



Durham E-Theses

Study of decentralised decision models in distributed environments

Ahmed, Quamar F.

How to cite:

Ahmed, Quamar F. (1994) *Study of decentralised decision models in distributed environments*, Durham theses, Durham University. Available at Durham E-Theses Online: <http://etheses.dur.ac.uk/5674/>

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.

**STUDY OF DECENTRALISED DECISION MODELS
IN DISTRIBUTED ENVIRONMENTS**

QUAMAR F AHMED

Doctor of Philosophy

**UNIVERSITY OF DURHAM
SCHOOL OF ENGINEERING AND COMPUTER SCIENCE**

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.

September 1994

**A Thesis submitted in partial fulfillment of the requirements
of the Council of the University of Durham for the Degree of
Doctor of Philosophy (Ph.D.)**

1



26 JUN 1995

STUDY OF DECENTRALISED DECISION MODELS IN DISTRIBUTED ENVIRONMENTS

Quamar F Ahmed
University of Durham
School of Engineering and Computer Science

Abstract

Many of today's complex systems require effective decision making within uncertain distributed environments. The central theme of the thesis considers the systematic analysis for the representation of decision making organisations. The basic concept of stochastic learning automata provides a framework for modelling decision making in complex systems. Models of interactive decision making are discussed, which result from interconnecting decision makers in both synchronous and sequential configurations. The concepts and viewpoints from learning theory and game theory are used to explain the behaviour of these structures. This work is then extended by presenting a quantitative framework based on Petri Net theory. This formalism provides a powerful means for capturing the information flow in the decision making process and demonstrating the explicit interactions between decision makers. Additionally, it is also used for the description and analysis of systems that are characterised as being concurrent, asynchronous, distributed, parallel and/ or stochastic activities. The thesis discusses the limitations of each modelling framework.

The thesis proposes an extension to the existing methodologies by presenting a new class of Petri Nets. This extension has resulted in a novel structure which has the additional feature of an embedded stochastic learning automata. An application of this approach to a realistic decision problem demonstrates the impact that the use of an artificial intelligence technique embedded within Petri Nets can have on the performance of decision models.

Acknowledgements

I would like to thank my supervisor, Professor Phil Mars, for his guidance, support and encouragement during the course of this project.

The research project was sponsored by British Aerospace Plc (BAe), Lancashire. I am very grateful to Mrs Judith Dodds and Mr Tom Morris of BAe, for their invaluable discussions in this field of study, and their enthusiasm in promoting and supporting the project.

Thanks are also due to the Defence Research Agency (DRA), Kent, for allowing the use of their facilities and the undertaking of various projects in the field of Command, Control, Communications and Intelligence (C³-I) systems which has contributed to a better understanding of this study.

Finally, I would also like to thank my colleagues, friends and family for their discussions and support.

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

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

Dedication

To My Parents

Table of Contents

Abstract	2
Acknowledgements	3
Table of Contents	6
List of Figures	12
List of Tables	17
Chapter 1 Introduction	20
1.1 Objectives	20
1.2 Overview of Thesis	20
1.2.1 Overview: Distributed Artificial Intelligence and Approaches to Coordination	21
1.2.2 Basic Stochastic Automaton Model	22
1.2.3 Multiple Automata and Decentralised Decision Making Models	22
1.2.4 Petri Net Theory	23
1.2.5 Learning Petri Net Models	23
1.2.6 Application to Distributed Decision Systems	23
1.2.7 Conclusions and Recommendations for Future Work ..	24
1.3 Conclusion and Summary	24

Chapter 2 Overview: Distributed Artificial Intelligence and Approaches to Coordination	25
2.1 Introduction	25
2.2 Overview of Distributed Artificial Intelligence (DAI)	26
2.2.1 Rationales for DAI	27
2.2.2 Motivations for Learning in DAI Systems	28
2.3 Approaches to Coordination	31
2.3.1 Negotiation	32
2.3.2 Organisational Structuring	33
2.3.3 Multiagent Planning	35
2.3.4 Metalevel Information Exchange	37
2.3.5 Commitments/ Conventions	38
2.3.5 Formal Frameworks	39
2.4 Conclusion and Summary	40
 Chapter 3 Basic Stochastic Automaton Model	 42
3.1 Introduction	42
3.2 Stochastic Learning Automaton Model	43
3.2.1 Stochastic Automaton	43
3.2.2 Environment	45
3.2.3 Reinforcement	46
3.2.4 Linear Reward/ Inaction Reinforcement Scheme	47
3.2.5 Performance	49
3.3 Conclusion and Summary	50

Chapter 4 Multiple Automata and Decentralised Decision Making	
Models	54
4.1	Introduction 54
4.2	Automata Games 55
4.3	Interactive Decision Making Models 56
4.4	Synchronous Models 57
4.5	Simple Feedback 58
4.5.1	Simulation – Simple Feedback 59
4.6	Weighted Feedback 59
4.6.1	Simulation – Weighted Feedback 61
4.7	Synchronous Models : Actions Determine Environment 61
4.8	Interconnection 1 : A_1 Determines A_2 's Environment 62
4.8.1	Simulation – Interconnection 1 63
4.9	Interconnection 2 : A_1 Determines Game for A_2 and A_3 64
4.9.1	Simulation – Interconnection 2 65
4.10	Sequential Models 66
4.11	Tree Structure 68
4.11.1	Simulation – Tree Structure 68
4.12	Directed Network 69
4.12.1	Simulation – Directed Network 70
4.13	General Network 70
4.14	Conclusion and Summary 71
Chapter 5 Petri Net Theory 97	

5.1	Introduction	97
5.2	Structure of a Petri Net	98
5.2.1	Petri Net Graphs	99
5.2.2	Petri Net Markings	100
5.2.3	Execution Rule for Marked Petri Nets	100
5.2.4	Modelling Examples	102
5.2.5	Analysis of Petri Nets	103
5.3	Time-Related Models	106
5.3.1	Stochastic Petri Nets (SPN)	106
5.3.2	An Example of Stochastic Petri Net	107
5.3.3	Generalised Stochastic Petri Nets (GSPN)	107
5.3.4	An Example of Generalised Stochastic Petri Net	109
5.4	Conclusion and Summary	112
 Chapter 6 Learning Petri Net Models		124
6.1	Introduction	124
6.2	Basic Stochastic Learning Petri Net (Basic SLPN)	124
6.2.1	Simulation Results : Basic SLPN	125
6.3	Stochastic Learning Petri Net (SLPN)	126
6.3.1	Reachability Tree : Stochastic Automata Embedded ..	127
6.3.2	Hierarchical System of Automata	129
6.3.3	Operation of SLPN	130
6.3.4	Simulation Results : SLPN	131
6.4	Generalised Stochastic Learning Petri Net (GSLPN)	132

6.4.1	Simulation Results : GSLPN	136
6.5	Conclusion and Summary	137
Chapter 7 Application to Distributed Decision Systems		165
7.1	Introduction	165
7.2	Model of the Decision Making Process	166
7.2.1	Model of an Organisation with a Decision Aid	167
7.3	Application : Small-scale C ³ -I System	168
7.3.1	Performance of Single Decision Module	169
7.3.2	Performance of Two Node Organisation	170
7.4	Experimental Results	171
7.5	Conclusion and Summary	176
Chapter 8 Conclusions and Recommendations for Future Work		199
8.1	Conclusions and Summary	199
8.2	Recommendation for Future Work	207
8.2.1	Models (Byzantine Generals)	207
8.2.2	Modelling Human Factor in C ³ -I Systems	208
8.2.3	Automatic Data Fusion	209
8.2.4	Migration of Control	211
References		214
Appendix 1 Computer Simulation Structure		221

A1.1	Introduction	221
A1.2	Structure of Simulation	221
A1.3	Stochastic Automaton Algorithm	221
Appendix 2 Game Theoretic Concept		229
A2.1	Introduction	229
A2.2	What Is Game Theory?	229
A2.3	Basic Definitions	232
Appendix 3 Petri Net Concepts		235
A3.1	Introduction	235
A3.2	Some Petri Net Properties	235
A3.3	Reachability (Coverability) Tree Algorithm	237
Appendix 4 Models (Byzantine Generals)		240
A4.1	Introduction	240
A4.2	Reliable Systems	240
A4.3	Byzantine Generals Problem	241
	A4.3.1 Impossibility Results	243
	A4.3.2 Solution with Oral Messages	245
	A4.3.3 Solution with Signed Messages	247
A4.4	Byzantine Generals Algorithm	249
Appendix 5 Publications		254

List of Figures

Figure 3.1	Stochastic Learning Automaton Model	52
Figure 3.2	Stochastic Automaton	53
Figure 3.3	Environment	53
Figure 4.1	Automata Game Schematic	73
Figure 4.2	Synchronous Models - The Basic Structure	74
	Simple Feedback	74
Figure 4.3	Average Action Probability vs Iterations : (Table 4.1a)	75
	(a) Action Probability Pr [0.5]	75
Figure 4.3	Average Action Probability vs Iterations : (Table 4.1b)	76
	(b) Action Probability Pr [0.8]	76
Figure 4.4	Synchronous Models - The Basic Structure	77
	Weighted Feedback	77
Figure 4.5	Average Action Probability vs Iterations : (Table 4.2)	78
Figure 4.6	Modification of Synchronous Models	79
	Interconnection 1 : A_1 Determines A_2 's Environment ..	79
Figure 4.7	Average Action Probability vs Iterations	80
	(a) No Coordination (One Equilibrium) : (Table 4.3a)	80
	(b) No Coordination (Two Equilibria) : (Table 4.3b)	81
	(c) Coordination (One Equilibrium) : (Table 4.3c)	82
	(d) Coordination (Two Equilibria) : (Table 4.3d)	83
Figure 4.8	Average Action Probability vs Iterations	84
	(a) No Coordination : (Table 4.4a)	84

	(b) Coordination : (Table 4.4b)	85
Figure 4.9	Modification of Synchronous Models	86
	Interconnection 2 : A_1 Determines Game for A_2 and A_3	86
Figure 4.10	Average Action Probability vs Iterations : (Table 4.5) ..	87
Figure 4.11	Sequential Models	88
	Tree Structure	88
Figure 4.12	Tree Structure : Selected Path	90
	(a) Optimal Path 001	90
	(b) Optimal Path 100	90
Figure 4.13	Optimal Path Probability Changes	91
	(a) Optimal Path 001 : (Table 4.6a)	91
	(b) Optimal Path 100 : (Table 4.6b)	91
Figure 4.14	Sequential Models	92
	Directed Network	92
Figure 4.15	Directed Network : Selected Path	94
	(a) Optimal Path 001:101	94
	(b) Optimal Path 010:110	94
Figure 4.16	Optimal Path Probability Changes	95
	(a) Optimal Path 001:101 : (Table 4.7a)	95
	(b) Optimal Path 010:110 : (Table 4.7b)	95
Figure 4.17	Sequential Models	96
	General Network	96
Figure 5.1	Petri Net Structure	114
Figure 5.2	Petri Net Graph	114

Figure 5.3	Marked Petri Net	115
	(a) Transition t_1 Fires	116
	(b) Transition $t_2 : t_3$ Fires	116
Figure 5.4	Reachability Tree Construction of Marked PN (Figure 5.3)	117
	(a) First Step in Building Tree	117
	(b) Second Step in Building Tree	117
	(c) Third Step in Building Tree	118
	(d) Fourth Step in Building Tree	118
Figure 5.5	Stochastic Petri Net (SPN)	119
Figure 5.6	Reachability Tree SPN	120
Figure 5.7	Generalised Stochastic Petri Net (GSPN)	121
Figure 5.8	GSPN Reachability Tree	122
Figure 6.1a	Basic Stochastic Learning Petri Net (Basic SLPN)	138
Figure 6.1b	Reachability Tree - Basic SLPN	138
Figure 6.2	Optimal Path Probability Changes : Basic SLPN	139
Figure 6.3	Structure of SLPN	140
Figure 6.4	Reachability Tree: Embedded Stochastic Automata	141
Figure 6.5	Two/Three-State Automaton	142
	(a) Two-State Automaton	142
	(b) Three-State Automaton	142
Figure 6.6	Sequence of Decision/ States	143
Figure 6.7	Stochastic Learning Petri Net (SLPN)	144
Figure 6.8	GSPN Reachability Tree: Embedded Stochastic Automata	154
Figure 6.9	GSPN : Sequence of Decisions/ States	155

Figure 7.1	Four-Stage Model of Interacting Decision Maker	177
Figure 7.2	Petri Net Representation of Interacting Decision Maker	178
Figure 7.3	Situation Assessment Module	179
Figure 7.4	Block Diagram: Two Node Organisation Supported by D.S.S.	180
Figure 7.5	Petri Net: Two Node Organisation Supported by D.S.S.	181
Figure 7.6	Topology for Simulation: Single Decision Module	182
Figure 7.7a	Petri Net Representation Decision Module DM1	183
Figure 7.7b	Reachability Tree for Decision Module DM1	183
Figure 7.8	Topology for Simulation: Two Node Organisation	184
Figure 7.9	Single Decision Module (Table 7.1)	185
Figure 7.9a	Average Action Probability vs Iterations : (Table 7.1)	186
Figure 7.10	Two Node Organisation (Table 7.2)	187
Figure 7.10a	Average Action Probability vs Iterations : (Table 7.10)	189
Figure 7.11	Switch of Environment: Before and After Switch	190
Figure 7.12	Average Action Probability vs Iterations	193
	(a) Before Switch P1.Q1 (Table 7.3a)	193
	(b) After Switch P1 (Table 7.3b)	193
	(c) After Switch P3.Q1 (Table 7.3b)	193
Figure 7.13	Communication Between Decision Modules	194
Figure 7.14	Average Action Probability vs Iterations	198
	(a) Upper Level Path P1.Q1 (Table 7.4a)	198
	(b) Path P1.Q1 (Table 7.4b)	198
	(c) Path P3.Q3 (Table 7.4b)	198

Figure 8.1 Factors Affecting the Use of Recognitional/ Analytical Decision Making Strategy	213
Figure A1.1 Overall Structure of Program	224
Figure A1.2 Programme Routines	225
(a) Routine Auto1()	225
(b) Routine Envir1()	226
(c) Routine Lri-prob12()	227
(d) Routine Lri-prob22()	228
Figure A3.1 Coverability (Reachability) Tree Algorithm	239
Figure A4.1 Impossibility Results	251
(a) Lieutenant 2 a traitor	251
(b) Commander a traitor	251
Figure A4.2 Solution with Oral Messages	252
(a) Lieutenant 3 a traitor	252
(b) Commander a traitor	252
Figure A4.3 Solution with Signed Messages: Commander a traitor .	253

List of Tables

Table 4.1	Simulation of Simple Feedback (Figure 4.2)	75
	(a) Action Probability Pr [0.5]	75
Table 4.1	Simulation of Simple Feedback (Figure 4.2)	76
	(b) Action Probability Pr [0.8]	76
Table 4.2	Simulation of Weighted Feedback (Figure 4.4)	78
Table 4.3	Simulation of Interconnection 1 (Figure 4.6)	80
	(a) No Coordination (One Equilibrium)	80
	(b) No Coordination (Two Equilibria)	81
	(c) Coordination (One Equilibrium)	82
	(d) Coordination (Two Equilibria)	83
Table 4.4	Simulation of Interconnection 1 : (Three Action Case) .	84
	(a) No Coordination	84
	(b) Coordination	85
Table 4.5	Simulation of Interconnection 2 (Figure 4.9)	87
Table 4.6	Simulation of Tree Structure (Figure 4.11)	89
	(a) Convergence Path To Minimum d112	89
	(b) Convergence Path To Minimum d211	89
Table 4.7	Simulation of Directed Network (Figure 4.14)	93
	(a) Convergence Path To Minimum d212	93
	(b) Convergence Path To Minimum d221	93
Table 5.1	Interpretations of Transitions and Places	123
Table 5.2a	Switching Probabilities of GSPN	123

Table 5.2b	Reachability Set of GSPN	123
Table 6.1	Optimal Path : Basic SLPN	139
Table 6.2	Optimal Path SLPN : Sequence 0	145
Table 6.3	Optimal Path SLPN : Sequence 1	146
Table 6.4	Optimal Path SLPN : Sequence 2	147
Table 6.5	Optimal Path SLPN : Sequence 3	148
Table 6.6	Optimal Path SLPN : Sequence 4	149
Table 6.7	Optimal Path SLPN : Sequence 5	150
Table 6.8	Optimal Path SLPN : Sequence 6	151
Table 6.9	Optimal Path SLPN : Sequence 7	152
Table 6.10	Optimal Path SLPN : Sequence 8	153
Table 6.11	Optimal Path GSLPN : Sequence 0	156
Table 6.12	Optimal Path GSLPN : Sequence 1	157
Table 6.13	Optimal Path GSLPN : Sequence 2	158
Table 6.14	Optimal Path GSLPN : Sequence 3	159
Table 6.15	Optimal Path GSLPN : Sequence 4	160
Table 6.16	Optimal Path GSLPN : Sequence 5	161
Table 6.17	Optimal Path GSLPN : Sequence 6	162
Table 6.18	Optimal Path GSLPN : Sequence 7	163
Table 6.19	Optimal Path GSLPN : Sequence 8	164
Table 7.1	Simulation of Single Decision Module (Figure 7.9)	186
Table 7.2	Simulation of Two Node Organisation (Figure 7.10) ...	188
	Optimal Strategy Pair P4.Q2	188
Table 7.3	Simulation of Two Node Organisation (Figure 7.11) ...	191

	(a) Before Switch : Optimal Strategy Pair P1.Q1	191
	(b) After Switch : Optimal Strategy Pair P3.Q1	192
Table 7.4	Communication Between Automata (Figure 7.13)	195
	(a) Top Level Communication : (SA1 and SA2)	195
	(b) Top and Lower Level Communication :		
	(SA1:SA2) (RS11:RS21)	196
	(c) Re-locate Unique Maximum: (SA1:SA2)(RS11:RS21)		
	Communicate	197

Chapter One

Introduction

1.1 Objectives

This thesis considers the development of appropriate algorithmic tools for the systematic analysis of distributed decentralised decision systems. Such systems are characterised by a high degree of complexity, a distribution of the decision making process among several ‘agents’, the need for reliable operations in the presence of multiple failures and the inevitable interactions of humans with computer-based decision support systems and decision aids. The analysis and development of such systems requires novel advances in the area of distributed decision making under uncertainty. It is essential to develop quantitative methodologies, theories and algorithms for the representation of such complex systems. Current research in this field is generating much interest and has been prompted by studies from related disciplines, such as computer science, control sciences, engineering and cognitive psychology, [1], [2], [3], [4], [5], [6], [7].

1.2 Overview of Thesis

The thesis describes the progress and results obtained during a research programme to study distributed decision making systems. An example of such a system is the so called C³-I (Command, Control, Communications and

Intelligence) system. In essence, it is the process of information management: how to obtain, process and distribute information quickly and accurately from a network or other hierarchy of systems. Initial work involved an overview of the field of Distributed Artificial Intelligence (DAI), focussing on coordination techniques and highlighting an area of research to be addressed by the DAI community, [8], [9], [10]. An approach based on the stochastic learning automata is proposed to provide the basic conceptual framework for a model of decentralised decision making, [11], [12]. The thesis then describes topologies of synchronous and sequential models by the interconnection of automata in various configurations, [13], [14]. The theory of Petri Nets is then reviewed, [15], [16]. A discussion indicates the limitations of each framework in the development of appropriate decision making models. To resolve these limitations, a new modelling technique is proposed by combining principles from stochastic learning automata and Petri net theory. The application of this new form of hybrid Petri Net model to a small-scale realistic problem is discussed. Original simulation results are presented and discussed. The thesis concludes by discussion of proposed future work. A brief discussion of each chapter is provided in the following subsections.

1.2.1 Overview: Distributed Artificial Intelligence and Approaches to Coordination

Chapter Two provides an overview of the field of DAI and highlights the importance of such systems. In addition, the chapter presents several approaches for effective coordination of nodes in a distributed network. The

survey reveals that there has been minimal DAI research in the collective learning process. Thus, the motivations for learning in DAI environments have also been addressed.

1.2.2 Basic Stochastic Automaton Model

Chapter Three reviews the basic stochastic learning automata model. It describes how a single decision maker operates in a random environment and updates its strategy for choosing actions on the basis of the elicited response. The mathematical description of the input and output sets of the automaton and the P-model environment are introduced. Several learning algorithms and also measures of performance including expediency, optimality and ϵ -optimality are defined. Stochastic learning automata are expected to provide the basic conceptual framework for future research on distributed decision systems.

1.2.3 Multiple Automata and Decentralised

Decision Making Models

Chapter Four provides a detailed study of decentralised decision making in unknown random environments using stochastic learning automata as the basic decision model. This chapter describes the analytical models for interactive decision making systems of increasing complexity and the relevant simulations. These interconnections consider both synchronous and sequential models. The concepts of stochastic learning theory and game theory are used to explain the results of extensive simulations.

1.2.4 Petri Net Theory

Chapter Five considers the potential of Petri Nets for the representation of decision models. This chapter surveys the known results in this area and identifies the Petri Net formalism as a potentially effective graphical and mathematical tool. Moreover, the thesis considers time-related models by examining stochastic timed nets. This chapter then discusses the limitations of existing Petri Nets models needed for the effective representation of decision models.

1.2.5 Learning Petri Net Models

Chapter Six presents a new class of Petri Nets, namely, *Stochastic Learning Petri Nets (SLPN)*. This extension to Petri Net models introduces a new model which has the additional feature of an embedded stochastic learning automata. The hybrid combination was shown to overcome the limitations of existing Petri Net theory and stochastic learning automata used in isolation. This chapter discusses the potential benefits of a new modelling technique by examining various forms of *Learning Petri net models*. The chapter also presents original simulation results for each model.

1.2.6 Application to Distributed Decision Systems

Chapter Seven considers the application of the new Stochastic Learning Petri Net (SLPN) model to a small-scale distributed decision problem. The basic model involves two decision modules interacting with a stochastic environment. This chapter describes simulation studies which demonstrate the

impact that the use of such a modelling tool can have on the performance of decision making organisations.

1.2.7 Conclusions and Future Work

Chapter Eight concludes the thesis by summarising the work that has been presented and also provides an insight to possible future areas of research.

1.3 Conclusions and Summary

This chapter has presented a brief introduction and overview of the contents of the thesis. It has highlighted the key areas of research involved in a study of decentralised decision making in distributed environments. The stochastic learning automata approach has been identified as the fundamental framework for modelling decision making in complex systems. As the research progresses additional layers of sophistication are incorporated within the basic model. Thus, a novel contribution in this field of study has been provided for the representation of effective distributed decentralised decision making models.

Chapter Two

Overview: Distributed Artificial Intelligence and Approaches to Coordination

2.1 Introduction

This chapter presents an overview of Distributed Artificial Intelligence (DAI), with special attention to coordination in Distributed Problem Solving (DPS) and Multi-Agent (MA) systems. The potential benefits in the application of such systems where information, resources or expertise are distributed, or where they are inherently distributed to improve speed, modularity or reliability of the system are considered. However, these potential benefits are not realised if the agents are uncoordinated. The major part of this chapter addresses what many consider to be the key research issue for DAI: how to coordinate the activities of a collection of semi-autonomous problem solvers. Several approaches for effective coordination of nodes in such systems have been reviewed. These include negotiation, organisational structuring, multi-agent planning, metalevel information exchange, commitments/conventions and formal frameworks.

The chapter does not aim to provide comprehensive coverage of the entire field of DAI, such reviews have been generated by Bond and Gasser, [8], and smaller collections have also been published by Huhns, [17] and Gasser, [10]. Rather, the objective is to provide a brief review of coordination techniques in DAI systems. The survey also reveals a specific research

problem that the DAI community have yet to address. Bond and Gasser, [8], stated that there has been virtually no DAI research in collective learning processes. To date this gap in DAI research remains. This thesis presents a new perspective in studying collective learning processes in DAI.

2.2 Overview of DAI

DAI is concerned with the study and construction of semi-autonomous concurrent processing nodes, or agents which perform intelligent operations by interacting with each other and their environments as a community. Each agent is responsible for maintaining a different perspective of the world model, and these agents communicate with each another. Consequently, this organisation has a more diverse perception of the world, is more robust and enables the strengths of several processing paradigms to be exploited.

Research in DAI may be divided into two primary arenas; Distributed Problem Solving and Multi-Agent Systems. Research in the field of *Distributed Problem (DPS)* involves a study of how the work of solving a problem can be divided among a number of modules or 'nodes' so that they can work together to solve problems beyond their individual capabilities. Whilst, research in *Multi-Agent (MA)* systems is concerned with coordinating the knowledge, goals, skills of intelligent agents so that they can jointly take actions or work together to solve problems. These agents may be working towards a single global goal or towards separate individual goals that interact in some way. Similar, to the DPS system, agents must share knowledge and problem solving capabilities, in addition they must reason about their local actions and the

actions of the other agents in the network.

The work described in the thesis better fits the MA system sub-area of DAI. The thesis presents an Artificial Intelligence (AI) approach based on the stochastic learning automata which provides the conceptual framework for research on distributed decision making models. The study of stochastic learning automata as distributed 'agents' is considered to exhibit the characteristics of an MA system. This is particularly applicable to the interconnections examined in Chapter Four. These interconnections consist of a multitude of automaton-environment pairs that interact to achieve specific goals. The agents are shown to operate together in an uncertain environment either in a cooperative or competitive manner and the game situation is represented by synchronous models. Similarly, sequential models are considered whereby, the agents operate on various levels with interaction between the different levels to seek optimal performance. However, the modelling framework does not include the reasoning capabilities which are an essential feature of an MA system, instead, an intelligence capability permits the agents to adapt to changing environments.

2.2.1 Rationales for DAI

The following attributes summarise the potential benefits of using this type of environment:

Parallelisation/ Concurrency : Faster problem solving by exploiting parallelism. No order is assumed in the invocation of agents in the network. They may be running in some arbitrary sequence on a single processor by a

multi-processing operating system, or may be running on physically separate processors.

Communication : Agents communicate with one another using a message passing protocol.

Modular Design/Naturalness : The principles of modular design and the ability to structure problems into relatively self-contained processing modules leads to systems that are easier to build and maintain. The decomposition of large tasks into manageable subtasks which, in themselves, are well bounded but which when allowed to interact, are capable of creating a powerful model of the world.

Robustness : A DAI system has both hardware and software robustness. If one or more agents becomes disabled through a hardware fault, the system will degrade gracefully. If the system misbehaves, due to a problem with the software, the consequences will be contained: if the agent fails to adhere to the specified message passing protocol then it will not be listened to by the other agents; if it produces an error in reasoning then it is likely to be outvoted by the rest of the community of agents.

2.2.2 Motivation for Learning in DAI Systems

Learning denotes changes in the system that are adaptive in the sense that they enable agents to acquire knowledge with time and adapt their reasoning to improve their performance at specific tasks, more efficiently and effectively at the next stage. The ability to learn is an essential feature of any intelligent system which has to operate in a changing environment.

However, this concept has been rarely discussed in the DAI literature, [8]. Research on learning in DAI systems should consider ways to improve the agent's knowledge and skill to enable the whole DAI system to improve its performance as a result.

The earliest attempt for incorporating any learning mechanisms in DAI systems was in the Multiple Intelligent Node Document Servers (MINDS) system, [18]. This system operates in the domain of intelligent document retrieval. However, the concept of learning in this system was observed to be a localised activity without any cooperation between the agents to learn globally useful attributes. Shaw and Whinston,[19], describe a method treating DAI systems as adaptive organisations with the ability to improve learning from past experience. The proposed method is composed of two processes: an extension of the Contract Net protocol (discussed in Section 2.3.1) and using a genetic transformation process within agents to find a more efficient solution. The Contract Net framework is extended as follows, tasks are awarded to the most appropriate bidders; the tasks are traded with hypothetical payments which is equivalent to the bid; this in turn affects the strengths of the agents involved which are updated accordingly. The concept of the bidding scheme as a feedback mechanism to rate each agent is used as the basis for learning and adaptation. The process of learning is performed by a genetic algorithm. This technique used the strengths as the indication of suitability to find desirable attributes of successful agents, and the weaker agents were eliminated by new agents inheriting the desirable characteristics. This process improved the overall performance of the system. Shaw and

Whinston have shown the application of this methodology to the scheduling of flexible manufacturing systems. Sian, [20], has developed a model for adaptation in MA systems that allow cooperative learning among autonomous agents. The symbolic approach to adaptation is based on explicit interaction between agents for the purpose of learning useful information. Each agent may only be able to infer partial hypotheses by using local information and requires cooperation to produce a 'complete picture'. The interaction with the other agents provides a more consistent, accurate hypotheses and increases the level of confidence in the hypotheses. The model has been implemented in a system called Multi-Agent Learning Environment (MALE).

The general model of DAI systems is one in which a collection of agents (distributed spatially, logically or temporally) are engaged in the performance of coordination of activities. In such complex systems which are organised in a hierarchical or decentralised manner, the agents must deal with large uncertainties regarding either the structure, parameters or the nature of external events. In particular, it is these external uncertainties that add to the difficulty of the control problem and their presence necessitates the use of learning schemes. It should be emphasised that decentralisation by its very nature introduces uncertainty into the system. The remote components of the same system can only have limited information about each other and the overall system. Hence, the decisions must be made by individual agents that have access only to partial information regarding the state of the overall system. However, this results in an inconsistency between local and global optimality. To deal with such systems effectively, it is essential

that the agents adapt to their environment by utilising a learning paradigm. This thesis has adopted a basic learning paradigm for the representation of decentralised decision making models.

