simulated annealing algorithm has asymptotic properties of convergence and in most practical applications the convergence time is very large, thus reducing the practical use of the method. Thus research has looked into ways of speeding up the basic simulated annealing algorithm. One such algorithm is the *Fast Simulated Annealing* proposed by Szu and Hartley [SR87a, SR87b]. Their approach is to use a different probability distribution in the generation of new states and they have proved the increased speedup of the algorithm. More details of this approach are presented in a subsequent section. Recently, a new algorithm has been proposed which has convergence properties orders of magnitude greater than the standard simulated annealing. This approach known as the *Very Fast Simulated Reannealing* was proposed by Ingber and Rosen [Ing89]. This approach again uses of a different generating function as well as a *reannealing procedure* where the sensitivities of different parameters are taken into account and a rescaling procedure is employed to allow for this.

In this chapter, we propose two new hybrid schemes where concepts of simulated annealing are used in standard genetic algorithms. These new approaches show very promising simulation results and have been termed *Genetic Annealing*. Simulation results are presented which show the performance of these algorithms for adaptive filtering. The next section reviews the technique of simulated annealing.

## 7.2 Simulated Annealing

As was stated above, the technique of simulated annealing is derived from concepts in statistical mechanics wherein a crystalline low energy state of a solid is obtained by initially heating it to large value to reduce it to a liquid state, and then gradually cooling the liquid state whereby the substance crystallizes into the required low energy state. For each temperature value  $T$  at thermal equilibrium, the probability that the substance is in a state with energy  $E$  is given by the Boltzmann Distribution.

$$Pr\{\mathbf{E} = E\} = \frac{1}{Z(T)} e^{\frac{-E}{k_B T}} \quad (7.1)$$

$Z(T)$  is a normalisation factor,  $k_B$  is *Boltzmann constant* and the expression  $e^{\frac{-E}{k_B T}}$  is known as the *Boltzmann factor*. As the temperature reduces, the Boltzmann distribution chooses only states with low energies and when the temperature approaches the limiting value of zero, only the minimum energy states have a non-zero probability of occurrence. To simulate the evolution of the process to thermal equilibrium at a particular temperature  $T$ , Metropolis *et al.* [Mea53] used a *Monte Carlo* method to generate the sequence of states. This was achieved as follows: The current state of the solid which was characterised by the positions of the the particles of which it was composed of, was given a small random perturbation to result in a new state. If the difference in the energies between the current state and the new state,  $\Delta E$ , was negative, then the new state was used as the current state and the process continued. If  $\Delta E \geq 0$ , then the new state was accepted with a probability which was given by

$e^{-\frac{\Delta E}{k_B T}}$ . This acceptance rule is referred to as the *Metropolis criterion*. After a large number of perturbations the probability distribution of the states approach that of the Boltzmann Distribution as given in Equation [7.1]. This algorithm known as the *Metropolis Algorithm* has been used extensively in statistical mechanics to estimate averages and integrals by means of random sampling [Bin78, Has70].

To use this technique in combinatorial optimisation, the different configurations of the optimisation problem would correspond to the states of the solid while the objective function and a *control parameter* would assume the roles of energy and temperature. Thus the simulated annealing approach is a sequence of Metropolis algorithms evaluated at decreasing values of the control parameter. The algorithm starts with a large value of the control parameter. From a given state  $i$ , a new state  $j$  is generated using a generation mechanism. This corresponds to the perturbation step of the Metropolis algorithm. The cost function of both the states are determined and the difference between the cost functions  $\Delta C_{i,j}$  calculated. Then if  $\Delta C_{i,j} \leq 0$ , the new state is accepted with probability 1. If  $\Delta C_{i,j} > 0$  the probability of acceptance is given by  $e^{-\frac{\Delta C_{i,j}}{c}}$  (*Metropolis criterion*). This step is the crucial factor in the simulated annealing approach as it allows probabilistically to accept solutions that are worse (higher cost) than the previous solution. Thus there exists a non-zero probability of jumping out of local optima. This process is continued for a certain number of steps until an equilibrium has been reached for that value of the control parameter indicating that the probability of the system being in any particular energy state is given by the Boltzmann distribution (Equation [7.1]). The control parameter  $c$  is then reduced in steps, with the system allowed to reach an equilibrium state at each value of the control parameter. The algorithm is terminated when the control parameter  $c$  reaches a predetermined small value. A mathematical model of the algorithm is presented in Appendix B.

The three important features defining the simulated annealing algorithm are

- A Generation distribution which selects new points from a neighbourhood of the current point. The usual choice for the generation distribution function is a Gaussian probability distribution centered around the current point.

- An Acceptance mechanism which decides whether to accept or reject a newly generated point. The Metropolis criterion is usually employed for the this purpose.
- A Cooling Schedule which suitably decrements the value of the control parameter. The cooling schedules have been studied with a lot of interest and many schemes are currently used [vLA87]. A simple cooling schedule is given by

$$c_{k+1} = \alpha \times c_k, \quad k = 1, 2, \dots \quad (7.2)$$

where  $\alpha < 1$ . This cooling has been used widely by researchers with values of  $\alpha$  ranging from 0.5 to 0.99. It has been proved by Geman and Geman [GG84] that for the inhomogeneous algorithm (Appendix B), the algorithm is able to locate the global minimum provided the cooling is done not faster than

$$c(k+1) = \frac{c(0)}{\log(k)} \quad (7.3)$$

where  $c_0$  is the starting value of the control parameter.

Thus the simulated annealing algorithm can be concisely stated as follows: Using the generation distribution (usually a Gaussian), a new point defined around a neighbourhood of the current point is generated. The acceptance criterion defined by the acceptance matrix is then used to decide whether to accept or reject the new point. Initially, as the control parameter has a large value, all new points including points with a large cost are likely to be accepted. As the value of the control parameter is reduced, only points resulting in low costs will be accepted, thus eventually leading to the global optimum of the cost function. To realise this eventual state, certain conditions are imposed on the generation and acceptance matrices and on the cooling schedule. More details of these conditions and mathematical analysis of the algorithm is given in [vLA87].

### 7.3 Fast Simulated Annealing

The *Fast Simulated Annealing* algorithm was proposed in 1987 by Szu and Hartley [SR87a] and has been proved to have a faster rate of convergence. It was initially proposed as a solution to a continuous optimisation problem in which the cost function  $C$  was defined over a  $n$ -dimensional continuous space. As was detailed in the Appendix B and the previous section the generating distribution of the classical simulated annealing used a Gaussian probability function. This was in some sense a local search around the current operating point and was defined by the variance of the Gaussian distribution used. The *Fast Simulated Annealing* algorithm uses the same concepts as that of the classical simulated annealing except it uses a different distribution for generating the next state. In particular it uses a *Cauchy Distribution* which is defined by the equation

$$G(x) = \frac{c(k)}{c(k)^2 + x^2} \quad (7.4)$$

where  $c$  is the control parameter. The advantage of using the Cauchy distribution is that the Cauchy process is a infinite variance distribution and thus has a fatter *tail* as compared to the Gaussian process. This permits occasional long search steps amidst local sampling thus leading to faster convergence. Similar to the condition proved for the classical simulated annealing (Equation [7.3]), there exists a rule for the rate of cooling for the Cauchy annealing which is given by

$$c(k+1) = \frac{c(0)}{k} \quad (7.5)$$

It has been proved that if the control parameter is reduced no faster than Equation [7.5] given above, the algorithm is able to locate the global optimum. It can be seen from Equations [7.3] and [7.5] that the rate of convergence of the Fast simulated annealing algorithm is faster than that of the classical simulated annealing. The proofs for the rate of convergence of the method are given in [SR87a, SR87b].

## 7.4 Very Fast Simulated Reannealing

The *Very Fast Simulated Reannealing* algorithm was proposed by Ingber and Rosen in 1989 ([Ing89]) as a technique of fitting empirical data to a theoretical cost function which is defined over a D-dimensional parameter space. This algorithm has been used in diverse applications such as combat analysis, finance and neuroscience. The main motivation for the approach has been the knowledge of the fact that both classical simulated annealing and fast annealing use generating distributions that do not take into account that different parameters may have different annealing sensitivities.

Very fast simulated reannealing introduced two differences from the standard and fast annealing approaches. The first was a new generating function which was easy to generate for D-dimensions as the D-dimensional form was just the products of the single dimensional form. The D-dimensional generating function was thus given by

$$\begin{aligned}
 G(x) &= \prod_{i=1}^D \frac{1}{2(|x_i| + c_i) \ln(1 + 1/c_i)} \\
 &\equiv \prod_{i=1}^D G_c^i(x^i)
 \end{aligned} \tag{7.6}$$

It can be seen from the above equation that the control parameter  $c_i$  is not the same for the different dimensions but has a different value for each dimension. The cooling schedule for the above function, which has been statistically proved to enable the algorithm to reach the global optimum, is given by

$$c_i(k) = c_i(0)e^{-z_i k^{1/D}} \tag{7.7}$$

where  $c_i(0)$  is the starting value for the control parameter for dimension  $i$  and  $z_i$  is a constant for each dimension. The new value of a parameter  $x_i$  at iteration  $(k+1)$  is given by

$$x_i(k+1) = x_i(k) + y_i(B_i - A_i) \tag{7.8}$$

where  $y_i$  is generated using Equation [7.6] and  $A_i, B_i$  are the limits of the parameter  $x_i$ .

The second concept introduced by the method was a way to incorporate the different sensitivities of parameters into the annealing procedure. It was an attempt to stretch out the range over which relatively insensitive parameters were being searched, relative to the ranges of the more sensitive parameters. This was achieved by a process referred to as *reannealing* which was essentially a rescaling procedure. Thus periodically the annealing time  $k$  was rescaled for each parameter dimension. The procedure for doing this is presented in [Ing89]. Although not specifically studied in this thesis, the *Very Fast Simulated Reannealing* algorithm is worthy of further study.

## 7.5 Genetic Annealing

### 7.5.1 Introduction

In this section, two new techniques are proposed which combine concepts from simulated annealing and genetic algorithms. A way to view this approach is to look at the basic process which describe these two approaches. The simulated annealing process uses the Boltzmann distribution while genetic algorithms are based on the Darwinian principle of *survival of the fittest*. There have been earlier efforts in developing optimisation schemes which are based on concepts derived from both annealing and genetic algorithms. Boseniuk and Ebeling in [BEA87] have attempted to improve the simulated annealing process by incorporating the concepts of competition and selection. This followed an earlier work by Ebeling and Engel ([EA86]), where a systematic comparison was drawn between Boltzmann and Darwinian strategies by analysing the underlying equations which described the two process. The conclusion arrived at was that both methods show significant differences when the transitional behaviour was analysed. Thus in [BEA87], Boseniuk and colleagues have used the Darwinian elements of competition and selection in simulated annealing. The important result which they arrive at is that using this hybrid scheme, the region of good solutions are reached with higher probability than that is achieved using only a single scheme of either annealing or genetic selection. A similar approach was used in [BE91], where in addition to the hybrid schemes incorporating the two strategies given above, a

new hybrid scheme was proposed. This scheme combined concepts from Boltzmann, Darwinian and Haeckel strategies. The Haeckel strategy ([EAM86]) was based on the observation from natural evolution where it was noticed that each biological organism undergoes a life cycle consisting of a period of early growth, a period of learning, a period of reproduction and finally death. This strategy highlighted the fact that in the early stages the mutation operator is more active while in the later stages it is the selective pressures which dominate. Thus a Haeckel strategy is composed of two stages:

- A period of *youth* where mutations are frequent and selection seldom.
- A period of *maturity* where mutations are seldom and selection occurs frequently.

The conclusion drawn from this study also indicated that the mixed strategies yielded a better performance than in comparison with the single strategies by themselves.

In the next sections, two new hybrid schemes are proposed which are based on concepts from both annealing and genetic algorithms. A motivation for these schemes has been the observation that although genetic algorithms were able to locate the optimal solution rapidly, the whole population did not converge to the optimal solution. These schemes overcome this limitation whereby all the members of the population converge to the optimal solution. This has been shown using simulation experiments for the adaptive filtering case.

### 7.5.2 Hybrid Scheme - I

As was stated in an earlier chapter, the role played by mutation in genetic algorithms has been largely secondary. This has been challenged by researchers and is also evident from the simulation results presented in Chapter 6. Too large a value of the mutation rate, though increasing the exploratory power of the algorithm, renders it similar to a random search algorithm, where there is no exploitation of the solutions which have been obtained until then. An approach to overcome this problem would be to use a large value of mutation at the initiation of the algorithm, but then to gradually

reduce the mutation rate as the generations evolve. Thus mutation now plays a role akin to that played by the control parameter in the simulated annealing algorithm where a large value of the control parameter enables the algorithm to initially search a wide area, but with lower values of control parameter to concentrate on the more promising but smaller regions. As the mutation rate is now varying, the proposed scheme is similar to the Haeckel optimisation strategy explained before, the main difference being that the selection process is not changed during the generations and remains the same (proportional selection).

To reduce the mutation rate during the adaptation process, the mutation rate is made a function of the generation number. Thus initially the algorithm uses a large value of mutation which is gradually reduced as the generation number increases. Two approaches were used to decrease the mutation rate - in the first approach the mutation rate was a linear function of the generation number while in the second approach the mutation rate was varied in an exponential manner depending on the generation number. The first approach resulted in either premature convergence to a non-optimal solution or resulted in a random search algorithm. The reason for this was found to be the rate at which the mutation value was reduced. Too fast a reduction of the mutation value resulted in premature convergence while with too slow a reduction, the algorithm is not able to exploit near optimal solutions which may have been discovered. This led to the formulation of the second scheme where the mutation rate was an exponential function of the generation number. The decrease in the mutation was performed using the following equations:

$$\begin{aligned} p_m &= \frac{p_m(\text{start}) \times tmp}{1 + tmp} \\ tmp &= e^{\frac{Gen_{st} - Gen_{No}}{decay}} \end{aligned} \quad (7.9)$$

$p_m$  is the probability of mutation and  $Gen_{No}$  is the generation number. The equation has three parameters which are initialised at the start of the algorithm. These are the starting probability  $p_m(\text{start})$ ,  $Gen_{st}$  and the decay parameter  $decay$ . As a result of the exponential nature of Equation [7.9], the value of the probability of mutation remains near the starting value of  $p_m(\text{start})$  until the number of generations reach

the value  $Genst$ . Thereafter the probability of mutation reduces, the rate of decrease depending on the decay parameter  $decay$ . The initial values of these parameters determine the accuracy and rate of convergence of the algorithm.

### 7.5.3 Hybrid Scheme - II

The second hybrid scheme proposed also combined concepts from simulated annealing and genetic algorithms. In simulated annealing an important idea was to use a probabilistic expression to decide whether to accept or reject a new configuration. This was achieved by using the Metropolis criterion as given by Equation [B.5]. The second hybrid scheme proposes the use of this criterion in genetic algorithms. Specifically this is achieved as follows: After the selection operation, two strings  $P_1, P_2$  are drawn randomly from the population for the genetic operations of crossover and mutation. After the crossover and mutation operations two new strings  $C_1, C_2$  are formed. If the new strings ( $C_1, C_2$ ) have a larger value of fitness (lower error value) than the parent strings ( $P_1, P_2$ ) then they are retained as the members of the next population. However if they have a smaller fitness value than the parent strings, then they are only retained probabilistically using a condition similar to the Metropolis condition. This is the significant change from the standard genetic algorithm where the new strings are always used to generate the members of subsequent populations. As the algorithm uses the Metropolis criterion, an important parameter of the process is the temperature or control parameter. The reduction in the control parameter is done using the simple cooling schedule as given by Equation [7.2], where  $\alpha = (0.9 - 0.99)$  is the rate of cooling. An important condition of the homogeneous simulated annealing algorithm (Appendix B) was that at each value of control parameter, the length of the resulting Markov chain should be infinite. However in practical applications this condition is made less stringent by reducing the control parameter value after a certain number of new points have been *accepted* using the acceptance criterion. This scheme is used in the second hybrid scheme. Thus at the start of the algorithm, the control parameter has a large value and all the offspring strings are retained as parents for the new generation. But as the algorithm proceeds, the value of the control

parameter reduces, thus only offspring strings which perform better than the parent strings are retained. As a result of the selective pressures and the acceptance criterion detailed above, members of the populations will converge towards a single string structure having the optimal value of fitness. Thus the average error in a population (generation) approaches the minimum error in a population. Thus the second hybrid scheme has two defining parameters - the starting value of the control parameter and the rate of decay  $\alpha$ .

## 7.6 Simulation Configuration and Results

In order to use the above algorithms for the adaptive IIR filtering paradigm, the system identification configuration was used wherein the unknown system was a sixth order IIR filter. The sixth order IIR filter was composed of a parallel bank of three second order IIR filters. This configuration was used in order to overcome the problems of stability as was detailed in chapter 2. The performance surface of such a configuration can result in a multimodal surface with local optimas [NJ89]. Thus the use of gradient algorithms may result in a non-optimal performance.

For both classical and fast annealing, the important equation is the one which describes how the next point is generated from the current point. The defining equation for this step is given by

$$x_i(\text{new}) = x_i(\text{prev}) + R \times s_i ; i = \{1, \dots, n\} \quad (7.10)$$

$n$  is the number of coefficients of the filter (dimension of the problem), while  $R$  is generated using the generating distribution. For classical annealing,  $R$  is generated using the Gaussian distribution, while for fast annealing  $R$  is generated using the Cauchy distribution.  $s_i$  is the step size for the coefficient  $i$ . If the newly generated coefficient  $x_i(\text{new})$  is outside the limits set for that coefficient, then Equation [7.10] is used again until the new coefficient generated satisfies the constraints. These constraints are usually imposed in order to keep the filter stable as was explained in chapter 2. Each change in a coefficient value using Equation [7.10] results in a

new state of the annealing process. This new state is retained using the Metropolis criterion. The algorithm cycles around the set of coefficients, perturbing each using Equation [7.10], and retaining the new state using the Metropolis criterion. The reduction in the control parameter was done using the cooling schedule given in Equation [7.2]. Both the homogeneous and inhomogeneous forms of the annealing algorithm was simulated. For the homogeneous case, the control parameter value was not reduced until a certain number of new states were accepted, while for the inhomogeneous implementation the control parameter was continuously reduced.

Results of using the classical and fast annealing are shown in Figures [7.1] and [7.2]. As can be seen, the fast annealing approach results in faster convergence than the classical annealing algorithm. Though this is an improvement, from the results obtained for the genetic algorithms, it is clear that the annealing approaches take a large number of time samples for accurate convergence to the global optimum. The initial value of the temperature in both the simulations was 1000, the decay parameter being varied. It can be seen that the decay parameter is responsible for the rate of convergence of the algorithm. Comparative results between classical and fast annealing are shown in Figure [7.3]. The value of the decay parameter in this case was 0.9.

Results of using the new hybrid algorithms (genetic annealing) are shown in Figures [7.4-7.13]. Figures [7.4-7.9] present the result of using the hybrid strategy (I) for varying values of the decay parameter. For slow value of the decay rate (Figure [7.4]) (decay parameter = 100), the average error is still high though the minimum error in the generation has reduced down to the optimal value. The same result is shown in Figure [7.5] at a higher resolution. Figures [7.6,7.7] show the result at a decay parameter value of 50. However too rapid a reduction of the mutation rate (indicating fast cooling) results in the algorithm getting locked in a non-optimal state as is shown in Figure [7.8, 7.9] (decay parameter value = 15). This fact can be inferred from error value to which the algorithm has converged.

From these results it can be seen that the average error in a generation now approaches and equals the minimum error which signifies the fact that all the members of the population have converged to a single structure. Whether this structure is the

global optimum has only been verified using simulation results and theoretical analysis of the method is not yet available. It can be noticed that the variance of the average error and minimum error is large. This arises because of the large initial value of the mutation probability which results in the algorithm exploring over a large area of search.