2.3 Approaches to Coordination

The concept of coordination in DAI research has most often been described as the process of control decision making that guides the overall behaviour and performance of a collection of semi-autonomous problem solvers. The existing literature on DAI provides various definitions of this concept, namely, coordination may be referred to as the *process* of structuring decisions so as to maximise the overall effectiveness of a collection of problem solving nodes. Alternatively, the *outcomes* of a collection of control decisions may also be referred to as coordination, [21]. The coordination of the actions of a collection of decentralised agents has been posited as a formidable problem of DAI research. At present there is a diverse range of techniques which can and do facilitate coordination in DAI systems. These mechanisms can be broadly divided into the following categories:

- Negotiation
- Organisational Structuring
- Multi-agent Planning
- Metalevel Information Exchange
- Commitments/ Conventions
- Formal Frameworks

Each of these approaches is examined in turn; a brief description about how it facilitates coordination behaviour is provided. The most relevant to the work in this thesis is the area of formal frameworks, which is discussed in Section 2.3.6. Most coordination techniques have been motivated and evaluated in terms of an application domain, often by building a simulator for the domain. These implementations are prototypes and simulations; to date, there have been only two MA systems which have been used in real-world applications, [22], [23], [24].

2.3.1 Negotiation

Negotiation is a fundamental part of human cooperation, that allows people to resolve conflicts that could interfere with cooperative behaviour. The term 'negotiation' may be defined as 'the process of improving agreement (reducing inconsistency and uncertainty) on common viewpoints or plans through the structured exchange of relevant information', [25]. The following provides a more concise description of negotiation.

Smith and Davis, [26], [27], developed the Contract-Net framework, which is one of the earliest and most influential research projects in cooperative DPS. This represents a framework that specifies communication and control in a distributed problem solver. The process of negotiation involves three important components: a two-way exchange of information between interested parties; an evaluation of the information by each party from their own perspective and a final agreement achieved by mutual selection.

Conry and her colleagues, [28], describe a multistage negotiation

paradigm for planning in a distributed environment with decentralised control and limited inter-node communication. This process considers another use of a limited form of negotiation in task allocation. The multistage negotiation protocol is useful for cooperatively resolving resource allocation conflicts which arise in a distributed network of problem solvers. This framework may be viewed as a generalisation of the contract net protocol. The contract net was devised as a means for accomplishing task distribution among agents in a distributed problem solving system. Task distribution takes place through a negotiation process involving contractor task announcement followed by bids from competing subcontractors and finally announcement of awards. The multistage negotiation extends the basic contract net protocol to allow iterative negotiation during the bidding and awarding of tasks.

2.3.2 Organisational Structuring

An organisational structure is a network level coordination mechanism that can be implemented in a number of ways. In most DAI research, an organisational arrangement imposes guidelines about the distribution of specialisations among the collective agents. It provides a framework for activity and interaction through defined roles, behavioural expectations and authority relationships(eg control). The control relationship between the agents can be represented in terms of topologies such as hierachical, heterarchical, flat(lateral) structures. These organisations are responsible for designating the relative authority of the agents and for specifying the types of interactions that can occur. Organisational structures can be used as a high-level specification

of the distribution of problem solving capabilities among the community members, [29].

An organisational structure provides more general long-term information about the relationships between agents. As stated previously, an organisation can be reviewed as a distribution of capabilities which is a precise way of dividing the problem space without having to go into depth about the particular problem subtrees.

Organisational structures provide a control framework that increases the likelihood that agents operate as a coherent team by identifying the roles of each individual. Lesser and Corkill, [30], applied organisational structures to efficiently implement network coordination strategies. Their ideas have been implemented and evaluated in one of the most flexible simulation testbeds developed to date: the Distributed Vehicle Monitoring Testbed (DVMT). This simulates a spatially organised network of agents which perform distributed interpretation to track vehicles moving among them. By this process of coordination agents build a map of vehicle movement through an entire area. Lesser and Corkill suggest that each agent needs to decide on its own activities, based on the current local view of the problem being solved, but organisational knowledge should be applied about its problem solving role in the network and the roles of others to guide its decision so that it is a more effective participant in the network. This approach divides coordination into two concurrent activities: the construction and maintenance of a network wide organisational structure into precise activities using the local knowledge and control capabilities of each agent.

2.3.3 Multiagent Planning

In a multiagent planning approach to cooperation, nodes (agents) form a multiagent plan that specifies all their future actions and interactions. The coordination of nodes through multiagent plans is different from organisational structuring and metalevel information exchange in terms of the level of detail to which it specifies every agents activities. In this case one or more nodes possess a plan that indicates exactly the actions and interactions each node will take for the duration of the network activity. Agents know *a priori* exactly what actions they will take, one or more nodes have information about each node's activities and what actions will occur, recognising and preventing the duplication of effort. Multiagent planning insists on detecting and avoids inconsistencies before they can occur. Finally, a multiagent plan dictates exactly what actions should be taken by each node and when the actions should be taken; which is unlike the guidelines imposed by an organisation structure. The approach requires more computation and communication resources than other approaches, since nodes are expected to share and process substantial amounts of information.

There are two basic approaches to multiagent planning: centralised and distributed. Georgeff, [31] develops a multiagent planning approach where the plans of individual nodes are first formed which is collected by some central planning node. It is then analysed to identify potential interactions such as conflicts between the nodes over limited resources. This provides an efficient method of interaction, and safety analysis is then developed by central node to determine which potential interactions could lead to conflicts. The central

planning node next groups together sequences of unsafe situations to create critical regions. Finally, the idea is to insert communication commands into the plans so that nodes can synchronise activities and avoid harmful interaction, [32], [33]. Cammarata et. al., [34] also devised a centralised multiagent planning system for the air traffic control (ATC) domain. In this ATC application, each aircraft (agent) sends information about its intended actions to a coordinator. The coordinator is responsible for developing a plan which specifies all the agents' actions, including the actions to be taken to avoid harmful collisions.

Whilst, with distributed multiagent planning, the plan is developed by several agents. Rosenschein and Genesereth, [35] studied a logic-based approach studying how agents with a common goal but different local information can exchange propositions to converge on identical plans. They developed strategies for convergence. These strategies were based on assumptions about the correctness and completeness of agents' information, whether additional information can cause a previously acceptable plan to be unacceptable and also what each agent knows about other agents' knowledge. Their results indicate that it is infeasible to expect sometimes unpredictable agents working in dynamic domains to always coordinate optimally, perhaps the best to be expected is that they will coordinate acceptably well and will tolerate any uncoordinated activity.

2.3.4 Metalevel Information Exchange

The exchange of metalevel information is another way that the agents in a network can improve their coordination. Gasser, [36], describes metalevel information as the control level information about the current priorities and focus of a problem solver. This indicates the approximate regions of the search space on which agents focus their efforts.

Durfee, [37], developed a metalevel information exchange to coordination, called *Partial Global Planning*. Their partial global planning approach presents a unified, flexible framework which brings together a range of distinct coordination techniques. The technique can be viewed as planning, but it differs from traditional planning that rigidly dictates specific actions to be performed at specific times. The partial plans can change so fluidly and adapt to changing information and environments. The plans are used to detail an agent's problem solving strategy, and its expectations. Each agent follows the specified strategies for as long as it is feasible, and they have the capability to change strategies as problem solving progresses.

This process of coordination involves sharing sufficient tentative plans. This enables at least one agent to establish a global view to recognise how changes to local plan could improve coordination among them. Any number of nodes can collect plan information from others; the coordination of the plans by specific nodes is dependent on the domain requirements and constraints. It is not necessary for each node to have a global view in order to improve coordination. As agents collect plan information from various agents in the network, the *partial* knowledge about its global situation is combined to form

Partial Global Plans (PGPs). Agents maintain their own set of PGPs, which may be used independently or asynchronously to coordinate its activities. Agents use its models of itself and others to identify when nodes have PGPs whose objectives could be part of some larger network objective called *Partial Global Goal* and combines the related PGPs into a single, larger PGP to achieve it.

2.3.5 **Committments/ Conventions**

Jennings, [38], presents a unifying coordination model which considers the notion of commitments and conventions as the foundation of coordination in MA systems. The term commitments are considered to be pledges to undertake a specified course of action, whilst conventions provide a means of monitoring commitments in changing circumstances. In the former case, agents can make pledges both about actions and beliefs. These beliefs can relate to the future or the past. In addition, commitments provide a degree of predictablity, so that agents can take the (future) activities of other agents into consideration when dealing with inter-agent dependencies, global constraints or resource utilisation constraints. In the latter case, conventions provide cooperating agents with the flexibility they need to operate in dynamic environments. In such environments the external world may change, agents may receive new information which may constantly change their own beliefs. Thus, to operate successfully and intelligently, agents need a mechanism for assessing whether commitments are valid. Conventions provide this mechanism so that agents can reconsider its commitments and specify the appropriate

course of action to either retain, rectify or abandon the commitment. The proposed model is based upon the *Centrality of Commitments and Conventions Hypothesis* which states that: *all coordination mechanisms can ultimately be reduced to (joint) commitments and their associated (social) conventions.*

2.3.6 Formal Frameworks

This section focusses on formal models, using logic-based or game-theoretical models. Some of this work has concentrated on how nodes form multiagent plans, including the work of Georgeff, and of Rosenschein and Genesereth, [39].

The formalisms developed for logic-based agents, that work alone must be extended in two ways. As a first extension these systems must be able to model and reason about the concurrent activities of multiple agents. The second requirement is that the agents must perform in situations where they have incomplete knowledge or limited computational resources. However, both modifications lead to a possibility of producing incorrect inferences which result in agents having inconsistent beliefs about the world. Thus, agents may never converge on shared, coordinated plans, [39].

Rosenschein and Genesereth, [35], [40], proposed another approach towards developing a formal theory for understanding the nature of cooperation among multiple agents. Their models were based on game theory techniques and have shown the utility of communication to resolve conflicts among agents having disparate goals. By using a game theoretic model, each agent attempts to choose an option to maximise its payoff, and since no combination of agents'

options can lead to maximal payoffs for them all, they must somehow select an option that results in acceptable payoffs given the circumstances. They studied how different assumptions about the rationality of the agents can lead to more or less effective choices.

As stated previously, game theoretic issues provide a fundamental basis for the study of decentralised decision making. Wheeler and Narendra, [13], consider the basic multiple automata game interacting through an uncertain environment. At each stage the automaton selects an action, and this determines the distribution of the random process involved. It should be noted that in contrast to the usual game-theoretic formulation, no player is aware of the other players, the actions selected by or the responses from the environment to the players. The research has involved synchronous models in which the time instants for automata actions and updates are synchronised, and sequential models which are asynchronous. These models can be analysed by game-theoretic concepts. A detailed discussion of this work is presented in Chapter Four.

2.4 Conclusion and Summary

This chapter has presented an overview of DAI and also focusses on the approaches for coordinating nodes in DAI systems. Based on this survey, it is evident that effective coordination is based on three essential factors. Firstly, it requires *structure* because without structure the nodes cannot interact in predictable ways. Secondly, it requires *flexibility* because nodes typically exist in dynamically changing environments where each node

might have incomplete, inaccurate, or obsolete information. Finally, effective coordination requires *knowledge* and *reasoning* capabilities to intelligently use the structure and flexibility. These factors also apply to the quantitative framework adopted in this thesis. The main features include structure and flexibility, which is illustrated by the different interconnections presented in Chapter Four. However, the limitations of the basic stochastic learning automata for the representation of a generalised network have forced an extension to this modelling framework, which is discussed in Chapter Six. The final features, knowledge and reasoning is not a matter of coordination, it enables agents to reason about the information and decision making in their problem solving activities. Although, the framework proposed in this thesis does not exhibit this characteristic, instead the model has an intelligence capability which enables agents to adapt to changing environments.

The survey has also emphasised that there are certain gaps in DAI research that are worthy of further investigation. One such area that should be addressed is related to the benefits to be gained by implementing a collective learning process in DAI research. This thesis addresses this particular topic of research and provides a new insight to the virtually unexplored field of DAI. The thesis proposes a different perspective to collective learning in a DAI environment. It will be described in the subsequent chapters how collectives of automata have been designed to function as a distributed, yet coordinated intelligent control system. These models utilise learning schemes to display intelligent behaviour in an uncertain environment, [12].

Chapter Three

Basic Stochastic Automaton Model

3.1 Introduction

The previous chapter has provided an overview of the field of DAI, identifying the key requirements for effective coordination and highlighting an area of research to be addressed by the DAI community. This chapter proposes the stochastic learning automata approach which provides a fundamental framework for a model of decision making under uncertainty, [41], [42]. The concept of learning is defined as any relatively permanent change in behaviour resulting from past experience. An extensive literature and a well established mathematical foundation now exists for models of learning systems. The learning system was first introduced to model the behaviour of biological systems [43]. At a later stage it was shown that such models can use a variety of learning schemes to display intelligent behaviour under uncertainty, [11], [12]. This early work and related research formed the basis for what has become known as the learning automaton approach. These automata effectively use past experience and interaction with a random environment to optimise their response to external factors.

This chapter introduces the basic concept of stochastic learning automata, providing relevant definitions of stochastic automata and random environments. It discusses the properties of reinforcement schemes (or up-

dating algorithms) which determine the performance of stochastic automata. These learning arrays will be combined with that of Petri nets in the later chapters, to model decision making systems.

3.2 Stochastic Learning Automaton Model

In general, a learning automaton may be defined as a simple model for decision making in an unknown random environment, Figure 3.1 shows the basic model. The stochastic automaton has a finite set of actions, and these actions form the inputs to the environment. Initially, the probability of selecting any of the available actions is equal. One action is selected at random, which interacts with a random environment. The environment responds to that action, and based on this response the action probabilities are sequentially updated. A new action is then selected according to the updated action probabilities, this procedure being repeated. Through this process of interaction with the environment, the automaton learns to choose asymptotically with a high probability the optimal action, if such an action exists. The components of the stochastic automaton model can be characterised as follows.

3.2.1 Stochastic Automaton

Figure 3.2 shows a stochastic automaton with its input and action set. A stochastic automaton is a sextuple $(\beta, \phi, \alpha, p, F, G)$ and the components can be defined as follows:

(i) The input set to the automaton (output from the environment), denoted $\beta(n)$

$$\beta = (\beta_1, \beta_2, \dots, \beta_k)$$

where k may be finite or infinite.

(ii) The state of an automaton at any instant n , $\phi(n)$

$$\phi = (\phi_1, \phi_2, \dots, \phi_s)$$

where s is finite.

(iii) The output action set selected by the automaton (inputs to the environment, $\alpha(n)$

$$\alpha = (\alpha_1, \alpha_2, \dots, \alpha_r)$$

where r is finite, and $r \leq s$.

(iv) The state probability vector governing the choice of the state at each stage, denoted $p(n)$

$$p(n) = (p_1(n), p_2(n), \dots, p_s(n))^t$$

where

$$p_i(n) = Pr(\alpha(n) = \alpha_i)$$

and

$$\sum_{i=1}^s p_i(n) = 1 \quad \forall n$$

thus, preserving the probability measure.

(v) The state transition function which relates the current state and input at stage n to the next state at stage $n + 1$.

$$F : \phi x \beta \rightarrow \phi$$

(vi) The output function G relates the state of the automaton to the resulting output action at stage, n

$$G : \phi \rightarrow \alpha$$

The functions F and G may be deterministic or stochastic mappings. If F and G are both deterministic, the automaton is denoted a 'deterministic automaton'. In this case the succeeding state ($n+1$) and output action are uniquely defined for a given current state and input. In contrast, if there are only probabilities associated with each successive state and output actions, the automaton determines a 'stochastic automaton' in which F or G or both are stochastic functions.

3.2.2 Environment

The environment can be defined as a random process or medium in which the automaton itself operates. Figure 3.3 represents the environment which accepts output actions of the automaton as inputs and produces responses which are in turn fed back to the automaton. The environment is described by the triple (α, c, β) where $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_r)$ are the input action set (input to the environment), the set $c = (c_1, c_2, \dots, c_r)$ represents the penalty probabilities and β is the output set (input to the automaton).

The nature of the response output from the environment, determines three possible types of environment. The first type of environment which is considered is the P model, this consists of a binary environment which is

defined by a finite set of inputs $\alpha = (\alpha_1, \dots, \alpha_r)$ (outputs from the automaton); a set of penalty probabilities associated with each action $c = (c_1, \dots, c_r)$; and an output set $\beta(n) = (0, 1)$. The $\beta(n) = 0$ at stage n denotes a favourable response (reward) and $\beta(n) = 1$ an unfavourable response (penalty). The $c_i(n)$ are called penalty probabilities and are defined as:

$$c_i = \text{Pr}[\beta(n) = 1 / \alpha(n) = \alpha_i] \quad (3.1)$$

Therefore c_i represents the probability of a penalty being output in response to the input α_i , while the probability of a reward is $(1 - c_i)$.

Other possible types of environments have included Q models (finite number of outputs) and S models (continuous outputs in range 0 to 1). In practice the choice of environmental models is obviously dictated by the particular application. If the penalty probabilities from the environment do not depend on stage number n , the environment is classified as stationary; otherwise the environment is non-stationary.

3.2.3 Reinforcement

The reinforcement scheme is a crucial factor in determining the performance of the learning automaton. In general terms a reinforcement scheme can be represented by:

$$p(n+1) = T[p(n), \alpha(n), \beta(n)] \quad (3.2)$$

where T is an operator (learning algorithm) that denotes the rule by which the

automaton updates the probability of selecting certain actions; $\alpha(n)$ represents the action of the automaton; $\beta(n)$ represents the input to the automaton from the environment at instant n , respectively.

The manner in which $p(n)$ is updated is governed by the learning algorithm T , [12]. Both linear and non-linear forms of updating algorithms T have been considered. If $p(n+1)$ is a linear function of the components of $p(n)$, the reinforcement scheme is said to be linear, otherwise it is non-linear. The most widely used are the class of linear algorithms which include linear reward/ penalty (L_{RP}), linear reward/ ϵ penalty ($L_{R\epsilon P}$) and linear reward/ inaction schemes (L_{RI}). For the L_{RP} scheme if an automaton selects an action α_i which results in success $p_i(n)$ is increased and all other $p_j(n)$ ($j \neq i$) are decreased. Similarly if action α_i produces a penalty response $p_i(n)$ is decreased and all other $p_j(n)$ are modified to preserve the probability measure. An L_{RI} scheme ignores penalty responses from the environment and $L_{R\epsilon P}$ only involves small changes in $p_i(n)$ for penalty responses compared with changes based on success.

3.2.4 Linear Reward/ Inaction Reinforcement Scheme

This section considers one particular reinforcement scheme known as the Linear Reward/ Inaction or L_{RI} method, since most simulations in the thesis have employed this particular learning scheme. This is due to the fact that L_{RI} schemes are known to exhibit the ability to converge to an optimal action, if the optimal action exists, [12]. The behavioural properties of a variable structure L_{RP} stochastic automaton can be analysed by the following

linear algorithm using various parameter values:

For $\alpha(n) = \alpha_i$ and $\beta(n) = 0$ (reward)

$$\begin{aligned} p_i(n+1) &= p_i(n) + a[1 - p_i(n)] \\ p_j(n+1) &= (1 - a)p_j(n) \quad j \neq i \end{aligned}$$

For $\alpha(n) = \alpha_i$ and $\beta(n) = 1$ (penalty)

$$\begin{aligned} p_i(n+1) &= (1 - b)p_i(n) \\ p_j(n+1) &= \frac{b}{(r-1)} + (1 - b)p_j(n) \end{aligned} \tag{3.3}$$

where $0 < a < 1$ and $0 < b < 1$ are constants called reward and penalty parameters, respectively. Special cases of the algorithm result when 'a' and 'b' take on certain values as stated above; also an L_{RI} scheme is produced if the penalty parameter $b=0$.

Equation 3.3 which accomodates a binary environment and this may be modified to include a general environment. In this case $\beta(n)$ takes on values in the interval (0, 1), and the success probabilities d_i are replaced by success distributions, one associated with each action. The following algorithm presents the general environment case, for an L_{RI} scheme when $b=0$:

For $\alpha(n) = \alpha_i$

$$\begin{aligned} p_i(n+1) &= p_i(n) + a\beta(n)[1 - p_i(n)] \\ p_j(n+1) &= p_j(n) - a\beta(n)p_j(n) \quad j \neq i \end{aligned} \tag{3.4}$$

Note that this formulation may be reduced to equation 3.3 if $\beta(n)$ is a binary environment.

These equations describe how the probabilities of selecting the appropriate actions are adjusted so that if successful, they are selected with greater probability, otherwise with less. Also note how the probability measure is preserved.

3.2.5 Performance

The basic operation carried out by a learning automaton is the updating of the action probabilities on the basis of the responses from the environment. The convergence characteristics of learning automata are dependent on the properties of the algorithm used in the updating scheme. A useful measure for judging the performance of the learning automaton is the average penalty received. At a certain stage n , if the action α_i is selected with probability $p_i(n)$, the expected penalty is:

$$M(n) = E[\beta(n)/p(n)] \quad (3.5)$$

Assuming a stationary environment and the actions are randomly selected with equal probability, the value of the average penalty M_o is given by:

$$M_o = \frac{(c_1 + c_2 + \dots + c_r)}{r} \quad (3.6)$$

Definition 1

A learning automaton is said to be *expedient* if:

$$\lim_{n \rightarrow \infty} E[M(n)] < M_0 \quad (3.7)$$

When a learning automaton is expedient it only does better than one which chooses actions in a purely random manner. If the average penalty is minimised by a proper selection of actions then the learning automaton is said to be *optimal*, where:

Definition 2

A learning automaton is called *optimal* if

$$\lim_{n \rightarrow \infty} E[M(n)] = \min_i [c_i] \quad (3.8)$$

Although optimal performance is a desirable property it cannot always be achieved. In such a case one would aim for sub-optimal performance, defined as follows:

Definition 3

A learning automaton is called ϵ - *optimal* if

$$\lim_{n \rightarrow \infty} E[M(n)] = c_{\min} + \epsilon \quad (3.9)$$

This property can be obtained for any arbitrary $\epsilon > 0$ by a suitable choice of the parameters of the reinforcement scheme. In this case ϵ - *optimal* implies that the performance of the automaton can be made as close to the optimal as required. These properties are said to be conditional if the values hold only when penalty probabilities c_i satisfy certain restrictions, eg. that they should lie in certain intervals.

3.3 Conclusion and Summary

This chapter has introduced the basic model of a stochastic learning automata. It has defined the structure of the stochastic automaton, the nature of the random environment and norms for judging the behaviour of the automaton. These concepts are relevant in studying the behaviour of interactive decision makers.

It will be shown in Chapter Four, that the stochastic learning automata approach will provide the fundamental framework for a model of decentralised decision making in C³-I environments.

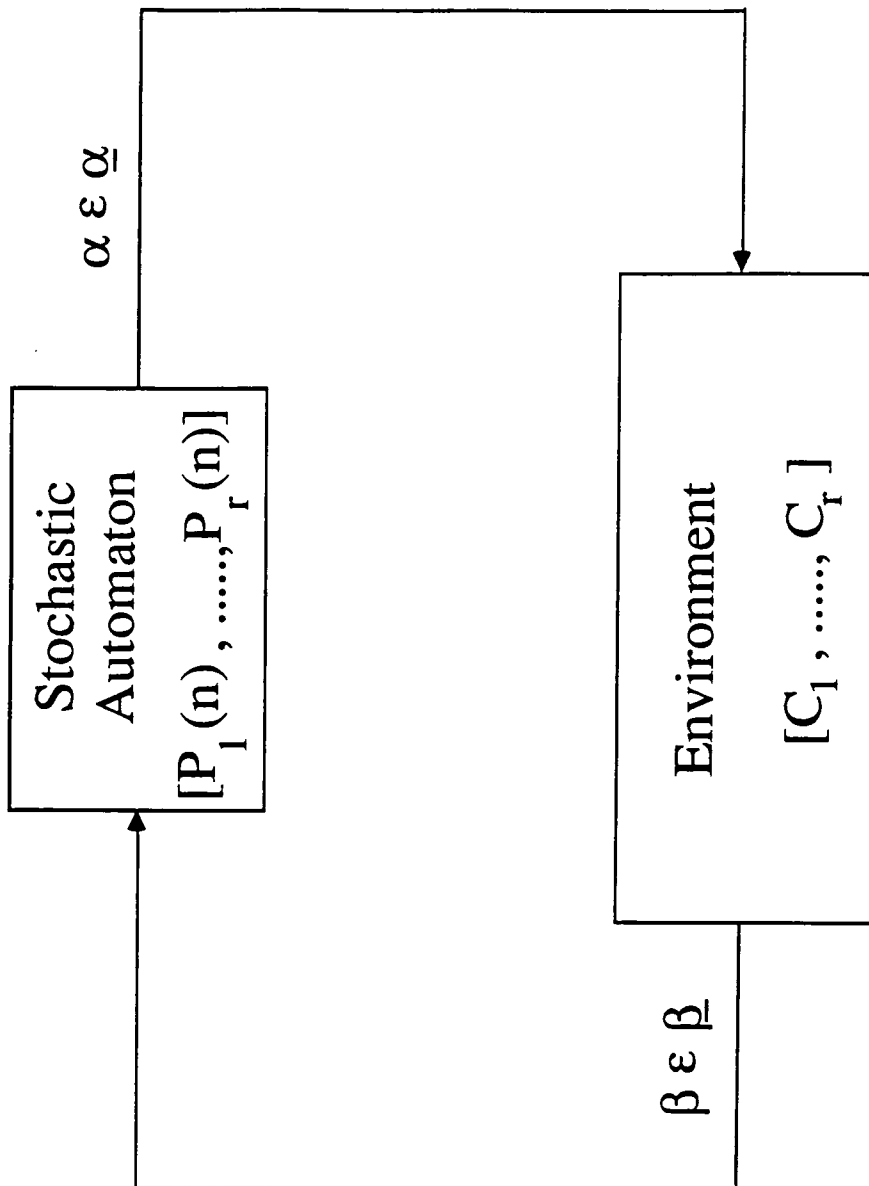


Figure 3.1 – Stochastic Learning Automaton Model

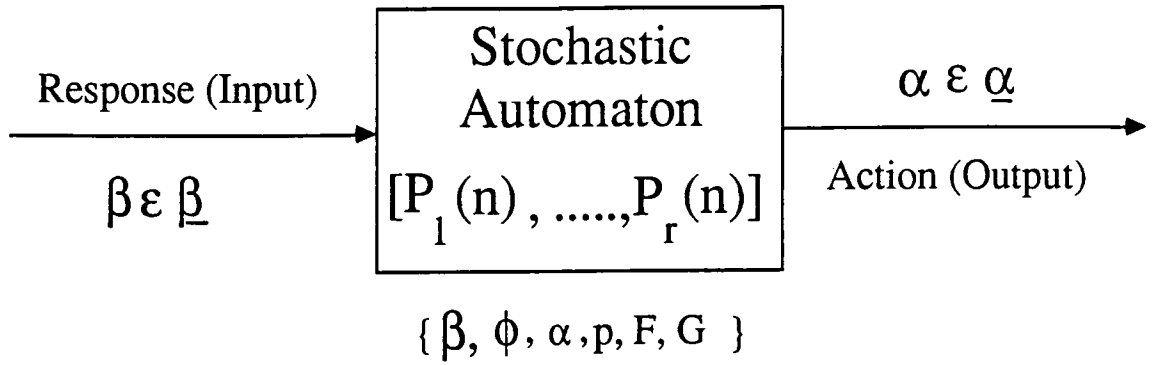


Figure 3.2 – Stochastic Automaton

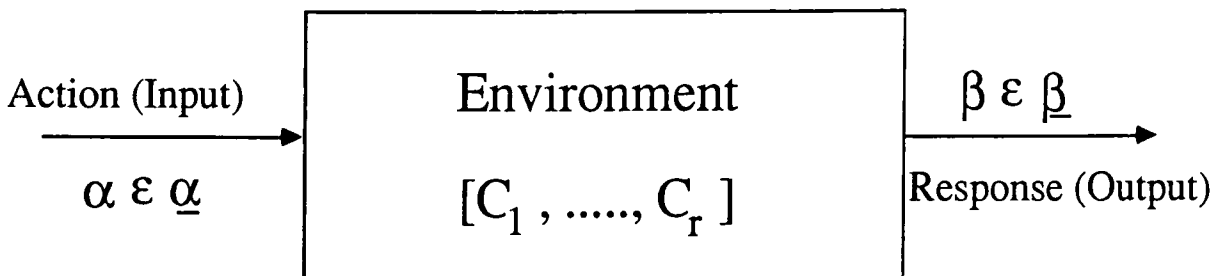


Figure 3.3 – Environment

Chapter Four

Multiple Automata and Decentralised Decision Making Models

4.1 Introduction

The previous chapter considered models of a single decision maker (an automaton) interacting with an uncertain environment. This discussion can now be extended to consider multiple decision makers and environment pairs in various interactive configurations. As such, the models are descriptive and have the property of analytical tractability. It is shown that each interconnection gives rise to a corresponding automata game, which lead to very different game structures. In some cases, the game can be analysed directly using results from automata game theory in which the players are considered to be learning automata. However, in some models the game lacks a structure for which automata behaviour is not known. Two types of interactions are of particular interest. In the first case, several automata are operating together in an environment either in a competitive or cooperative manner and this game situation may be represented by synchronous models. The second case considers automata operating on various levels with interaction between different levels in a hierarchical structure.

This chapter defines an automata game and analyses the behaviour of multiple automata in an abstract game played repeatedly. A major part of this chapter introduces models in which decision makers are not autonomous

and their decisions affect each other. It is shown that automata games have no prior knowledge of the game or number of players available and the players choose their strategies on-line. Such interactive structures result from interconnecting many decision maker-environment pairs to produce synchronous and sequential models, [13], [14], [44], [45]. These models are discussed in detail and simulation results are provided for each interactive configuration. The analysis of such models is based on results from learning theory and game-theoretic issues, [46].

4.2 Automata Games

An automata game, Figure 4.1, involves N automata (or players) A_i ($i = 1, \dots, N$) each with an action (strategy) set $\alpha^i = \alpha_{1i}^i, \dots, \alpha_{r_i}^i$ interacting repeatedly through a stationary random environment. Each automaton A_i selects an action according to its current probability distribution, at time instant n . The joint action (or play) $\alpha(n) = \alpha = (\alpha_{i1}^1, \alpha_{i2}^2, \dots, \alpha_{iN}^N)$ determines the success probabilities for a binary environment or success distributions for a general environment. Note that a binary environment is assumed for the models studied, unless otherwise stated. The environment is stationary since the $d^i(\alpha)$ are fixed over time. In the multiple automata case, each automaton has access only to its own response. It should be noted that in contrast to the usual game theoretic formulation, no player is aware of the other players, the actions selected by or any of the environment success probabilities ($d_{i1i2\dots iN}^i$).

Similar, to the single automaton environment model, the basic feature of an automata game is that at each instant the probabilities of choosing

actions are updated. The probability of an action is increased when the selected action results in a success and is decreased or left unchanged when it results in a failure.

4.3 Interactive Decision Making Models

As stated previously, this section introduces plausible models of decentralised decision making under uncertainty. It is considered that the stochastic learning automata approach, and interactive decision making in automata games will provide the fundamental framework for a model of decentralised decision making in complex systems. This approach constructs models which result from interconnecting many of the automaton-environment pairs in simple ways. In such systems, the decision makers (modelled as automata) update their actions using learning schemes on the basis of responses from many local environments, this gives rise to specific strategic games. Some games are easily analysed using results either known or derived from automata game theory, other interconnections lead to a structure for which automata behaviour is not currently known. The objective is to build models with both analytical tractability as well as providing realistic models.

The models studied in this section express typical ways in which decision makers can interact, by considering both feedback and hierarchical structures. Particular emphasis has been given to feedback structures in the form of synchronous models. In such models, the actions of all automata occur simultaneously, as do the subsequent responses. A sequential model is also considered, where there are various levels of automata and there is

interaction between different levels. In such systems, one automaton acts at a time with the action chosen determining which automaton acts next.

4.4 Synchronous Models

These models represent ways in which decision makers interact. For such models, the time instants when the automata choose actions and update probabilities are synchronised. The concepts of game theory are used to analyse the convergence of the learning schemes. The relevant concepts of game theory have been provided in Appendix Two. Figure 4.2 and Figure 4.4 show some simple examples of synchronous models. These basic structures can be modified to include examples in which the actions of the automata determine specific types of nonstationary environments and they are discussed in Section 4.7. A description of each model is presented in the following subsections.

Simulation results are provided for each model in the subsequent sections. The results for all decision making models - synchronous and sequential are presented in the form of tables and graphs. It is assumed that all the feedback models use L_{R-I} algorithms of the form, Equation 3.3; unless otherwise stated. In all cases 'a' is the reward parameter; $p_1(n)$ and $q_1(n)$ (and $r_1(n)$ in the three player examples) are probabilities for selecting the first action A_1 and A_2 (and A_3), respectively. The game matrix uses reward probabilities to represent the game structure of each environment, in the case of synchronous models. The number of sample paths over which averages were taken, is related to 'm'. A single iteration computes one loop

at each stage 'n'. Expected values are denoted by, eg $p_1(n) = E[p_1(n)]$. The general structure of each simulation program is provided in Appendix One.

4.5 Simple Feedback

The most basic feedback arrangement shown in Figure 4.2 interconnects two automaton-environment pairs $A_1 - E_1$ and $A_2 - E_2$. Each automaton A_1 and A_2 are assumed to have two actions interacting into their respective binary environments at each stage 'n'. The main feature exhibited by this model is that the response $\beta(n)$ from one automaton's environment is the input to the other automaton.

The synchronous nature of the model can be viewed as a standard automata game which can be represented by the following game matrix. The notation α_j^i implies that an action α_j is selected by automaton i ; similarly, the environment success probabilities are given by d_j^i where i and j are the automaton and action indices, respectively.

$$\tau = \begin{matrix} & \alpha_1^2 & \alpha_2^2 \\ \alpha_1^1 & \left(d_1^2, d_1^1 \right) & \left(d_2^2, d_1^1 \right) \\ \alpha_2^1 & \left(d_1^2, d_2^1 \right) & \left(d_2^2, d_2^1 \right) \end{matrix} \quad (4.1)$$

The strategies of A_1 and A_2 correspond to the rows and columns, respectively of τ . Each ordered pair of the game matrix represents the expectation of success (reward probability, since reward = 1 and penalty = 0) for A_1 and A_2 resulting from the corresponding strategy pair. It is evident from the game matrix that all four strategy pairs are equilibria. If $d_1^i > d_2^i$ ($i = 1, 2$),

2), it is also true that (α_1^1, α_1^2) is the only Pareto optimal play. Note that irrespective of the action selected by A_2 , A_1 's action is equally good and vice versa.

4.5.1 Simulation - Simple Feedback

Table 4.1a and Table 4.1b display the simulation results for two different initial probabilities and their corresponding learning curves are provided in Figure 4.3a and Figure 4.3b. It is evident from both simulations that, the action probabilities for automaton A_1 and A_2 fluctuate close to the initial probability value. This confirms that irrespective of the choice of initial probability all action probabilities remain close to their initial values, independently of the reward parameter and number of sample paths. Recall that in this model, Figure 4.2, the response from one automata's environment E_1 is the input to another automaton. Since direct feedback of responses into the original automaton does not occur, the action probabilities do not show convergence behaviour. This indicates that learning has not been performed.

4.6 Weighted Feedback

In Figure 4.4 each automaton receives responses from two environments, this results in a more involved game. For the N-automata case, the weighting factor $w_i = (w_{i1}, w_{i2}, \dots, w_{iN})$, $\sum_{j=1}^N w_{ij} = 1$ is associated with each response output from E_i . This produces a normalised scalar input to each automaton A_j . In the weighted feedback model $\tilde{\beta}^i \varepsilon(0, 1)$ is the response of environment E_i , while $\beta_i \varepsilon(0, w_{i1}, \dots, w_{iN}, 1)$ is the normalised scalar input to

A_i . If the responses are weighted equally then the input is the number of successes divided by the total number of responses, as in the multi-environment model for the single automaton. Zero weights imply feedback from only some environments. The effective environment success probabilities are defined as:

$$\delta_{j_1 j_2 k} \triangleq \Pr(\beta(n) = \beta_k / \alpha_1(n) = \alpha_{j_1}^1, \alpha_2(n) = \alpha_{j_2}^2) \quad k = 1, 2, \dots, K \quad (4.2)$$

where β_k is the k^{th} element of the input set to each automaton. If equal weights are considered for Figure 4.4, then $K=3$ and the input set is (0, 0.5, 1). The expected value of A_i 's input conditioned on the action choices is:

$$s_{j_1 j_2} \triangleq E[\beta(n) / \alpha_1(n) = \alpha_{j_1}^1, \alpha_2(n) = \alpha_{j_2}^2] = \sum_{k=1}^3 \beta_k \delta_{j_1 j_2 k} \quad (4.3)$$

Since $s_{j_1 j_2}$ is analogous to $d_{j_1 j_2}$ in the automata game formulation; it can be used to construct an identical payoff game with the game matrix having elements $s_{j_1 j_2}$. These values represent the environment reward probabilities of E_1 and E_2 , as shown below:

$$s_{11} = \sum_{k=1}^3 \beta_k \delta_{11k} = \frac{1}{2}[d_1^1(1 - d_1^2) + d_1^2(1 - d_1^1)] + d_1^1 d_1^2 = \frac{1}{2}(d_1^1 + d_1^2) \quad (4.4)$$

The common factor of $\frac{1}{2}$ maybe omitted to provide the following identical payoff game matrix:

$$\tau = \begin{matrix} & \alpha_1^2 & \alpha_2^2 \\ \alpha_1^1 & \left(\begin{matrix} d_1^2 + d_1^1 & d_2^2 + d_1^1 \end{matrix} \right) \\ \alpha_2^1 & \left(\begin{matrix} d_1^2 + d_2^1 & d_2^2 + d_2^1 \end{matrix} \right) \end{matrix} \quad (4.5)$$

Assuming that $d_1^1 > d_2^1$ and $d_1^2 > d_2^2$, it follows that $d_1^1 + d_1^2$ is the largest element of τ and $d_2^1 + d_2^2$ is the smallest. It is apparent that (α_1^1, α_1^2) is the only pure strategy equilibrium. However, τ has even more structure. Each player has a dominant strategy α_1^1 and α_1^2 , which is better than any other strategy regardless of what the other player does.

4.6.1 Simulation – Weighted Feedback

The simulation result for the weighted feedback model is provided in Table 4.2, each automaton has three actions with the same initial probabilities and a different weighting on the two responses. Since, the automata receives responses from two environments, a normalised scalar input is received by each A_i . In this case, all action probabilities are updated by using the generalised version of Linear Reward/ Inaction algorithm, as stated in Equation 3.4. The results indicate that each automata's first strategy corresponds to the best action (which relates to the highest reward probability) in its local environment, then $(\alpha_1^1, \alpha_1^2, \dots, \alpha_1^N)$ is the set of dominant strategies. As mentioned, the set of dominant strategies is denoted by (α_1^1, α_1^2) which is better than any other strategy regardless of what the other player does. This example shows that dominance holds for multiple automata in which the automaton A_i has r_i actions. Figure 4.5 presents the graphical illustration for the tabulated results.

4.7 Synchronous Models: Actions Determine Environment

This section presents two types of interconnections as shown in Figure

4.6 and Figure 4.9. These interconnections are modifications of the basic structure in which the actions selected by the automaton determine specific types of nonstationary environments.

4.8 Interconnection 1: A_1 Determines A_2 's Environment

No Coordination

The feedback configuration as shown in Figure 4.6, indicates that the actions selected by automaton A_1 determines the stationary random environment E_1^2 or E_2^2 that is observed by A_2 , but the actions from A_2 do not influence A_1 . In this model there is no coordination of A_1 and A_2 ; automaton A_1 receives responses from environment E_1 whilst A_2 interacts with E_1^2 or E_2^2 , therefore the response received by each player is different at each time instant. The game structure can be represented by the following game matrix:

$$\tau = \begin{matrix} & \alpha_1^2 & \alpha_2^2 \\ \alpha_1^1 \left(\begin{matrix} d_1, d_{11} & d_1, d_{12} \\ d_2, d_{21} & d_2, d_{22} \end{matrix} \right) & & \end{matrix} \quad (4.6)$$

Clearly, A_1 will converge to its best action independently of A_2 , while A_2 converges to its best action in the environment that A_1 has determined. Note that it is possible that in optimising for itself A_1 prevents A_2 from receiving its optimal payoff.

Coordination

In contrast to the configuration mentioned above, each player (automata) receives the same input at each time instant. This represents a coordinated structure, such that the actions selected by automata A_2 also determines

the environment. By coordination of A_1 and A_2 the game matrix may be modified to provide the sum of the payoffs as the input to each player. Thus the game matrix τ_1^* can be thought of as having the game structure:

$$\tau_1^* = \begin{matrix} & \alpha_1^2 & \alpha_2^2 \\ \alpha_1^1 & \left(\begin{matrix} d_1 + d_{11} & d_1 + d_{12} \end{matrix} \right) \\ \alpha_2^1 & \left(\begin{matrix} d_2 + d_{21} & d_2 + d_{22} \end{matrix} \right) \end{matrix} \quad (4.7)$$

(omitting a factor of $\frac{1}{2}$). If this coordination τ_1^* , has a unique equilibrium then at least one player must have a dominant strategy. Thus, the automata will converge to the equilibrium with probability arbitrarily close to one. This may result in A_1 selecting the action that was the worst without coordination. If A_2 does not have a dominant strategy in τ_1 , it is possible that τ_1^* may have two equilibria.

4.8.1 Simulation – Interconnection 1

In this model, the actions of the automata determine responses from the environment. The results in Table 4.3a – Table 4.3d consider a two-state automaton, without coordination between automata A_1 and A_2 ; and examine a model which involves coordination. A graphical representation of each table is also included in Figure 4.7a - Figure 4.7d, respectively.

No Coordination

Table 4.3a presents simulation results for interconnection 1 with one equilibrium. It can be seen that the strategy pair (α_1^1, α_1^2) is the unique equilibrium and Pareto optimal play. The results obtained confirm that the action prob-

ability for automata A_1 converges to its best action independently of A_2 ; and A_2 receives optimal payoff and both automata converge to the equilibrium. However, this does not apply when two equilibria exist as shown in Table 4.3b. In this case (α_1^1, α_1^2) is still the unique equilibrium but (α_1^2, α_2^2) is also Pareto optimal (joint maximum). Since A_1 's best action does not imply that A_2 receives its optimal payoff, then the convergence of A_2 is slower in comparison to A_1 .

Coordination

Table 4.3c and Table 4.3d present simulation results for Interconnection 1 with coordination. The tables illustrate convergence behaviour when the game matrix τ_1^* , has a unique equilibrium and two equilibria. Note that each table uses the same environments as Table 4.3a and Table 4.3b; but the game structure is modified by τ_1^* . In the unique equilibrium case, Table 4.3c, it can be seen that both automaton steadily converge to the equilibrium. However, in the two equilibrium case the action probabilities for each automaton are decreasing in value, showing that for specific initial conditions the drift is towards the global optimum. If however, the players have more than two actions, the point of equal initial probabilities may not have this property. The example in Table 4.4b suggests this for the three action case with coordination.

Simulation results are also included for the players having three actions each, shown in Table 4.4a and Table 4.4b; the corresponding learning curves are presented in Figure 4.8a and Figure 4.8b.

4.9 Interconnection 2: A_1 Determines Game for A_2 and A_3

Figure 4.9, represents interaction of automata in which the actions selected by A_1 determine the environment E_1 or E_2 (now a game) seen by A_2 and A_3 . In this case A_1 may be thought of as a coordinator without its own environment whose actions produce uncertain results. The objective of A_1 is to maximise the weighted sum of the payoffs to A_2 and A_3 . The environments, E_i , can be expressed as follows:

$$E_i = (d_{jk}^{i1}, d_{jk}^{i2}) \quad (4.8)$$

the payoffs, M^i , to each player is given by

$$M^1 = \frac{1}{2}(d_{jk}^{i1} + d_{jk}^{i2}); M^2 = d_{jk}^{i1}; M^3 = d_{jk}^{i2} \quad (4.9)$$

where i, j and k are the actions of A_1, A_2 and A_3 , respectively. In general many equilibria can exist. Even in the identical payoff case:

$$d_{jk}^{i1} = d_{jk}^{i2} = d_{jk}^{i3} \quad (4.10)$$

if

$$d_{11}^1 > d_{22}^1 > d_{12}^2 > d_{21}^2 > d_{12}^1 \geq d_{21}^1 \geq d_{11}^2 \geq d_{22}^2 \quad (4.11)$$

then $d_{11}^1, d_{22}^1, d_{12}^2$ and d_{21}^2 all correspond to equilibria. As mentioned for Interconnection 1, the theory is incomplete when there are many equilibria.

4.9.1 Simulation – Interconnection 2:

This simulation is based on Figure 4.9, the action selected by A_1 determines the same response for A_2 and A_3 . To simulate this structure,

each response produced by environment (E_1, E_2) has an associated weighting factor. Each response β^2, β^3 are weighted equally - $w_1 = (w_{11} = 0.5, w_{12} = 0.5)$. Hence, a normalised scalar input $\beta^1 \varepsilon(0, 0.5, 1)$ is received by automaton A_1 . In this example A_1 , now uses the general environment L_{RI} scheme for updating probability vectors. The results for Table 4.5 are produced using the following equilibria and corresponding payoffs:

$$(1, 1, 1)(d_{11}^1 = 0.9); (1, 2, 2)(d_{22}^1 = 0.7); (2, 1, 2)(d_{12}^2 = 0.5); (2, 2, 1)(d_{21}^2 = 0.3) \quad (4.12)$$

This model shows that the automata converges to the largest equilibrium in the N-automata case if each has two actions, starting with equal action probabilities. From Table 4.5, it can be seen that the rate of convergence for action probabilities corresponding to automaton A_2 and A_3 are within close approximation to each other, since both automata receive identical responses from the environment. A graphical representation is included in Figure 4.10. It is evident that the learning curves for the action probabilities which correspond to automaton A_2 and A_3 coincide with each other. In comparison there is a rapid convergence to unity for automaton A_1 action probability vector.

4.10 Sequential Models

The hierarchical structure stochastic automata system may represent many important realistic situations. In such a case, collections of automata are organised to model the behaviour of a hierarchical learning system where learning proceeds at a number of distinct levels with each level capable of

eliciting a response from the environment. This concept was introduced by Thathachar and Ramakrishnan, [47], [48]. They proposed a simple modification of the absolutely expedient algorithm, which provided a reinforcement scheme for a hierarchical system of automata. This approach significantly reduced the high dimensionality problem associated with a learning automaton. Further research efforts resolve this problem by considering a reorganisation scheme that uses ϵ -optimal learning automata to heuristically select hierarchical structures with minimal computation, [49]. The learning behaviours of the generalised sequential model operating in the multi-teacher environment were also considered, [50].

A sequential model is depicted in Figure 4.11, Figure 4.14 and Figure 4.17. In these models only one decision maker acts at any time, such that a sequence of decisions propagate down the tree structure and the bottom level automata produces a response from the environment which is fed back to all automata responsible for the selected path. It is possible to analyse sequential models as networks of decision makers in which control passes from node to node. The nodes in a sequential model can represent a synchronous model, so that a more general model can be produced which includes both types of structures. Three types of network structures are briefly described.

The following sections also present computer simulation results for the sequential models. In the simulation study, a three-level hierarchical system with each automaton having two actions are examined. Such systems are in the form of a tree structure and directed network. To simulate these structures, the penalty probabilities in the environment were selected from the

range [0.5, 0.95], except the unique minimum penalty probability which was set to 0.1. The L_{RI} scheme was adopted to update action probabilities for the optimal path. Similar to the previous notation 'a' indicates the reward parameter; the total number of experiments is given by 'm' and the expected values are denoted by, eg $p_1(n) = E[p_1(n)]$. Two sets of simulations were performed for each tree structure, the results are produced in both table and graph format.

4.11 Tree Structure

A tree structure is a multilevel system of automata consisting of several levels, each comprising of many automata. There is a definite order in which the automata can act. Each action of an automaton at a certain level, selects automata at the next lower level. Figure 4.11 illustrates, automata arranged in three levels. The hierarchy consists of a single automata at the first level, two in the second level and four in the third level. Each automaton has two actions. Considering the structure, A_0 acts first, choosing either A_1 or A_2 , who then acts to select an automaton at the next lower level. The action selected at the lowest level, generates a response from a stationary random environment. The action probabilities on the selected path are updated on the basis of this response.

4.11.1 Simulation - Tree Structure

Table 4.6 provides the penalty probabilities of the environment which are used for each simulation. Note that the set of penalty probabilities are

different for each simulation. The location of the unique minimum penalty probability has been changed, whilst all other penalty probabilities remain unchanged.

Consider the three level hierarchical system in Figure 4.12, the optimal path probability changes for Table 4.6 can be easily analysed. It is shown that all action paths associated with the unique minimum penalty probability converge close to unity. Considering the penalty probabilities in the Table 4.6a observe that the optimal path is 001. This notation implies that path 0 is selected by level 1 and level 2 automaton; and path 1 is selected by automaton in level 3. However, Table 4.6b selects an alternative route in determining the optimal path. In this case the action probabilities converge close to unity by selecting path 100; since the unique minimum penalty probability is associated with this path. Graphs for each table are presented in Figure 4.13a and Figure 4.13b.

4.12 Directed Network

In comparison to the previous model, the network of Figure 4.14 also represents a three-level hierarchical system with two actions per automaton. It also illustrates a case in which control always passes back to A_0 at the end of each cycle. However, in this structure the automaton in the second level may select any automaton in the lower level. It is possible to use many forms of updating schemes since local responses occur at different times for different levels.

4.12.1 Simulation – Directed Network

Similarly, this presents computer simulation results for sequential model in the form of a directed network. The results in Table 4.7 examine the learning behaviour of a directed network by adopting the same techniques as mentioned previously.

Note that in the case of a directed network, the action path probabilities may converge to the unique minimum by selecting alternative routes in the first and second level. Both results in Table 4.7 confirm that the optimal path associated with the unique minimum penalty probability converge close to unity. Figure 4.15, clearly indicates the different routes which may be selected in determining the optimal path for each case. For this network the number of times a particular path converges to the unique minimum was also evaluated. Thus, Table 4.7a converges to the unique minimum by selecting a combination of paths 001 and 101, in the ratio of 0.3:0.2 from a total of fifty experiments. In Table 4.7b, the optimal path is determined by selecting a combination of paths 010 and 110 in a ratio of 0.5:0.5 from fifty experiments.

The results in Table 4.7 are produced in the form of graphs, Figure 4.16a and Figure 4.16b, respectively.

4.13 General Network

In contrast to the previous sequential models, Figure 4.17 illustrates a general network. In this model an action sequence does not exist giving rise to a path through the network. It is possible for any automata to select any other automata. In the network as shown, each automaton has three

actions, which corresponds to the selection of one of the automata. The action selected generates a random response which may be received by one or many automata.

In such a network, certain issues need to be resolved; who receives what information and who is assigned to carry out which decisions. The general network exhibits the characteristics of a decentralised system, since the information may be collected from many sources, distributed to appropriate units in the organisation for processing and then used by selected nodes to reach a suitable decision. At this stage, the stochastic learning automaton approach was considered to be limited in modelling capability for arbitrary topologies of decision models. To resolve this limitation, the potential of Petri Nets for modelling complex systems are presented in the next chapter.

4.14 Conclusion and Summary

This chapter has shown how the stochastic learning automaton model may be considered as a basic framework for a model of decentralised decision making. It has introduced models in which many automaton-environment pairs are interconnected in various ways to achieve desirable global performance, even though decisions are made on the basis of simple updating schemes. The models that have been studied, include synchronous and sequential models.

The synchronous models represent feedback configurations in which the actions of all decision makers are synchronised. Such models give rise to corresponding automata games, of particular interest is the interconnection of decision makers which lead to very different game structures. It is

important to analyse how the corresponding game changes as interconnections are varied. In some cases, concepts of learning theory and automata games may be used to analyse the behaviour of a particular model. Often, however the interconnections may be difficult to analyse, which indicates the necessity for further research.

The learning behaviour of the hierarchical system of automata in the form of a tree structure and directed network has also been considered. Simulation results indicate a rapid convergence for the optimal path. Further research has shown that a modified algorithm for this structure is required when the number of actions is large. This chapter concludes by describing a general network which resembles a decentralised system. The basic conceptual framework based on the stochastic learning automaton approach is restricted in its modelling capabilities for the representation of such systems. The limitations in the modelling technique is evident, since it lacks structure, flexibility and the ability to demonstrate the explicit interactions between the various agents in the network. These shortcomings have established the need for a more high-level modelling framework. A later Chapter Six presents novel work which incorporates stochastic learning automata as described in Chapter Three with a graphical modelling concept based on Petri net theory to overcome these limitations.

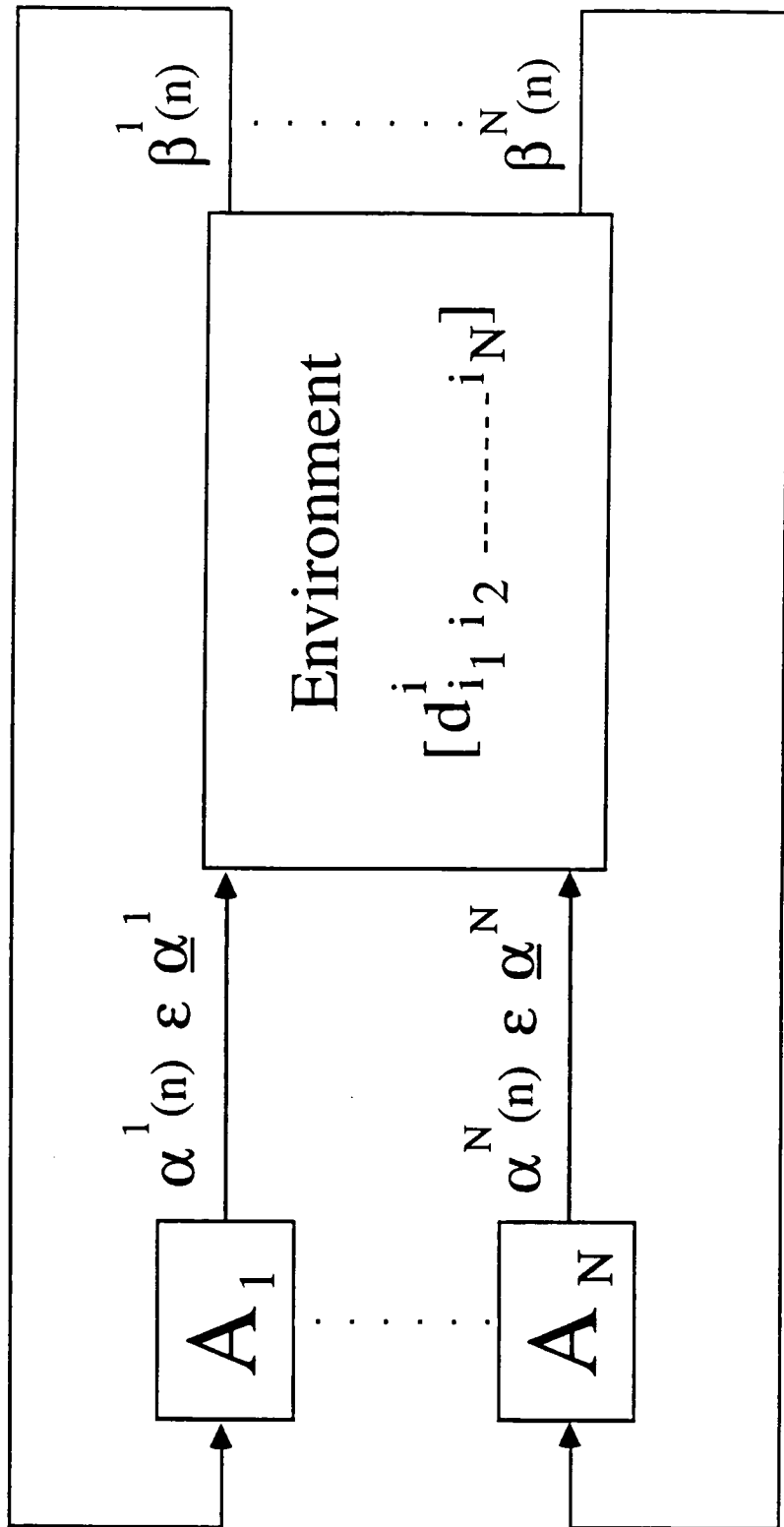
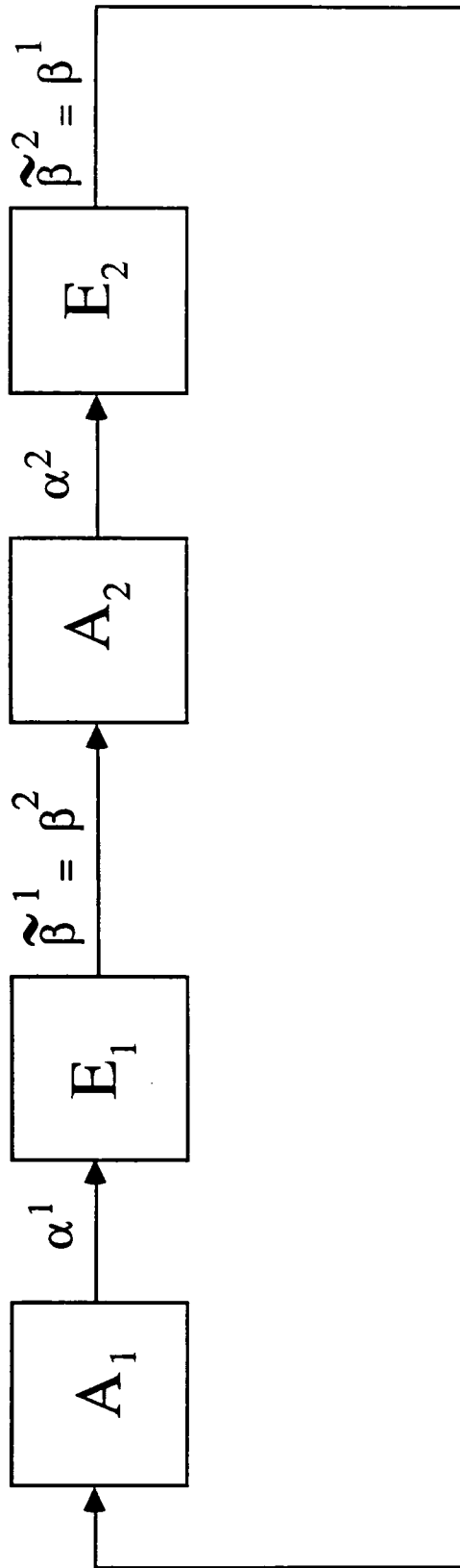