The results of using the hybrid scheme (II) are shown in Figures [7.10-7.13]. As in case of hybrid scheme (I), the average error in a generation approaches that of the minimum error indicating that all the members of the population has converged to a single string. The immediate observation from these set of results is that the variance of both the average error and minimum error is very much reduced as compared to hybrid scheme (I). This is because of the relatively small value of the mutation probability in the second hybrid scheme. Figures [7.10] and [7.11] shows the result of using a varying values of the decay parameter and a value of 0.075 for the mutation probability. It can be seen that the decay parameter decides the rate of convergence of the algorithm. Figures [7.12] and [7.13] show the same result but with a smaller value of the mutation probability (0.025). It can be seen from the final error values that too small a value of the probability of mutation results in convergence to non-optimal state (larger value of error). All the simulations results presented above were the average values obtained after 20 runs of the algorithm.

## 7.7 Conclusions

This chapter presented the results in using the annealing approach to adaptive IIR filtering. In particular the classical and fast simulated annealing algorithms were used. Although the fast annealing approach located the optimum set of coefficients faster than the standard algorithm, the number of time samples required for convergence was very large making it an impractical method to use in real world applications. Using concepts from annealing in genetic algorithms as was the case in the proposed hybrid schemes, it was possible to determine when to stop the algorithm. Of the two hybrid schemes proposed, the second hybrid scheme is more promising as the rate of convergence is comparable to that of the standard genetic algorithm while at the

same time providing a stopping criterion for the algorithm. This was an important consequence as with the standard genetic algorithms convergence of a *population* to the global optimum structure was not observed in the simulation experiments carried out in Chapter 6. However the theoretical analysis of the new hybrid schemes is still incomplete and it remains to be proved that the algorithms do converge to the global optimum. For the adaptive IIR filtering case this has been shown to be true using simulation experiments. The hybrid schemes provide for a method of combining the methods of annealing and genetic algorithms. Perhaps more interesting results will be obtained if the annealing principles used above are used in tandem with evolutionary strategies and evolutionary programming. This would remove the problem of discretisation which is present when genetic algorithms are used.

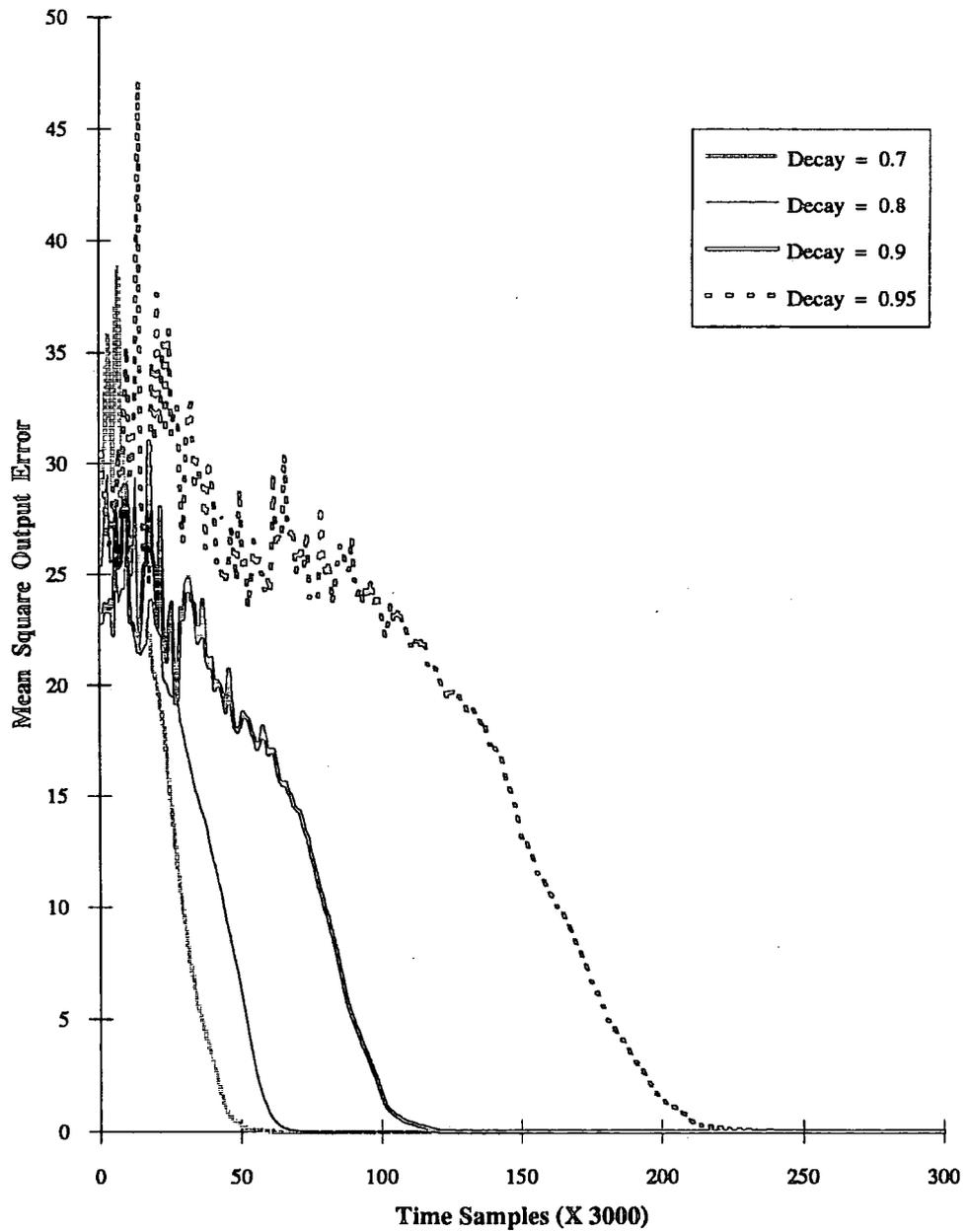


Figure 7.1: Results using Classical Simulated Annealing

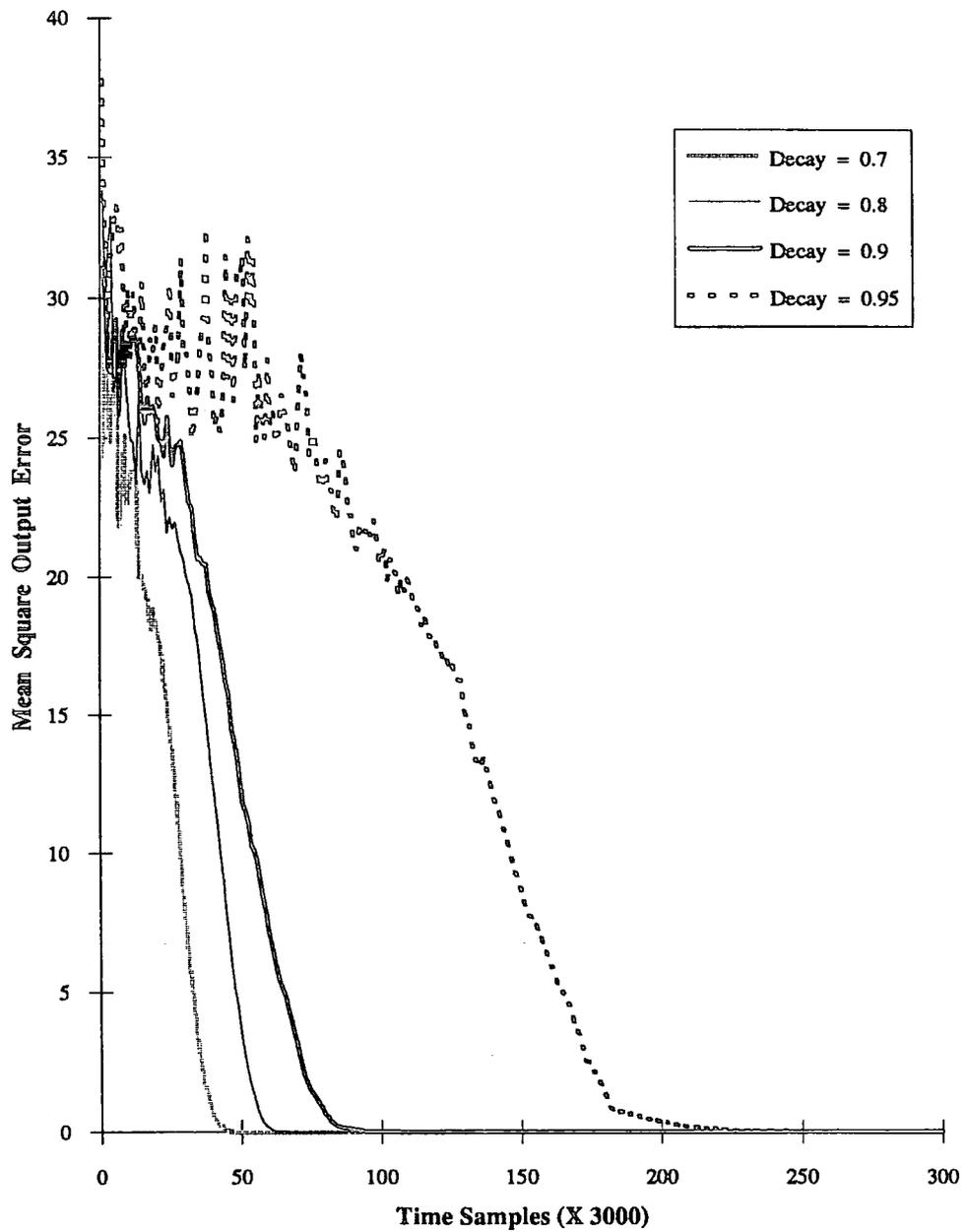


Figure 7.2: Results using Fast Simulated Annealing

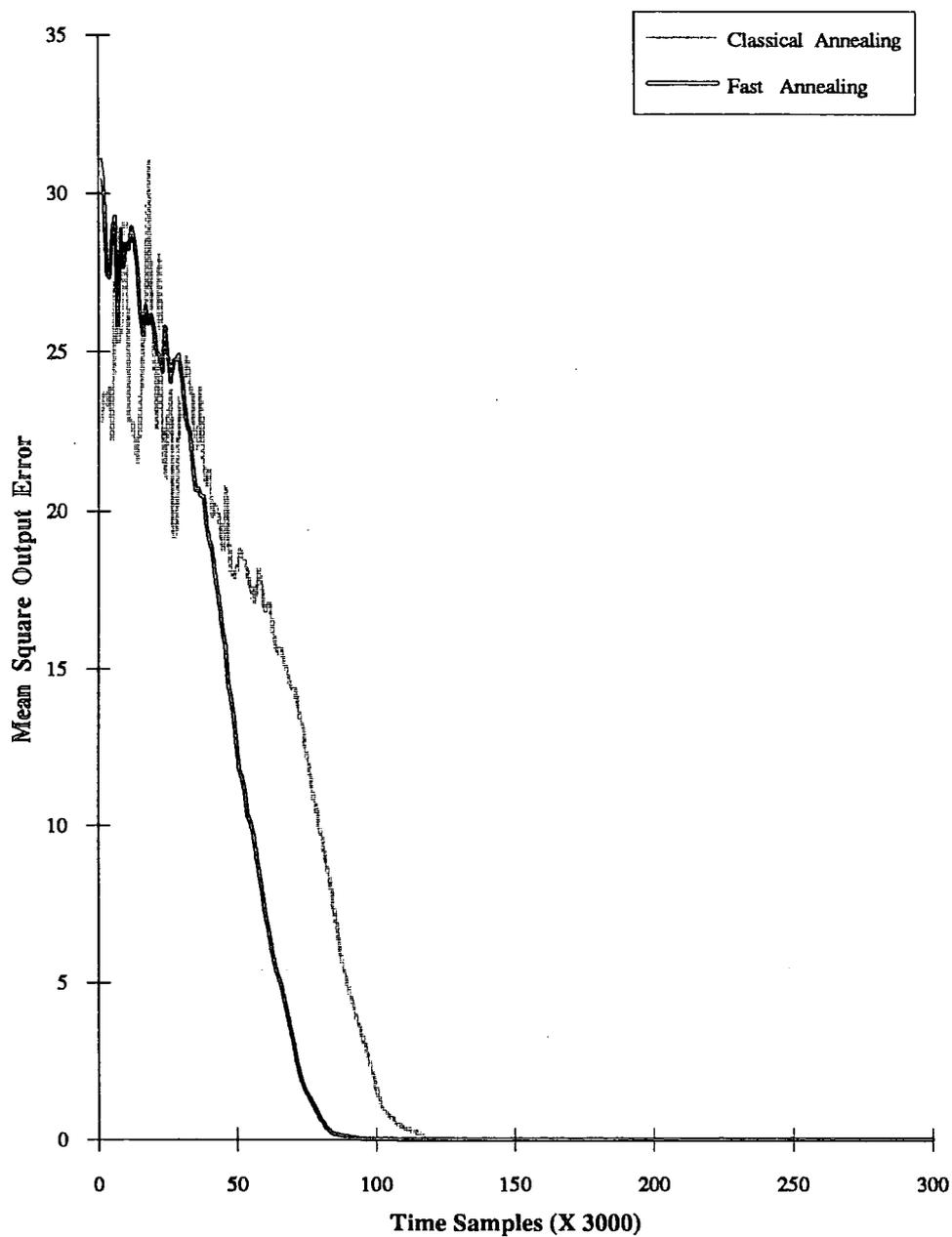


Figure 7.3: Comparative Results using Classical and Fast Simulated Annealing (Decay Parameter = 0.9)

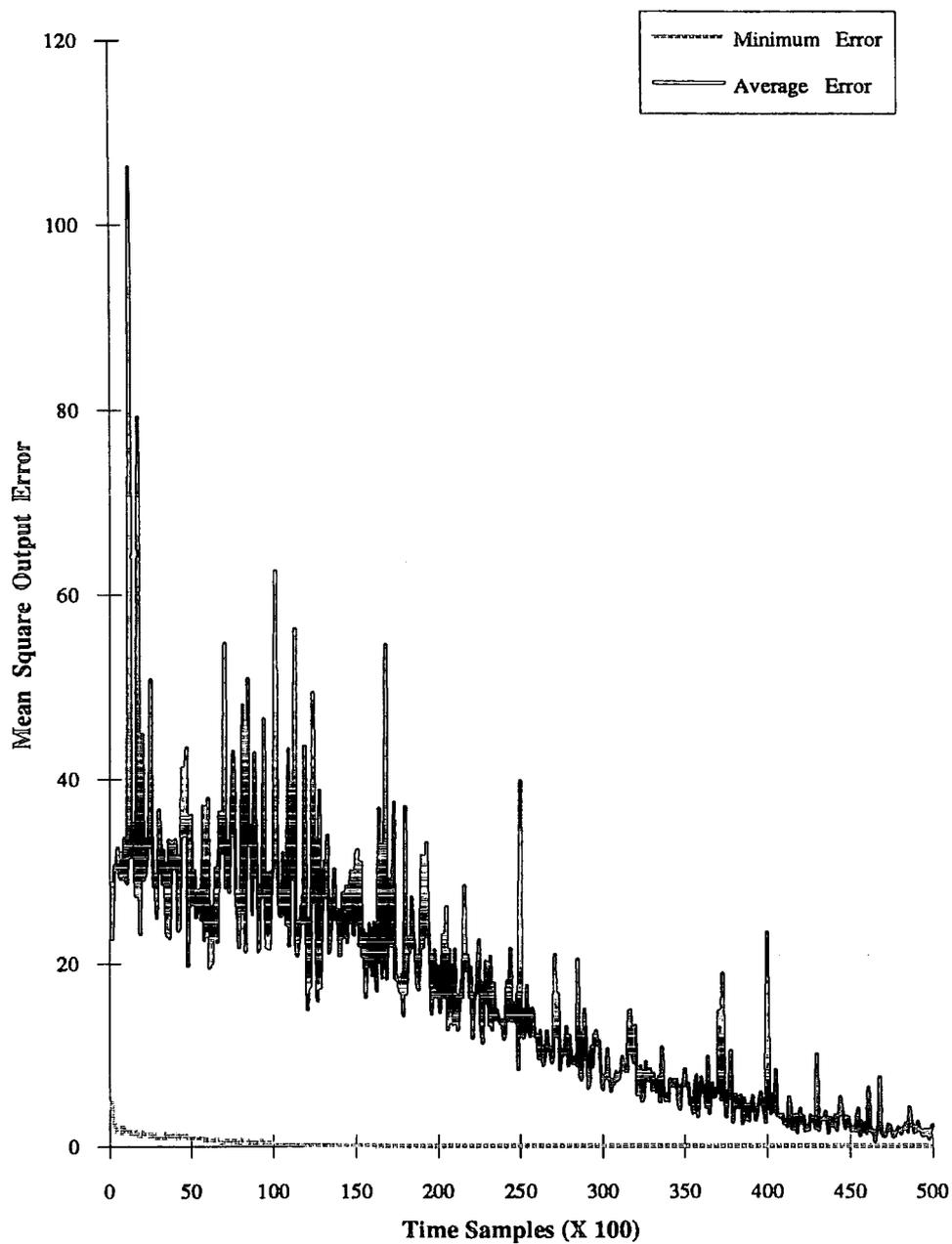


Figure 7.4: Results using Hybrid Scheme - I (Decay Parameter = 100)

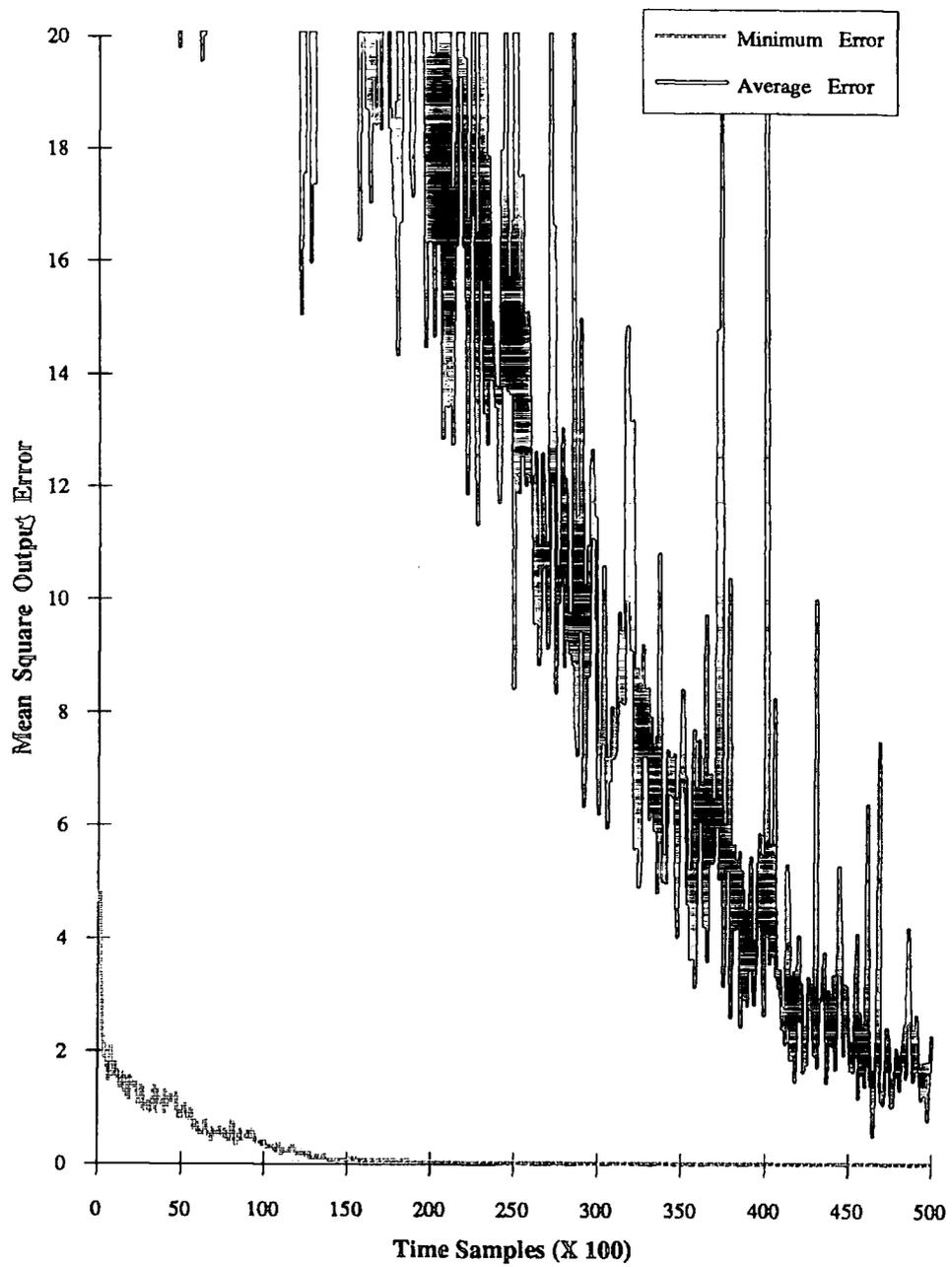


Figure 7.5: Results using Hybrid Scheme - I (Decay Parameter = 100)