Figure 4.1 - Automata Game Schematic



Simple Feedback

Figure 4.2 - Synchronous Models - The Basic Structure

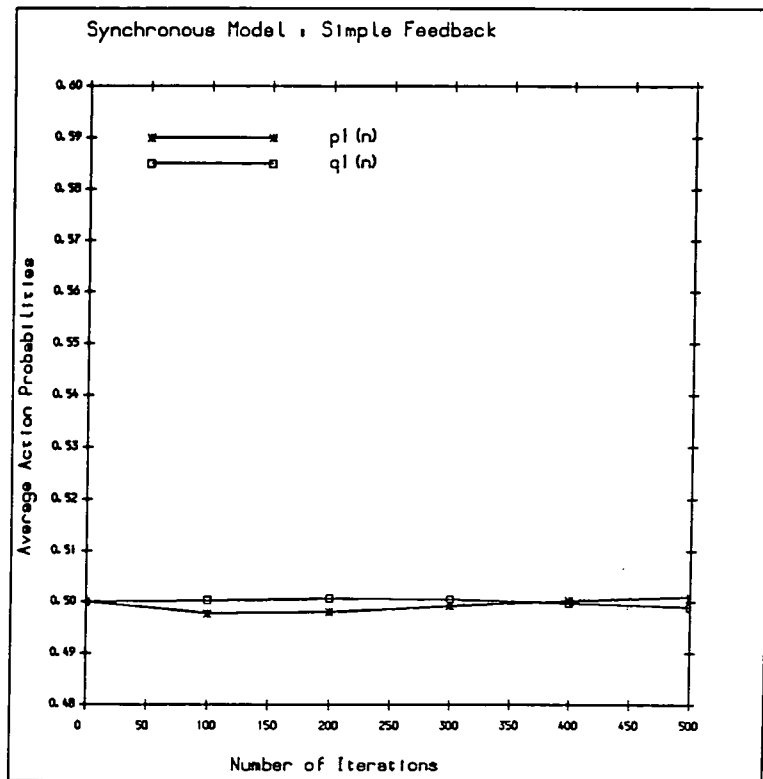
$$E_1 = \{.6, .1\}, \quad E_2 = \{.7, .2\}$$

$$\tau = \begin{pmatrix} .7, .6 & .2, .6 \\ .7, .1 & .2, .1 \end{pmatrix}$$

a = 0.01, m = 100		
n	p1(n)	q1(n)
0	0.500000	0.500000
100	0.497706	0.500284
200	0.498057	0.500694
300	0.499280	0.500533
400	0.500284	0.499794
500	0.500994	0.499016

(a) Action Probability Pr [0.5]

Table 4.1 – Simulation of Simple Feedback (Figure 4.2)



(a) Action Probability Pr [0.5]

Figure 4.3 – Average Action Probability vs Iterations : (Table 4.1a)

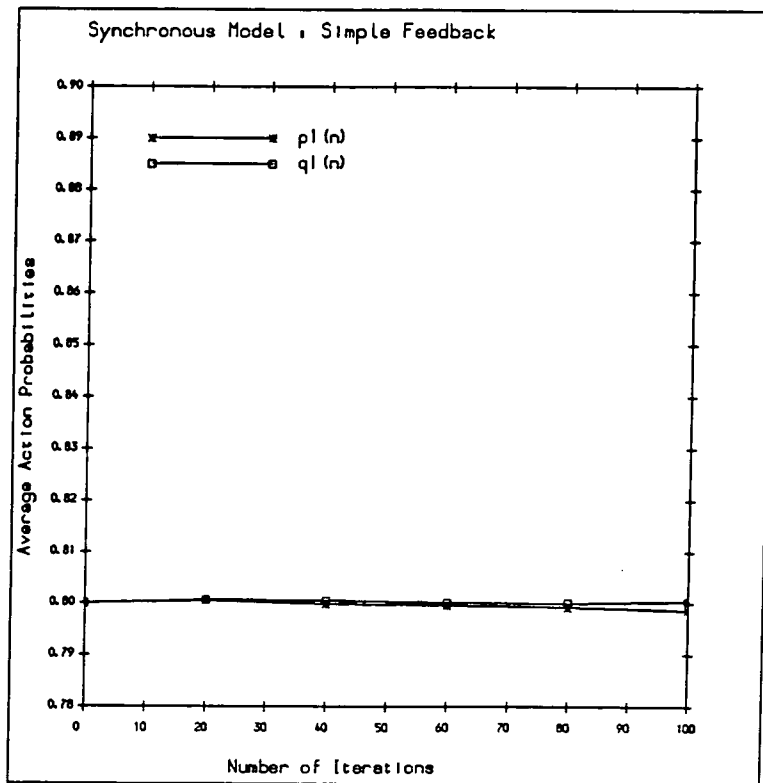
$$E_1 = \{.6, .1\}, \quad E_2 = \{.7, .2\}$$

$$\tau = \begin{pmatrix} .7, .6 & .2, .6 \\ .7, .1 & .2, .1 \end{pmatrix}$$

a = 0.02, m = 200		
n	p1(n)	q1(n)
0	0.800000	0.800000
20	0.800580	0.800623
40	0.799841	0.800518
60	0.799606	0.800128
80	0.799242	0.800030
100	0.798685	0.800361

(b) Action Probability Pr [0.8]

Table 4.1 - Simulation of Simple Feedback (Figure 4.2)



(b) Action Probability Pr [0.8]

Figure 4.3 - Average Action Probability vs Iterations : (Table 4.1b)

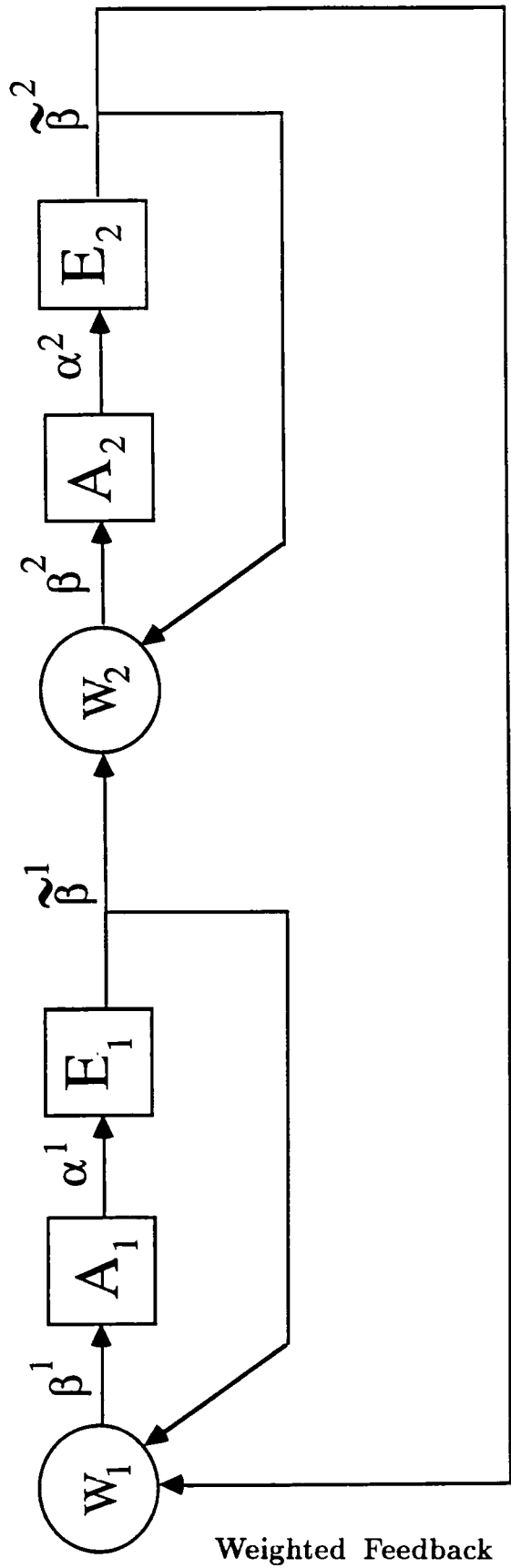


Figure 4.4 - Synchronous Models - The Basic Structure

$$E_1 = \{.8, .5, .3\}, \quad E_2 = \{.9, .1, .7\}$$

$$w_1 = \{.4, .6\}, \quad w_2 = \{.6, .4\}$$

$$\tau = \begin{pmatrix} .86, .84 & .38, .52 & .74, .76 \\ .74, .66 & .26, .34 & .62, .58 \\ .66, .54 & .18, .22 & .54, .46 \end{pmatrix}$$

a = 0.04, m = 50						
n	p1(n)	p2(n)	p3(n)	q1(n)	q2(n)	q3(n)
0	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333
200	0.553642	0.178255	0.267103	0.519196	0.147588	0.332215
400	0.750079	0.100287	0.148634	0.727566	0.076755	0.194679
600	0.832411	0.067227	0.099362	0.812883	0.051300	0.134817
800	0.873850	0.050505	0.074646	0.859174	0.038539	0.101287
1000	0.898755	0.040454	0.059791	0.887000	0.030870	0.081131

Table 4.2 - Simulation of Weighted Feedback (Figure 4.4)

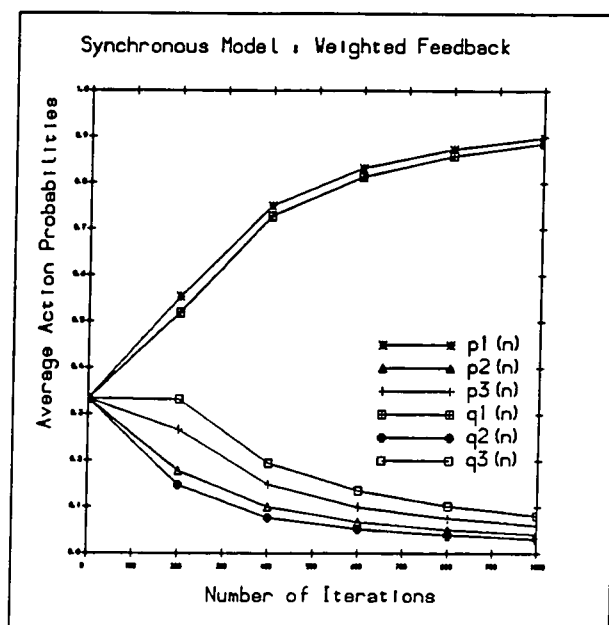
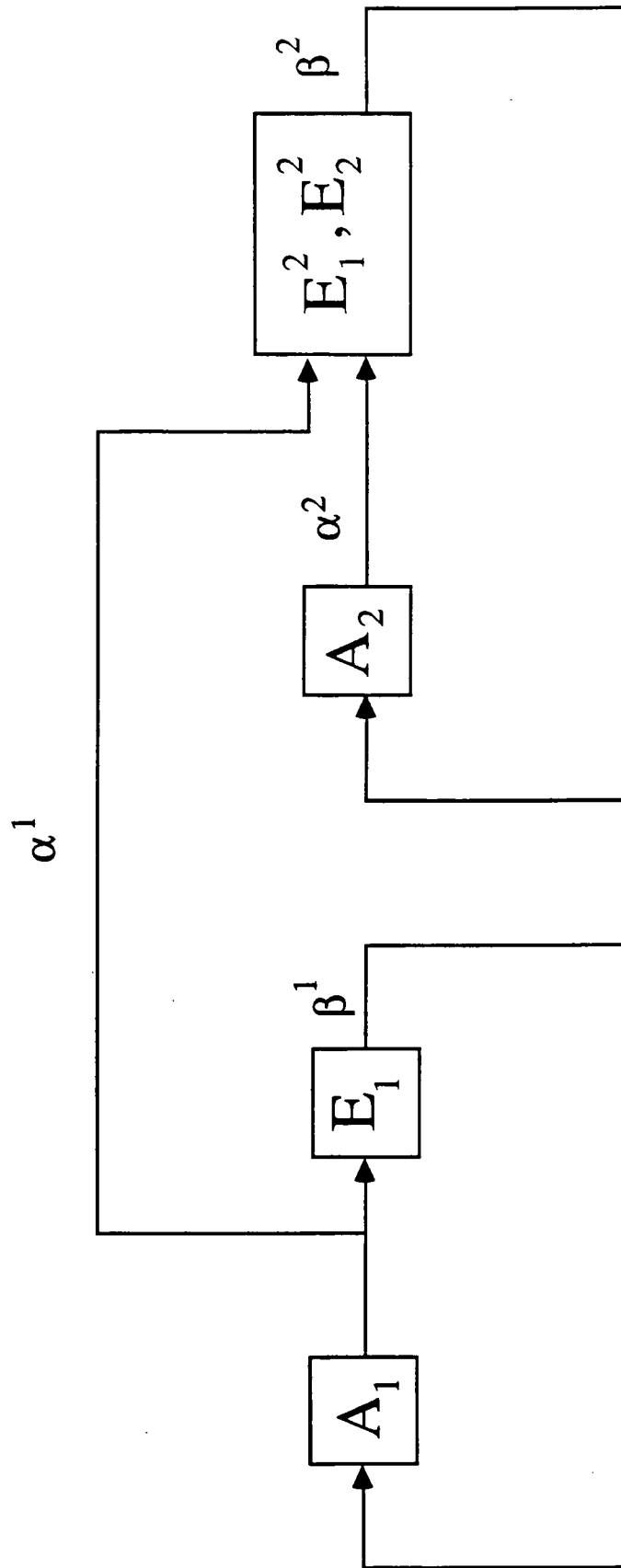


Figure 4.5 - Average Action Probability vs Iterations : (Table 4.2)



Interconnection 1: A_1 Determines A_2 's Environment

Figure 4.6 - Modification of Synchronous Models

$$E_1 = \{.8, .6\}, \quad E_1^2 = \{.8, .4\}$$

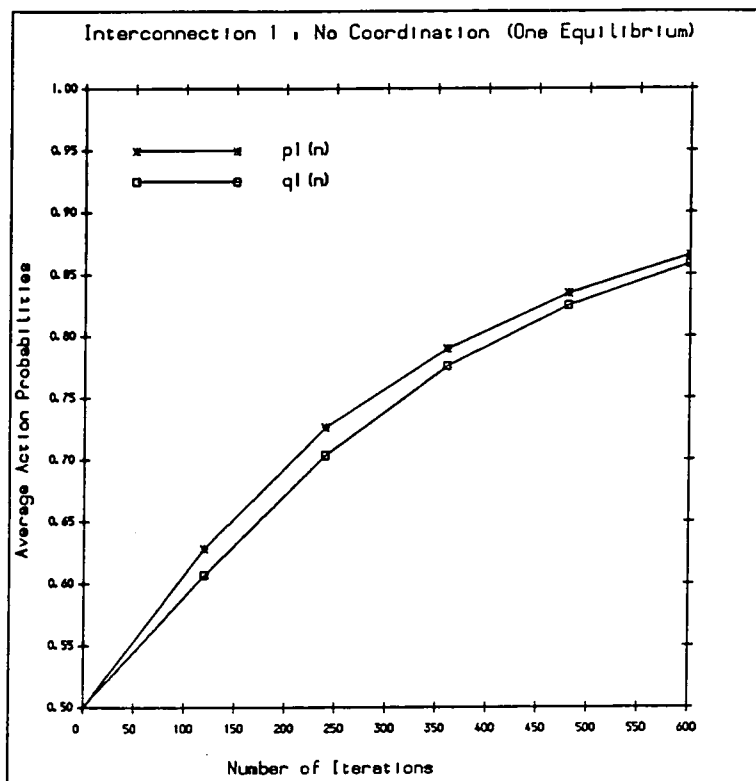
$$E_2^2 = \{.1, .3\}$$

$$\tau = \begin{pmatrix} .8, .8 & .8, .4 \\ .6, .1 & .6, .3 \end{pmatrix}$$

a = 0.04, m = 50		
n	p1(n)	q1(n)
0	0.500000	0.500000
120	0.628541	0.607027
240	0.726216	0.703727
360	0.789747	0.775936
480	0.835037	0.825118
600	0.865719	0.858582

(a) No Coordination (One Equilibrium)

Table 4.3 – Simulation of Interconnection 1 (Figure 4.6)



(a) No Coordination (One Equilibrium) : (Table 4.3a)

Figure 4.7 – Average Action Probability vs Iterations

$$E_1 = \{.8, .6\}, \quad E_1^2 = \{.3, .1\}$$

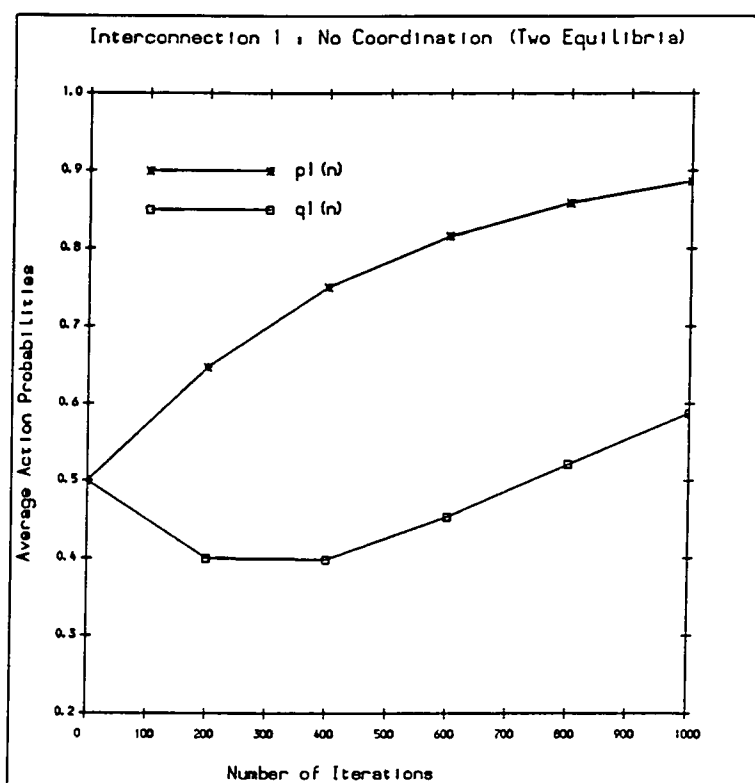
$$E_2 = \{.1, .8\}$$

$$\tau = \begin{pmatrix} .8, .3 & .8, .1 \\ .6, .1 & .6, .8 \end{pmatrix}$$

a = 0.03, m = 50		
n	p1(n)	q1(n)
0	0.500000	0.500000
200	0.647297	0.399764
400	0.750913	0.398302
600	0.816202	0.453326
800	0.859188	0.521408
1000	0.886665	0.587276

(b) No Coordination (Two Equilibria)

Table 4.3 – Simulation of Interconnection 1 (Figure 4.6)



(b) No Coordination (Two Equilibria) : (Table 4.3b)

Figure 4.7 – Average Action Probability vs Iterations

$$E_1 = \{.8, .6\}, \quad E_1^2 = \{.8, .4\}$$

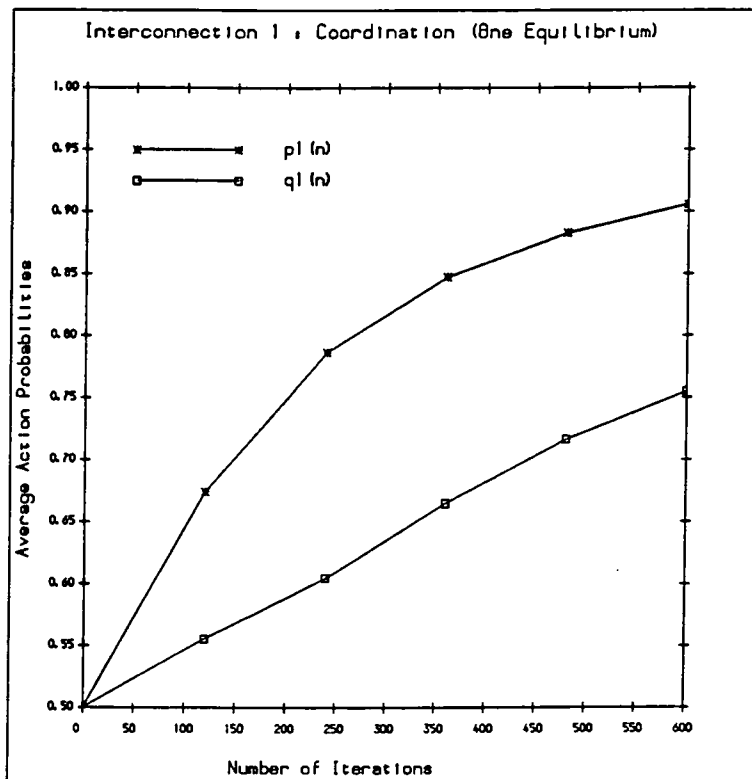
$$E_2^2 = \{.1, .3\}$$

$$\tau^* = \begin{pmatrix} .8 & .6 \\ .35 & .45 \end{pmatrix}$$

a = 0.04, m = 50		
n	p1(n)	q1(n)
0	0.500000	0.500000
200	0.674407	0.555802
400	0.786698	0.604913
600	0.847590	0.664721
800	0.883028	0.716545
1000	0.905703	0.754938

(c) Coordination (One Equilibrium)

Table 4.3 – Simulation of Interconnection 1 (Figure 4.6)



(c) Coordination (One Equilibrium) : (Table 4.3c)

Figure 4.7 – Average Action Probability vs Iterations

$$E_1 = \{.8, .6\}, \quad E_1^2 = \{.3, .1\}$$

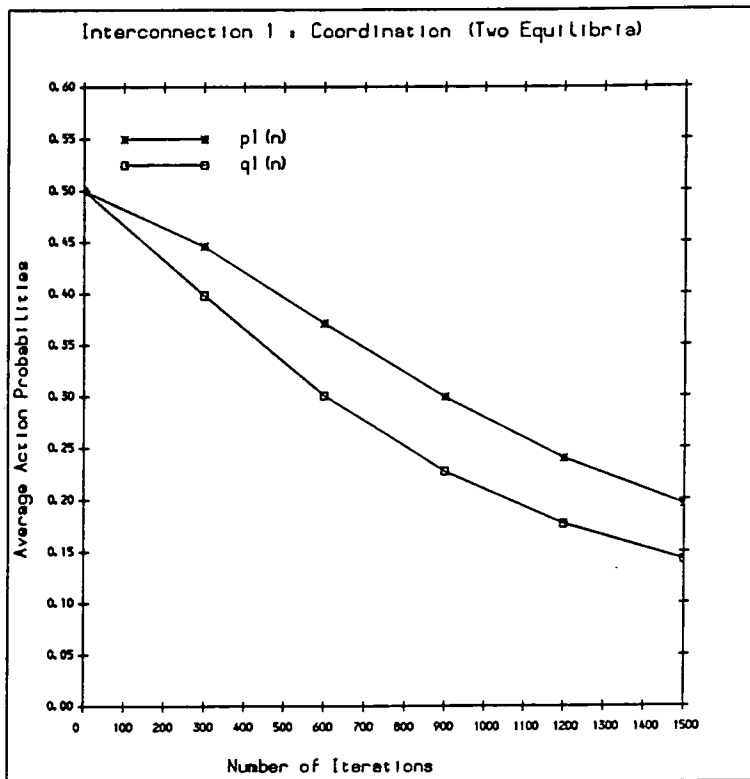
$$E_2^2 = \{.1, .8\}$$

$$\tau^* = \begin{pmatrix} .55 & .45 \\ .35 & .7 \end{pmatrix}$$

a = 0.02, m = 50		
n	p1(n)	q1(n)
0	0.500000	0.500000
200	0.445904	0.398251
400	0.371236	0.300475
600	0.299128	0.227108
800	0.239998	0.176754
1000	0.195622	0.142509

(d) Coordination (Two Equilibria)

Table 4.3 – Simulation of Interconnection 1 (Figure 4.6)



(d) Coordination (Two Equilibria) : (Table 4.3d)

Figure 4.7 – Average Action Probability vs Iterations

$$E_1 = \{.6, .1, .9\}$$

$$E_1^2 = \{.9, .1, .1\}$$

$$E_2^2 = \{.1, .9, .9\}$$

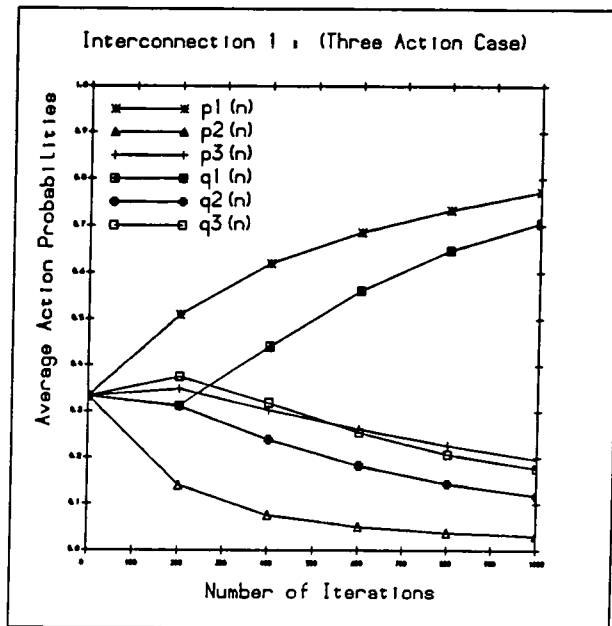
$$E_3^2 = \{.1, .8, .9\}$$

$$\tau = \begin{pmatrix} .6, .9 & .6, .1 & .6, .1 \\ .1, .1 & .1, .9 & .1, .9 \\ .5, .1 & .5, .8 & .5, .9 \end{pmatrix}$$

a = 0.03, m = 50						
n	p1(n)	p2(n)	p3(n)	q1(n)	q2(n)	q3(n)
0	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333
200	0.509682	0.140095	0.349223	0.312386	0.311682	0.374938
400	0.619705	0.075711	0.303584	0.441012	0.238897	0.319091
600	0.686459	0.050711	0.261797	0.561464	0.182801	0.254735
800	0.733834	0.038124	0.227042	0.648078	0.143385	0.207537
1000	0.772299	0.030538	0.196163	0.705207	0.116538	0.177255

(a) No Coordination

Table 4.4 – Simulation of Interconnection 1 : (Three Action Case)



(a) No Coordination : (Table 4.4a)

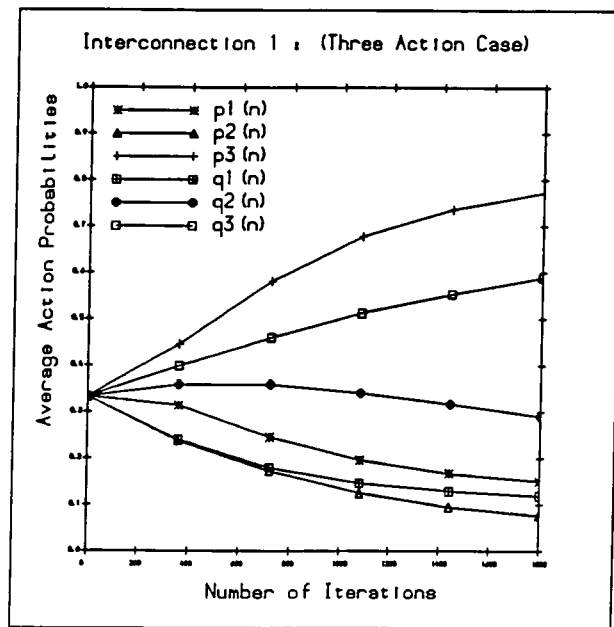
Figure 4.8 – Average Action Probability vs Iterations

$$\tau^* = \begin{pmatrix} .75 & .35 & .35 \\ .1 & .5 & .5 \\ .3 & .65 & .7 \end{pmatrix}$$

a = 0.02, m = 50						
n	p1(n)	p2(n)	p3(n)	q1(n)	q2(n)	q3(n)
0	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333
360	0.314742	0.237620	0.446638	0.240913	0.359054	0.399033
720	0.245473	0.172188	0.581339	0.179645	0.359128	0.460227
1080	0.195852	0.124719	0.678429	0.145706	0.340993	0.512301
1440	0.167442	0.095202	0.736356	0.128907	0.316816	0.553283
1800	0.150007	0.076333	0.772660	0.119028	0.290459	0.589513

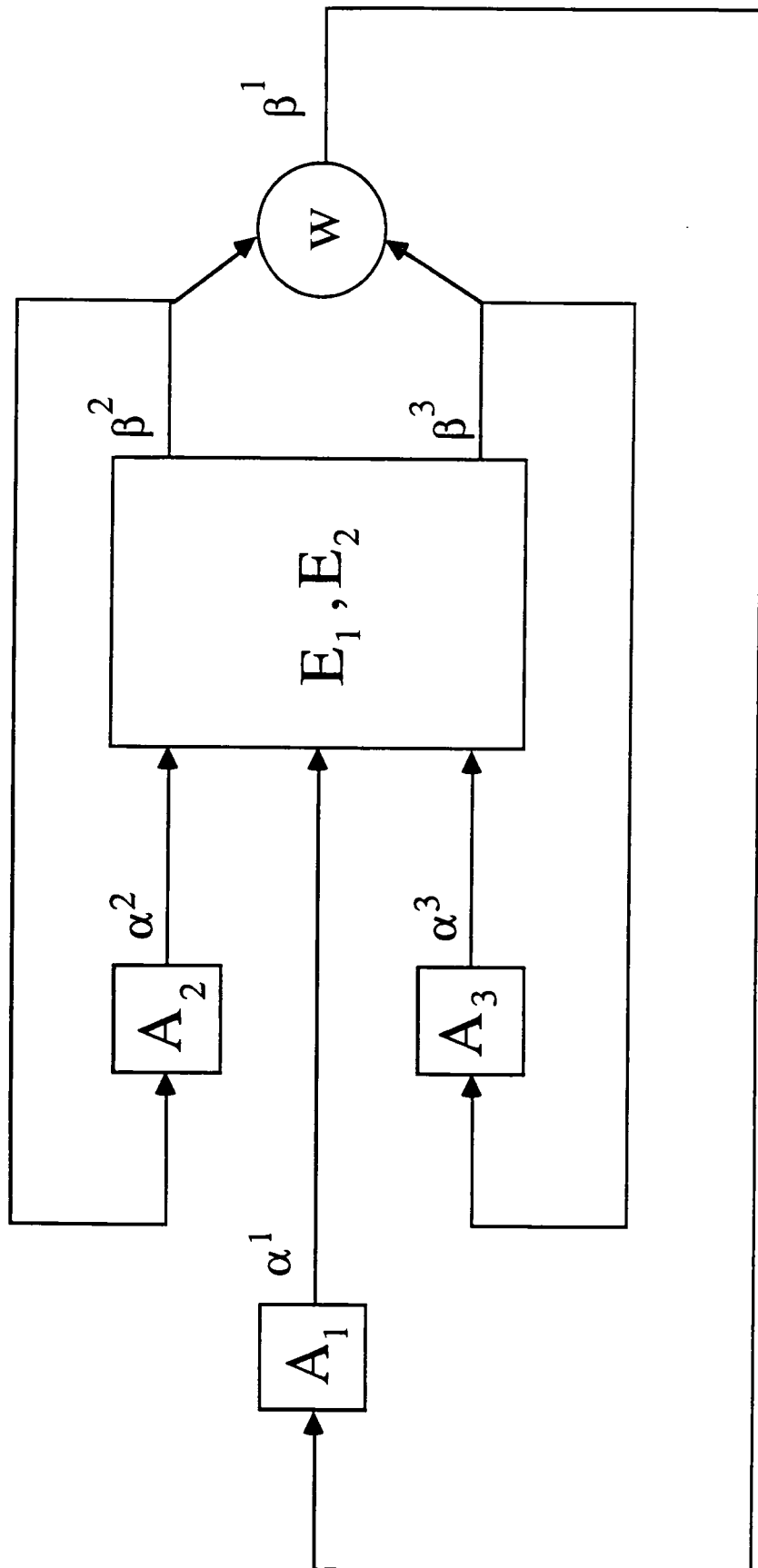
(b) Coordination

Table 4.4 – Simulation of Interconnection 1 : (Three Action Case)



(b) Coordination : (Table 4.4b)

Figure 4.8 – Average Action Probability vs Iterations



Interconnection 2: A_1 Determines Game for A_2 and A_3

Figure 4.9 – Modification of Synchronous Models

$$E_1 = \begin{pmatrix} .9 & .2 \\ .2 & .7 \end{pmatrix}$$

$$E_2 = \begin{pmatrix} .1 & .5 \\ .3 & .1 \end{pmatrix}$$

a = 0.01, m = 100			
n	p1(n)	q1(n)	r1(n)
0	0.500000	0.500000	0.500000
300	0.714258	0.546181	0.546508
600	0.823564	0.615667	0.616319
900	0.878749	0.696609	0.697244
1200	0.908693	0.759451	0.759969
1500	0.926874	0.801021	0.801443

Table 4.5 – Simulation of Interconnection 2 (Figure 4.9)

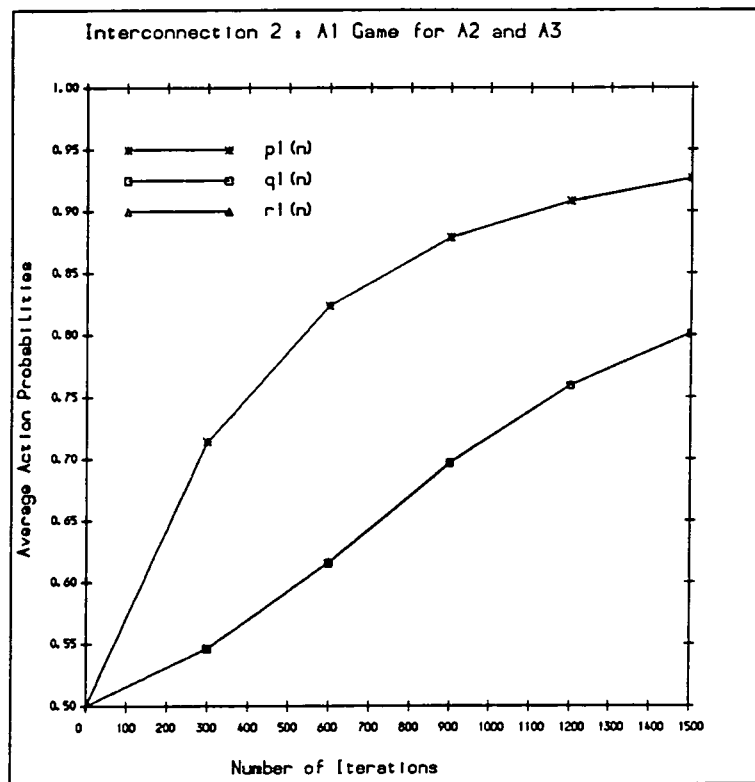
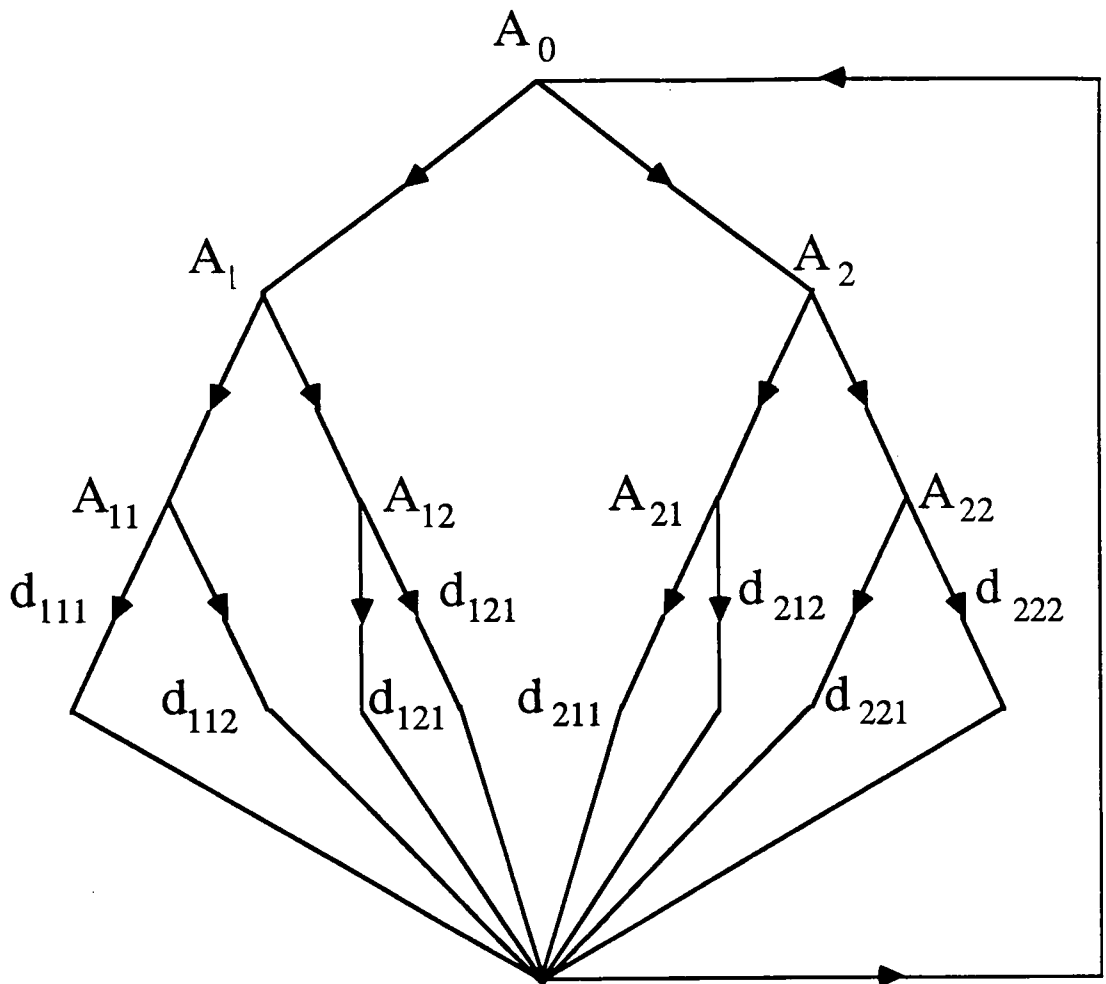


Figure 4.10 – Average Action Probability vs Iterations : (Table 4.5)



Tree Structure

Figure 4.11 - Sequential Models

$$E_1 = \{.9, .1, .7, .6, .8, .85, .95, .75\}$$

a = 0.1, m = 50			
n	p1(n)	q1(n)	r1(n)
0	0.500000	0.500000	0.500000
200	0.898385	0.855382	0.893066
400	0.949192	0.927690	0.946533
600	0.966128	0.951793	0.964355
800	0.974596	0.968845	0.973266
1000	0.979677	0.971076	0.978613

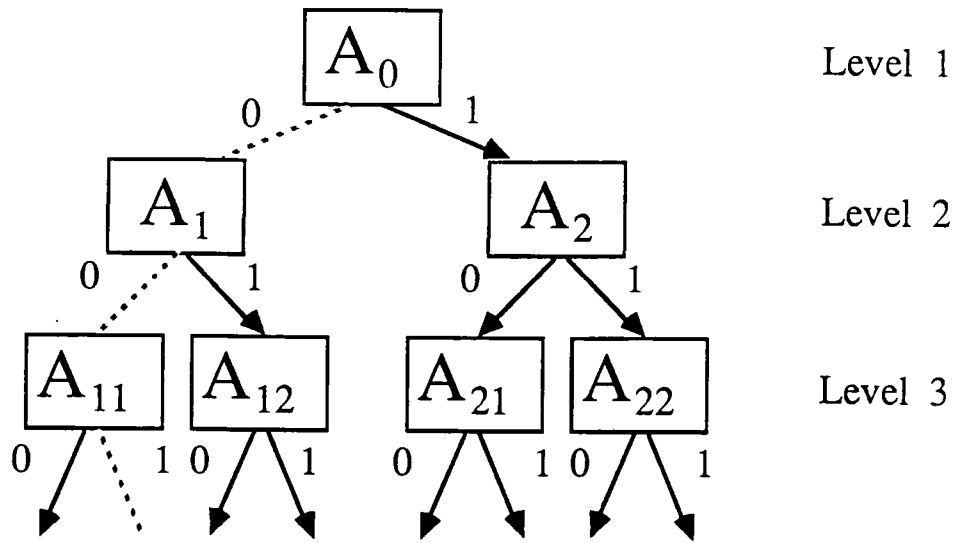
(a) Convergence Path To Minimum d112

$$E_1 = \{.9, .8, .7, .6, .1, .85, .95, .75\}$$

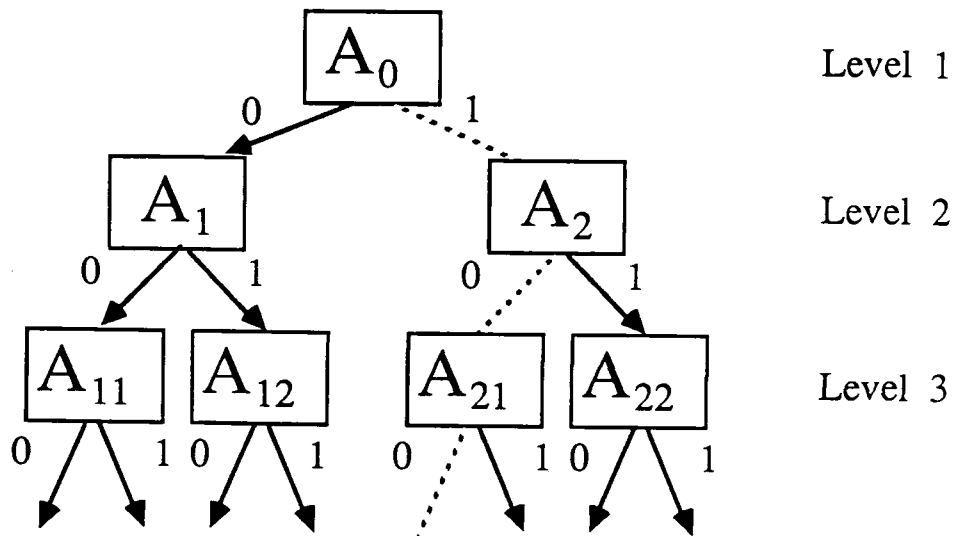
a = 0.1, m = 50			
n	p1(n)	q1(n)	r1(n)
0	0.500000	0.500000	0.500000
200	0.811304	0.862549	0.866206
400	0.886884	0.923761	0.925468
600	0.909790	0.944198	0.945247
800	0.922343	0.954417	0.955137
1000	0.929874	0.960548	0.961070

(b) Convergence Path To Minimum d211

Table 4.6 - Simulation of Tree Structure (Figure 4.11)

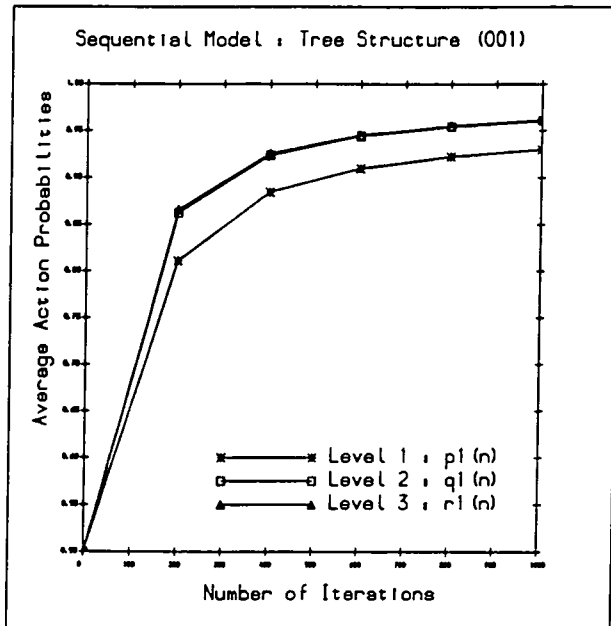


(a) Optimal Path 001

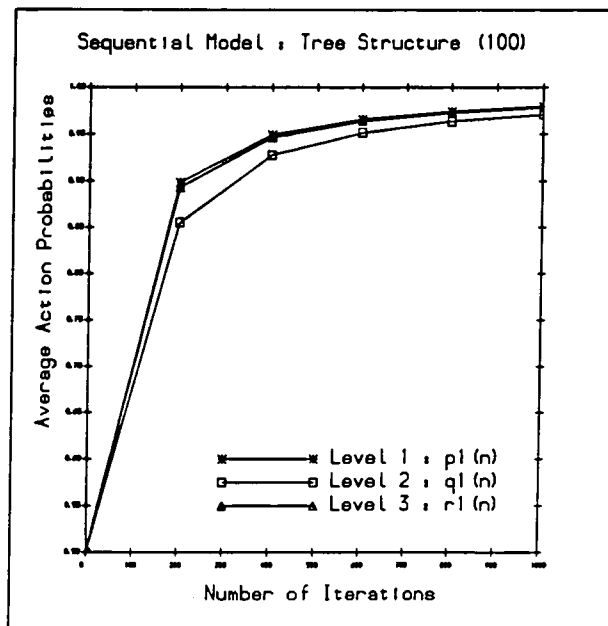


(b) Optimal Path 100

Figure 4.12 - Tree Structure : Selected Path

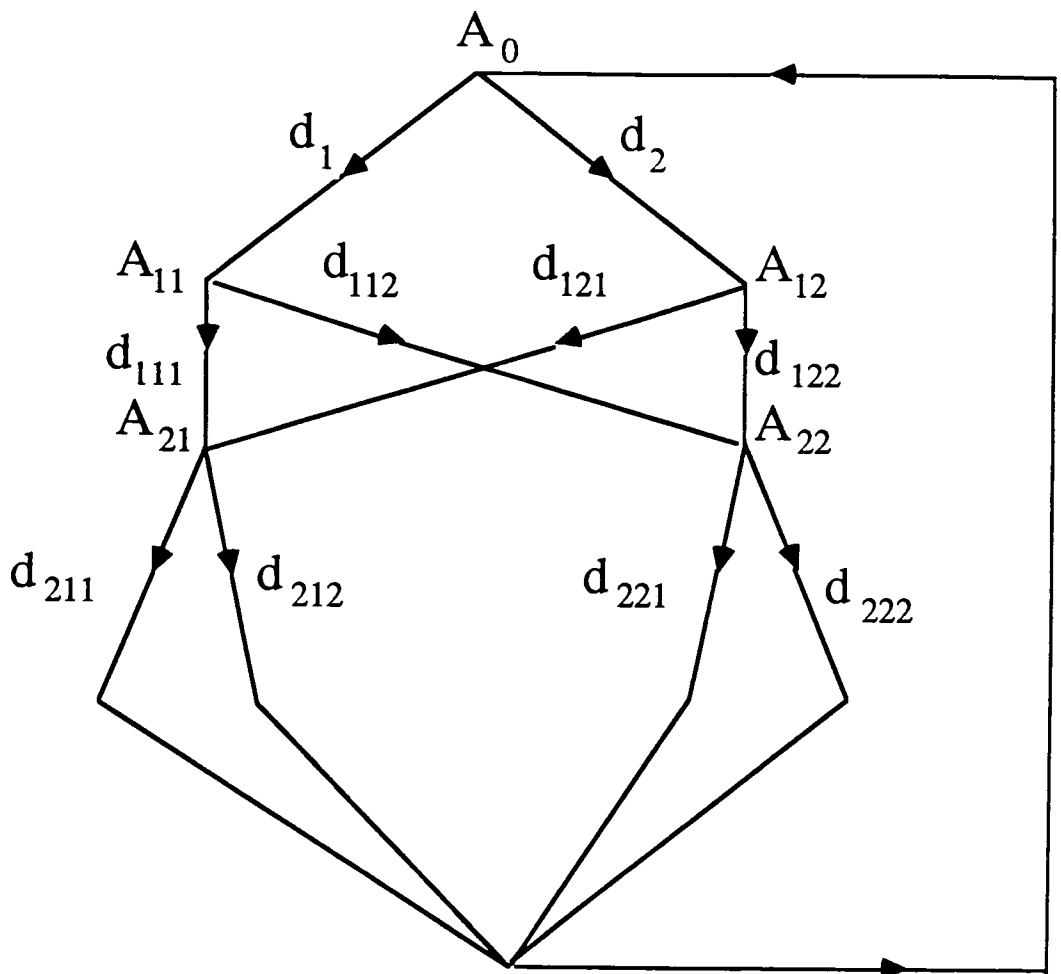


(b) Optimal Path 001 (Table 4.6a)



(a) Optimal Path 100 (Table 4.6b)

Figure 4.13 – Optimal Path Probability Changes



Directed Network

Figure 4.14 - Sequential Models

$$E_1 = \{.95, .1, .9, .8\}$$

a = 0.1, m = 50					
n	p1(n)	q1(n)	r1(n)	s1(n)	t1(n)
0	0.500000	0.500000	0.500000	0.500000	0.500000
200	0.553064	0.446936	0.849155	0.805656	0.928465
400	0.574791	0.425209	0.898792	0.853788	0.964233
600	0.589425	0.410575	0.915343	0.869834	0.976155
800	0.597068	0.402932	0.923619	0.877857	0.982116
1000	0.601655	0.398345	0.928584	0.882671	0.985693

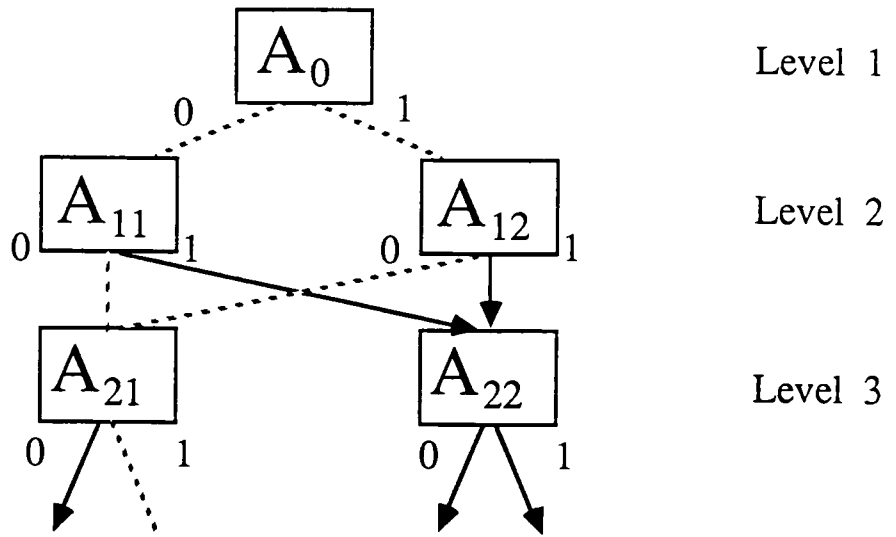
(a) Convergence Path To Minimum d212

$$E_1 = \{.95, .9, .1, .8\}$$

a = 0.1, m = 50					
n	p1(n)	q1(n)	r1(n)	s1(n)	t1(n)
0	0.500000	0.500000	0.500000	0.500000	0.500000
200	0.474115	0.525885	0.796715	0.821705	0.917460
400	0.474132	0.525868	0.827840	0.856285	0.958730
600	0.476088	0.523912	0.838221	0.867813	0.972487
800	0.477066	0.522934	0.843412	0.873576	0.979365
1000	0.477653	0.522347	0.846526	0.877035	0.983492

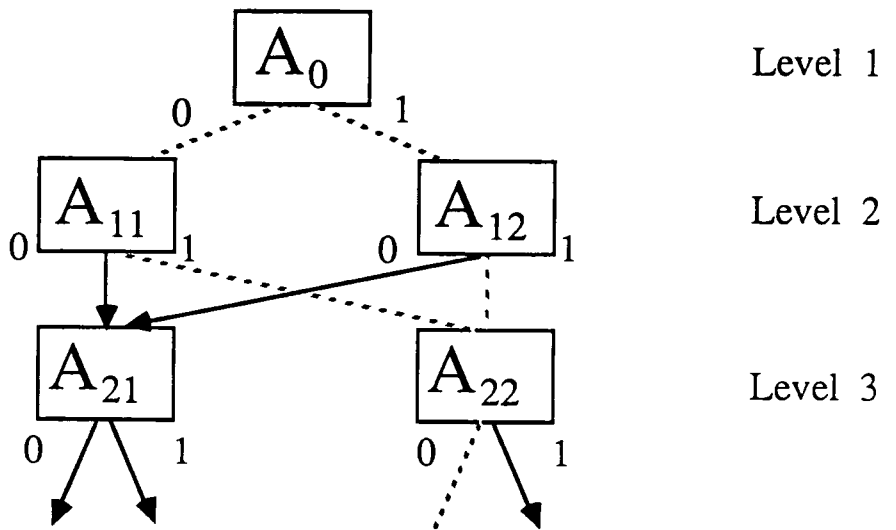
(b) Convergence Path To Minimum d221

Table 4.7 - Simulation of Directed Network (Figure 4.14)



--- Optimal Path (001: 101)

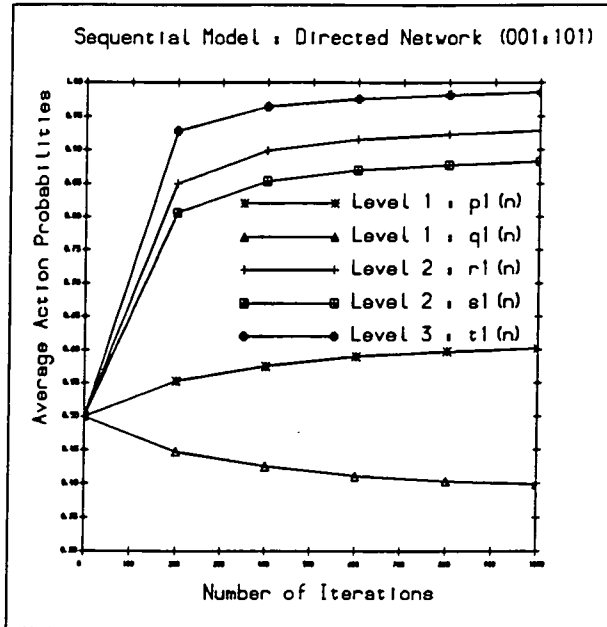
(a) Optimal Path 001:101



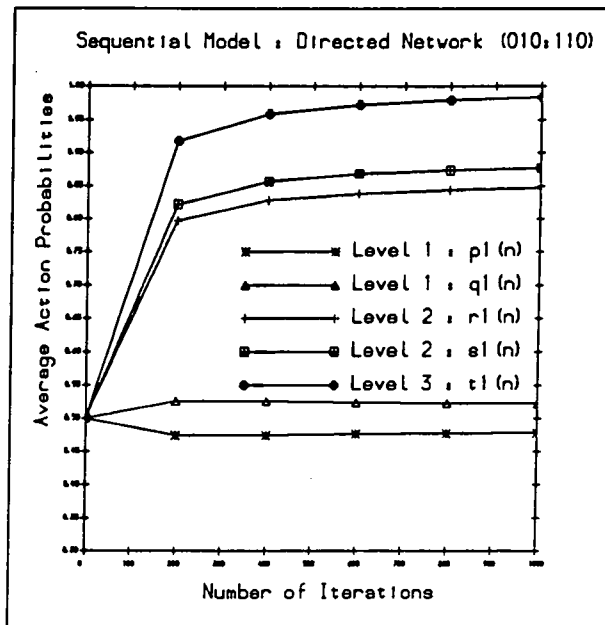
--- Optimal Path (010: 110)

(b) Optimal Path 010:110

Figure 4.15 – Directed Network : Selected Path

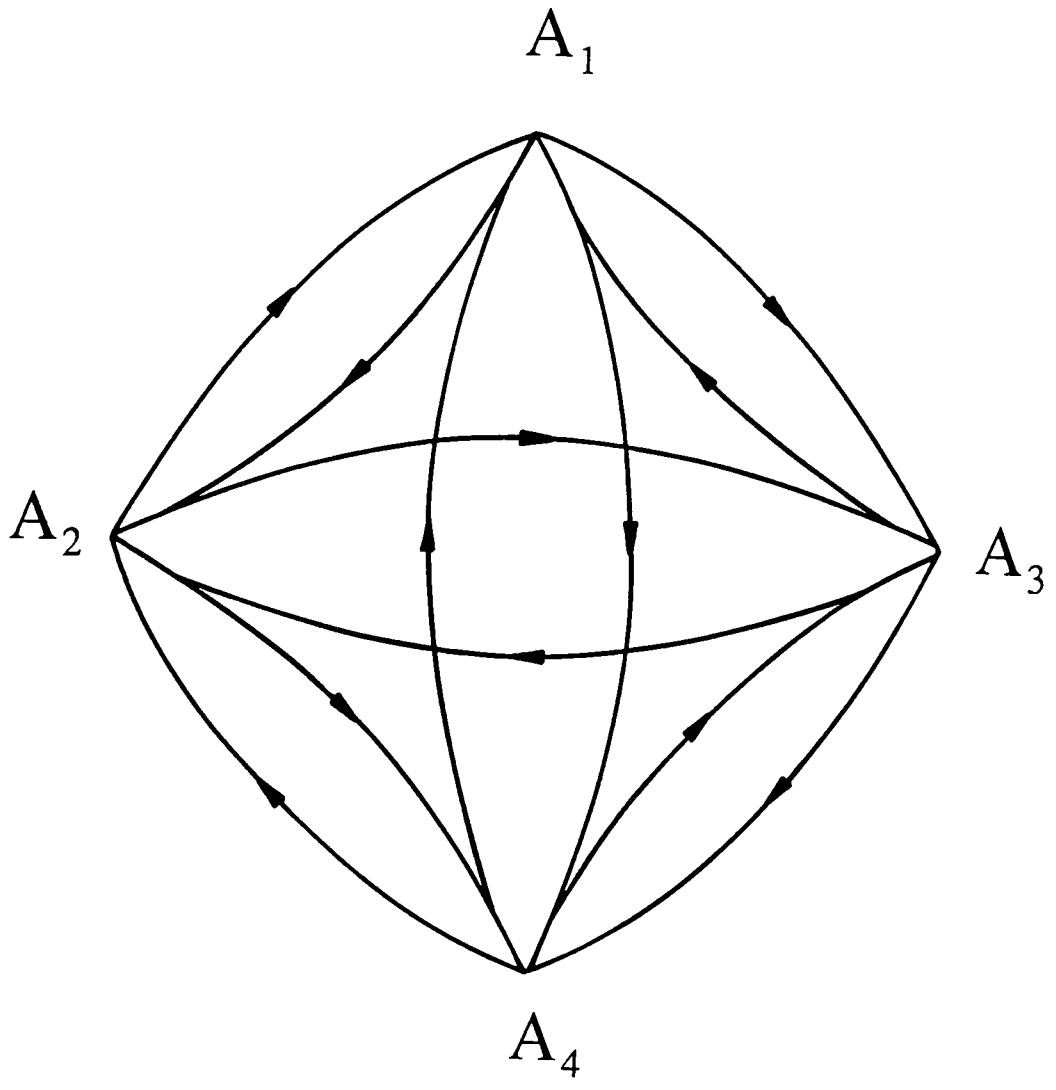


(a) Optimal Path 001:101 (Table 4.7a)



(b) Optimal Path 010:110 (Table 4.7b)

Figure 4.16 - Optimal Path Probability Changes



General Network

Figure 4.17 - Sequential Models

Chapter Five

Petri Net Theory

5.1 Introduction

The previous chapters have provided a survey of the state of the art of distributed decision making systems, and outlined the degree of complexity involved in the development of such systems. In approaching this problem, the stochastic learning automata was adopted as a basic framework. Further, the discussion in Chapter Four includes an analysis of the collective behaviour of multiple learning automata in a game situation. Thus, several configurations are introduced and simulation results are included for models in which the actions of all decision makers are synchronised as well as sequential models. However, it was shown that such models lack structure and the ability to describe explicit interactions between decision makers. It was considered that such models were inadequate for the representation of complex systems which may involve concurrent, asynchronous, parallel or distributed activities.