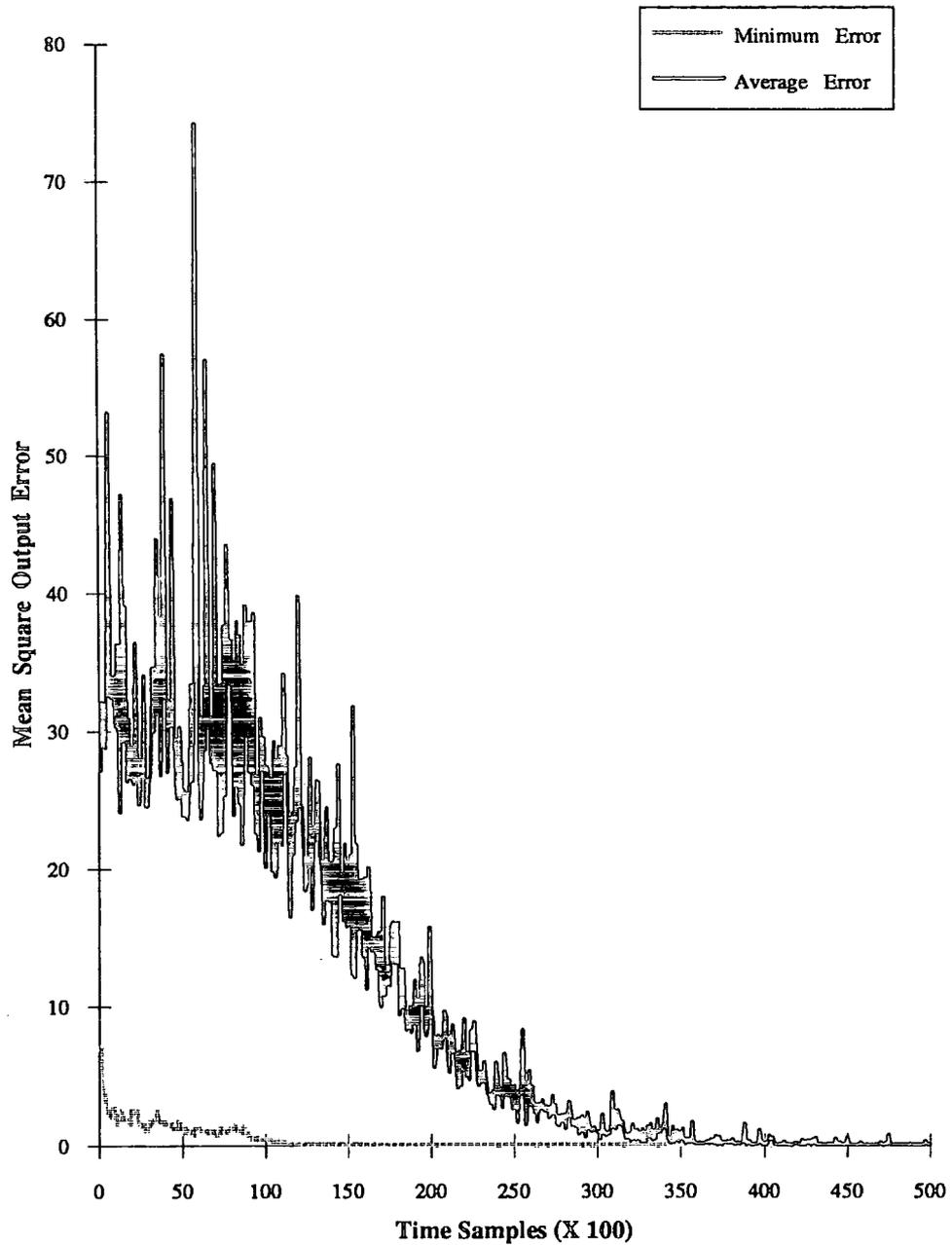


Figure 7.6: Results using Hybrid Scheme - I (Decay Parameter = 50)

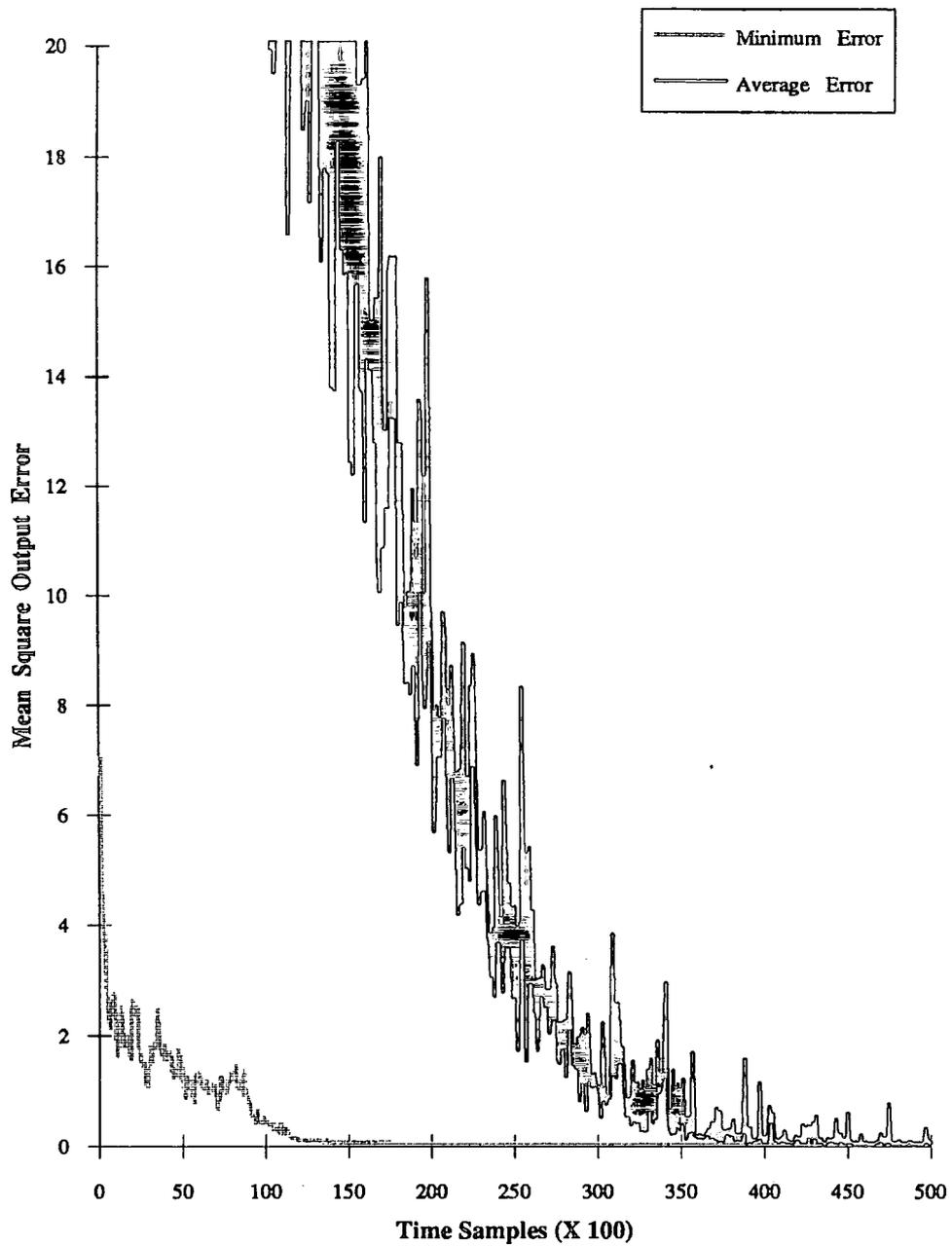


Figure 7.7: Results using Hybrid Scheme - I (Decay Parameter = 50)

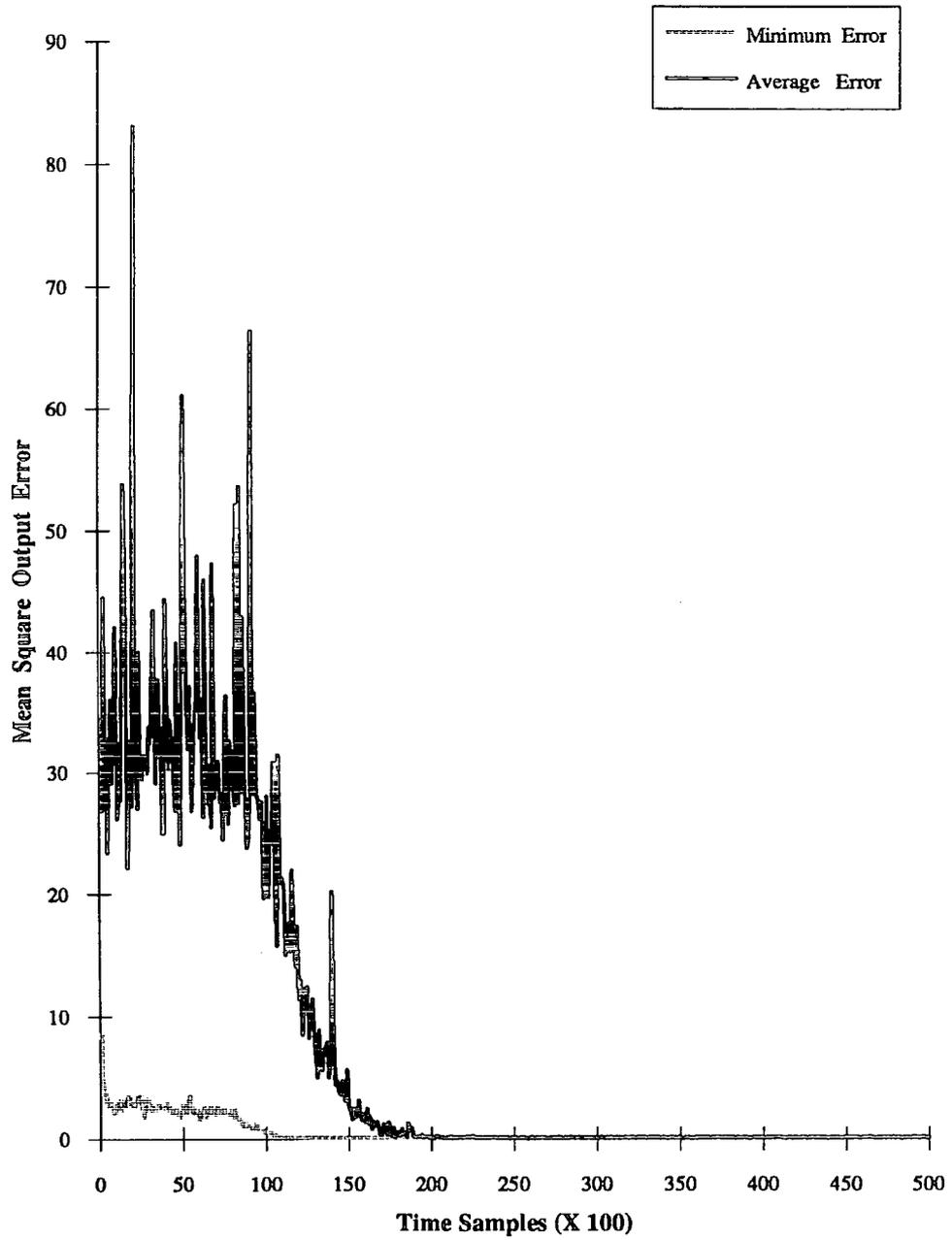


Figure 7.8: Results using Hybrid Scheme - I (Decay Parameter = 15)

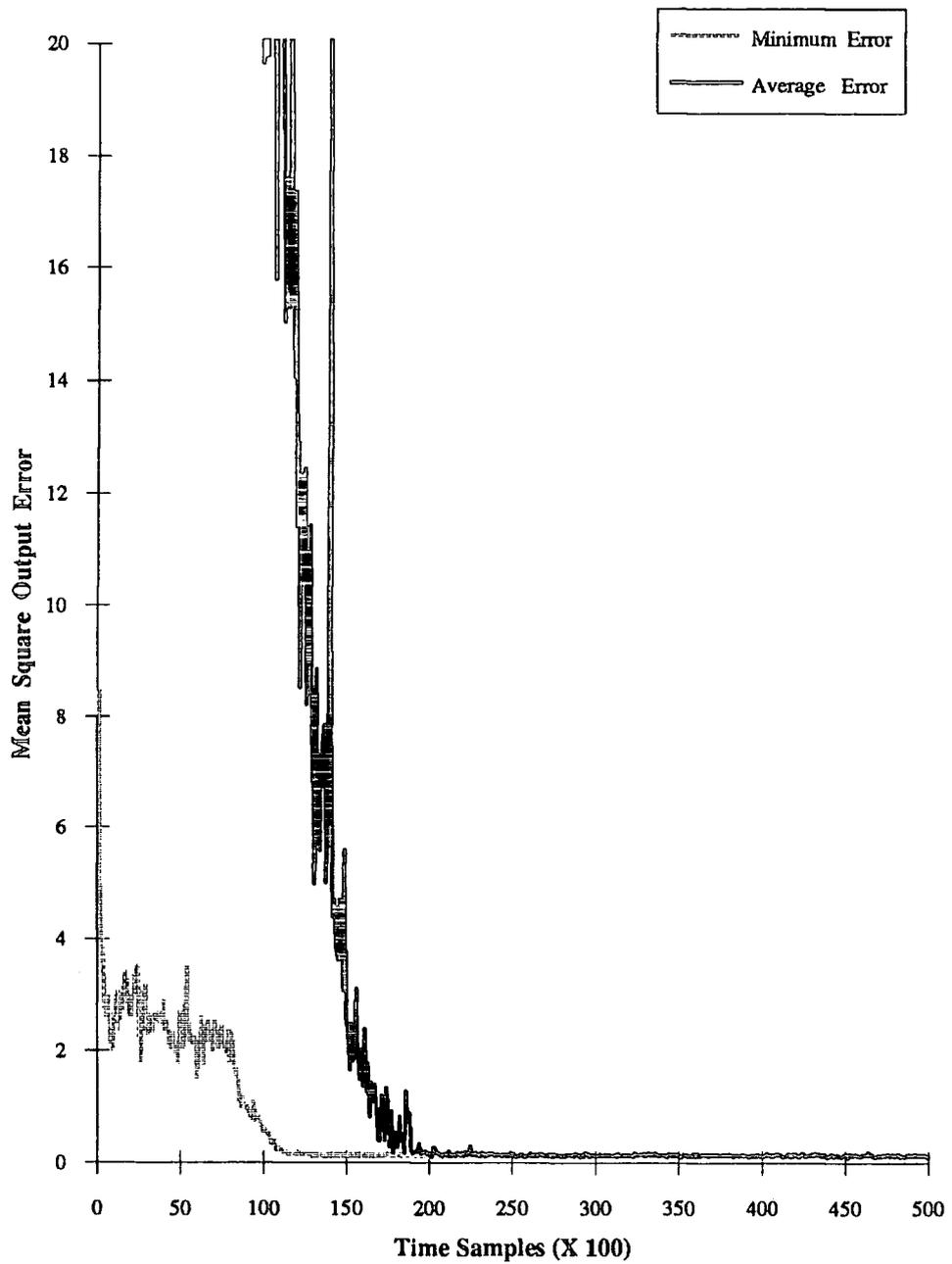


Figure 7.9: Results using Hybrid Scheme - I (Decay Parameter = 15)

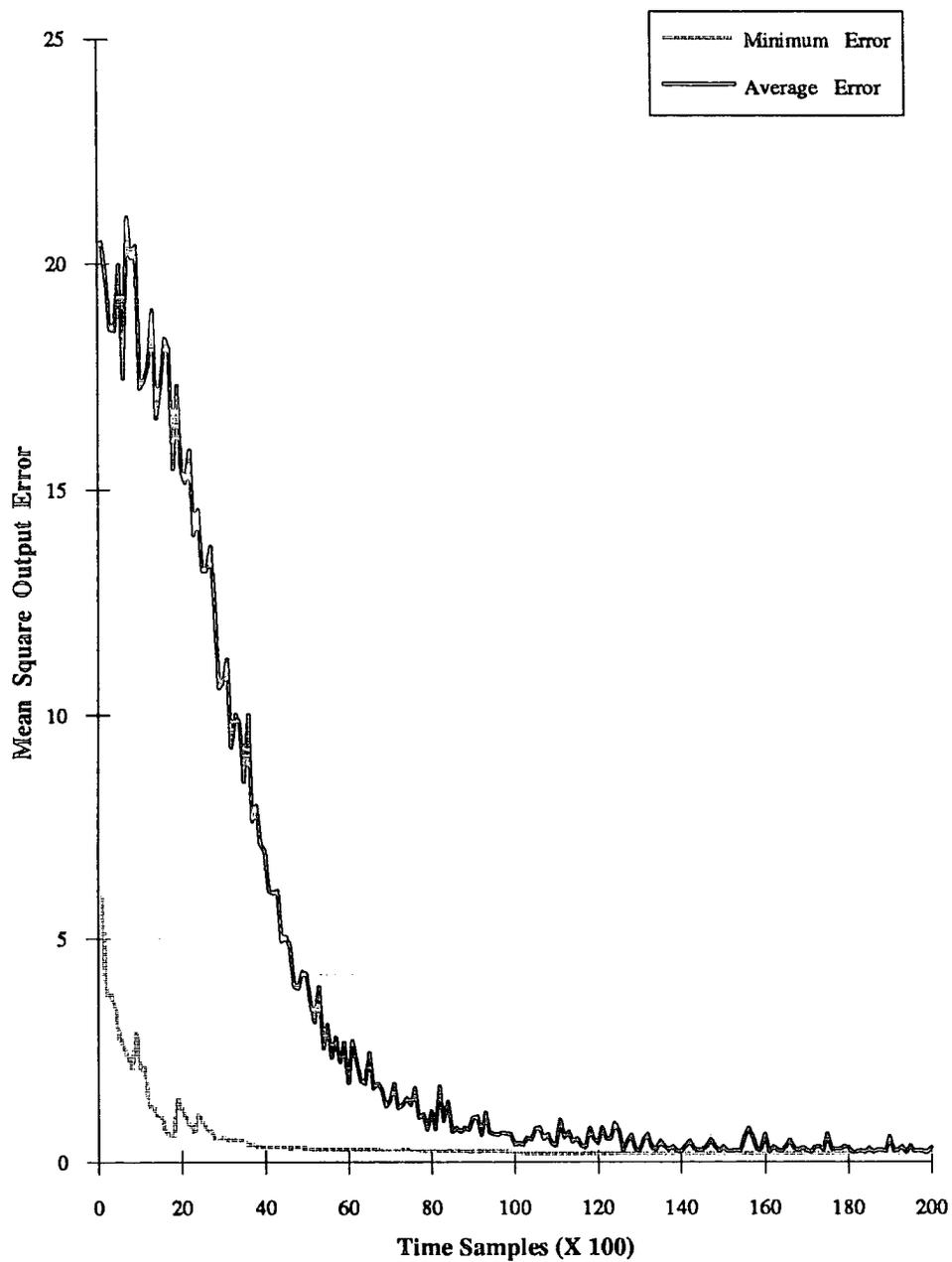


Figure 7.10: Results using Hybrid Scheme - II ( $p_m = 0.075$ , Decay = 0.9)