This chapter introduces a mathematical framework based on Petri net theory as a suitable basic model for representing and studying complex systems. Petri Nets (PN) models are a graphical and mathematical tool applicable to the analysis of a diverse range of systems. They have been proposed by many authors as a useful tool for describing and analysing the flow of information and control in systems that exhibit concurrent, asynchronous,

distributed, parallel, nondeterministic, and/ or stochastic activities, [15], [16], [51], [52]. In the graphical representation, PNs can be considered as a visual communication aid in the form of flow charts, block diagrams, and networks. In mathematical terms, many systems can be described numerically and the relations between certain features may be interpreted by equations or inequalities. The concept of Petri nets originated in the early work of C. A. Petri in his doctoral dissertation, submitted in 1962 to the faculty of Mathematics and Physics at the Technical University of Darmstadt, West Germany, [53].

This following section presents formal definitions for basic PN concepts. An introductory modelling example is provided, which illustrates the modelling capability of PNs within the context of this thesis. It also describes an analysis technique which may be used to study the behaviour of the system. Finally, the notion of time is also introduced by discussing stochastic nets, [54], [55].

5.2 Structure of a Petri Net

The structure of *standard* Petri nets are composed of a set of *places* $P (p_1, p_2, \dots, p_n)$, a set of *transitions* $T (t_1, t_2, \dots, t_m)$ and to complete the definition, it is relevant to consider the relationship between the places and the transitions. The relationship is achieved by outlining a set of input and output functions. The *input* function I defines, for each transition t_j , the set of input places for the transition $I(t_j)$. The *output* function O defines for each transition t_j , the set of output places for the transition $O(t_j)$. Figure 5.1 shows the structure of a Petri net.

A formal definition of a Petri net structure represents a 3-tuple, $PN = (P, T, F)$ as follows:

$P = (p_1, p_2, \dots, p_n)$ is a finite set of places;

$T = (t_1, t_2, \dots, t_m)$ is a finite set of transitions;

$I(t_j) \subset (P \times T)$ is the input function; (5.1)

$O(t_j) \subset (T \times P)$ is the output function;

$F \subseteq (I(t_j) \cup O(t_j))$ is a set of input and output functions;

5.2.1 Petri Net Graphs

The above Petri net structure may be represented in the form of a *directed, weighted, bipartite graph*. In the graphical representation of Petri nets, places are drawn as circles and transitions as bars or boxes. The input and output functions are represented by directed arcs which are drawn as arrows, connecting the transitions to places and places to transitions. A place is an *input* to a transition if an arc exists from the place to the transition. A place is an *output* of a transition if an arc exists from the transition to the place. Since the arcs are directed, and labelled with their weights (nonnegative integer), PNs are referred to as *directed, weighted graph*. Labels for unity weight are usually omitted. Additionally, since the nodes can be partitioned into two sets (places and transitions) such that each arc is directed from an element of one set (place or transition) to an element of the other set (transition or place), it is known as a *bipartite graph*. Figure 5.2 shows an example of a Petri net graph corresponding to the structure described above. The Petri net consists of five places and five transitions.

5.2.2 Petri Net Markings

The dynamic feature of a PN is represented by tokens. A marking (state) assigns tokens to each place in a net. Tokens are graphically drawn as black dots, which reside in the circle nodes representing the places of the net. A Petri net with tokens is called a *marked Petri net*. The number m_i of tokens and its position in a net may change during execution, thus defining the state of a system. The PN state is usually called the Petri net *marking*, and is denoted by the vector $M = (m_1, m_2, \dots, m_n)$. A marked Petri net is depicted in Figure 5.3.

A formal definition of marked PN is thus the following 4-tuple, $PN = (P, T, F, M_0)$ as follows:

$$\begin{aligned} P &= (p_1, p_2, \dots, p_n) \text{ is a finite set of places;} \\ T &= (t_1, t_2, \dots, t_m) \text{ is a finite set of transitions;} \\ F &\subseteq (PxT) \cup (TxP) \text{ is a set of input and output functions;} \\ M_0 &= (m_{01}, m_{02}, \dots, m_{0n}) \end{aligned} \tag{5.2}$$

where m_{0i} denotes the number of tokens in place p_i in the initial marking M_0 .

5.2.3 Execution Rule for Marked Petri Nets

The dynamic behaviour of a PN can be described by the execution of the net. The execution of a PN is controlled by the number and the distribution of tokens in the PN. A Petri net executes by firing transitions. The firing of a transition changes a state or marking of the PN to a new

marking according to the following transition enabling and firing rule:

- (1) A transition t is said to be *enabled* if each input place contains at least one token.
- (2) A *firing* of an enabled transition removes one token from each of its input place, and then adds one token into each of its output places.
- (3) Each firing of a transition represents a change in the state of the net by modifying the distribution of tokens in a nets' place; thus producing a new marking.

This results in a new marking M' where:

$$M'(p) = \begin{cases} M(p) + 1 & \text{if } p \in O(t), \quad p \notin I(t); \\ M(p) - 1 & \text{if } p \in I(t), \quad p \notin O(t); \\ M(p) & \text{otherwise} \end{cases} \quad (5.3)$$

Consider the dynamic behaviour of the marked Petri net in Figure 5.3. The initial marking is $M_0 = [10000]$, and by definition only transition t_1 is enabled in this marking. Thus t_1 is the only transition that can fire and change the state of the system. The result of firing t_1 is shown in Figure 5.3a, no tokens are present in p_1 while places p_2 and p_3 each contain a token; the marking now becomes $M_1 = [01100]$. In this case both transitions t_2 and t_3 are enabled and can fire independently (*concurrently*), since they do not share any input places. The firing of t_2 enables transition t_4 ; the firing of t_3 puts a token in p_5 . On completing the firings of both transitions t_2 and t_3 , a new marking is reached as shown in Figure 5.3b. This situation represents a *conflict*. Both transitions t_4 and t_5 are enabled. However, only one of these two transitions can fire and the firing of one transition disables the other.



In such a case, the decision as to which one fires is non deterministic. If t_5 fires the system returns to the initial marking. The ability to represent both concurrency and conflict makes PNs a very powerful modelling tool.

5.2.4 Modelling Examples

Petri nets were designed for and are used mainly for modelling. Many systems can be modelled by a Petri net, including computer software, hardware or physical systems. In particular they may be used to model the flow of information or other resources within a system. This section outlines one of the basic concepts of PNs that are useful in modelling. It provides a description and a more realistic application of the individual components.

Events and Conditions

The simple Petri net view of a system focusses on two concepts; events and conditions. *Events* are actions which take place in the system. The occurrence of an event is controlled by the state of the system. The state of the system may be described by a set of conditions. A *condition* is a logical description of the state of the system, which may *hold* (true) or *not hold* (false).

In modelling using this concept, places represent conditions and, transitions in a PN represent events in a real system. A transition (an event) has a certain number of *input* and *output* places representing the pre-conditions and post-conditions, respectively. A marked PN then corresponds to a state of the system being modelled. The firing of a PN transition corresponds to

the occurrence of an event in the system. The occurrence of an event causes the system to move to a different state. Therefore, the successive firings of transitions (and the resulting changes in markings) in a PN represent the evolution of the system through different states. Some typical interpretation of transitions and their input and output places are shown in Table 5.1.

5.2.5 Analysis of Petri Nets

The major strength of PNs is in the modelling of systems. However modelling of systems is itself not useful. It is necessary to *analyse* the modelled system. This analysis provides important insights into the behaviour of the system. There are several approaches in the analysis of PNs. The major analysis technique which has been used with PNs, in this project is based on the coverability (reachability) tree. This technique involves finding a finite representation for the reachability set of a PN. It consists of a tree whose nodes represent markings of the PN and whose arcs represent the possible changes in state resulting from the firing of transitions. The following presents the reachability property and a discussion of the appropriate analysis technique.

Reachability

The reachability set of a PN can be represented in the form of a tree structure, as discussed later. This concept is a fundamental basis for studying the dynamic properties of a system. The firing of an enabled transition changes the state (marking) of a system according to the transition firing rule. As execution of the firing rule proceeds, a sequence of firings will result in

a sequence of markings. Thus, a sequence of transitions (t_1, t_2, \dots) and a sequence of markings (M_0, M_1, M_2, \dots) can be defined. By definition, a marking M_n is said to be *reachable* from a marking M_0 if there exists a sequence of transition firings that transforms the PN state from M_0 to M_n . A *firing* or *occurrence* sequence is denoted by

$$\sigma = M_0 t_1 M_1 t_2 M_2 \dots t_n M_n \quad (5.4)$$

or simply

$$\sigma = t_1 t_2 \dots t_n \quad (5.5)$$

Hence, M_n is reachable from M_0 by σ and this may be written as follows:

$$M_0[\sigma > M_n \quad (5.6)$$

Thus, the set of all possible markings reachable from M_0 in a net is denoted by $R(M_0)$.

The Coverability (Reachability) Tree

Given a PN with an initial marking M_0 , it is possible to obtain as many 'new' markings as the number of enabled transitions. From each new marking, more new markings can be reached. If this procedure is repeated many times, a tree representation of all the markings will be produced. The *coverability tree* consists of nodes which represent markings of the PN and whose arcs represent a transition firing, which transforms one marking to another. Note that if the net is bounded, the coverability tree is called a *reachability tree*.

Consider the example of the marked PN of Figure 5.3. The reachability tree can be constructed by starting from the initial marking M_0 and

considering the markings immediately reachable from this state. The initial marking is $M_0 = [10000]$, and transition t_1 is enabled. Since the entire reachability set is required, new nodes may be defined in the reachability tree for the (reachable) markings which result from firing of transition t_1 . An arc labelled by the transition fired leads from the initial marking to each of the new markings, Figure 5.4a shows all markings that are immediately reachable from the initial marking. Consider all markings that are reachable from these new markings. From $M_1 = [01100]$, transitions t_2 and t_3 may be fired producing $M_2 = [00110]$ and $M_3 = [01001]$. These firings produce the tree of Figure 5.4b. The immediately reachable marking from M_2 is now M_1 and M_4 by firing transition t_4 and t_3 , respectively. In this case a new marking $M_4 = [00011]$ is created, and the old marking of M_1 is created. From M_1 the same operation could be repeated and this would obviously lead to an infinite structure. This process is repeated, producing new markings to add to the tree shown in Figure 5.4c and Figure 5.4d.

By repeating this procedure over and over, every reachable marking will eventually be produced. However, the resulting reachability tree may be infinite. Every marking in the reachability set will be produced, and for any PN with an infinite reachability set, the corresponding tree would also be infinite. It is important to note that if the tree is going to be useful, it is necessary to limit the tree to a finite size. Appendix Three provides relevant PN properties, details for the reduction to finite form and also the algorithm necessary to construct the reachability tree.

5.3 Time-Related Model

The concept of time is not explicitly given in the standard Petri net model. Therefore with standard PNs it is possible to describe only the logical structure of systems and not their time evolution. As such, PNs did not convey information about the duration of each activity or on the way in which the transitions to be fired is actually selected from among these enabled in a marking. Many authors have extended PN models by introducing the notion of time, [54], [55], [56]. This section introduces time delays in a Petri net model. These time delays are specified probabilistically and the model is known as a Stochastic Petri Net (SPN).

5.3.1 Stochastic Petri Nets (SPN)

A Stochastic Petri Net (SPN) is a Petri net where each transition is associated with an exponentially distributed random variable that expresses the delay from the enabling to the firing of the transition, [54], [55], [56].

A formal definition of a Stochastic Petri net, is as follows:

$$\text{SPN} = (\text{P}, \text{T}, \text{A}, M_0, \lambda) \quad (5.8)$$

where $(\text{P}, \text{T}, \text{A}, M_0)$ is the *marked PN* underlying the SPN; and $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_m)$ is the set of possibly *marking-dependent* firing rates associated with the Petri net transitions.

In the case where several transitions are simultaneously enabled; the transition with which is associated the shortest delay will fire first. The SPN then reaches a new marking in which transitions that were already enabled

in the previous marking, but did not fire, may be still enabled.

5.3.2 An Example of a Stochastic Petri Net

Consider the SPN shown in Figure 5.5, transition t_1 fires at a marking-dependent rate equal to αm_1 , where m_1 is the number of tokens in p_1 . The other transitions t_2 , t_3 and t_4 have (marking-independent) firing rates β , γ and δ , respectively; these are indicated close to the corresponding transitions. The associated reachability tree is shown in Figure 5.6. The system operations can be precisely described by means of a graph that translates into a Markovian model useful for obtaining performance estimates, [54].

5.3.3 Generalised Stochastic Petri Net (GSPN)

The limitation of SPNs is that the graphical representation of systems becomes more difficult as system size and complexity increase, [57]. Moreover, the number of states of the associated reachability tree rapidly multiplies as the dimensions of the graph increase. Thus, SPNs can be used to model only systems of limited size.

The SPNs have been extended to a class of Generalised Stochastic Petri Nets (GSPN) to overcome these limitations. The GSPNs are useful in modelling system operations which comprise activities whose durations differ by orders of magnitude. It is then permissible that the short activities can be processed immediately, whilst time is associated with the longer lasting activities. This approach is most appropriate, since the number of states of the associated reachability tree is reduced, hence reducing the complexity of

the model.

GSPN models comprise two types of transitions, *immediate transitions* and *timed transitions*. Immediate transitions fire in zero time with priority over timed transitions. Timed transitions fire after a random, exponentially distributed enabling time. In all figures, the convention used for drawing timed transitions is thick bars or box nodes, and immediate transitions as thin bars.

A formal definition of a GSPN is thus,

$$\text{GSPN} = (\text{P}, \text{T}, \text{A}, M_0, \pi, W) \quad (5.9)$$

where $(\text{P}, \text{T}, \text{A}, M_0)$ is the *marked PN* underlying the GSPN; π is a *priority* function defined over the set of immediate transitions; $W = (w_1, w_2, \dots, w_n)$ is an array whose entries correspond to the *firing rates* of the timed transitions (as in the case of SPN); the *weights* of immediate transitions. Similarly, the interpretation of the model is very similar to the case of SPN, with the additions resulting from the introduction of immediate transitions.

In the case of a GSPN, a reduction of the reachability tree is possible by classifying markings into two types: vanishing and tangible markings. A marking is called *vanishing marking* if it enables (at least) one immediate transition. A vanishing marking is so named since no time is spent in this marking; as soon as such a marking is reached (one of) the immediate transitions fire in zero time. However, when a marking enables timed transitions, it is called *tangible marking*, and the behaviour is the same as in the case of SPNs.

Several transitions may be simultaneously enabled by a marking. The following rules may be applied: if the set of enabled transitions H comprises only timed transitions, then the enabled transition fires t_i ($i \in H$) with probability as follows

$$Pr\{t_i\} = \frac{\lambda}{\sum_{k \in H} \lambda_k} \quad (5.10)$$

exactly as with SPNs. If H consists of both immediate and timed transitions then only immediate transitions can fire. If H comprises zero or more timed transitions and only one immediate transition, then this is the one that fires. However, if H comprises several immediate transitions it is necessary to specify a probability density function on the set of enabled immediate transitions according to which transition is selected. This is called a *random switch* and the associated probability distribution is called a *switching distribution*.

It may also be noted that the reachability set of GSPN is significantly reduced in comparison to the associated PN, because the priority rules introduced with immediate transitions do not allow some states to be reached. The reachability set of a SPN is identical to the set constructed for the associated PN.

5.3.4 An Example of Generalised Stochastic Petri Net

Consider, an example of the GSPN shown in Figure 5.7; comprises of seven places and transitions. The three timed transitions t_1 , t_6 and t_7 fire at fixed rates u , v and z respectively. The immediate transitions t_4 and t_5 are enabled simultaneously if tokens are present in places p_4 and p_5 . Thus,

a switching distribution must be defined for each marking in which m_2 , m_4 and m_5 are greater than zero. It is also necessary to define a switching distribution for the two conflicting immediate transitions, namely, t_2 and t_3 . These transitions are always enabled simultaneously, such that a switching distribution for each marking in which m_3 is greater than zero is required. For this particular structure, two random switches can be defined, Table 5.2a provides the switching distribution.

Execution of GSPN

By starting from the initial marking $M_0[2001100]$ shown in Figure 5.7; the evolution of states results in a reachability tree as depicted in Figure 5.8. Clearly, transition t_1 fires after an exponentially distributed random time u , and this removes one token from place p_1 and placing one in p_2 . At this stage the immediate transitions t_4 or t_5 are enabled. The transition that fires is selected according to the switching distribution defined in Table 5.2a, in this case equal probabilities are assigned to the firing of each transition. Now, assume that t_4 fires, this moves a token contained in place p_2 and p_4 , and includes one in p_6 . The enabled transitions are t_1 and t_6 , each of which can fire first with the following probabilities.

$$Pr\{t_1\} = \frac{u}{(u + v)} \quad (5.11)$$

$$Pr\{t_6\} = \frac{v}{(u + v)} \quad (5.12)$$

If t_1 fires first, a token moves from p_1 to p_2 , thus enabling the immediate transition t_5 . Since t_5 is the only immediate transition that is enabled, this

fires at zero time by moving one token from p_2 to p_7 and removing one token from p_5 . This produces a new marking containing token in p_6 and p_7 . The two timed transitions t_6 and t_7 are now enabled. Transition t_6 fires with probability

$$Pr\{t_6\} = \frac{v}{(v + z)} \quad (5.13)$$

whilst transition t_7 fires with probability

$$Pr\{t_7\} = \frac{z}{(v + z)} \quad (5.14)$$

Assume that t_6 fires, so that one token moves from p_6 to p_3 , and a token is put in p_1 . Thus, the two immediate transitions t_2 and t_3 are now simultaneously enabled; the transition that fires is selected according to switching distribution defined in Table 5.2a. Similarly, in this case equal probabilities are assigned to the firing of each transition, so that the token can move either to p_4 or to p_5 . Now transitions t_1 and t_7 are enabled, and the PN evolution continues; thus developing the corresponding reachability tree, as shown in Figure 5.8.

The reachability set of the GSPN example is provided in Table 5.2b. It comprises 16 markings, whereas that of the associated PN comprises 33 states. As stated previously the reachability set of a GSPN is a subset of the reachability set of the associated PN, due to the precedence rules introduced with immediate transitions which do not allow some states to be reached. Thus, it must be pointed out that the reachability set of a SPN is, instead, the same as for the associated PN. Furthermore, Table 5.2b illustrates that the reachability set of the GSPN may be divided into two disjoint subsets,

one of which consists of markings that enable timed transitions only, and also markings that enable immediate transitions.

5.4 Conclusion and Summary

This chapter has presented a brief review of knowledge in the field of PNs. It has defined a high-level quantitative framework based on PN methodology, and introduced appropriate terminology although not all aspects in the field of PN theory have been discussed. The practical applications of such state-transition models have been considered by extending and/or modifying the basic model definitions to obtain more convenient modelling tools. In particular, the possibility of representing in the model the time involved in system operations has been discussed by studying stochastic timed nets.

A detailed description of the SPN has been presented. It is shown that SPNs are obtained by associating with each transition in a PN an exponentially distributed firing time. SPNs are a very useful tool for the analysis of computer systems since they allow the system operations to be precisely described by means of a graph and the model is useful for obtaining performance estimates. However, there are limitations to the use of SPN, they can be used to model only systems of limited size. This is due to the complexity involved in the graphical representation of systems, and also there is a rapid increase in the number of states of the associated reachability tree as the dimensions of the graph increase. Thus, a Generalised SPN is introduced which contain two types of transitions: timed and immediate. It

is shown that by considering GSPNs, the number of states of the associated reachability tree is reduced and also the solution complexity is reduced.

To conclude, this chapter has identified the potential modelling capability of the Petri Net formalism. The framework is considered to be an effective graphical and mathematical tool. In particular, they provide a powerful means for the description and analysis of systems that are characterised as being concurrent, asynchronous, distributed, parallel, nondeterministic and/or stochastic. However, at this phase of the research programme it is evident that existing PN theory do not exhibit an intelligence capability which is needed for the effective representation of decision models. The next chapter addresses the shortcomings of the PN methodology by introducing a new class of PNs, known as the *Learning Petri Net Models*.

$$N = (P, T, F)$$

$$P = (p_1, p_2, p_3, p_4, p_5)$$

$$T = (t_1, t_2, t_3, t_4, t_5)$$

$$I(t_1) = p_1$$

$$I(t_2) = p_2$$

$$I(t_3) = p_3$$

$$I(t_4) = p_4$$

$$I(t_5) = (p_4, p_5)$$

$$O(t_1) = (p_2, p_3)$$

$$O(t_2) = p_4$$

$$O(t_3) = p_5$$

$$O(t_4) = p_4$$

$$O(t_5) = p_1$$

Figure 5.1 - Petri Net Structure

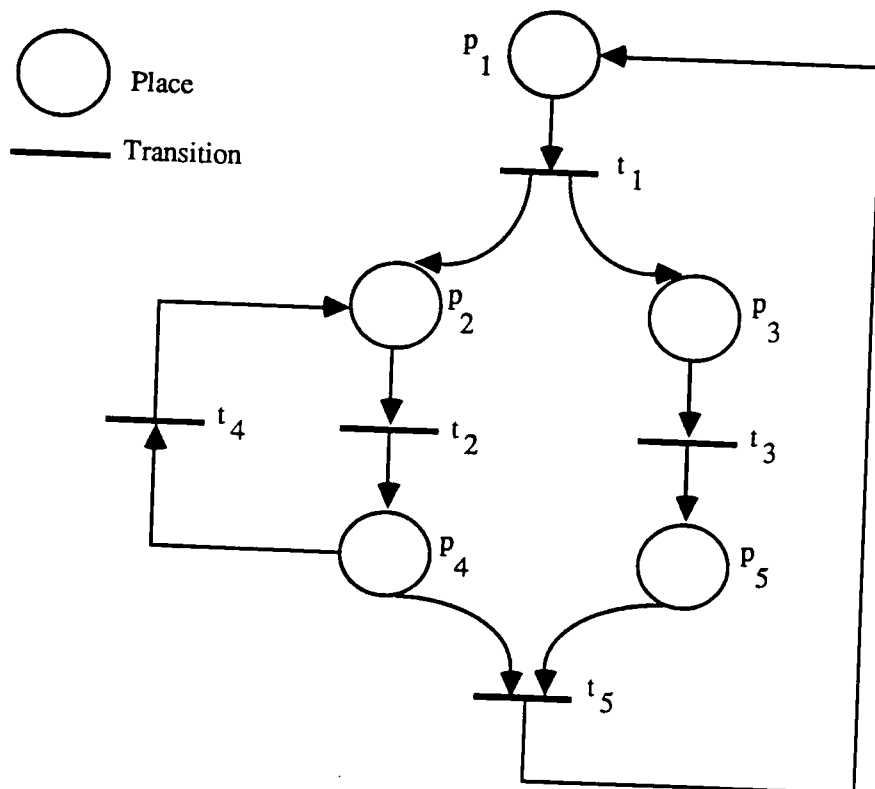


Figure 5.2 - Petri Net Graph

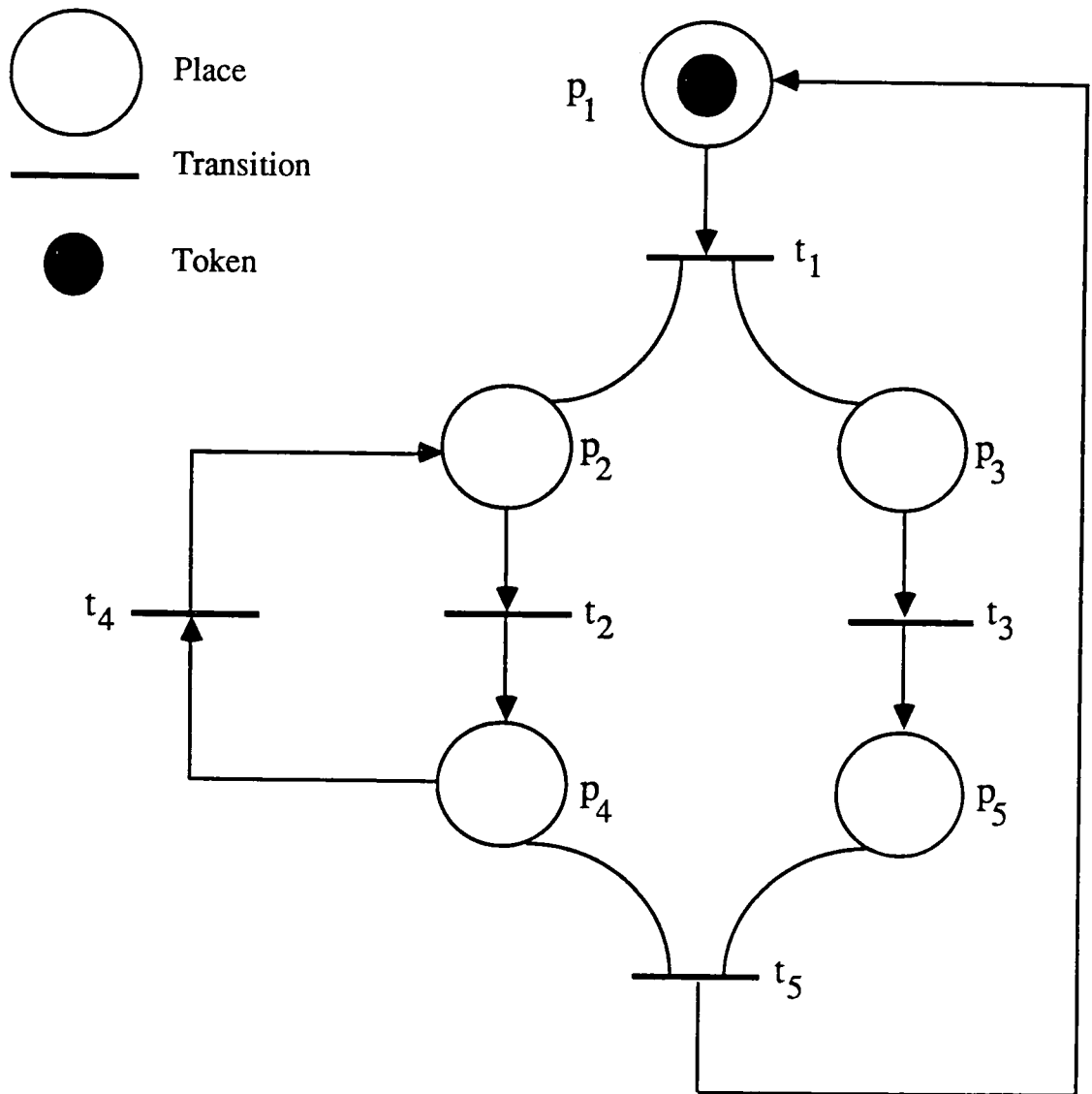
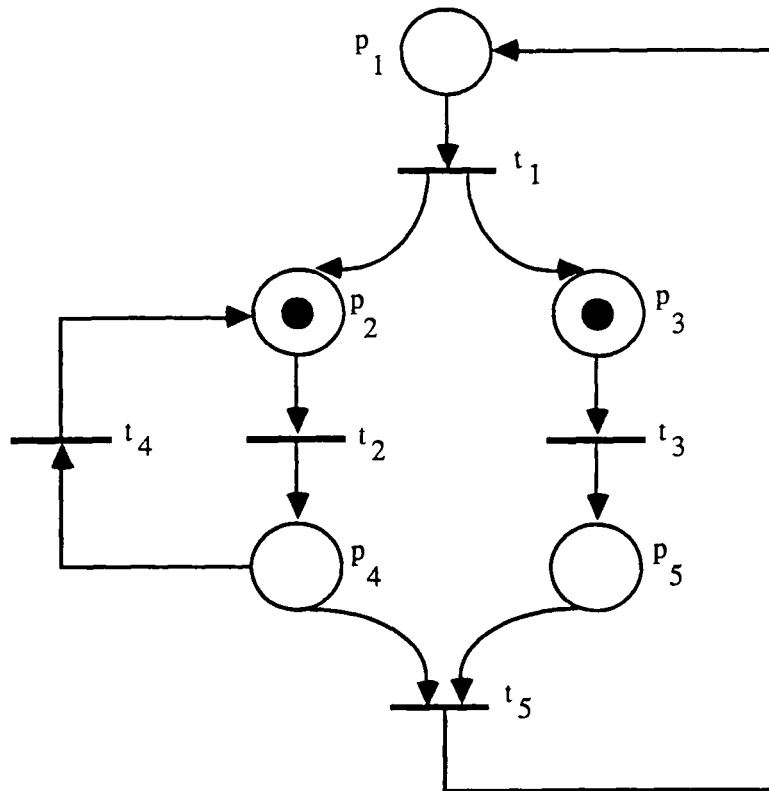
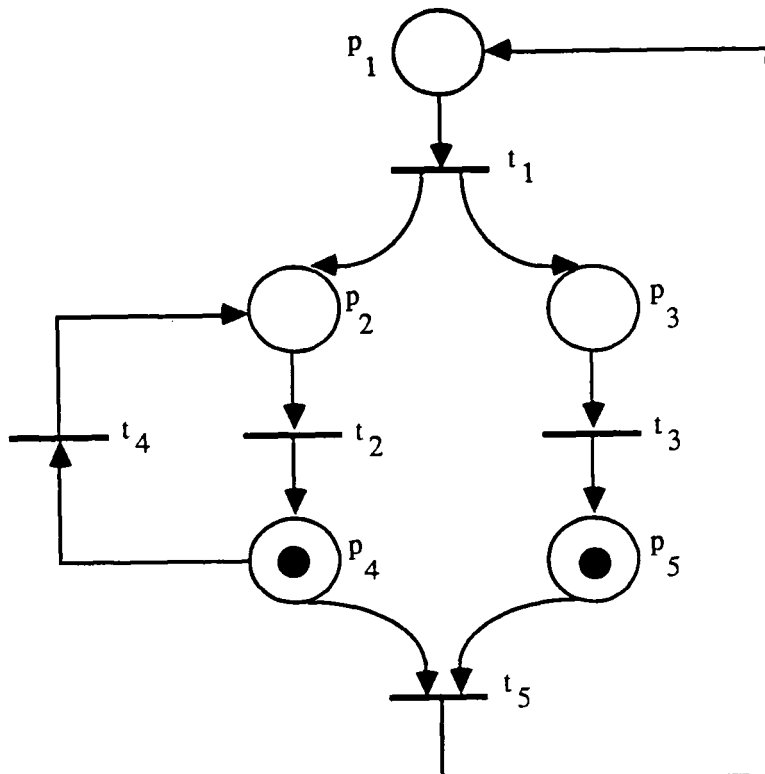