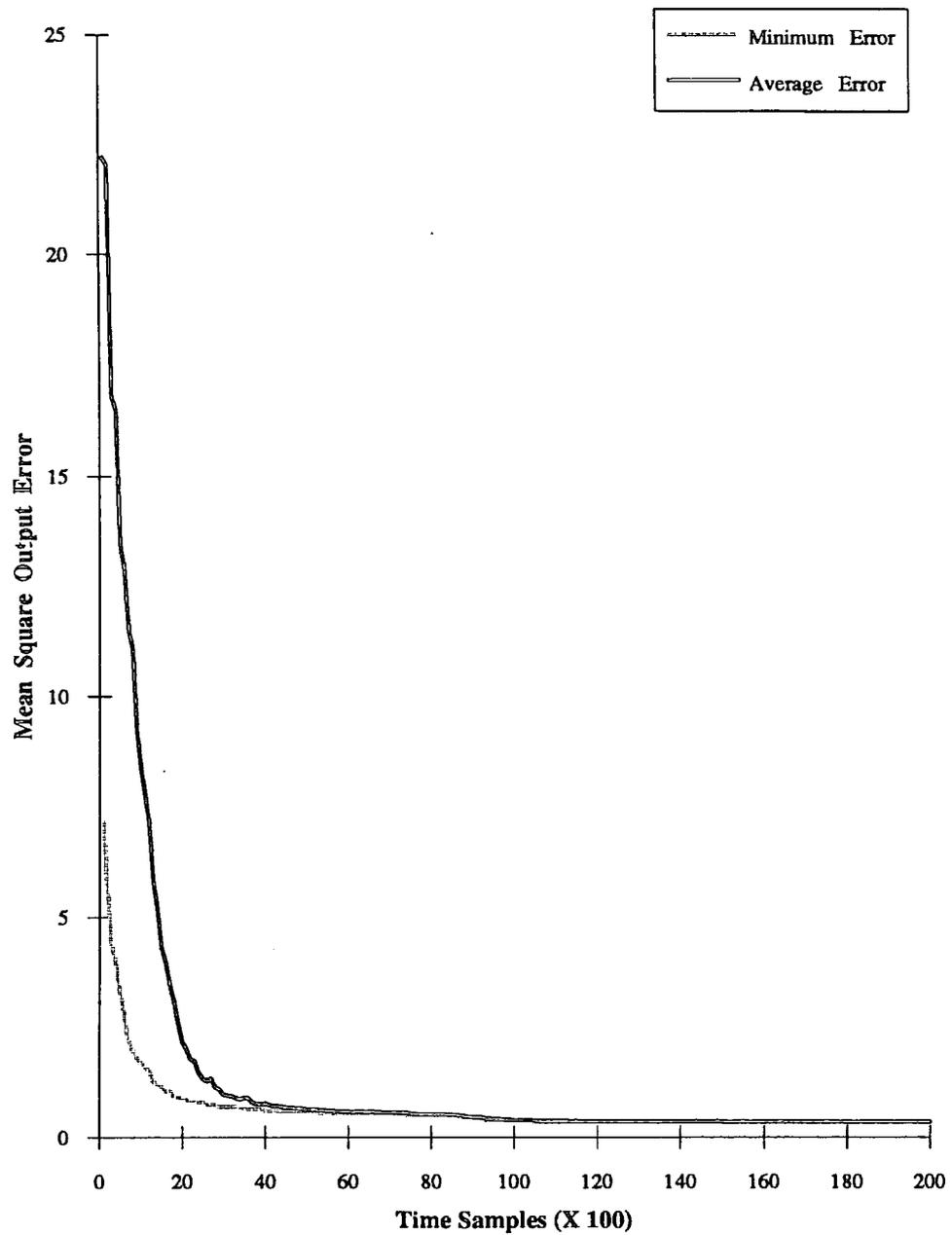


Figure 7.11: Results using Hybrid Scheme - II ( $p_m = 0.075$ , Decay = 0.7)

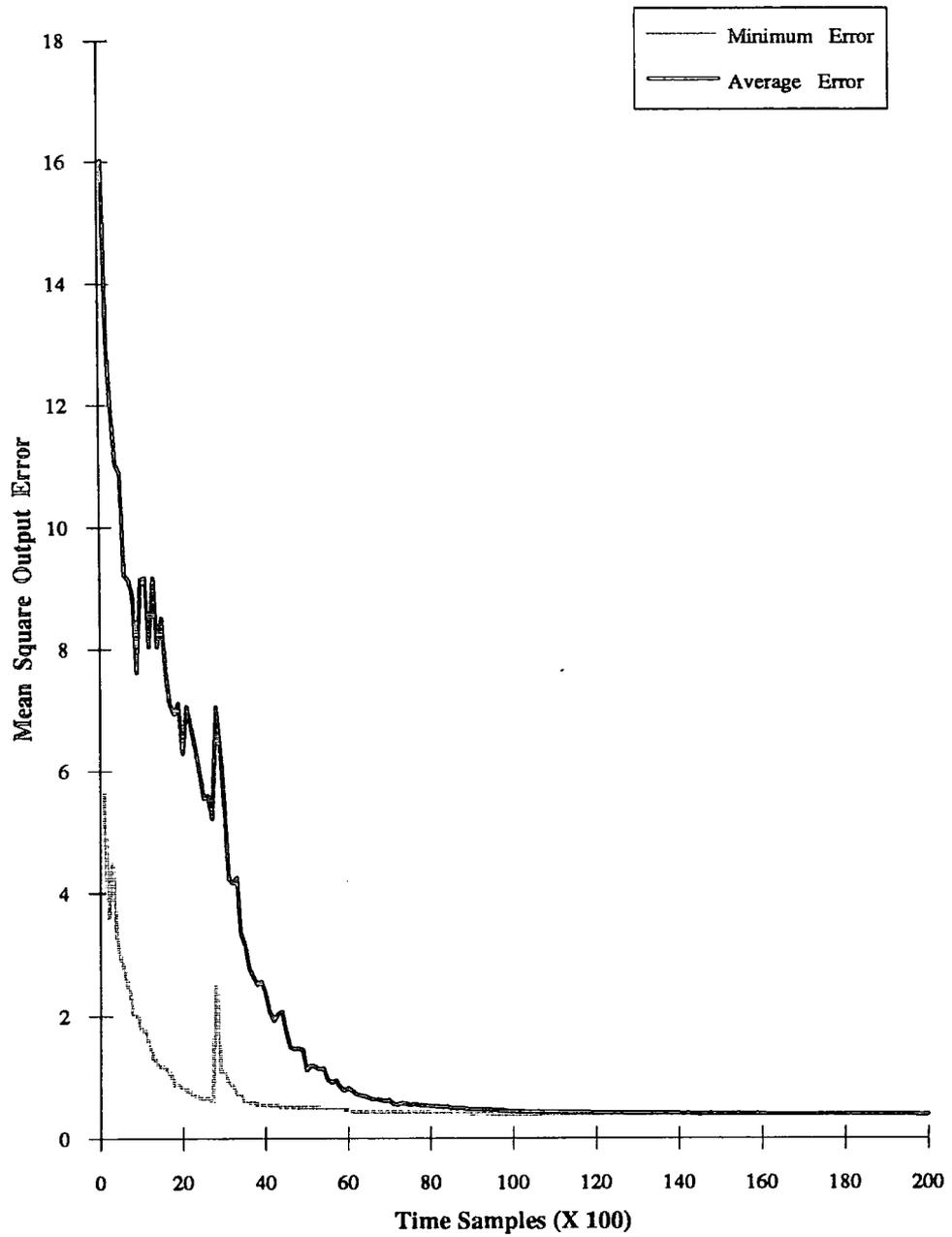


Figure 7.12: Results using Hybrid Scheme - II ( $p_m = 0.025$ , Decay = 0.9)

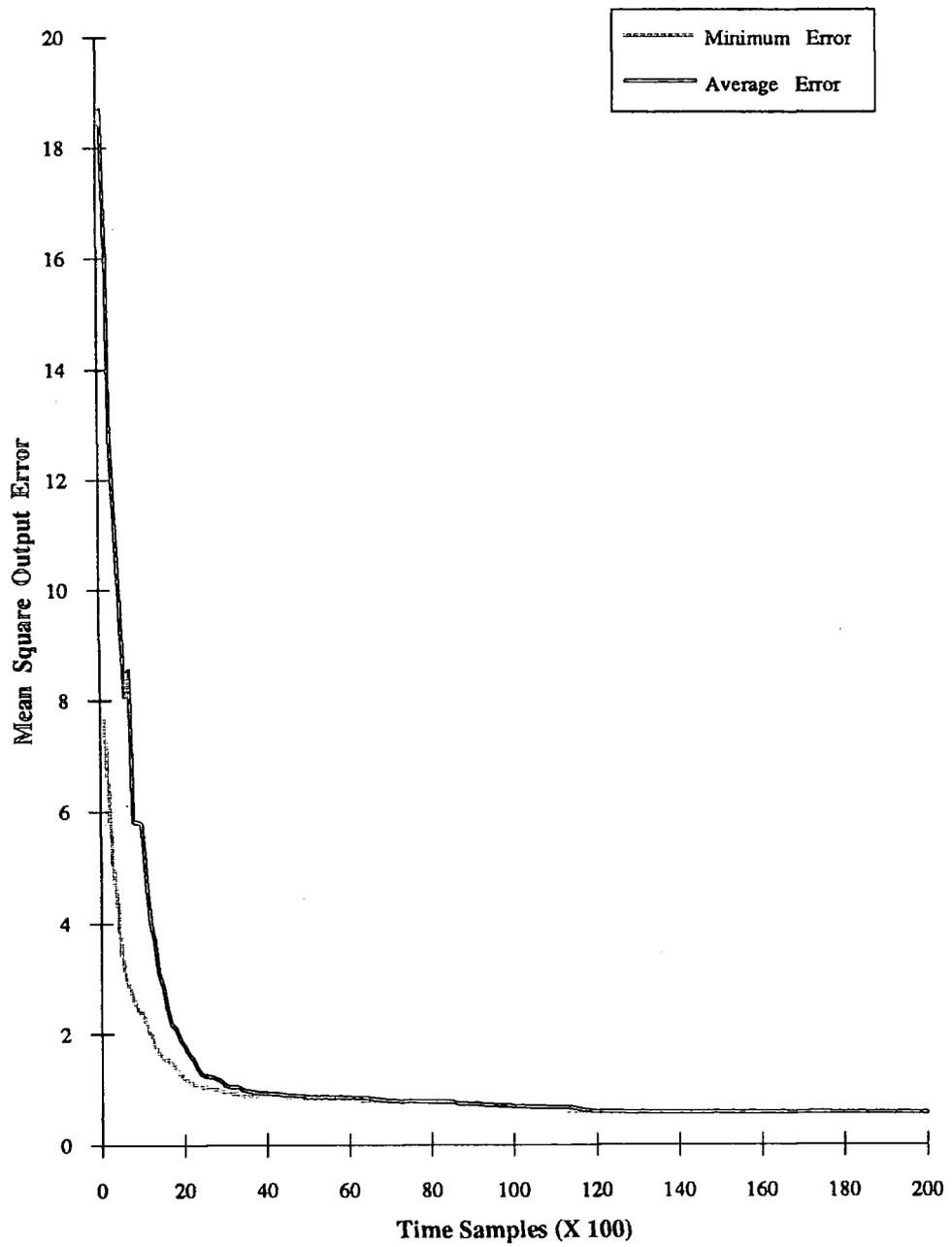


Figure 7.13: Results using Hybrid Scheme - II ( $p_m = 0.025$ , Decay = 0.7)

# Chapter 8

## Conclusions and Further Work

### 8.1 Conclusions

The work undertaken in this thesis can be broadly classified into two main categories:

- o Study of Adaptive IIR filtering algorithms.
- o Application and development of novel optimisation algorithms for use in adaptive IIR filtering.

The initial study of adaptive IIR filtering algorithms indicated that the main problem with current adaptive algorithms for IIR filtering is the inability to deal with multimodal error surfaces. Algorithms which have been designed to tackle this problem work under constrained conditions and are computationally very involved. Thus the potential appeal of using IIR filters to replace FIR filters was lost. This necessitated a different approach to developing adaptive IIR filtering algorithms.

The Stochastic Learning Automata approach was able to overcome the problems of global optimality as has been shown from the simulation results in chapter 4. However this success was achieved only for the case of a second order IIR filter. When adapting higher order IIR filters, two problems were encountered using the SLA approach - firstly the rate of convergence of the algorithm dropped drastically as the method did not scale well with increased number of parameters, and secondly the stability of the adaptive IIR filter especially for the higher order case became an important feature

of the adaptive algorithm. To overcome the problem of dimensionality, the automata games approach was attempted. Although this approach tackled the problems of dimensionality, theoretical results regarding the global optimality of such an approach are not available. Thus the automata games approach may result in a non-optimal solution. The second problem of stability of a high order IIR filter was overcome by using the alternative parallel form realization. Though SLA algorithms are a powerful set of tools, their use for the specific case of adaptive IIR filtering seems rather limited, especially for on-line applications. Another drawback with the SLA approach is that the algorithm forces parameters to take discrete values, combinations of which form the actions of an automata as has been explained in chapter 4. This results in the algorithm obtaining only an approximation to the exact global optimum. However, this fact could be used to construct a hybrid scheme whereby the initial search at a coarse level is carried out using a SLA. Thereafter the results obtained by the SLA may be used as the starting values for established techniques such as gradient descent to locate the exact optimum. Perhaps the more advantageous method would be to combine individual automata into interconnected structures which may be able to model complex functions. This approach would lead to the use of the automata algorithms in neural networks.

The simulated evolutionary approach to optimisation although developed a couple of decades ago, has only recently been used in engineering problems. The main advantage of the method especially for the adaptive IIR filtering case is the ease with which the dimensionality problem is handled. The complete theoretical analysis of the various paradigms of simulated evolution are still forthcoming, though in some case asymptotic convergence proofs are available. These indicate that with a long time frame of reference, the algorithm would be able to locate the global optimum. The use of these algorithms for the adaptive IIR filtering case as shown by the results in the previous chapters is very promising. In particular, genetic algorithms have the potential to be implemented in digital logic as the algorithms mainly operate using binary strings. This would entail real time applications with the genetic algorithms being micro-coded into silicon. However the use of binary strings would entail the necessary discretisation of the parameters and the ensuing loss of accuracy. This

problem can be avoided using the evolutionary strategy (evolutionary programming) approach since these algorithms use the phenotypic representation and thus do not use a coded form of the parameters. Though the adaptive IIR filtering problem has been studied in this thesis, the more general setting for the work would be optimising *stochastic, noisy and multimodal performance surfaces*. This very general setting can be used in a variety of engineering applications. The main drawback with the simulated evolutionary approaches are the dependence of the strategic parameters of the algorithm on the particular problem being solved. Though researchers have attempted to solve this problem by incorporating the parameters themselves as genetic material, more analysis needs to be done to quantify the results obtained so far. The computational time of the simulated evolutionary algorithms when simulated on a sequential machine is large. However, the real power of the method arises in using parallel techniques as each structure of a population could be evaluated at the same time instant.

## 8.2 Further Work

The problem of adaptive IIR filtering which was used in this thesis forms a special case of the more general problem, namely *the optimisation of a noisy, stochastic, multimodal error surface*. The evolutionary schemes have been shown to have significant promise for this problem as shown from the results obtained for the adaptive IIR filtering problem. In the subsequent sections, we present some future areas for research, which seem to hold significant promise for the general problem stated above.

### 8.2.1 Use of Genetic Algorithms in Non-stationary Environments

Non-stationary environments are of significant practical importance as most real world problems have performance surfaces which are not constant but may change values with time. Thus if an unknown system has been identified correctly by a modeling system, the modeling system must be able to track any changes in the unknown

system characteristics. These changes can vary from slow long-term changes to continuously changing performance surfaces. One of the approaches to this problem was to include the parameters of the genetic algorithm as genetic material which undergoes the process of genetic manipulation. An initial study of this approach was attempted and the results have been reported in this thesis. Another approach to non-stationary environments would be to use the concept of *diploidy* and *dominance*. *Diploidy* in genetics refers to the use of a pair of chromosomes which contain information for the same function while *haploid* organisms are composed of a single strand of chromosome which contain information about a particular function. Though diploidy seems to suggest redundancy, it could perhaps be used as a mechanism to take into account the non-stationary characteristics of an environment. For the case of diploid chromosomes, each locus can be occupied by one of the two allele values. This conflict is resolved by use of the *dominance* operator which decides which of the allele value is dominant and which are recessive. The dominant allele value is expressed in the phenotype. The main theories given for diploidy and dominance are that diploidy provides a mechanism to remember past history while dominance protects those previously remembered allele values from a currently unfavourable phase. Thus diploidy and dominance allow for an alternative solution to be held in the background. Although preliminary work on this aspect has been accomplished, more complete analysis and results are still required.

### 8.2.2 Parallel Implementation

Parallel implementations of evolutionary optimisation schemes have received a great deal of interest as their operation make them very suitable for such techniques. As the basic unit of an evolutionary scheme is a population, members of a population can be evaluated in parallel. This method needs to be explored and analysed in greater detail. A possible implementation would be to realise the genetic algorithm using dedicated hardware. This is based on the fact that the main string structures comprising the members of a population are binary in nature for a genetic algorithm and thus all the members may be evaluated in parallel. In the case of evolutionary strategies, if

the recombination parameter is not used, then even the genetic operations may be performed in parallel. Preliminary work on this front has been reported [HB92] where the parallel implementation details of evolutionary algorithms are explained in detail.

### 8.2.3 Genetic Algorithms and Neural Networks

The use of genetic algorithms in neural networks would tie together two schemes which have been inspired by biological systems. The main use of genetic algorithms would be to train the neural networks - i.e. the genetic algorithm is used as the learning scheme. This method has obvious advantages since the genetic algorithm as a stochastic technique does suffer from problems of local minima which established algorithms like back propagation find difficult to overcome. The neural network approach aims at generating a functional, possibly nonlinear, relationship between an input and output set of data. Current results in mating these two techniques have met with modest results and most of the problems which have been solved have been relatively small. A possible area of application would be to use choose problems where gradient information is not available and thus the neural net has to be trained using different approaches. Some encouraging results using this idea has been obtained in reinforcement learning for neurocontrol - this approach has been termed as genetic reinforcement learning [WDD91]. There is considerable scope for research in this area with a lot of challenging control problems which are yet to be fully solved.

### 8.2.4 Theoretical Analysis using Natural Genetics

Theoretical analysis of evolutionary optimisation schemes are still in their infancy and more research is required to give the precise nature of the approach, under what conditions they fail, and the nature of the problems which they are suited to optimise. This perhaps could be achieved by using results which have been derived by researchers who have been working in the area of mathematical genetics. Some of these results relate to global stability of populations and relations between the various parameters to achieve this.

### 8.2.5 Hybrid Schemes

It is the firm conclusion of the author that in future the schemes which likely to provide good results for real world problems are hybrid schemes composed of different optimisation techniques perhaps operating at different stages. One such approach was detailed in thesis where the techniques of annealing and genetic algorithms were used to devise new schemes which had some desirable properties. Use of stochastic learning automata in hierarchical hybrid schemes appears to have promising prospects as the automaton can be used to conduct a coarse level search the results of which can then be used by the algorithms further down the hierarchy to obtain a more accurate and precise result. Theoretical results for the new hybrid schemes which were proposed in Chapter 7 are not yet available and more work is required in order to obtain the global optimality conditions.

# Appendix A

## ALGORITHM: Direct Form Realization to Lattice Form

Suppose

$$H(z) = \frac{\sum_{i=0}^M b_i z^{-i}}{\sum_{j=0}^M a_j z^{-j}} ; a_0^M = 1$$

is the direct form filter. Then the program given below computes the lattice form coefficients  $(k_i, v_i)$  corresponding to the direct form coefficients  $(a_j, b_i)$ .

---

### Program Lattice-Coeff

Begin

$s_M = 1$

For  $i = M$  to 1

Begin

$k_i = -a_i^*$

$v_i = b_i^i$

$s_{i-1} = s_i(1 - k_i)$

For  $j = 1$  to  $(i-1)$

Begin

$a_j^{i-1} = a_j^i + (k_i a_{i-j}^i)/(1 - k_i^2)$

$b_j^{i-1} = b_j^i - (v_i a_{i-j}^i)$

End

$b_0^{i-1} = b_0^i + v_i k_i$

End

$v_0 = b_0^0$

End

The above algorithm calculates the lattice coefficients  $k_i$  and  $v_i$ , given the coefficients of the direct form filter. The coefficients  $k_i$  are referred to as the *reflection coefficients*. The condition for the stability of a lattice filter is that the magnitude of all the reflection coefficients  $k_i$  must be less than unity, i.e.  $k_i < 1 ; \forall i$ . This criteria could be easily incorporated into an adaptive algorithm by restricting the values a particular coefficients can take.

# Appendix B

## Mathematical Model of Simulated Annealing

The simulated annealing algorithm is a mechanism that continuously attempts to transform a current point into one of its neighbouring points. The mathematical model which best describes this process is a *Markov chain*: a sequence of steps where the probability of a move is dependent only on the previous state or move. This is applicable in the case of simulated annealing, as the transitions correspond to a move and that the outcome of a transition is dependent only on the previous state. A Markov chain is usually described by a set of conditional probabilities  $P_{ij}(k, k + 1)$  for every pair of outcomes  $(i, j)$ .  $P_{ij}(k, k + 1)$  describes the probability of reaching the state  $j$  at instant  $(k + 1)$  from state  $i$  at instant  $k$ . Suppose  $\mathbf{X}(k + 1)$  denotes the outcome of the trial at time instant  $(k + 1)$ , then

$$P_{ij}(k, k + 1) = \Pr\{\mathbf{X}(k + 1) = j \mid \mathbf{X}(k) = i\} \quad (\text{B.1})$$

The above Markov chain is said to be *homogeneous* if the conditional probabilities  $P_{ij}$  do not depend on the iteration  $k$ , otherwise it is called *inhomogeneous*.

In case of the simulated annealing algorithm, the probabilities  $P_{ij}$  are referred to as the transition probabilities and the matrix composed of these transition probabilities is called the transition matrix. The transition probabilities define the properties of the algorithm and is a function of the control parameter  $c$ . If the parameter  $c$  is kept constant, then the corresponding Markov chain is homogeneous. This follows from the above definition of homogeneity, as a constant value of the control parameter implies that the transition matrix is not dependent on the iteration index  $k$ . The transition probability  $P_{ij}(c)$  of the simulated annealing process can then be defined

by

$$P_{ij}(c) = \begin{cases} G_{ij}(c) \times H_{ij}(c) & \forall j \neq i \\ 1 - \sum_{l=1, l \neq i}^{|R|} G_{il}(c) \times H_{il}(c) & j = i \end{cases} \quad (\text{B.2})$$

The two matrices  $G_{ij}$  and  $H_{ij}$  are very important with regard to the global optimisation capability and the rate of convergence of the algorithm. The generating probability matrix  $G_{ij}$  is defined by the generating distribution and is used to generate the next point  $j$  by perturbing the current point  $i$ . A Gaussian distribution is usually used for this process. Thus if a parameter  $x$  of the process has a value  $x(k)$  at iteration  $k$ , then at iteration  $(k+1)$  its value is determined by

$$x(k+1) = x(k) + G_{(0,\sigma)}(x) \quad (\text{B.3})$$

where  $G(0, \sigma)$  is a Gaussian distribution with mean value 0 and variance  $\sigma$  i.e.

$$G(x) \approx \exp(-x^2/\sigma^2) \quad (\text{B.4})$$

The variance  $\sigma$  is function of the control parameter  $c$ . The use of the Gaussian distribution has not been always followed in the implementations of simulated annealing where sometimes a uniform distribution has been used [BMU92, Cor87]. The original formulations of the simulated annealing algorithm [KGV83, Cer85] also had used uniform distributions to generate the new points of a sequence. The acceptance probability matrix  $H_{ij}$  is derived from the Metropolis criterion which has been explained before and is given by

$$H_{ij}(k) = \begin{cases} 1 & \text{if } (\Delta E_{ij}) \leq 0 \\ \exp(\frac{\Delta E_{ij}}{c}) & \text{if } (\Delta E_{ij}) > 0 \end{cases} \quad (\text{B.5})$$

where  $\Delta E$  is difference in energies (cost) between the current state and the new state.  $H_{ij}(k)$  is used to decide whether to accept the new point which has been generated using  $G_{ij}(k)$ .

The control parameter plays an important role in rate of convergence and accuracy of the algorithm and is gradually reduced during the course of the algorithm. This

decrement can result in two formulations of the algorithm which are based on the resulting Markov chain:

- o Homogeneous Algorithm: The algorithm is described by a sequence of homogeneous Markov chains. Each Markov chain is generated at a fixed value of the control parameter  $c$ , which is reduced between subsequent Markov chains.
- o Inhomogeneous Algorithm: In this formulation, the algorithm is described by a single inhomogeneous algorithm where the value of  $c$  is continuously reduced between transitions.

# Bibliography

- [Ale86] S.T. Alexander. *Adaptive Signal Processing*. Springer-Verlag, 1986.
- [BE91] T. Boseniuk and W. Ebeling. Boltzmann, Darwin and Haeckel Strategies in complex optimisation. *Lecture Notes in Computer Science*, 496:430–444, 1991.
- [BEA87] T. Boseniuk, W. Ebeling, and Engel A. Boltzmann and Darwin strategies in complex optimisation. *Physics Letters A*, 125:307–310, 1987.
- [Bin78] K. Binder. *Monte Carlo methods in statistical physics*. Springer, New York, 1978.
- [BMU92] N. Benvenuto, M Marchesi, and A. Uncini. Applications of simulated annealing for the design of special digital filters. *IEEE Transactions on Signal Processing*, 40:323–332, 1992.
- [Cer85] V. Cerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Opt. Theory Appl.*, 45:41–51, 1985.
- [CG85] C. Cowan and P. Grant, editors. *Adaptive Filters*. Prentice-Hall, Englewood Cliffs, N.J., 1985.
- [Cor87] A. et al. Corana. Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. *ACM Transactions on Mathematical software*, 13:262–280, 1987.
- [CS69] B. Chandrashekar and D.W.C. Shen. Stochastic automata games. *IEEE Transactions on Systems, Science and Cybernetics*, 5:145–149, 1969.

## BIBLIOGRAPHY

- [CS88] R.A. Caruana and J.D. Schaffer. Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In *Proc. of the 5<sup>th</sup> Int'l Conf. on Machine learning*, Morgan Kaufman Publishing, San Mateo, California, 1988.
- [Dav91] L. Davis (Editor). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.
- [DeJ75] K.A. DeJong. *An analysis of the behaviour of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.
- [DeJ80] K. A. DeJong. Adaptive System Design: A Genetic Approach. *IEEE Transactions of System, Man and Cybernetics*, 10:566-574, September 1980.
- [DS90] K.A. DeJong and W.M. Spears. An analysis of multi-point crossover. In *Proc. of the foundations of Genetic algorithms*, Indiana, 1990.
- [DS91] K.A. DeJong and W.M. Spears. On the virtues of parameterised uniform crossover. In *Proc. of the 4<sup>th</sup> Int'l Conf. on Genetic algorithms*, Morgan Kaufman Publishing, San Mateo, California, 1991.
- [EA86] W. Ebeling and Engel A. Models of evolutionary systems and their applications to optimisation problems. *Syst. Anal. Model. Simul.*, 3:377-385, 1986.
- [EAM86] W. Ebeling, Engel A., and V.G. Mazenko. Modeling selection process with age-dependent birth and death rates. *BioSystems*, 19, 1986.
- [EHC82] D. M. Etter, M.J. Hicks, and K. H. Cho. Recursive adaptive filter design using an adaptive genetic algorithm. In *Proc. of the IEEE Int. Conf. on ASSP*, pages 635-638, 1982.
- [Fei76] P. L. Feintuch. An adaptive recursive LMS filter. *Proceedings of the IEEE*, pages 1622-1624, November 1976.
- [FFA91] D. B. Fogel, L.J. Fogel, and W.J. Atmar. Meta-Evolutionary programming. In *Proc. of the 25<sup>th</sup> Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, California, 1991.

## BIBLIOGRAPHY

- [FJ86] H. Fan and W. K. Jenkins. A new adaptive IIR filter. *IEEE Transactions on Circuits and Systems*, 33:939–947, 1986.
- [FM66] K.S. Fu and G.J. McMurthy. A study of stochastic automata as a model for learning and adaptive controllers. *IEEE Transactions on Automatic Control*, 11:379–387, 1966.
- [FN89] H. Fan and M. Nayeri. On error surfaces of sufficient order adaptive IIR filters: proofs and counter examples to a unimodality conjecture. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37:1436–1442, 1989.
- [Fog62] L. J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.
- [Fog91a] D. B. Fogel. Evolutionary modeling of underwater acoustics. In *Proc. of OCEANS'91*, pages 453–457, 1991.
- [Fog91b] D. B. Fogel. *System Identification through Simulated Evolution: A Machine Learning Approach to Modeling*. Ginn Press, Needham Heights, MA 02194, 1991.
- [FOW66] L.J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through simulated evolution*. John Wiley & Sons, New York, 1966.
- [GDK89] D. Goldberg, K. Deb, and B. Korb. Messy genetic algorithms: Motivation, Analysis and 1<sup>st</sup> Results. *Complex Systems*, 3:493–530, 1989.
- [GDK90] D. Goldberg, K. Deb, and B. Korb. Messy genetic algorithms: Studies in mixed size and scale. *Complex Systems*, 4:415–444, 1990.
- [GG84] S. Geman and D. Geman. Stochastic relaxation, gibbs distribution and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.
- [Gil85] A.M. Gilles. *Machine learning procedures for generating image domain features*. Doctoral Dissertation, University of Michigan, 1985.

## BIBLIOGRAPHY

- [Gol89] D.H. Goldberg. *Genetic Algorithms - In Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, 1989.
- [Goo83] R. P. Gooch. *Equation-Error Approach to Adaptive IIR Filtering*. PhD thesis, Stanford University, 1983.
- [Gre86] John J. Grefenstette. Optimisation of control parameters for genetic algorithms. *IEEE Transactions of System, Man and Cybernetics*, 16:122-128, January 1986.
- [Gri78] L.J. Griffiths. An adaptive lattice structure for noise-canceling applications. In *Proc. IEEE Int. Conf. Acoust., Sp., and Sig. Processing*, pages 87-90, April 1978.
- [Has70] W. Hastings. Monte carlo sampling methods using markov chains and their application. *Biometrika*, 57:97-109, 1970.
- [Hay86] S. Haykin. *Adaptive filter theory*. Prentice-Hall Inc., Englewood Cliffs, N.J., 07632, 1986.
- [HB92] F. Hoffmeister and T. Back. *Genetic algorithms and evolution strategies: Similarities and differences*. Technical Report, No. SYS-1/92, University of Dortmund, Germany, 1992.
- [HM84] M.L. Honig and D.G. Messerschmitt. *Adaptive Filters: Structures, algorithms and applications*. Kluwer Academic, Hingham, MA, 1984.
- [Hol71] R.B. Hollstein. *Artificial genetic adaptation in computer control systems*. PhD thesis, University of Michigan, 1971.
- [Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press, University of Michigan, 1975.
- [Hor76] S. Horvath, Jr. Adaptive IIR digital filters for on-line time-domain equalization and linear prediction. In *IEEE Arden House Workshop on Digital Signal Processing*, Harriman, N.Y., February 1976.

## BIBLIOGRAPHY

- [Ing89] L. Ingber. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Mathematical Computer modeling*, 412:41–51, 1989.
- [JL77] C. R. Johnson, Jr. and M. G. Larimore. Comments on and additions to 'An adaptive recursive LMS filter'. *Proceedings of the IEEE*, 65:1399–1401, September 1977.
- [Joh79] C. R. Johnson, Jr. A convergence proof for a hyperstable adaptive recursive filter. *IEEE Transactions on Information Theory*, 25:745–749, November 1979.
- [Joh84] C. R. Johnson, Jr. Adaptive IIR filtering: Current results and open issues. *IEEE Transactions on Information Theory*, 30:237–250, March 1984.
- [KGV83] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi. Optimisation by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [KT63] V. Yu Krylov and M.L. Tsetlin. Games between automata. *Automat. Telemekh.*, 24:975–987, July 1963.
- [Lak81] S. Lakshmivarahan. *Learning Algorithms: Theory and Applications*. New York: Springer-Verlag, 1981.
- [LN81] S. Lakshmivarahan and K.S. Narendra. Learning algorithms for two-person zero-sum stochastic games with incomplete information. *Mathematics of operations research*, 6:379–386, 1981.
- [LN82] S. Lakshmivarahan and K.S. Narendra. Learning algorithms for two-person zero-sum stochastic games with incomplete information: a unified approach. *SIAM Journal of control and optimisation*, 20:541–552, 1982.
- [LS83] L. Ljung and T. Söderström. *Theory and practise of recursive identification*. MIT Press, Cambridge, M.A., 1983.

## BIBLIOGRAPHY

- [LT72a] S. Lakshmivarahan and M.A.L. Thathatchar. Bayesian learning and reinforcement schemes for stochastic automata. In *Proc. Int. Conf. on Cybernetics and Society*, Washington D.C., October 1972.
- [LT72b] S. Lakshmivarahan and M.A.L. Thathatchar. Optimal non-linear reinforcement schemes for stochastic automata. *Information Sciences*, 4:121-128, 1972.
- [LT73] S. Lakshmivarahan and M.A.L. Thathatchar. Absolutely expedient learning algorithms for stochastic automata. *IEEE Transactions on Systems, Man and Cybernetics*, 3:281-286, May 1973.
- [LT76] S. Lakshmivarahan and M. A. L. Thathatchar. Absolute expediency of q and s-model learning algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 6:222-226, 1976.
- [LTJ80] M. G. Larimore, J. R. Treichler, and C. R. Johnson, Jr. SHARF:an algorithm for adapting IIR digital filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28:428-440, August 1980.
- [Luc66] R. W. Lucky. Techniques for adaptive equalization of digital communication systems. *Bell System Technical Journal*, 45:255-286, 1966.
- [Mas73] L.G. Mason. An optimal learning algorithm for S-model environments. *IEEE Transactions on Automatic Control*, pages 493-496, October 1973.
- [Mea53] N. Metropolis and et. al. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087-1092, 1953.
- [Men73] J M. Mendel. *Discrete Techniques of Parameter Estimation: The Equation Error Formulation*. Marcel Dekker, New York, 1973.
- [MK84] Brian T. Mitchell and Dionysios I. Kountanis. A reorganisation scheme for a hierarchical system learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, 14(2):328-334, March/April 1984.

## BIBLIOGRAPHY

- [MP90] J.R. McDonell and W.C. Page. Mobile robot path planning using evolutionary programming. In *Proc. of the 24<sup>th</sup> Asilomar Conference on Signals, Systems and Computers*, pages 1025–1029, Pacific Grove, CA, 1990.
- [MT89] S. Mukhopadhyay and M.A.L. Thathatchar. Associative learning of boolean functions. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5):1008–1015, September/October 1989.
- [MV78] J. Makhoul and R. Viswanathan. Adaptive lattice methods for linear prediction. In *Proc. IEEE Int. Conf. Acoust., Sp., and Sig. Processing*, pages 83–86, April 1978.
- [NJ89] M. Nayeri and W. K. Jenkins. Alternative realizations to adaptive IIR filters and properties of their performance surfaces. *IEEE Transactions on Circuits and Systems*, 36:485–496, April 1989.
- [NT74] K. S. Narendra and M.A.L. Thathatchar. Learning automata - a survey. *IEEE Transactions on Systems, Man and Cybernetics*, 4(4):323–333, July 1974.
- [NT89] K. S. Narendra and M.A.L. Thathatchar. *Learning Automata - An Introduction*. Prentice-Hall International Inc., 1989.
- [OC88] B.J. Oommen and J.P.R. Christensen.  $\epsilon$ -optimal discretised linear reward-penalty learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, 18(3):451–458, May/June 1988.
- [OH84] B.J. Oommen and E.R. Hansen. The asymptotic optimality of discretised linear reward-inaction learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, 14:542–545, May/June 1984.
- [OL90] John B. Oommen and Kevin J. Lanctôt. Discretised pursuit learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, 20(4):931–938, July/August 1990.

## BIBLIOGRAPHY

- [OM88] B.J. Oommen and D.C.Y. Ma. Deterministic learning automata solutions to the equi-partitioning problem. *IEEE Transactions on Computers*, 37:2-14, January 1988.
- [PA78] D. Parikh and N. Ahmed. On an adaptive algorithm for IIR filters. *Proceedings of the IEEE*, 66:585-588, 1978.
- [PAS80a] D. Parikh, N. Ahmed, and S. D. Stearns. An adaptive lattice algorithm for recursive filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 28:pp. 110-111, February 1980.
- [PAS80b] D. Parikh, N. Ahmed, and S. D. Stearns. An adaptive lattice algorithm for recursive filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28:110-111, February 1980.
- [PM88] John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing*. Macmillan Publishing Company, New York, 1988.
- [Pop73] V. M. Popov. *Hyperstability of Control Systems*. Springer-Verlag, Berlin, 1973.
- [Rec73] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.
- [Reg92] Philip A. Regalia. Stable and efficient lattice algorithms for adaptive IIR filtering. *IEEE Transactions on Signal Processing*, 40:375-388, 1992.
- [Sch75] Hans-Paul Schwefel. *Evolutionsstrategie und numerische Optimierung*. Dissertation, Technische Universität Berlin, 1975.
- [Sch81] Hans-Paul Schwefel. *Numerical optimisation of computer models*. John Wiley, Chichester, 1981.
- [SD88] S. D. Stearns and R. David. *Signal Processing Algorithms*. Prentice-Hall Inc., Englewood Cliffs, NJ 07632, 1988.