Figure 5.3 - Marked Petri Net



(a) Transition t_1 Fires



(b) Transition $t_2 : t_3$ Fires

Figure 5.3 - Marked Petri Net

$$M_0 = [10000]$$



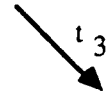
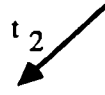
$$M_1 = [01100]$$

(a) First Step in Building Tree

$$M_0 = [10000]$$



$$M_1 = [01100]$$

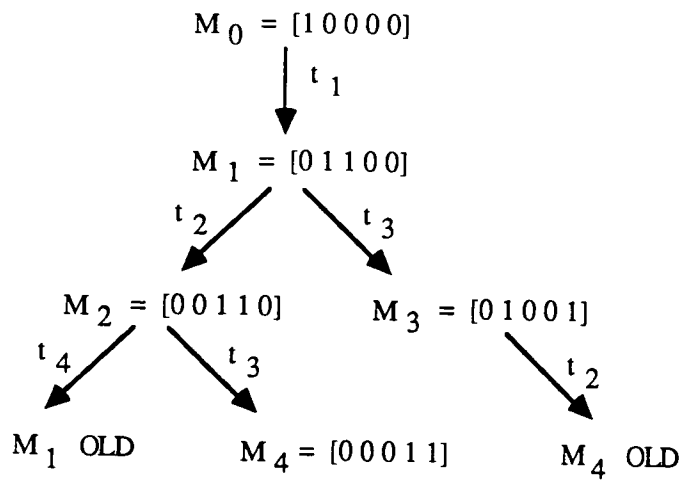


$$M_2 = [00110]$$

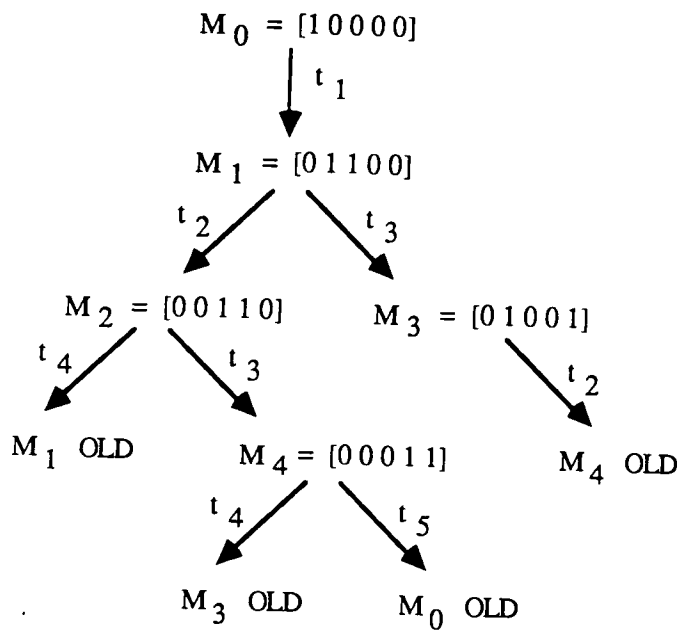
$$M_3 = [01001]$$

(b) Second Step in Building Tree

Figure 5.4 – Reachability Tree Construction of Marked PN (Figure 5.3)



(c) Third Step in Building Tree



(d) Fourth Step in Building Tree

Figure 5.4 – Reachability Tree Construction of Marked PN (Figure 5.3)

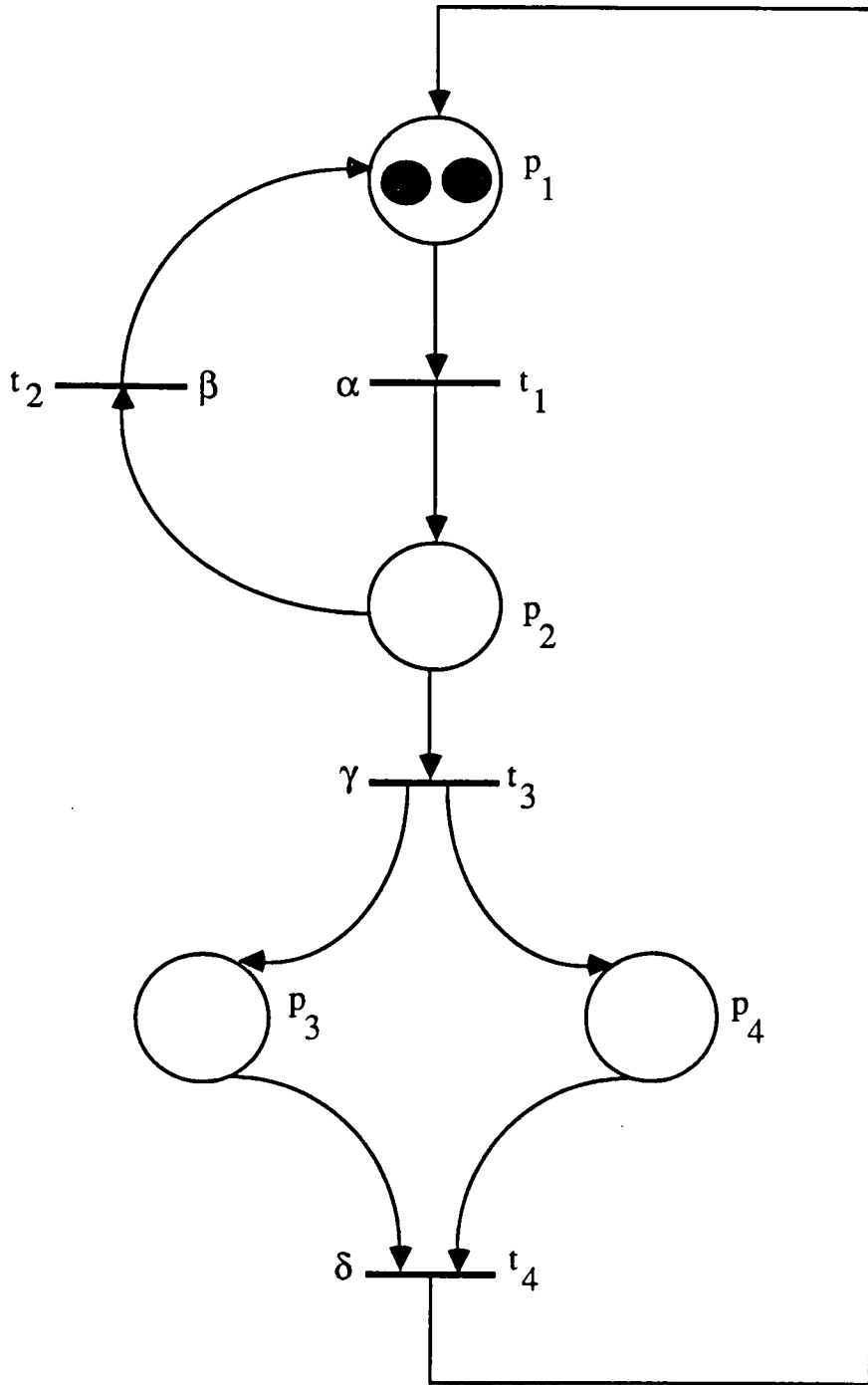


Figure 5.5 - Stochastic Petri Net (SPN)

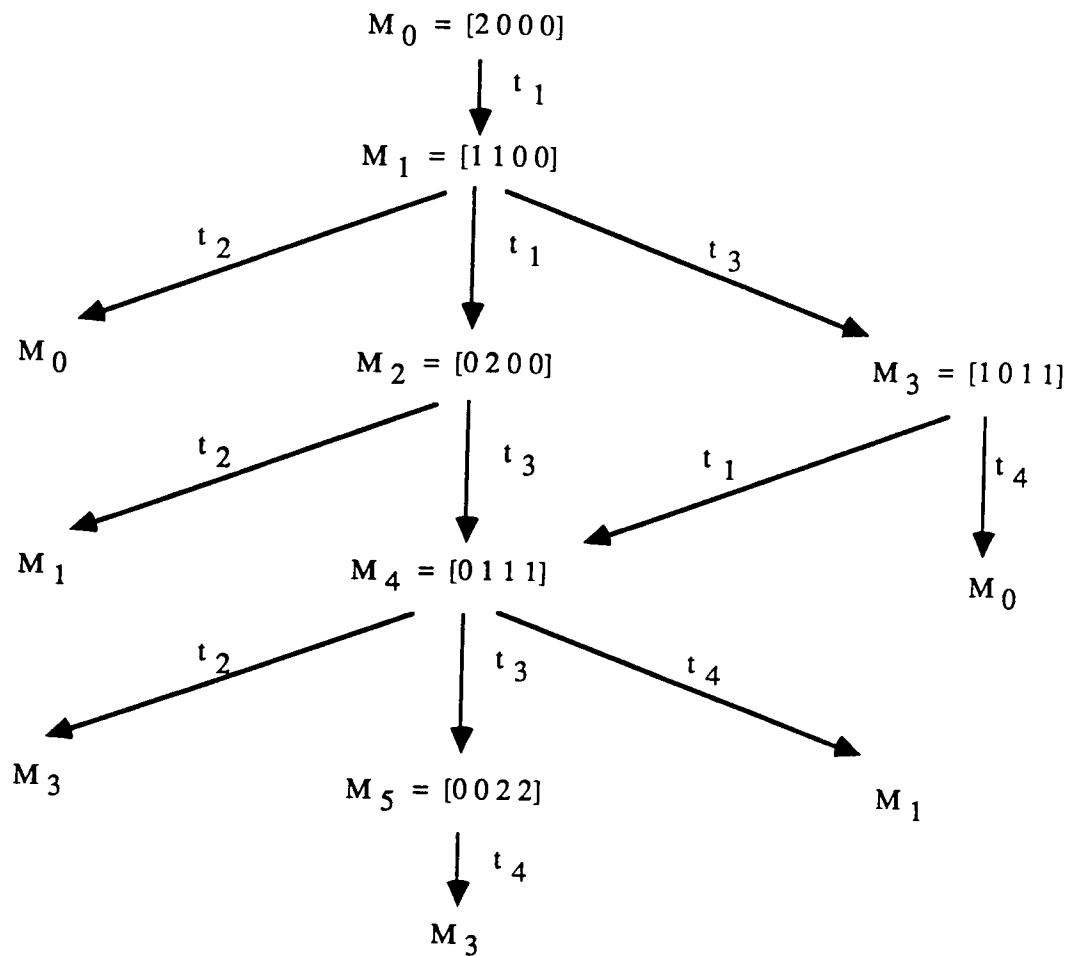


Figure 5.6 - Reachability Tree SPN

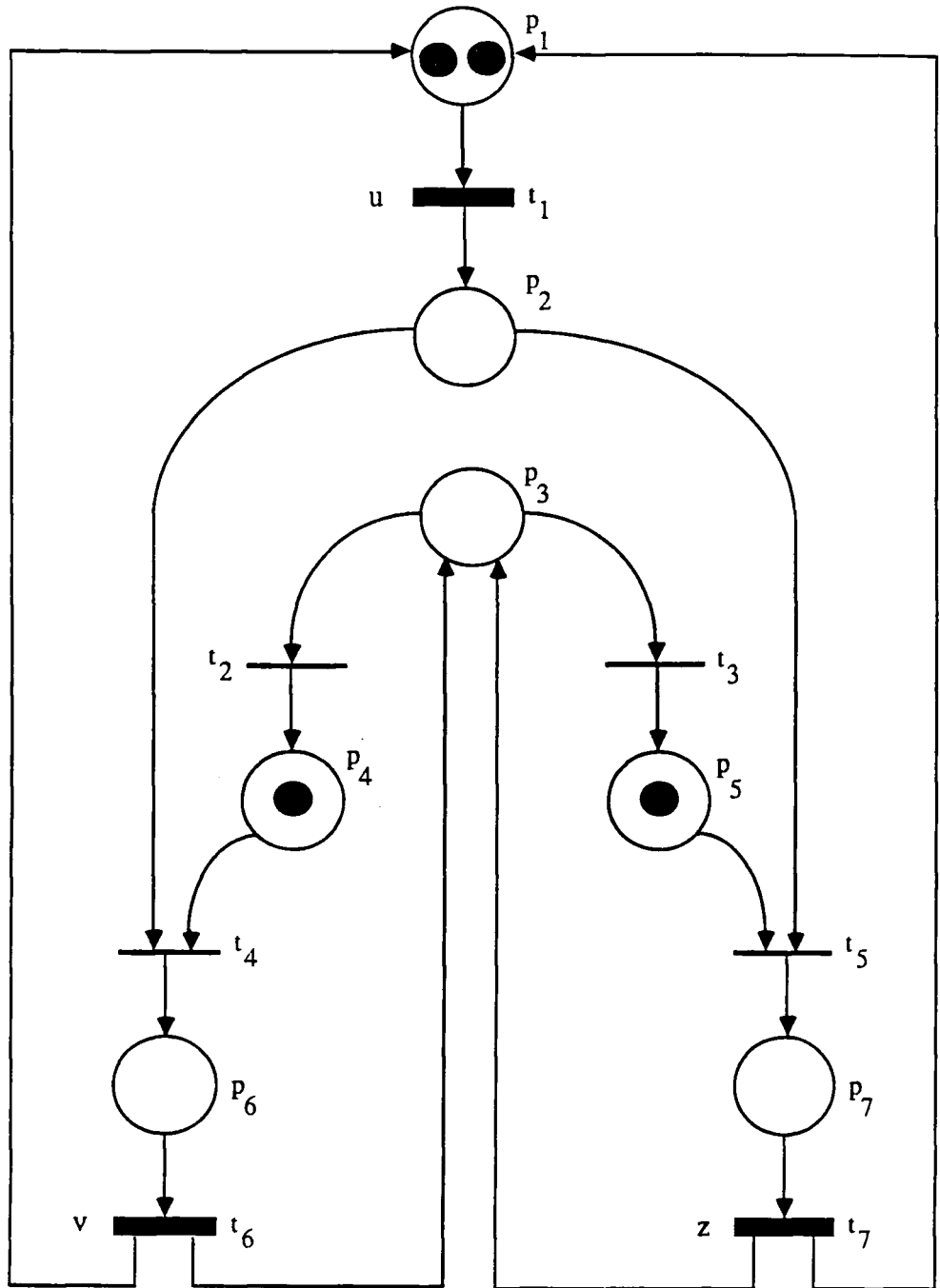
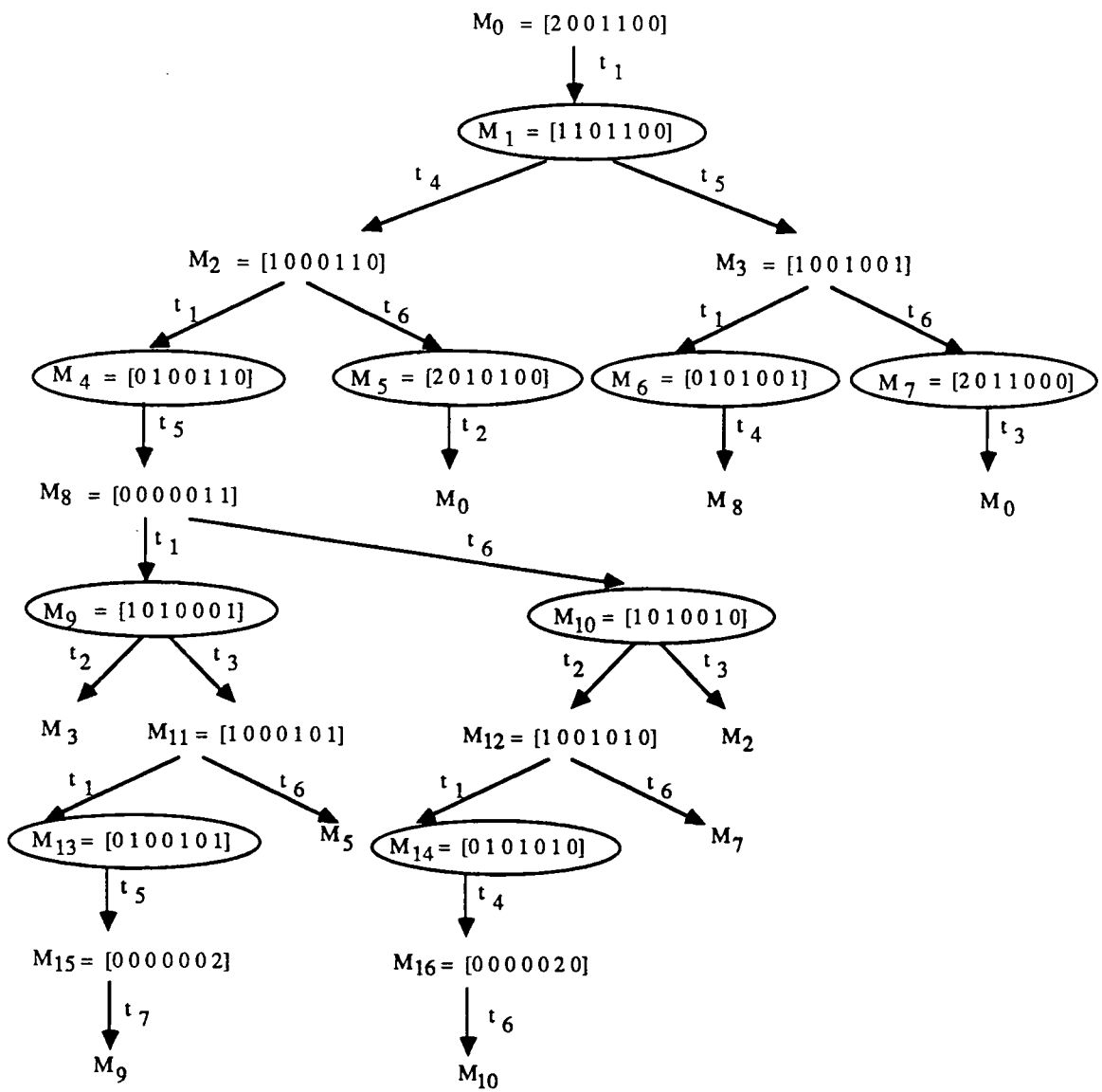


Figure 5.7 – Generalised Stochastic Petri Net (GSPN)



States - ($M_0 M_2 M_3 M_8 M_{11} M_{12} M_{15} M_{16}$) Tangible Marking


 Vanishing Marking Indicates Presence of Random Switch

Figure 5.8 – GSPN Reachability Tree

INPUT PLACES	TRANSITIONS	OUTPUT PLACES
Pre-conditions	Event	Post Conditions
Input data	Computation	Output data
Input Signal	Signal Processor	Output signal
Resource needed	Task or Job	Resource needed
Buffer	Processor	Buffer

Table 5.1 – Interpretations of Transitions and Places

$$\begin{aligned} \Pr(t_4) &= m_4 / (m_4 + m_5) \\ \Pr(t_5) &= m_5 / (m_4 + m_5) \\ \\ \Pr(t_2) &= m_5 / (m_4 + m_5) \\ \Pr(t_3) &= m_4 / (m_4 + m_5) && \text{if } m_4 = 0 \text{ and } m_5 = 0 \\ \\ \Pr(t_2) = \Pr(t_3) &= 1/2 && \text{if } m_4 = m_5 = 0 \end{aligned}$$

Table 5.2a – Switching Probabilities of GSPN

MARKINGS THAT
ENABLE
TIMED TRANSITIONS

	m1	m2	m3	m4	m5	m6	m7
M0	2	0	0	1	1	0	0
M1	1	0	0	0	1	1	0
M3	1	0	0	1	0	0	1
M8	0	0	0	0	0	1	1
M11	1	0	0	0	1	0	1
M12	1	0	0	1	0	1	0
M15	0	0	0	0	0	0	2
M16	0	0	0	0	0	2	0

MARKINGS THAT
ENABLE
IMMEDIATE TRANSITIONS

	m1	m2	m3	m4	m5	m6	m7
M1	1	1	0	1	1	0	0
M4	0	1	0	0	1	1	0
M5	2	0	1	0	1	0	0
M6	0	1	0	1	0	0	1
M7	2	0	1	1	0	0	0
M9	1	0	1	0	0	0	1
M10	1	0	1	0	0	1	0
M13	0	1	0	0	1	0	1
M14	0	1	0	1	0	1	0

Table 5.2b – Reachability Set of GSPN

Chapter Six

Learning Petri Net Models

6.1 Introduction

This chapter draws together methods described in the previous chapters to form a novel extension of PNs by embedding a stochastic learning automata within PN models. In the discussion that follows the progressive stages of the development of a powerful modelling tool for C³-I systems is provided. The decision making process can be modelled with a new type of hybrid PN, namely *Stochastic Learning Petri Net (SLPN)*, [58], [59]. In such a manner for the first time an AI based decision making process is embedded within PNs. This hybrid PN structure enables models of arbitrary topology to be simulated, and the application of this modelling tool is discussed in the next Chapter.

6.2 Basic Stochastic Learning Petri Net (Basic SLPN)

This section introduces a new class of PN, referred to as *Basic Stochastic Learning Petri net (Basic SLPN)* as depicted in Figure 6.1a. It has been formed by incorporating the concept of stochastic learning automata into a SPN model. Consider the model of the Basic SLPN in Figure 6.1a, and the corresponding reachability tree, as depicted in Figure 6.1b. It is clear from Figure 6.1a that the Basic SLPN provides structure to the original

stochastic learning automata by the additional concept of tokens in the net; thus describing precisely the interactions involved.

A formal definition of the Basic SLPN structure is thus the following:

$$\text{BasicSLPN} = (P, T, A, M_0, \beta, \phi, \alpha, p, F, G) \quad (6.1)$$

where (P, T, A, M_0) is the *PN* underlying the model; $(\beta, \phi, \alpha, p, F, G)$ is the stochastic learning automata underlying the Basic SLPN. In this representation $Pr\{\alpha_i\} = Pr\{t_i\}$ indicates the transition firing probabilities.

The interpretation of the model is similar to the case of stochastic learning automata with the additions resulting from the introduction of Stochastic Petri nets.

6.2.1 Simulation Results: Basic SLPN

The simulation results for a two-state Basic SLPN are provided in Table 6.1. Similar to the previous simulations an L_{RI} updating scheme is used; the reward parameter and expected values are provided. The probabilities associated with the firing of each transition are equal; the initial value is equal to 0.5. Consider the results in Table 6.1 and the corresponding learning curve shown in Figure 6.2, it is evident that the transition associated with the unique maximum reward probability converge close to unity. This highlights the intelligence capability embedded within a Petri net model, and also a modification of the basic stochastic automaton model arises due to the disposition of tokens in the net, thus providing a graphical description to the model.

6.3 Stochastic Learning Petri Net (SLPN)

This section discusses the various stages involved in the extension of SPN model to a new class of Petri nets, namely, *Stochastic Learning Petri nets (SLPN)* as depicted in Figure 6.3. Consider the model of the SPN, shown in Figure 5.5. By analysis of the reachability tree in Figure 5.6, it is evident that the SPN model may exhibit one of six different states, depending on the transition that fires. Several transitions may be simultaneously enabled by a particular marking. Assume that H is the set of enabled transitions, then a transition t_i ($i \in H$) fires with probability:

$$Pr\{t_i\} = \frac{\lambda}{\sum_{k \in H} \lambda_k} \quad (6.2)$$

exactly as in case of SPNs, λ is the firing rate associated with PN transitions. Thus, the different states of a SPN define probability ratios which correspond to the firing of each transition. In any state, the sum of the probability ratios is always equal to unity. For example, consider state $M_1 = [1100]$; the enabled transitions are t_1 , t_2 and t_3 and their respective firing probabilities may be defined as follows:

$$Pr\{t_1\} = \frac{\alpha}{(\alpha + \beta + \gamma)} \quad (6.3)$$

$$Pr\{t_2\} = \frac{\beta}{(\alpha + \beta + \gamma)} \quad (6.4)$$

$$Pr\{t_3\} = \frac{\gamma}{(\alpha + \beta + \gamma)} \quad (6.5)$$

Thus,

$$Pr\{t_1\} + Pr\{t_2\} + Pr\{t_3\} = 1 \quad (6.6)$$

6.3.1 Reachability Tree : Stochastic Automata Embedded

In the tree representation, several transitions may be simultaneously enabled in any particular marking. The concept of a stochastic automaton may be introduced to select probabilistically the transition that fires. A transition selected in a particular marking corresponds to an action selected by an automaton. The firing of the chosen transition determines the next state (marking) of the system, by modifying the token distribution. In the tree representation of the SPN, Figure 5.6, there exists both two-state and three-state automata. This is illustrated in Figure 6.4. Consider the following cases:

Two-state Automaton

It is clear that state $M_2[0200]$ and state $M_3[1011]$ in Figure 6.4 represent a two-state automaton as shown in Figure 6.5a. The SPN with marking M_2 enables transitions t_2 and t_3 , since tokens are present in the input place (p_2). Each transition has an equal initial probability of being selected. The firing of t_2 , determines the next state of SPN to be M_1 ; the firing of t_3 , determines that the next state is M_4 . The firing probabilities for each transition are given as follows:

$$Pr\{t_2\} = \frac{\beta}{(\beta + \gamma)} \quad (6.7)$$

$$Pr\{t_3\} = \frac{\gamma}{(\beta + \gamma)} \quad (6.8)$$

Similarly,

$$Pr\{t_2\} + Pr\{t_3\} = 1 \quad (6.9)$$

The concept of a two-state automaton also applies to state M_3 which has the possibility of firing two transitions, t_1 and t_4 ; the firing of these transitions determines the next state to be M_4 and M_0 , respectively.

Three-state Automaton

Clearly, the states M_1 and M_4 correspond to a three-state automaton. By considering the marking M_1 , the case is illustrated in Figure 6.5b. It is shown that the transitions t_1 , t_2 and t_3 are enabled; each transition has an equal initial probability of being selected. The possibility of firing t_1 , determines that the next state is M_2 ; the firing of t_2 determines that the next state of the SPN to be M_0 ; finally, if t_3 is selected by the automaton then the state transfers to M_3 .

A similar concept also applies to state M_4 . In this case, the three-state automaton has the possibility of selecting t_2 , t_3 or t_4 with equal initial probabilities. The firing of transitions t_2 , t_3 or t_4 determines the next states as M_3 , M_5 and M_1 , respectively.

Transitions Fire Instantly

Note that the transition firing probabilities in each state M_0 and M_5 are always equal to unity. Since in state M_0 , the only transition that is enabled

is t_1 ,

$$Pr\{t_1\} = \frac{\alpha}{\alpha} = 1 \quad (6.10)$$

Thus, it must fire with probability one. Similarly, in state M_5 the only transition that is enabled is t_4 ,

$$Pr\{t_4\} = \frac{\delta}{\delta} = 1 \quad (6.11)$$

so it must also fire with probability equal to unity.

6.3.2 Hierarchical System of Automata

The reachability tree may now be considered as a simple hierarchical system of automata; each state corresponding to an automaton. It may be noted that in a hierarchy each firing of a transition (action selected by automaton) has a unique path connecting it to the automaton (state) that has been selected previously, or to an automaton at the top level (state M_0). From the tree structure of Figure 6.4, it is possible to define nine unique paths which may be considered as sequence of states/ decisions, shown in Figure 6.6. To introduce the concept of an environment into this model, each sequence of states is associated with a reward probability, indicated by c_i values as illustrated in Figure 6.6. Such a system may be considered as a *Stochastic Learning Petri Net (SLPN)* model; this structure is shown in Figure 6.7.

6.3.3 Operation of SLPN

The operation of this hierarchical learning system is as follows. At any instant the first level automaton, state M_0 selects an action (fires t_1). This activates an automaton in the second level which fires a transition from its current transition probability distribution. This in turn activates, automata in the next level and so on. However, if a particular sequence of decisions corresponding to a unique path has been reached; the sequence is fed into the environment. The environment in turn generates a reward/punish signal as its reaction. The response of the environment is used to update the transition probabilities for the various levels of automata in the selected path. This process repeats until all the probabilities in one path converge close to unity (ie. path associated with unique maximum reward probability or unique minimum penalty probability) from the top level (M_0) to the lowest level (M_5).

Thus, the formal definition of a SLPN, is as follows:

$$SLPN = (P, T, A, M_0, \lambda, M_x) \quad (6.12)$$

where (P, T, A, M_0, λ) is the *stochastic Petri net* underlying the model; M_x indicates the presence of two/ three-state stochastic learning automata which consist of the components $(\beta, \phi, \alpha, p, F, G)$. In this representation $Pr\{\alpha_i\} = Pr\{t_i\}$ indicates the transition firing probabilities.

Similarly, the interpretation of the model is identical to the case of stochastic learning automata with the additions resulting from the introduction of Stochastic Petri nets.

6.3.4 Simulation Results : SLPN

This section presents computer simulation results for the SLPN model. The results are presented in the form of tables. In all cases the reward parameter is indicated; and $Pr(i,j)$ denotes the transition firing probabilities, where i represents the state of the system and j provides the notation for the transition that fires. For example, consider the notation for state M_1 firing transition t_3 ; the transition firing probability may be represented by $Pr(1,3)$. Expected values are denoted by the expression eg. $Pr(i,j) = E[Pr(i,j)]$.