## BIBLIOGRAPHY

- [SEA76] S. D. Stearns, G. R. Elliot, and N. Ahmed. On adaptive recursive filtering. In *Proc. 10<sup>th</sup> Asilomar Conf. on Circuits, Systems and Computers*, pages 5–10, Pacific Grove, CA, November 1976.
- [Shy87] John J. Shynk. Performance of alternative adaptive IIR filter realizations. In *Proc. 21<sup>st</sup> Asilomar Conf. on Circuits, Systems and Computers*, pages 144–150, Pacific Grove, CA, November 1987.
- [Shy89a] John J. Shynk. Adaptive IIR filtering. *IEEE ASSP Magazine*, pages 4–21, April 1989.
- [Shy89b] John J. Shynk. Adaptive IIR filtering using parallel form realizations. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37:519–533, April 1989.
- [SK89] Rahul Simha and James F. Kurose. Relative reward strength algorithms for learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, 19(2):388–398, March/April 1989.
- [SN69] I.J. Shapiro and K. S. Narendra. Use of stochastic automata for parameter self-optimization with multimodal performance criteria. *IEEE Transactions on Systems, Man and Cybernetics*, 5:352–360, 1969.
- [Sod75] T. Soderstrom. On the uniqueness of maximum likelihood identification. *Automatica*, 11:193–197, 1975.
- [SR87a] H. Szu and Hartley R. Fast simulated annealing. *Physics Letters A*, 122(3,4):157–162, 1987.
- [SR87b] H. Szu and Hartley R. Nonconvex optimisation by fast simulated annealing. *Proc. of the IEEE*, 75(11):1538–1540, 1987.
- [SS82] T. Söderström and P. Stoica. Some properties of the output error method. *Automatica*, 18:93–99, 1982.
- [Ste81] S. D. Stearns. Error surface of recursive adaptive filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 29:763–766, June 1981.

## BIBLIOGRAPHY

- [Suc91] D. Suckley. Genetic algorithms in the design of FIR filters. *IEE Proc. - (G)*, 138:234–238, April 1991.
- [Sys89] G. Syswerda. Uniform crossover in genetic algorithms. In *Proc. of the 3<sup>rd</sup> Int'l Conf. on Genetic algorithms*, Morgan Kaufman Publishing, San Mateo, California, 1989.
- [TCC87] Y.H. Tam, P.C. Ching, and Y.T. Chan. Adaptive recursive filters in cascade form. *IEE Proc.(F)*, 134:245–252, June 1987.
- [TJL87] J. R. Treichler, C.R. Johnson, Jr., and M.G. Larimore. *Theory and design of adaptive filters*. John Wiley & Sons, New York, 1987.
- [TLJ78] J. R. Treichler, M. G. Larimore, and C. R. Johnson, Jr. Simple adaptive IIR filtering. In *Proc. 1978 Int. Conf. Acoust., Speech, Signal Processing*, pages 118–122, Tulsa, OK, April 1978.
- [TO79] M.A.L. Thathatchar and B.J. Oommen. Discretised reward-inaction learning automata. *Journal of Cybernetics and Information Science*, pages 24–29, Spring 1979.
- [TP89] C.K.K. Tang and Mars P. Intelligent learning algorithms for adaptive digital filters. *Electronic Letters*, 25:1565–1566, 1989.
- [TP91] C.K.K. Tang and Mars P. Stochastic learning automata and adaptive digital filters. *IEE Proc. (F)*, 138(4):331–340, August 1991.
- [TR81] M.A.L. Thathatchar and K.R. Ramakrishnan. A hierarchical system of learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, 11:236–242, 1981.
- [TS85] M.A.L. Thathatchar and P.S. Sastry. A new approach to the design of reinforcement schemes for learning automata. *IEEE Transactions on Systems, Man and Cybernetics*, 15(1):168–175, January/February 1985.

## BIBLIOGRAPHY

- [TS86] M.A.L. Thathatchar and P.S. Sastry. Estimator algorithms for learning automata. In *Proc. of Platinum Jubilee Conference on Systems and Signal Processing, Bangalore, India*, 1986.
- [Tse62] M.L. Tsetlin. On the behaviour of finite automata in random media. *Automation and Remote Control*, 22:1210–1219, 1962.
- [vLA87] P.J.M. van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, 1987.
- [VN70] R. Viswanathan and K.S. Narendra. Expedient and optimal variable-structure stochastic automata. Technical Report CT-31, Dunham Lab., Yale University, New Haven, Conn., April 1970.
- [VN73] R. Viswanathan and K. S. Narendra. Stochastic automata models with applications to learning systems. *IEEE Transactions on Systems, Man and Cybernetics*, pages 107–111, January 1973.
- [VN74] R. Viswanathan and K. S. Narendra. Games of stochastic automata. *IEEE Transactions on Systems, Man and Cybernetics*, 4:131–135, 1974.
- [WDD91] D. Whitley, S. Dominic, and R. Das. Genetic reinforcement learning with multilayer neural networks. In *Proc. of the 4<sup>th</sup> International Conf. on Genetic Algorithms*, Morgan Kaufman Publishing, San Mateo, California, 1991.
- [WH60] B. Widrow and M.E. Hoff, Jr. Adaptive switching circuits. In *IRE WESCON Conv. Rec.*, pages 96–104, 1960.
- [Whi75] S. A. White. An adaptive recursive filter. In *Proc. 9<sup>th</sup> Asilomar Conf. on Circuits, Systems and Computers*, pages 21–25, Pacific Grove, CA, November 1975.
- [WM76] B. Widrow and J.M. McCool. A comparison of adaptive algorithms based on the method of steepest descent and random search. *IEEE Transactions on Antennas and Propagation*, 24:615–637, 1976.

## BIBLIOGRAPHY

- [WS85] B. Widrow and S. D. Stearns. *Adaptive Signal Processing*. Prentice-Hall Inc., Englewood Cliffs, N.J. 07632, 1985.

## Publications

# Genetic and Learning Automata Algorithms for Adaptive Digital Filters

Nambiar R. †

Tang C.K.K. °

Mars P. †

† School of Eng. and Computer Science  
University of Durham  
Durham DH1 3LE, U.K.

° GEC Marconi Research Centre  
West Hanning Field Road, Great Baddock,  
Chelmsford CM2 8HN, U.K.

## Abstract

*This paper details two different approaches to Adaptive Digital Filtering based on Learning Algorithms. The first approach is based on Stochastic Learning Automata where the discretised values of a parameter(s) form the actions of an Learning Automata which then obtains the optimal parameter setting using a suitably defined error function as the feedback from the environment. We detail the use of improved learning schemes published elsewhere and also point out the basic shortcoming of this approach.*

*The second approach is based on Genetic Algorithms. GAs have been used here in the context of multiparameter optimisation. We present simulation results to show how this approach is able to tackle the problems of dimensionality when adapting high-order filters. The effect of the different parameters of a GA on the learning process is also demonstrated. Comparative results between a pure random search algorithm and the GA are also presented.*

## 1 Introduction

The basic task of *Adaptation* in Adaptive Filtering is to determine the optimum setting of parameters defining the system so as to minimise a suitably defined error function. Thus the problem of adaptation can be reduced to a problem in optimisation. Algorithms used for this purpose mainly fall into two main classes: *Gradient Algorithms and Least Square Techniques*. Gradient Algorithms have been widely used in adaptive control but fail when the error function is multimodal. Their performance also deteriorates in the presence of noise and non-stationary environments. Least Square Techniques have faster convergence but are computationally more complex.

This paper gives details of two different approaches to adaptive filtering based on *Learning Algorithms*. After a brief introduction to the problem in section 2, details of the two approaches are presented in section 3 and 4. Simulation results and conclusions are given in section 5.

## 2 Adaptive Filtering

Adaptive Filtering has been used for various applications like adaptive equalisation, adaptive noise-cancelling, adaptive prediction etc.[1]. The system identification configuration has been used in this paper to illustrate the new approaches to adaptive filtering.

In adaptive filtering the adaptive filter used can be of two types: **Adaptive FIR Filter** or **Adaptive IIR Filter**. Algorithms relating to the adaptation of FIR filters are well established. In particular gradient algorithms are very suitable for adaptive FIR filtering as the error surface is quadratic and unimodal with respect to the filter coefficients. But the potential advantages of using an IIR filter in place of a FIR filter has encouraged the study of adaptive IIR filtering, a thorough review of which is presented in [2]. An IIR filter gives a better frequency response and less computational cost than an equivalent FIR filter. But the stability of an IIR filter is an important issue during its adaptation. However the problem which has prompted the use of *Learning Algorithms* in adaptive IIR filtering is: *The error surface in the case of IIR filtering may not be quadratic and unimodal with respect to the filter coefficients and may have multiple optima*. This renders the use of gradient techniques impractical as they could get stuck in a local minima.

When adapting high-order IIR filters, the stability of the filters generated during the adaptation is of vital importance. A method to check the stability is to factorise the denominator polynomial at each stage of the adaptation which is computationally expensive. To overcome this problem, alternative realisations like the parallel and cascade forms have been used to model the direct form filters as given in [3]. The basic sub-system in either the parallel or the cascade configuration is a 2<sup>nd</sup> order filter. This enables the stability check to be built into the algorithm itself by ensuring the coefficients of the 2<sup>nd</sup> order sub-system lie inside the *stability triangle* [2].

## 3 Stochastic Learning Automata

### 3.1 Introduction

Stochastic Learning Automata (SLA) may be defined as an abstract element which interacts with the environment in such a manner so as to improve a specified performance measure. It could be regarded as a finite state machine having a finite set of outputs  $\bar{\alpha} = \{\alpha_1, \dots, \alpha_r\}$ , each of which could be selected with a probability  $\bar{p} = \{p_1, \dots, p_r\}$ . The input set  $\beta$  of the automata could be binary i.e 0,1 [P-model], be finite [Q-model] or continuous between 0 and 1 [S-model]. The automata operates by selecting an action, then using the response from the environment to that action as an input to modify the existing probability vector  $p$ . At stage  $n$  of the learning process we have

$$p(n+1) = T[p(n), \alpha(n), \beta(n)]; \quad 1$$

where  $T$  is the action probability updating rule. More complete details of SLA are given in [4].

The use of SLA in adaptive filtering has been reported in [5]. This follows from an earlier paper where SLA has been used as a optimisation tool for multimodal noisy surfaces [6]. When used to adapt digital filters, the output set of actions of the automata form a set of filter coefficients, each action being regarded as a specific combination of filter coefficients. This is equivalent to the error space being partitioned into a number of hyperspaces, the number of hyperspaces being equal to the number of automata output actions. The environment is represented by the operating environment of the adaptive filter and the mean squared output error is used as a performance criterion. We add to the results already obtained in [5] by using new probability updating algorithms. These include the discretised LRI and the pursuit algorithms.

### 3.2 New Reinforcement Algorithms

#### 3.2.1 Discretised LRI

A general approach for improving the convergence of SLA is by discretising the action probabilities. Theoretical results for the discretised LRI algorithm for a 2-action automata are given in [7]. For the multi-action discrete case simulation results when used in adaptive filtering are presented. The concept of discretisation is achieved by restricting the action probabilities representing the internal state of the automata to a finite set of discrete values in the interval [0,1]. More details of the approach is given in [7].

#### 3.2.2 Pursuit Algorithms

Pursuit algorithms are a simpler subset of a new class of algorithms referred to as *estimator algorithms* introduced by Thathachar and Sastry [8]. As opposed to non-estimator algorithms, *Estimator algorithms* use a running estimate of the probability of reward for each action. Thus

the state vector of the SLA is now increased to include another parameter  $d$ .

Pursuit algorithms are characterised by the fact that the action probability vector pursues the optimal action. The steps of the algorithm are the same as the standard P-model LRI reinforcement but for two changes. Firstly, if an action is rewarded, then the action probability of that action is not necessarily increased, rather the automata increases the probability of the action having the largest estimate of reward. Secondly the algorithm updates the estimate vector  $\bar{d}$  at each iteration, where  $d_i$  is calculated as the ratio of the number of times an action  $i$  is rewarded to the number of times it is selected. More details of the scheme are presented in [8]. Discretised Pursuit Algorithms are the discretised counterparts of the Continuous Pursuit algorithms and were introduced in [9]. The algorithm functions similar to the continuous counterpart except that the action probabilities are discretised.

### 3.3 Simulation Experiments

In using SLA for adaptive IIR filtering the reduced order model given in [10], was used. As a result of the reduced order modeling the error surface is bimodal. The results in Fig. 1 show that the SLA is able to identify the global minimum.

When high-order filters are adapted using the SLA approach, the number of actions of the automata being used as an adaptive controller becomes large decreasing the speed of convergence. Although the automata games approach has been attempted [5], the construction of the game matrix for the high-order filter has proven to be the stumbling block. A new approach based on Genetic Algorithms is proposed which overcomes this limitation.

## 4 Genetic Algorithms

### 4.1 Introduction

Genetic Algorithms (GAs) [11,12] are search techniques which are based on the mechanics of natural selection and genetics involving a structured yet randomised information exchange resulting in the survival of the fittest amongst a population of string structures. GA have been developed by John Holland and his colleagues at the University of Holland.

The basic structure and operation of a GA is as follows: Genetic Algorithms operate on a population of structures which are fixed length strings representing all possible solutions of a problem domain. Using such a representation, an initial population is randomly generated. For each structure (trial solution) in the population, a fitness value is assigned. Each structure is then assigned a probability measure based on the fitness value which decides the contribution a parent solution makes to the new generation. This phase is referred to as the *Reproduction Phase*. Each of the offspring generated by the reproduction phase is then modified using *Genetic Operators*. The

two operators used here are the *Crossover operator* and the *Mutation operator*. In the *crossover operation*, two individual strings are selected randomly for the population. A crossover point is randomly selected to lie between the defining length of the string. The resulting substrings of the two parent strings are swapped resulting in two new strings. The mutation operator generates a new string by independently modifying the values at each loci of an existing string with a probability  $p_m$ . The parameter  $p_m$  is referred to as the probability of mutation. More details of the basic algorithm is given in [12].

## 4.2 Application of GAs in Adaptive Filtering

GAs have been used here for adapting IIR filtering particularly to overcome the problem of dimensionality when adapting high-order filters. An earlier application of GA in adaptive filtering has been reported in [13], and illustrated the viability of the approach. In using GA for adaptive filtering, the system identification configuration has been chosen where the unknown system is an fixed IIR filter while the adaptive system is an adaptive IIR filter whose coefficients are changed by the genetic algorithm.

The genetic algorithm operates with a population of string structures, each string structure in this case being the set of coefficients of the adaptive IIR filter. Each coefficient is coded as a binary string of 4 bits. Thus there are  $16(2^4)$  discrete values a coefficient can take. A mapping procedure is employed which maps the decoded unsigned integer linearly from  $[0, 2^4 - 1]$  to a specified interval  $[P_{min}, P_{max}]$ . For the multi-parameter case, the binary coded forms of all the coefficients are concatenated. This forms the string structure for the individuals of a population. To assign a fitness value to each string structure, the mean-squared-output-error  $e$ , averaged over a suitable window length obtained for that string structure is used. The method of power law scaling [12] has been used wherein the scaled error value is taken as some specified power of the raw error signal. A value of 4 was chosen for the power after extensive simulation experiments. Larger values of the power led to pre-mature convergence while lower values increased the convergence time. In order to convert the maximisation problem to a minimisation problem, an inverting function was used. The actual fitness value  $f$ , assigned to a string  $i$  was

$$f_i = 1/e_i^4 \quad 2$$

where  $e_i$  was as defined above.

## 4.3 Simulation Experiments

The three defining parameters of the GA had the following values :  $n$  (pop. size) = 50 ;  $p_c$  (prob. of crossover) = 0.8 ;  $p_m$  (prob. of mutation) = 0.075. In the simulation experiments, the adaptive filter has been in the form of a parallel bank of  $2^{nd}$  order filters. Thus a  $10^{th}$  order filter was modeled by a parallel bank of 5 second order

filters. Due to constraints on space, the transfer functions of these filters are not presented. All the results show the minimum error obtained after  $n$  generations versus the number of generations. In the simulation experiments performed to check the effect of the various parameters, a  $6^{th}$  order IIR filter was used as a model.

## 5 Results and Conclusions

Fig. 1 shows the result using the new reinforcement algorithms for SLA. The discretised versions of the algorithms are seen to perform better than the continuous counterparts with respect to the convergence time. Fig. 2 shows the result when GAs are used to adapt different order filters. It can be seen that GA have a fast initial learning rate. Figs 3, 4, and 5 show the effect of the different parameters of the GA on the learning rate. The effect of the mutation probability ( Fig. 4) is seen to play a crucial role as too low or too high a value increases the convergence time. Though increasing the population size ( Fig. 3) decreases the convergence time in terms of the number of generations needed, the actual time of computation increases as more time is spent on evaluating a single generation. Fig. 6 shows the comparison between a pure random search algorithm and the GA.

The above results show that GAs are a viable and practical approach in adaptive IIR filtering especially for adapting high-order filters. It is to be noted though that GA cannot locate the exact global optima on account of the discretisation of the parameter space and thus can be used as a first level of search to locate a point close to the global optimum. The optimal setting of the GA parameters is rather heuristic at present and depends heavily on the application on hand.

## 6 References

1. Widrow B., & Stearns S.D., *Adaptive Signal Processing*, Prentice Hall, Englewood Cliffs, 1985,
2. Shynk, J.J., *Adaptive IIR Filtering*, IEEE ASSP Mag., Apr. '89, pp. 4-21,
3. Nayeri M. & Jenkins W.K., *Alternative Realizations to Adaptive IIR Filters and Properties of their Performance Surfaces*, IEEE Trans. on Ckts. and Systems, Vol 36, No. 4, April 1989, pp. 485-496,
4. Narendra K.S & Thathachar M.A.L., *Learning Automata - An Introduction*, Prentice-Hall, Englewood cliffs, 1989,
5. Tang C.K.K & Mars P., *Stochastic Learning Automata and Adaptive Digital Filters*, IEE Proc. F, Vol 138, Aug. 1991, pp. 331-340,
6. Narendra K.S & Shapiro, *Use of Stochastic Automata for Parameter Self-Optimisation with Multi-Modal Performance Criteria*, IEEE Trans. on SMC, Oct. 1969, pp. 352-360,

7. Oomen B. & Hansen, *The asymptotic optimality of discretised LRI Learning Automata*, IEEE SMC, May/June 1984, pp. 542-545,
8. Thathachar M.A.L & Sastry P.S., *A Class of rapidly converging algorithms for Learning Automata*, IEEE Trans. on SMC, Vol 15, Jan. 1985, pp. 168-175,
9. Oomen B. & Lanctot, *Discretised Pursuit Learning Algorithms*, IEEE Trans. on SMC, July/Aug. 1990, pp.931-938,
10. Johnson & Larimore M.G, *Comments and Additions*

11. Holland J.H, *Adaptation in Natural and Artificial Systems*, Ann Arbor, The University of Michigan Press, 1975,
12. Goldberg D.H., *Genetic Algorithms - in Search, Optimisation and Machine Learning*, Addison-Wesley Publishing Comp. Inc., 1989,
13. Etter D.M et. al., *Recursive Adaptive Filter Design using an Adaptive Genetic Algorithm*, Proc. of IEEE Conf. on ASSP, 1982, pp.635-638.

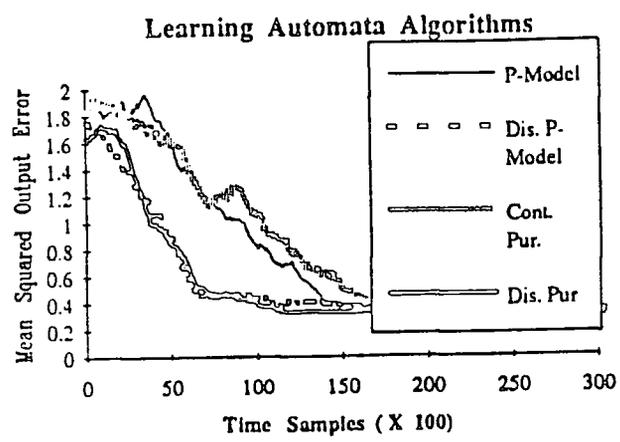


Fig. 1

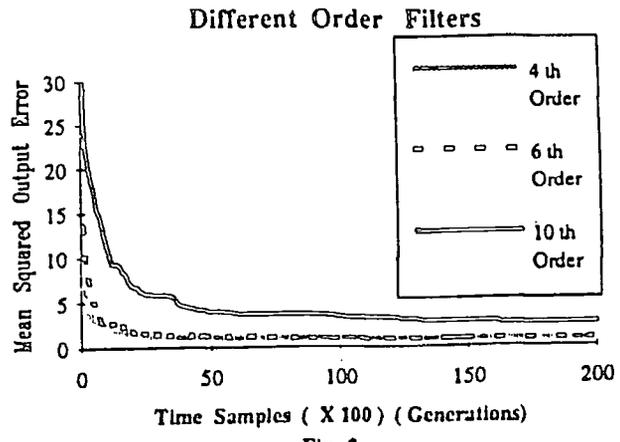


Fig. 2

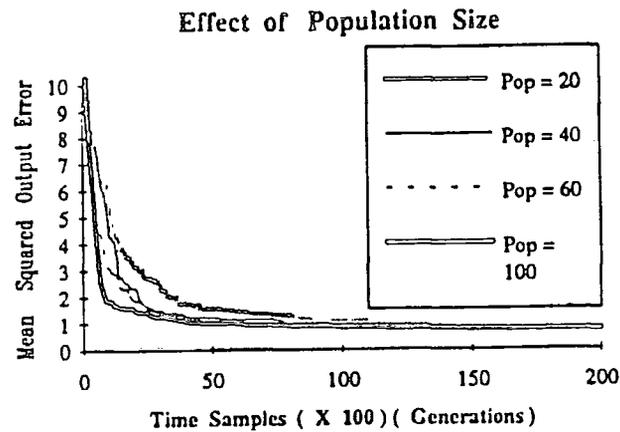


Fig. 3

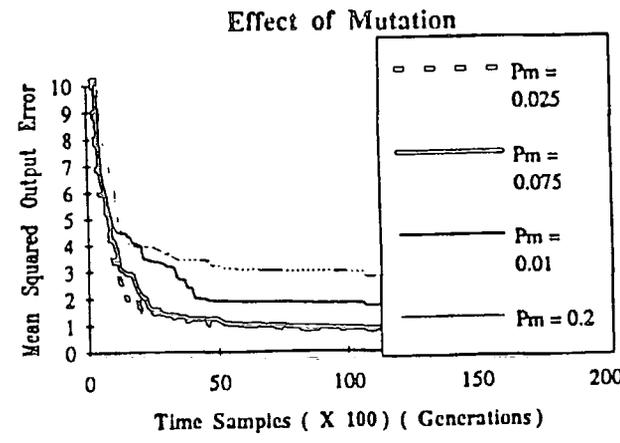


Fig. 4

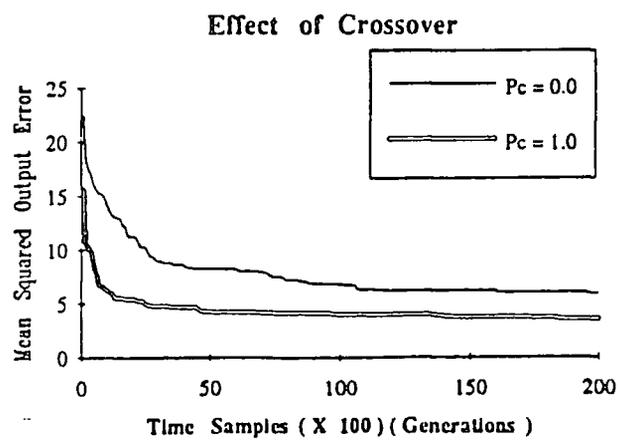


Fig. 5

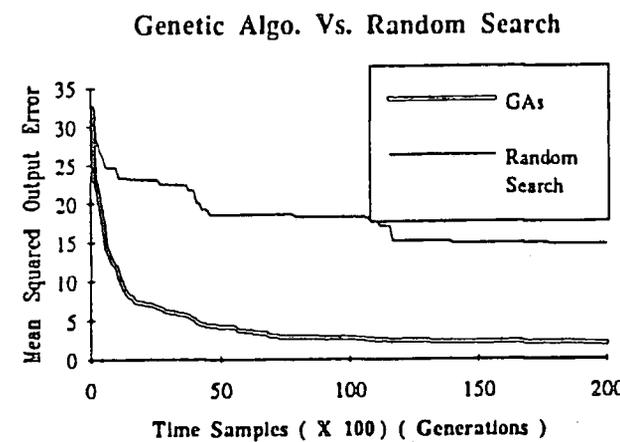


Fig. 6

# Genetic Algorithms for Adaptive Digital Filtering

Nambiar R. & Mars P. †

## Abstract

*This paper details a different approach to Adaptive Digital Filtering based on Genetic Algorithms. Algorithms used in Adaptive Digital Filtering have traditionally been based on the Gradient of the error surface or on Least Square principles. These methods have been found to have inadequacies when adapting IIR filters. The process of adaptation to determine the optimum coefficients can be cast as an optimisation problem wherein a search space is defined and the optimum parameter setting corresponds to the minimum/maximum on the search space. Thus, Genetic Algorithms (GAs), which are search techniques based on randomised techniques, have been used here in the context of multiparameter optimisation. Simulation results are presented to show how this approach is able to tackle the problems of dimensionality when adapting high-order IIR filters. The effect of the different parameters of a GA on the learning process is demonstrated. Comparative results between a pure random search algorithm and the GA are also presented.*

## 1 Introduction

The basic task of *Adaptation* in Adaptive Digital Filtering is to determine the optimum setting of parameters defining the system so as to minimise a suitably defined error function. Thus the problem of adaptation can be reduced to a problem in optimisation. Algorithms used for this purpose mainly fall into two main classes : *Gradient Algorithms* and *Least Square Techniques*. Gradient Algorithms have been widely used in adaptive control but fail when the error function is multimodal. Their performance also deteriorates in the presence of noise and non-stationary environments. Least Square Techniques have faster convergence but are computationally more complex. Thus new approaches based on *Learning Algorithms* were attempted. The use of *Stochastic Learning Automata* (SLA) in adaptive digital filtering has been reported in [1]. But the SLA approach did not give satisfactory results when adapting high-order filters, as the convergence times were very large.

This paper gives details of a different approach to adaptive filtering based on *Genetic Algorithms*. Genetic Algorithms are powerful search techniques which have been developed from principles of natural genetics. After a brief introduction to the problem in section 2, details of the new approach are presented in section 3. Simulation results and conclusions are given in section 4.

## 2 Adaptive Filtering

Adaptive Filtering has been used for various applications such as adaptive equalisation, adaptive noise-cancelling, adaptive prediction etc. [2]. The system identification configuration (Fig. 1) has been used in this paper to illustrate the new approach to adaptive filtering.

In adaptive filtering the adaptive filter used can be of two types : **Adaptive FIR Filter** or **Adaptive IIR Filter**. Algorithms relating to the adaptation of FIR filters are well established. In particular gradient algorithms are very suitable for adaptive FIR filtering as the error surface is quadratic and unimodal with respect to the filter coefficients. But the potential advantages of using an IIR filter in place of a FIR filter has encouraged the study of adaptive IIR filtering, a thorough review of which is presented in [3]. An IIR filter gives a better frequency response and less computational cost than an equivalent FIR filter. The problem which has prompted the use of *Learning Algorithms* in adaptive IIR filtering is: *The error surface in the case of IIR filtering may not be quadratic and unimodal with respect to the filter coefficients and may have multiple optimas*. This renders the use of gradient techniques impractical as they could get stuck in a local minima. The presence of multiple optimas and the conditions when they occur have been investigated in [4].

Another important issue in adaptive filtering is the stability of the filters generated during the adaptation. A method to check the stability is to factorise the denominator polynomial at each stage of the adaptation which is computationally expensive. To overcome this problem, alternative realisations like the parallel and cascade

†School of Engineering and Computer Science, University of Durham, Durham DH1 3LE, U.K.

forms have been used to model the direct form filters as given in [5]. The basic sub-system in either the parallel or the cascade configuration is a  $2^{nd}$  order filter. This enables the stability check to be built into the algorithm itself by ensuring the coefficients of the  $2^{nd}$  order sub-system lie inside the *stability triangle* [2].

### 3 Genetic Algorithms

#### 3.1 Introduction

Genetic Algorithms (GAs) [6,7] are search techniques which are based on the mechanics of natural selection and genetics involving a structured yet randomised information exchange resulting in the survival of the fittest amongst a population of string structures. GAs have been developed by John Holland and his colleagues at the University of Holland.

The basic structure and operation of a GA is as follows: Genetic Algorithms operate on a population of structures which are fixed length strings representing all possible solutions of a problem domain. Though Holland [6] has shown that the binary representation is the best method to form the string structures, there has been increasing evidence that real-valued strings also provide as good a representation. In this paper, the binary representation has been used wherein a parameter is coded as a bit string. Using such a representation, an initial population is randomly generated. For each structure (trial solution) in the population, a fitness value is assigned. Each structure is then assigned a probability measure based on the fitness value which decides the contribution that structure would make to the next generation. This phase is referred to as the *Reproduction Phase*. Each of the offspring generated by the reproduction phase is then modified using *Genetic Operators*. The two operators used here are the *Crossover operator* and the *Mutation operator*. In the *Crossover operation*, two individual strings are selected randomly from the population. A crossover point is randomly selected to lie between the defining length of the string. The resulting substrings of the two parent strings are swapped resulting in two new strings. The parameter governing the crossover operation is the crossover probability  $p_c$ . The *Mutation Operator* generates a new string by independently modifying the values at each location of an existing string with a probability  $p_m$ . The parameter  $p_m$  is referred to as the probability of mutation. Complete details of the algorithm are given in [7].

#### 3.2 Application of GAs in Adaptive Filtering

GAs have been used here for adapting IIR filtering particularly to overcome the problem of dimensionality when adapting high-order filters. An earlier application of GAs in adaptive filtering has been reported in [8], and illustrated the viability of the approach. In using GAs for adaptive filtering, the system identification configuration shown alongside in Fig. 1 has been chosen where the unknown system is a fixed IIR filter while the adaptive system is an adaptive IIR filter whose coefficients are changed by the genetic algorithm.

The genetic algorithm operates with a population of string structures, each string structure in this case being the set of coefficients of the adaptive IIR filter. Each coefficient is coded as a binary string of 4 bits. Thus there are  $16(2^4)$  discrete values a coefficient can take. A mapping procedure is employed which maps the decoded unsigned integer linearly from  $[0, 2^4 - 1]$  to a specified interval  $[P_{min}, P_{max}]$ . For the multi-parameter case, the binary coded forms of all the coefficients are concatenated. This forms the string structure for the individuals of a population. To assign a fitness value to each string structure, the mean-squared-output-error  $e_i$  averaged over a suitable window length obtained for that string structure is used. The length of the window played an important role in the convergence, as too small a window length resulted in convergence to incorrect parameter values. The method of power law scaling [7] has been used wherein the scaled error value is taken as some specified power of the raw error signal. A value of 4 was chosen for the power after extensive simulation experiments. Larger values of the power led to premature convergence while lower values increased the convergence time. In order to convert the maximisation problem to a minimisation problem, an inverting function was used. Thus, the actual fitness value  $f_i$  assigned to a string  $i$  was

$$f_i = 1/e_i^4$$

where  $e_i$  was as defined above.

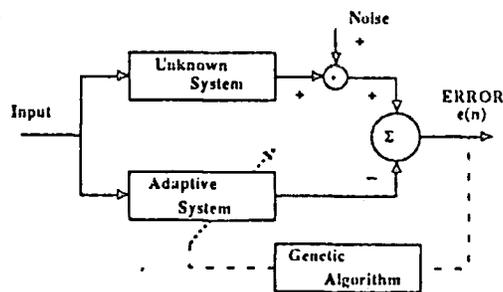


Fig. (1) System Identification Configuration

### 3.3 Simulation Experiments

The three defining parameters of the GA had the following values :  $n$  (pop. size) = 50 ;  $p_c$  (prob. of crossover) = 0.6 ;  $p_m$  (prob. of mutation) = 0.075. These values were obtained after extensive simulation experiments with varying values of the parameters. We also present results showing the effect of different parameter values on the convergence of the algorithm. The first simulation experiment was performed to check whether the GA was capable of locating the global minimum in the presence of local optima. The example used has been reported in [9] and consists of a 2<sup>nd</sup> order IIR filter being identified by 1<sup>st</sup> order IIR filter having a single pole. The transfer function of the 2<sup>nd</sup> order filter is:

$$H(z^{-1}) = \frac{0.05 - 0.4z^{-1}}{1.0 - 1.1314z^{-1} + 0.25z^{-2}} \quad 2$$

while the 1<sup>st</sup> order filter had the transfer function

$$H_a(z^{-1}) = \frac{a}{1 - bz^{-1}} \quad 3$$

The resulting error surface has been shown to be bimodal. The GA approach was successfully able to identify the global optima.

In subsequent simulation experiments to adapt higher order filters, the adaptive filter was in the form of a bank of 2<sup>nd</sup> order filters. Thus a 10<sup>th</sup> order filter was modeled by a bank of five 2<sup>nd</sup> order filters. Due to constraints on space, only the transfer function of the 6<sup>th</sup> order filter is given below. All the results show the minimum error obtained after  $n$  generations versus the number of generations. In the simulation experiments performed to check the effect of the various parameters, a 6<sup>th</sup> order IIR filter was used as a model, the transfer function of which is given below:

$$H(z^{-1}) = \frac{3.0 - 7.5822z^{-1} + 7.9202647z^{-2} - 3.9101332z^{-3} + 0.762588z^{-4}}{1 - 3.7911z^{-1} + 6.3959647z^{-2} - 6.0223078z^{-3} + 3.3151666z^{-4} - 0.99703899z^{-5} + 0.1248048z^{-6}}$$

This filter was adapted by means of a bank of three 2<sup>nd</sup> order filters the transfer functions of which had the form

$$H(z^{-1}) = \frac{1}{1 - a_1z^{-1} - a_2z^{-2}}$$

The stability of the filters during adaptation was achieved by constraining the filter coefficients  $a_1, a_2$  to lie within the stability triangle. Both the parallel and the cascade structures were used.

## 4 Results and Conclusions

Of the different alternative configurations which were used, the parallel form gave the best results. The cascaded form did not converge even after a large number of generations. The reason for this was found to be the propagation of quantisation error through the filter banks, resulting in an erroneous estimate of error for that particular filter. The main reason for using the lattice forms was that the stability check was easily incorporated in the algorithm by constraining the filter parameters to have unity magnitude. However the lattice configurations did not converge even after a large number of generations. As seen from the results, the parallel form gave the best results. The main reason for this was the existence of multiple global minima all of which were equivalent to one another. More details of this result and the results using other alternative configurations are given in [10].

Fig. 2 shows the result when GAs are used to adapt different order filters. It can be seen that GA have a fast initial learning rate. Figs. 3, and 4 show the effect of the different parameters of the GA on the learning rate. The effect of the mutation probability (Fig. 3) is seen to play a crucial role as too low or too high a value increases the convergence time. Though increasing the population size (Fig. 4) decreases the convergence time in terms of the number of generations needed, the actual time of computation increases as more time is spent on evaluating a single generation. Fig. 5 shows the comparison between a pure random search algorithm and the GA.

The above results show that GAs are a viable and practical approach in adaptive IIR filtering especially for adapting high-order filters. It is to be noted though that GA cannot locate the exact global optima on account of the discretisation of the parameter space. However they can be used as a first level of search to locate a point close to the global optimum. The optimal setting of the GA parameters is rather heuristic at present and depends heavily on the application on hand. Current work is incorporating concepts from Simulated Annealing into Genetic Algorithms with the aim of obtaining improved convergence.

## 5 References

1. Tang C.K.K & Mars P., *Stochastic Learning Automata and Adaptive Digital Filters*, IEE Proc. F, Vol 138, Aug. 1991, pp. 331-340,
2. Widrow B. & Stearns S.D., *Adaptive Signal Processing*, Prentice Hall, Englewood Cliffs, 1985,
3. Shynk J.J., *Adaptive IIR Filtering*, IEEE ASSP Mag., April 1989, pp. 4-21,
4. Fan H. & Jenkins W.K., *A New Adaptive IIR Filter*, IEEE Trans., CAS-33, 1986, pp. 939-947,
5. Nayeri M. & Jenkins W.K., *Alternative Realisations to Adaptive IIR Filters and Properties of their Performance Surfaces*, IEEE Trans., CAS-36, No. 4, April 1989, pp. 485-496,
6. Holland J.H., *Adaptation in Natural and Artificial Systems*, Ann Arbor, The University of Michigan Press, 1975,
7. Goldberg D.H., *Genetic Algorithms - in Search, Optimisation and Machine Learning*, Addison-Wesley Publishing Comp. Inc., 1989,
8. Etter D.M et al., *Recursive Adaptive Filter Design using an Adaptive Genetic Algorithm*, Proc. of IEEE Conf. on ASSP, 1982, pp. 635-638.
9. Johnson C.R. Jr. & Larimore M.G., *Comments and Additions to " An Adaptive Recursive LMS Filter"*, Proc. of IEEE, Sept. 1977, pp. 1399-1401,
10. Nambiar R., *Genetic Algorithms and Adaptive Digital Filtering*, Internal Report, Sept. 1991, School of Eng. & Comp. Science, University of Durham.

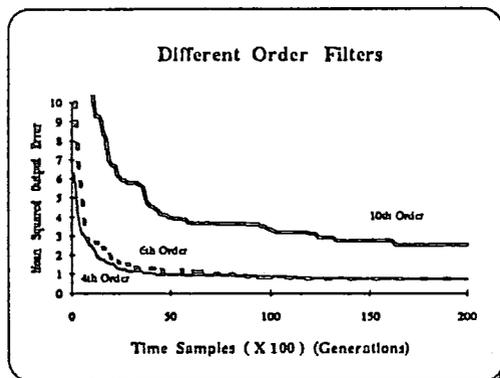


Fig. 2

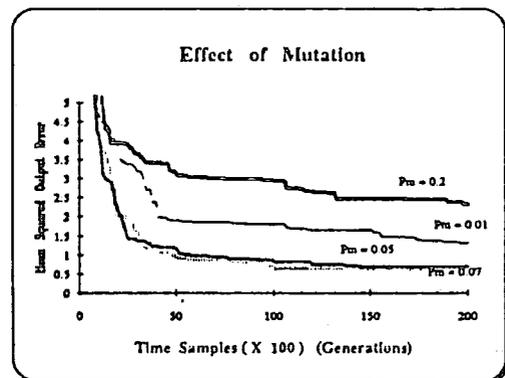


Fig. 3

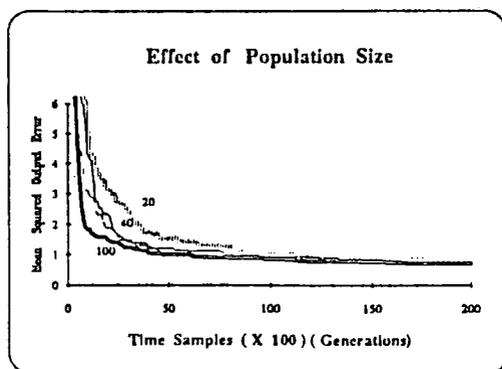


Fig. 4

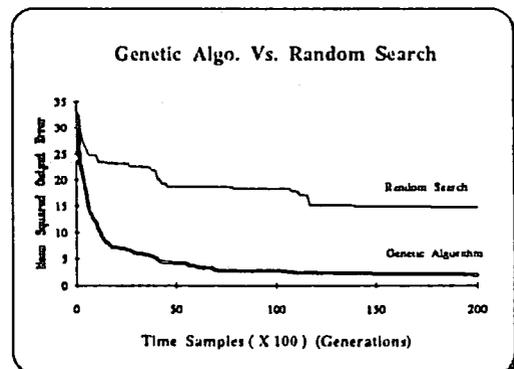


Fig. 5

# Genetic and Annealing Approaches to Adaptive Digital Filtering

R. Nambiar

P. Mars

School of Engineering and Computer Science  
University of Durham  
Durham DH1 3LE, U.K.

## Abstract

*Novel approaches to Adaptive Digital Filtering based on Genetic Algorithms and Simulated Annealing are proposed in this paper. Algorithms used in Adaptive Digital Filtering are usually based on using the Gradient of the Mean Square Error or on Least Square principles. These methods have been found to have inadequacies when adapting IIR filters. The process of adaptation to determine the optimum coefficients can be cast as an optimisation problem. Genetic Algorithms (GAs) and Simulated Annealing (SA), which are search techniques based on randomised techniques, have been used here in the context of multiparameter optimisation. Simulation results are presented which show how these approaches are able to tackle the problems of global optimality and dimensionality when adapting high-order IIR filters. New hybrid schemes where concepts of SA are incorporated into GAs are proposed.*

## 1 Introduction

The core problem in Adaptive Digital Filtering is to determine the optimum setting of parameters of the adaptive filter so as to minimise a suitably defined error function. Thus the problem of adaptation can be reduced to a problem in optimisation. Algorithms used for this purpose mainly fall into two main classes: *Gradient Algorithms and Least Square Techniques*. Gradient Algorithms have been widely used in adaptive control and filtering but fail when the error function is multimodal. Their performance also deteriorates in the presence of noise and non-stationary environments. Least Square Techniques have faster convergence but are computationally more complex [1].

This paper details different approaches to adaptive filtering based on Genetic Algorithms and Simulated Annealing. Novel hybrid schemes incorporating concepts from both these methods are proposed in this paper. After a brief introduction to the problem

in section 2, details of these approaches are presented in section 3, 4 and 5. Simulation results and conclusions are given in section 6 and 7.

## 2 Adaptive Filtering

In adaptive filtering the adaptive filter used can be of two types: Adaptive FIR Filter or Adaptive IIR Filter. Adaptive FIR filter algorithms have been well analysed and established. In particular gradient algorithms are very suitable for adaptive FIR filtering as the error surface is quadratic and unimodal with respect to the filter coefficients. But the potential advantages of using an IIR filter in place of a FIR filter has encouraged the study of adaptive IIR filtering, a comprehensive review of which is presented in [2]. The problem which has prompted the use of new techniques in adaptive IIR filtering is: *The error surface in the case of IIR filtering may not be quadratic with respect to the filter coefficients and thus may be multimodal*. This renders the use of gradient techniques impractical as they could get stuck in a local minima. The presence of local optimas and the conditions when they occur have been investigated in [3, 4]. In using these new approaches for adaptive filtering, the system identification configuration has been chosen where the unknown system is a fixed IIR filter while the adaptive system is an adaptive IIR filter whose coefficients are modified by the algorithm being used.

An important issue in adaptive IIR filtering is the stability of the filters generated during the adaptation. A method to check the stability is to factorise the denominator polynomial at each stage of the adaptation which is computationally expensive. To overcome this problem, alternative realisations like the parallel and cascade forms have been used to model the direct form filters as given in [5]. The basic sub-system in either the parallel or the cascade configuration is a  $2^{\text{nd}}$  order

filter. This enables the stability check to be built into the algorithm itself by ensuring the coefficients of the  $2^{\text{nd}}$  order sub-system lie inside the *stability triangle* [2]. Use of the parallel or cascade form may result in an error surface that has multiple global optimas [5].

### 3 Genetic Algorithms

Genetic Algorithms (GAs) [6, 7] are search techniques which are based on the mechanics of natural selection and genetics, involving a structured yet randomised information exchange resulting in the survival of the fittest amongst a population of string structures. GAs have been developed by John Holland and his colleagues at the University of Michigan.

The basic structure and operation of a GA is as follows: Genetic Algorithms operate on a population of structures which are fixed length strings representing all possible solutions of a problem domain. In this paper, the binary representation has been used wherein a parameter is coded as a bit string. Using such a representation, an initial population is randomly generated. For each structure (trial solution) in the population, a fitness value is assigned. Each structure is then assigned a probability measure based on the fitness value which decides the contribution that structure would make to the next generation. This phase is referred to as the *Reproduction Phase*. Each of the offspring generated by the reproduction phase is then modified using genetic operators of *Crossover* and *Mutation*. In the *Crossover operation*, sub-strings of two individual strings selected randomly from the population are swapped resulting in two new strings. The parameter governing the crossover operation is the crossover probability  $p_c$ . The *Mutation Operator* generates a new string by independently modifying the values at each location of an existing string with a probability  $p_m$  which is referred to as the probability of mutation. Further details of the algorithm are given in [7].

GAs have been used here for adapting IIR filtering particularly to overcome the problem of dimensionality when adapting high-order filters. An earlier application of GAs in adaptive filtering has been reported in [8, 9], and illustrated the viability of the approach. The string structure in this application is the set of coefficients of the adaptive IIR filter coded as a binary string of  $N$  bits. Thus there are  $(2^N)$  discrete values a coefficient can take. A mapping procedure is employed which maps the decoded unsigned integer linearly from  $[0, 2^N - 1]$  to a specified interval  $[P_{\min}, P_{\max}]$ . For the multi-parameter case (Higher

order IIR filter), the binary coded forms of all the coefficients are concatenated. This forms the string structure for the individuals of a population. To assign a fitness value to each string structure  $i$ , the Mean Squared Output Error (MSOE)  $e_i$  averaged over a suitable window length obtained for that string structure is used. The method of power law scaling [7] has been used wherein the scaled error value is taken as some specified power of the raw error signal. The maximisation problem was converted to a minimisation problem by using an inverting function. Thus, the actual fitness value  $f_i$  assigned to a string  $i$  was

$$f_i = 1/e_i^4 \quad (1)$$

where  $e_i$  was as defined above.

### 4 Simulated Annealing

One of the newer techniques for optimisation especially for multimodal functions is that of Simulated Annealing which was proposed in [10]. The method determines the optimal point of a cost function by simulating the annealing process of a metal, allowing probabilistic uphill moves thereby locating the global optimum. The cost function in the annealing process is usually the free energy of the system and the probabilistic uphill moves are determined by the temperature of the system. The process starts with high values of the temperature which allow more uphill moves thereby ensuring an efficient search of the search space. As the temperature is gradually reduced, the process probabilistically converges to the global optimum. Complete details of the method are given in [11].

The main drawback of the SA is that convergence to the global optimum is assured only asymptotically leading to very long convergence time, thus making it impractical to use in real world problems. To speed up the convergence to the algorithm, Szu and Hartley [12] proposed the following modification: The standard SA algorithm makes use of a *Gaussian distribution* as a generating function to search the neighbourhood of the current point. Szu and Hartley proposed the use of the infinite variance *Cauchy distribution* which has a wider tail than the *Gaussian distribution* thus permitting occasional long steps while searching the local neighbourhood. This method has theoretically been proved to have faster convergence [12] [proportional to  $1/(t)$ ] as opposed to the standard SA [proportional to  $1/(\log t)$ ], where  $t$  is the time parameter.

To use the above techniques for adaptive IIR filtering, the parallel form realization has been used in this

paper, where the adaptive filter is made up of parallel sections of second order IIR filters. Thus the stability of the filter structure can be ensured by restricting the parameters of the second order filter to lie within the stability triangle [2]. A typical second section is adapted in the following manner: Suppose the two parameters of the section are  $a, b$  and the current values are  $a_{cur}, b_{cur}$ . The new values of the parameters are then generated as follows:

$$\begin{aligned} a_{new} &= a_{cur} + r * step \\ b_{new} &= b_{cur} + r * step \end{aligned} \quad (2)$$

where  $r$  depends on the distribution being used and  $step$  determines the step-size of the search. If the new values of  $a, b$  lie outside the stability triangle, then the values are discarded and Equ. [2] is used again.

## 5 Hybrid Techniques

In this section, two novel hybrid schemes are proposed which use concepts of SA in GAs. A drawback of the GAs is that there is no definite way to detect when the algorithm has located the global optimum. Though the members of the population should all converge to a good solution, this is not always the case in practice. The proposed schemes are designed to overcome this problem.

### 5.1 Hybrid Technique - I

In this proposed modification to the standard GA, the mutation operator is now used as a primary operator. More specifically, the mutation operator now plays the role which the temperature plays in SA. We propose to use a high value of mutation at the start of the algorithm and as the generations evolve, to gradually reduce the value of the mutation. Thus the generation number is used to exponentially reduce the value of the mutation as the algorithm proceeds. The advantage of this scheme is that as the value of the mutation is gradually reduced, the average minimum error of the whole generation approaches the value of the minimum error in a generation. This could be used as a criterion to stop the algorithm. The decrease in value of mutation is performed by using a non-linear function (exponential) of the generation number as given below

$$\begin{aligned} p_m &= \frac{0.2 * tmp}{1 + tmp} \\ tmp &= e^{(100 - \text{Gen. No.}) / decay} \end{aligned} \quad (3)$$

### 5.2 Hybrid Technique - II

The SA uses the *Metropolis criterion* to probabilistically decide whether to retain or reject a new point. We propose the use of this criteria in GAs. Specifically, after the selection operation, the crossover and mutation operators generate two new members of a population. These two new members are then retained if they have a lesser value of error than the parent members from which they were generated. If they have a larger value, then they are retained depending on a probabilistic function which is a function of a temperature parameter and the difference in error between the parent and new strings. The temperature parameter is dependent on the generation number and is exponentially decreased as the algorithm proceeds using a cooling schedule similar to the schedule used in the standard Simulated Annealing. At the start of the algorithm, all the new members generated are retained as the temperature parameter has a large initial value, but as the algorithm proceeds only new members having an error value less than the parent members are retained. Thus, as the number of generations increase, the average minimum error of the whole generation approaches the value of the minimum error in a generation as was the case in the Hybrid Scheme (I), and thus could be used as a criteria to stop the algorithm.

## 6 Simulation Experiments and Results

In the simulation experiments to adapt higher order filters, the adaptive filter was in the form of a bank of  $2^{nd}$  order filters. Thus a  $10^{th}$  order filter was modeled by a bank of five  $2^{nd}$  order filters. All the results show the minimum error obtained after  $n$  generations versus the number of generations. In all the simulation experiments performed hence, a  $6^{th}$  order IIR filter was used as a model. This filter was adapted by means of a bank of three  $2^{nd}$  order filters, the transfer functions of which had the form

$$H(z^{-1}) = \frac{1}{1 - a_1 z^{-1} - a_2 z^{-2}} \quad (4)$$

The stability of the filters during adaptation was achieved by constraining the filter coefficients  $a_1, a_2$  to lie within the stability triangle.

The three defining parameters of the GA had the following values :  $n$  (pop. size) = 40 ;  $p_c$  (prob. of crossover) = 0.8 ;  $p_m$  (prob. of mutation) = 0.075. These values were obtained after extensive simulation experiments with varying values of the parameters.

Figure [1] shows the result of using the standard GA for the above simulation experiment, with varying number of bits used to code a parameter. The algorithm was able to locate the global minimum and shows rapid initial convergence.

Figure [2] shows the result of using Simulated Annealing as a adaptation technique to the simulation setup detailed above. As can be seen, the Cauchy distribution results in faster convergence. But compared to the standard GA, the SA algorithm takes a much larger number of time samples for convergence.

Figure [3,4] shows the result obtained using the hybrid scheme (I). The minimum error in this case is the minimum error obtained for that particular generation, while the average error is the value of the error averaged over all the members of the population of a generation. Depending on the value of the decay parameter which decides how fast the temperature reduces, the average error is seen to approach the minimum error. The initial value of the mutation operator was 0.2 which was then reduced using an exponential function depending on the generation number (Equ. [3]). The other values of the parameters defining the GA were as given before. The convergence time is seen to be larger than that obtained using the standard GA which results because of the large initial value of the mutation parameter.

Figure [5] shows the result of using the hybrid scheme (II) for different values of the probability of mutation. Results obtained show that the scheme has faster convergence than the hybrid scheme (I) and also that the variance of the error values are reduced. The reason why this happens in the hybrid scheme (I) is because of the large initial value of the mutation parameter.

## 7 Conclusions

Of the alternative configurations which were used, the parallel form gave the best results. The main reason for this was the existence of multiple global minima all of which were equivalent to one another [5]. The cascaded form did not converge even after a large number of generations. The reason for this was found to be the propagation of quantisation error through the filter banks, resulting in an erroneous estimate of error for that particular filter. The main reason for using the lattice forms was that the stability check was easily incorporated in the algorithm by constraining the filter parameters to have unity magnitude. However the lattice configurations did not converge even after a large number of generations.

The above results show that GAs are a viable and practical approach in adaptive IIR filtering especially for adapting high-order filters. It is to be noted though that GA could not locate the exact global optima on account of the discretisation of the parameter space. However they can be used as a first level of search to locate a point close to the global optimum. The optimal setting of the GA parameters is rather heuristic at present and depends heavily on the application on hand. The SA approach though leading to the precise location of the global optimum took a large number of time samples to converge. The hybrid schemes suggest a method by which a stopping criteria could be incorporated into the basic GA structure. In particular the Hybrid Scheme (II) seems to be very promising especially as it has convergence speed similar to that obtained with the standard GA.

## References

- [1] B. Widrow and S. D. Stearns. *Adaptive Signal Processing*. Prentice-Hall Inc., Englewood Cliffs, N.J. 07632, 1985.
- [2] John J. Shynk. Adaptive IIR filtering. *IEEE ASSP Magazine*, pages 4-21, April 1989.
- [3] S. D. Stearns. Error surface of recursive adaptive filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 29:763-766, June 1981.
- [4] H. Fan and M. Nayeri. On error surfaces of sufficient order adaptive IIR filters: proofs and counter examples to a unimodality conjecture. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37:1436-1442, 1989.
- [5] M. Nayeri and W. K. Jenkins. Alternative realizations to adaptive IIR filters and properties of their performance surfaces. *IEEE Transactions on Circuits and Systems*, 36:485-496, April 1989.
- [6] John H. Holland. *Adaptation in Natural and Artificial Systems*. The MIT Press, Cambridge, Massachusetts, 1992, (First Edition 1975).
- [7] D.H. Goldberg. *Genetic Algorithms - In Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, 1989.
- [8] D. M. Etter, M.J. Hicks, and K. H. Cho. Recursive adaptive filter design using an adaptive genetic algorithm. In *Proc. of the IEEE Int. Conf. on ASSP*, pages 635-638, 1982.

- [9] R. Nambiar and P. Mars. Genetic algorithms for adaptive digital filtering. In *IEE Colloquium on Genetic Algorithms for Control systems Engineering*, Savoy Place, London, May 1992.
- [10] S. Kirkpatrick, C. D. Gellat Jr., and M. P. Vecchi. Optimisation by simulated annealing. *Science*, 220(4598):671-680, 1983.
- [11] P.J.M. van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, 1987.
- [12] H. Szu and R. Hartley. Fast simulated annealing. *Physics Letters A*, 122(3,4):157-162, 1987.

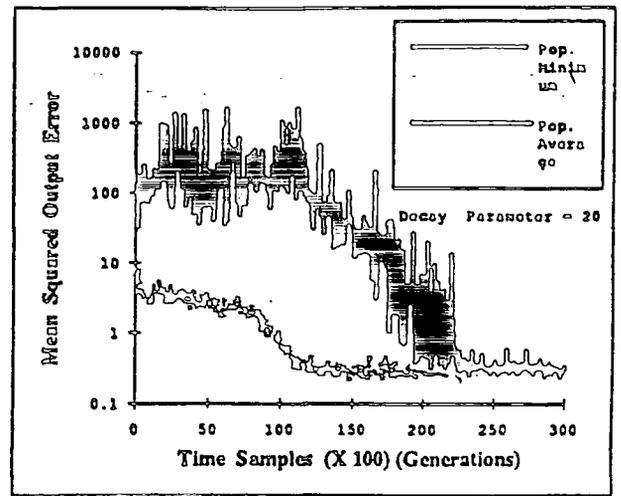


Figure 3

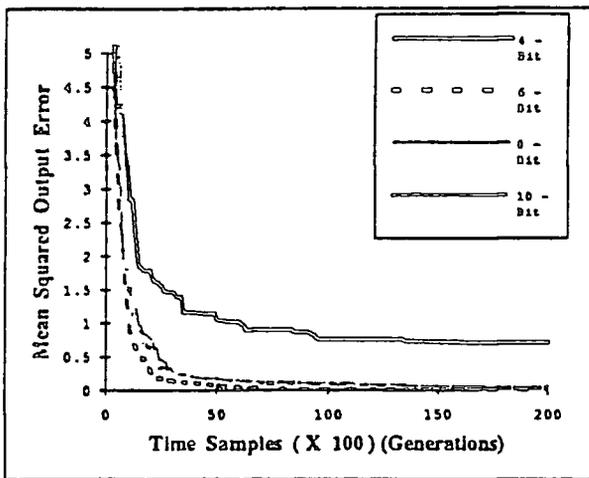


Figure 1

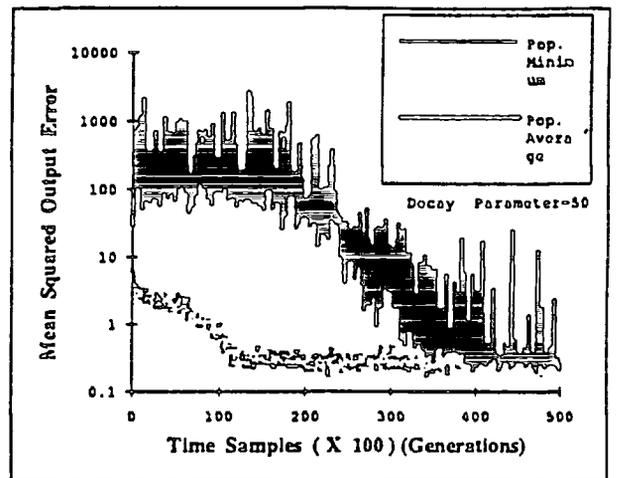


Figure 4

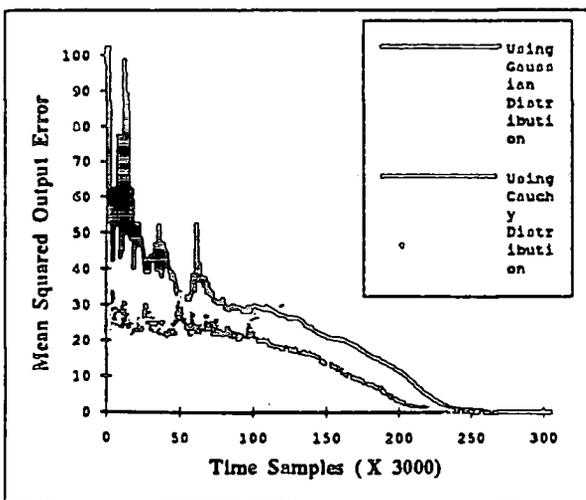


Figure 2

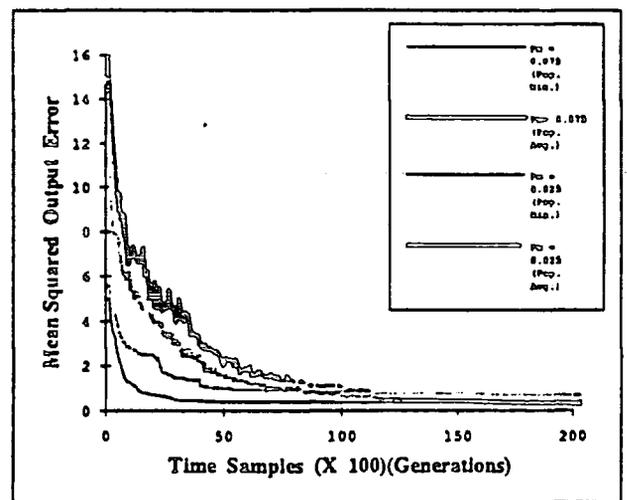


Figure 5