In the simulation study the hierarchical system in Figure 6.4 was examined. Such a tree representation was modified by introducing the stochastic learning automaton approach with the capability of selecting sequences of decisions, discussed previously. To simulate this SLPN, all of the reward probabilities in the environment were in the range [0.2 - 0.45] except the unique maximum reward probability which was set to 0.9. An L_{RI} updating scheme was used to update action probabilities for the selected path.

Tables [6.2 - 6.10] provide the reward probabilities of the environment which are used for simulation. Note that in each case the unique maximum reward probability is associated with the selected sequence of decisions. Consider Table 6.2, the maximum reward probability relates to sequence 0. This sequence corresponds to selecting the path: $M0 - Pr(0,1); M1 - Pr(1,2); M0$, repeatedly. Hence, the optimal path probability changes in Table 6.2 may be analysed. The SLPN in state $M0$ always fires t_1 with probability unity; thus the transition probability in $M0$ always remains constant. The firing of t_1 results in changing the next state to $M1$. In state $M1$, the transitions t_1, t_2

and t_3 are enabled. However, the rapid convergence of $Pr(1,2)$ in state M1 indicates that the optimal path converges to sequence 0, by the firing of t_2 .

Table 6.3 illustrates the convergence to the unique maximum reward probability, such that sequence 1 is selected. This sequence represents the path $M0 - Pr(0, 1); M1 - Pr(1, 1); M2 - Pr(2, 2); M1$. In this case, transition probability vector in state M0 is equal to unity; since t_1 must always fire with probability equal to one. Also the convergence of transition probability $Pr(1,1)$ in the three-state automaton M1; and $Pr(2,2)$ in the two-state automaton M2 show that the optimal path selected is sequence 1, which has the unique maximum reward probability.

Similarly, for Table [6.4 - 6.10]. It is observed that the transition probability vectors that converge close to unity, correspond to the sequence of decisions associated with the unique maximum reward probability.

6.4 Generalised Stochastic Learning Petri Net (GSLPN)

A similar approach is adopted for the development of GSLPN. By analysis of the reachability tree in Figure 5.8, the GSPN may exist in one of eighteen different states. These states provide a combination of immediate and timed transitions. However, for the development of a *Generalised* version of the *Stochastic Learning Petri Net* it is necessary to consider only timed transitions, since firing rates associated with immediate transitions are determined by switching distribution. Such that, if several timed transitions are simultaneously enabled in any tangible marking; and assuming that H is the

set of enabled transitions, then a transition t_i ($i \in H$) fires with probability

$$Pr\{t_1\} = \frac{\lambda}{\sum_{k \in H} \lambda_k} \quad (6.13)$$

as stated previously, λ is the firing rate associated with PN transitions. Thus, the different states relating to timed transitions of a GSPN define probability ratios which correspond to the firing of each transition. In any state, the sum of probability ratios is always equal to unity. Consider state $M_2 = [1000110]$; the enabled transitions are t_1 and t_6 and their respective firing probabilities may be defined as follows:

$$Pr\{t_1\} = \frac{u}{(u + v)} \quad (6.14)$$

and

$$Pr\{t_6\} = \frac{v}{(u + v)} \quad (6.15)$$

Thus,

$$Pr\{t_1\} + Pr\{t_6\} = 1 \quad (6.16)$$

For this model of a GSPN, the concept of a stochastic automaton has been introduced to control the firing of timed transitions on a probabilistic basis. The firing of the selected transition in a tangible marking determines that the next state (marking) of the system corresponds to a vanishing marking; thus enabling only immediate transitions. Such transitions are then controlled by the switching distribution technique. In the tree representation in Figure 6.8, there exists only two-state automata for each tangible marking. Clearly, the tangible states M_2 , M_3 , M_8 , M_{11} and M_{12} correspond to a two-state automata.

Two-state Automaton

Consider, Figure 6.8, the tangible marking $M_2[1000110]$ enables transitions t_1 and t_6 , since tokens are present in the input places (p_1, p_6). Each transition can fire first with equal initial probabilities, firing probabilities for each transition given below.

$$Pr\{t_1\} = \frac{u}{(u + v)} \quad (6.17)$$

$$Pr\{t_6\} = \frac{v}{(u + v)} \quad (6.18)$$

Similarly

$$Pr\{t_1\} + Pr\{t_6\} = 1 \quad (6.19)$$

The firing of t_1 , determines the next state of GSPN to be vanishing marking M_4 ; the firing of t_6 , also determines that the next state is vanishing marking M_5 . The concept of two-state automaton also applies to each tangible marking, namely, M_3, M_8, M_{11} and M_{12} .

It must be pointed out that the transition firing probabilities in each tangible marking $M_0; M_{15}$ and M_{16} is always equal to unity. Since in state M_0 , the only transition that is enabled is t_1 .

$$Pr\{t_1\} = \frac{u}{u} = 1 \quad (6.20)$$

Thus, it must fire with probability one. Similarly, in state M_{15} and M_{16} the only transitions enabled are t_7 and t_6 , respectively.

$$Pr\{t_7\} = Pr\{t_6\} = 1 \quad (6.21)$$

so it must also fire with probability equal to unity.

Hierarchical System of Automata – GSLPN

Thus, from the reachability tree a simple hierarchical system of automata is developed; each tangible state corresponding to an automaton. Similar to the case of SLPN, in a hierarchy each action has a unique path connecting it to the automaton (state) that has been selected previously, or to an automaton at the top level (state M_0). By considering the tree structure of Figure 6.8, nine unique paths may be defined which are considered as sequence of states/decisions, shown in Figure 6.9. To introduce the concept of an environment into this model, each sequence of states is associated with a reward probability.

The operation of this hierarchical learning system is similar to the SLPN, with the addition of switching distributions associated with vanishing markings. If a vanishing marking is reached, the next state is determined by considering switching distributions (random switch), presented in Table 5.2a. The firing of transitions according to a random switch alters the next state to a tangible marking, depending on the transition that fires. At any instant the first level automaton, state M_0 , selects an action (fires t_1). This activates an automaton in the second level which fires a transition from its current transition probability distribution. This in turn activates, automata in the next level and so on. However, if a particular sequence of decisions corresponding to a unique path has been reached; the environment in turn generates a reward/ punish signal as its reaction. The reaction of the environment is used to update the transition probabilities for the various

levels of automata in the selected path. This process repeats until all the probabilities in one path become close to unity from the top level (M_0) to the lowest level (M_5). Such a system may be considered as a *Generalised Stochastic Learning Petri Net (GSLPN)* model.

6.4.1 Simulation Results : GSLPN

This section presents computer simulation results for the GSLPN model. The results are presented in the form of tables; indicating reward parameter is equal to 0.1, also the reward probabilities associated with each sequence of states. The notation for $Pr(i, j)$ remains the same as for SLPN, as indicated in Section 6.3.4. In the simulation study the hierarchical system was modified by introducing the stochastic learning automaton approach for each tangible marking. By considering each tangible state as an automaton, this provides the capability of selecting sequences of decisions, discussed previously. For the simulation of GSLPN, all of the reward probabilities in the environment were in the range [0.2 - 0.45] except the unique maximum reward probability which was set to 0.9. An L_{RI} updating scheme was used to update action probabilities for the selected path.

Tables [6.11 - 6.19] provide the reward probabilities of the environment which are used for simulation. Note that in each case the unique maximum reward probability is associated with the selected sequence of decisions.

From these results it can be seen that the transition probability vectors of certain tangible states converge close to unity. In each case this convergence corresponds to the sequence associated with the unique maximum

reward probability.

6.5 Conclusion and Summary

This chapter has introduced a new class of hybrid Petri nets which have the additional feature of an embedded stochastic learning automata within Petri net models. By embedding the concept of stochastic learning automata in Petri nets the hybrid combination was shown to overcome the limitations of, existing Petri net theory and interconnected automata used in isolation. An extension of a standard PN, SPN and GSPN, have developed new hybrid models known as *Basic Stochastic Learning Petri Net (Basic SLPN)*, *Stochastic Learning Petri Net (SLPN)* and *Generalised Stochastic Learning Petri Net (GSLPN)*, respectively. In the case of a Basic SLPN it has been shown that the movement of tokens in the model provides structure to the stochastic learning automata described in Chapter Three. Whilst, the SLPN and GSLPN models have the ability to control the firing of transitions on a probabilistic basis; and enables convergence to a selected sequence of states/ decisions at each time instant. Preliminary simulation results are presented for each Learning Petri net model. The next chapter considers an application of the SLPN model to a specific two node decision making organisation.

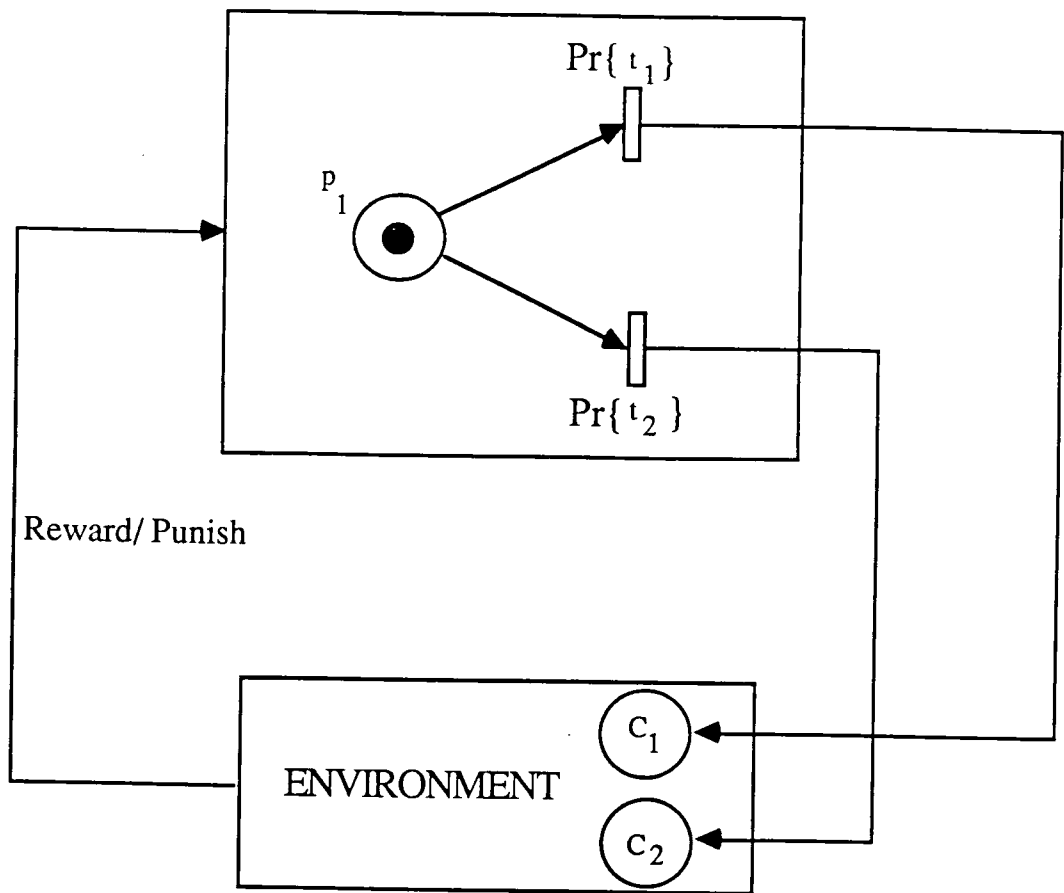


Figure 6.1a - Basic Stochastic Learning Petri Net (Basic SLPN)

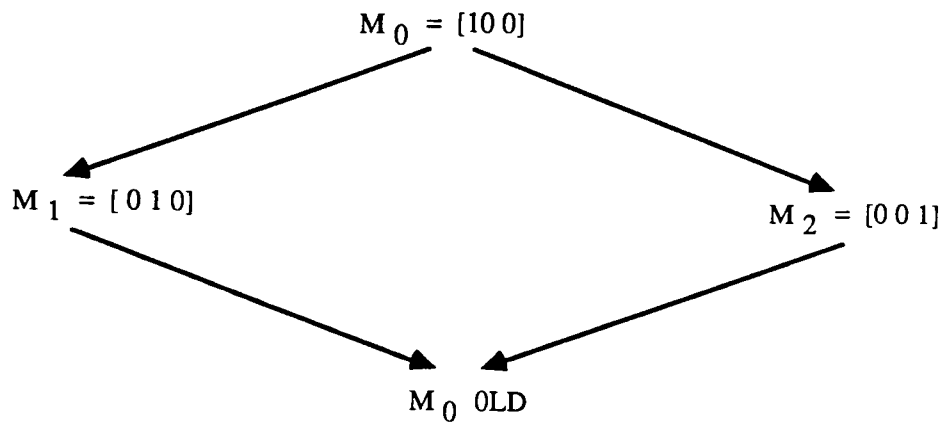


Figure 6.1b - Reachability Tree - Basic SLPN

Reward Parameter = 0.04		
Reward Probability		
i	0	1
Ci	0.2	0.9

n	p1	p2
0	0.500000	0.500000
200	0.185784	0.814216
400	0.096207	0.903793
600	0.064346	0.935654
800	0.048341	0.951659
1000	0.038721	0.961279

Table 6.1 – Optimal Path : Basic SLPN

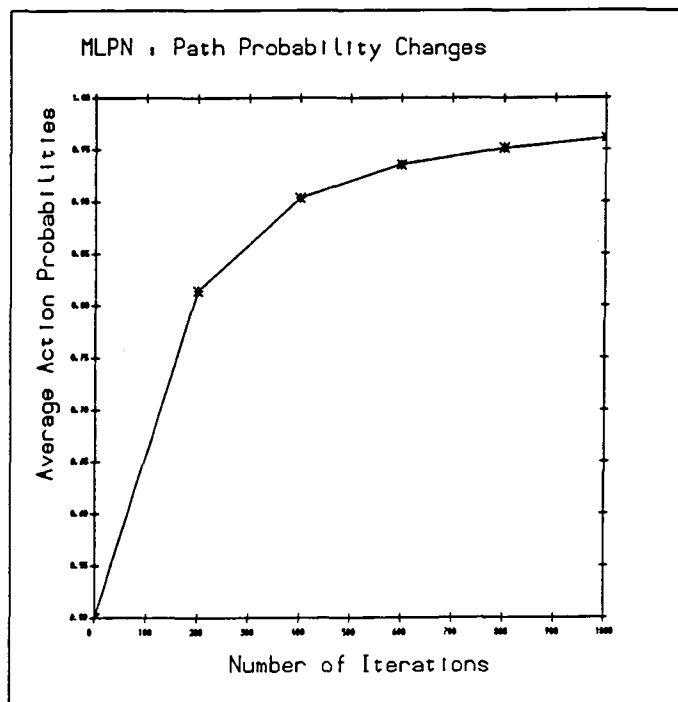


Figure 6.2 – Optimal Path Probability Changes : Basic SLPN

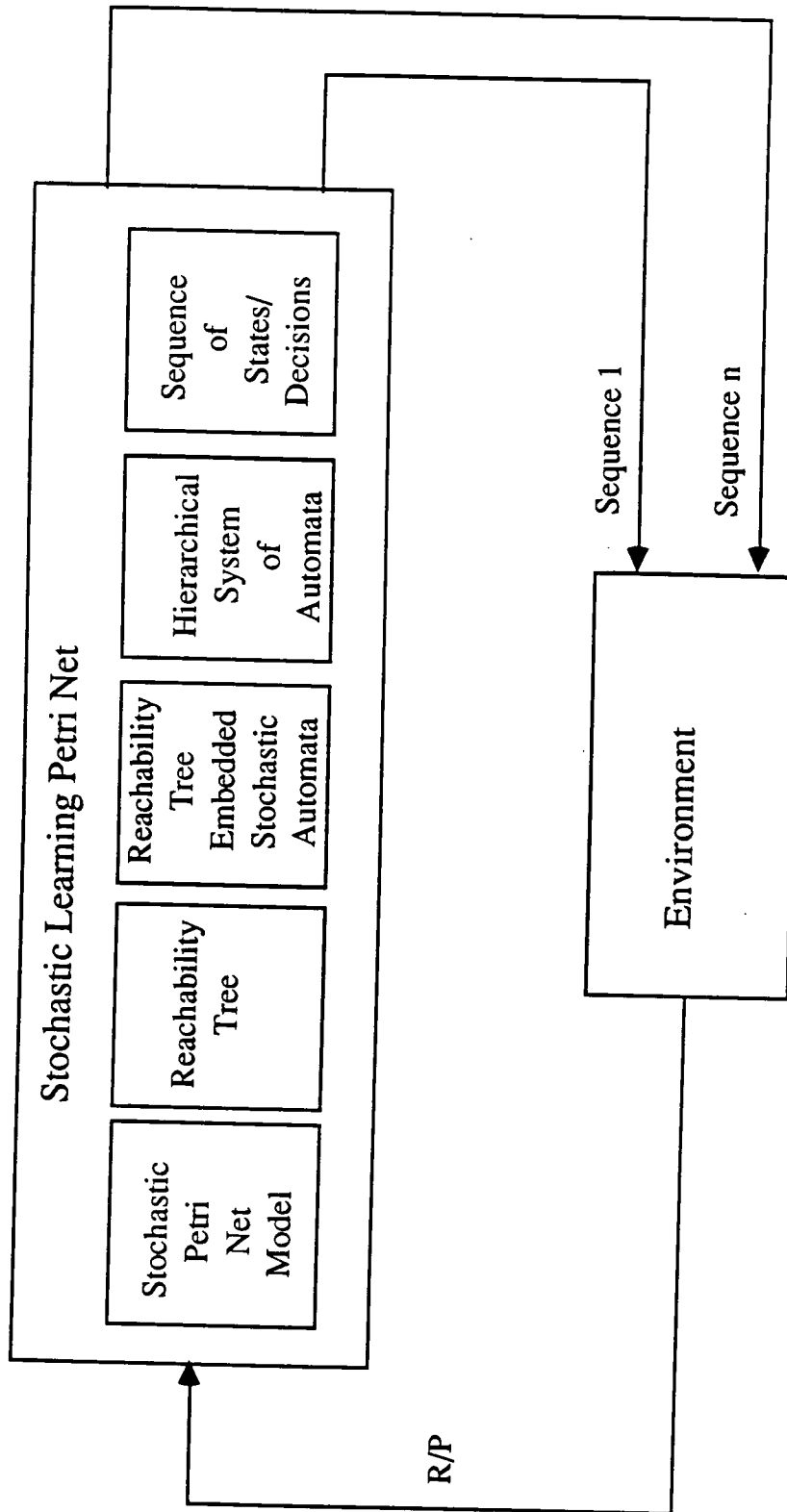
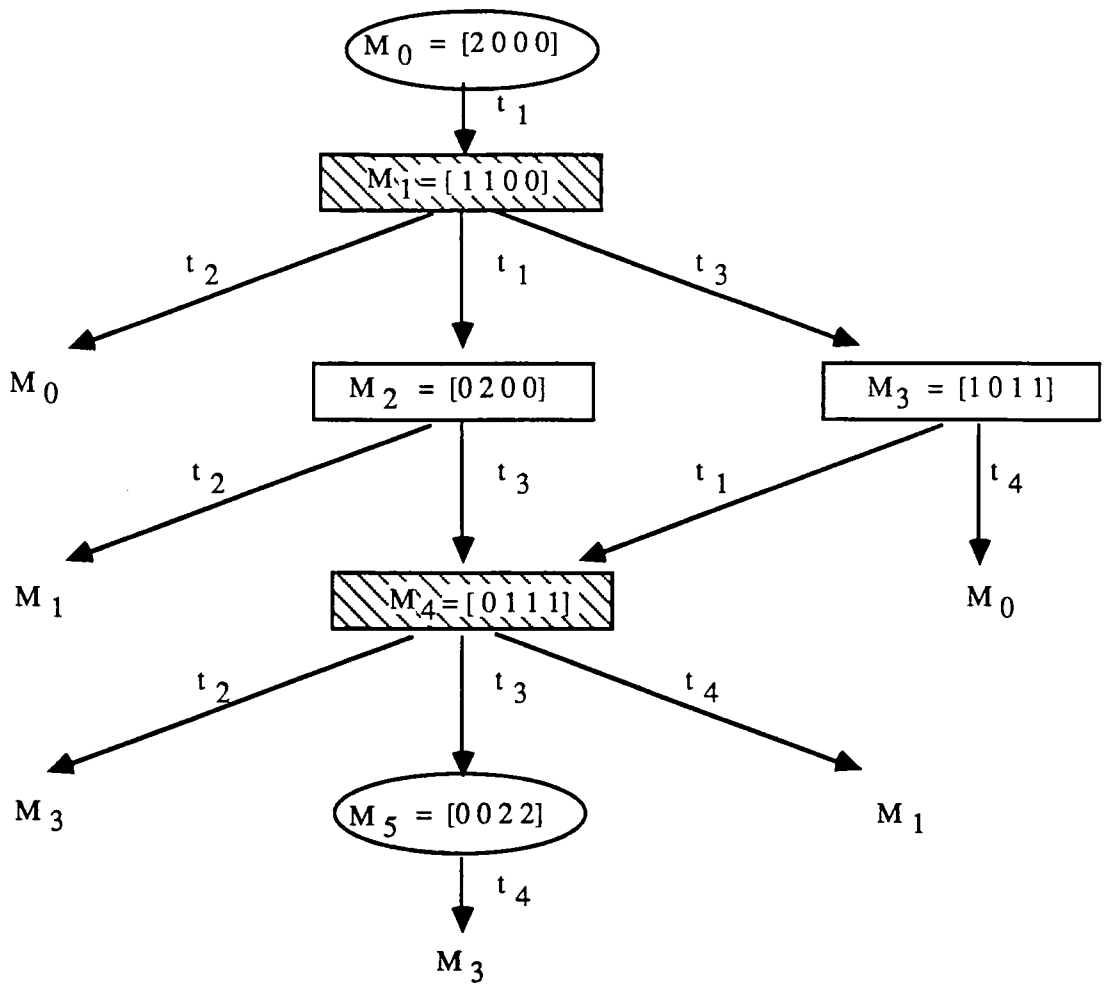


Figure 6.3 – Structure of SLPN



ENVIRONMENT									
Sequence i	0	1	2	3	4	5	6	7	8
C_i	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8

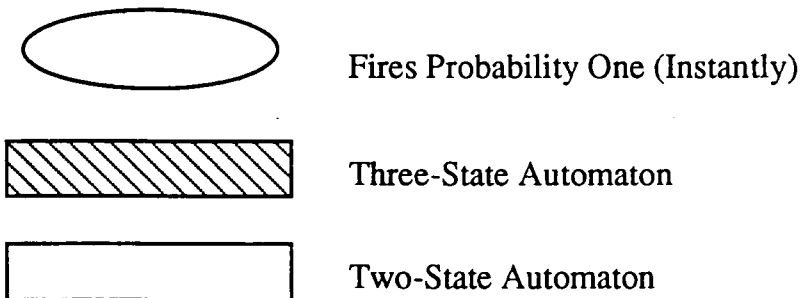
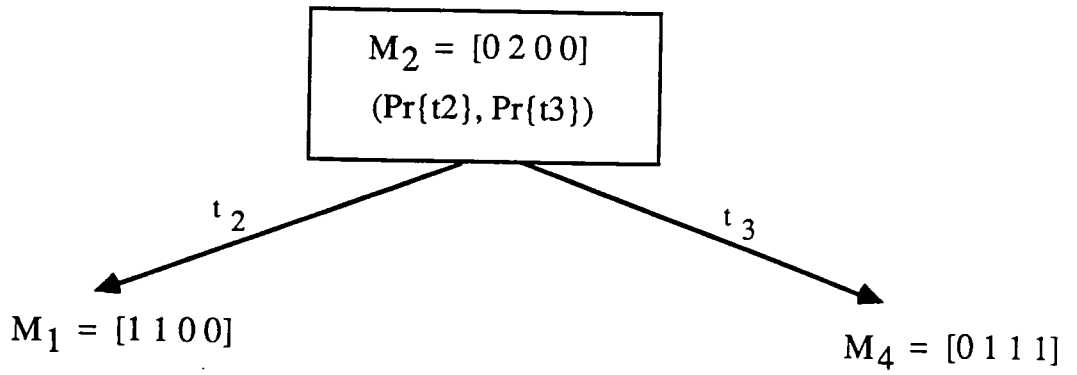
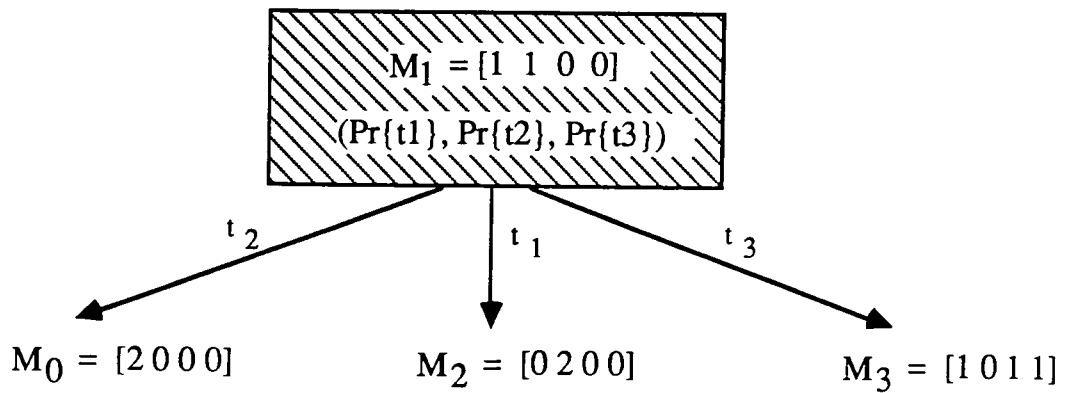


Figure 6.4 – Reachability Tree : Embedded Stochastic Automata

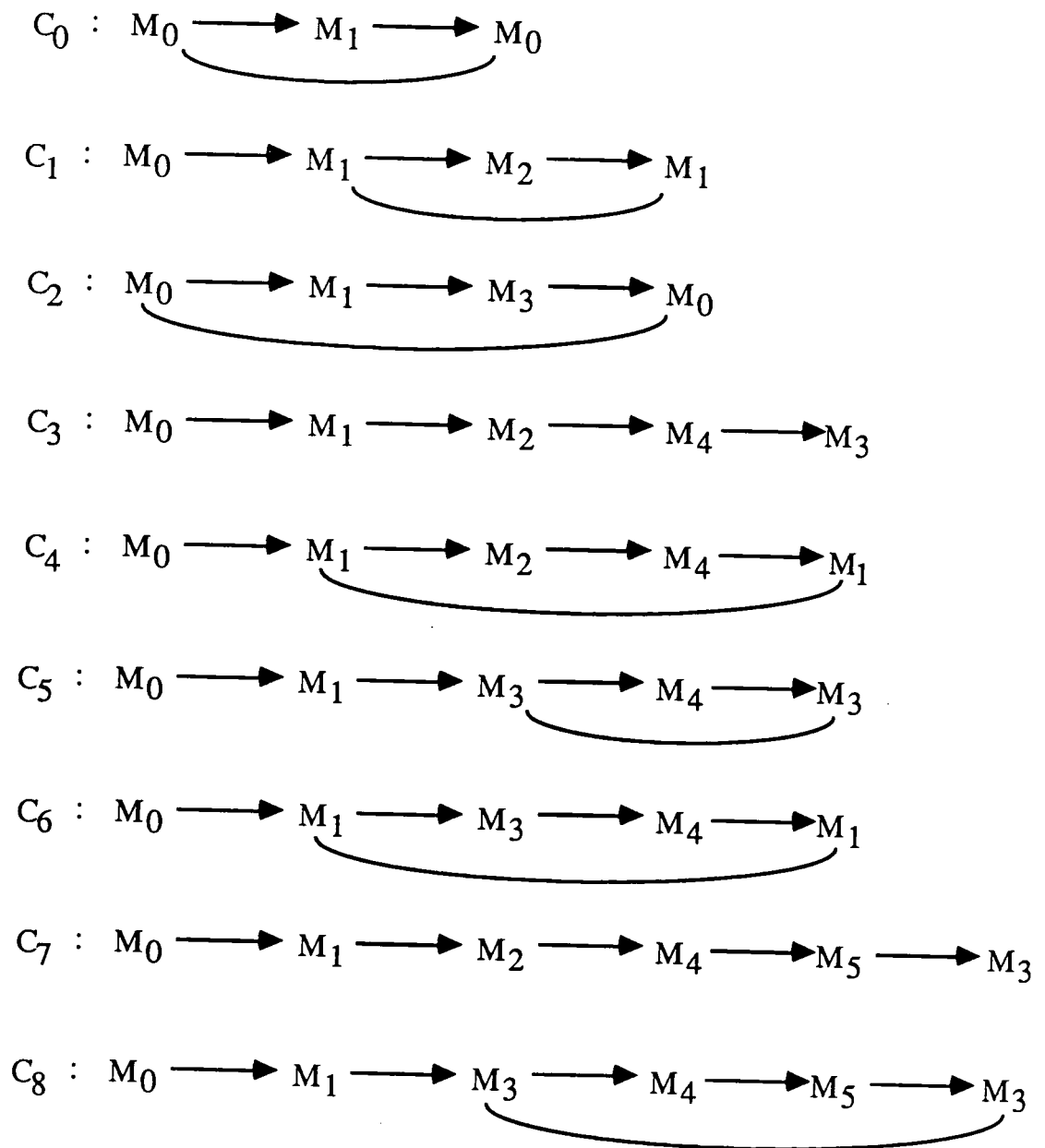


(a) Two-State Automaton



(b) Three-State Automaton

Figure 6.5 - Two/ Three-State Automaton



[$C_0 - C_8$] Penalty Probabilities - Range[0.2-0.5] Unique Maximum = 0.9

[$M_0 - M_5$] Sequence of States in Reachability Tree

Figure 6.6 – Sequence of Decisions/ States

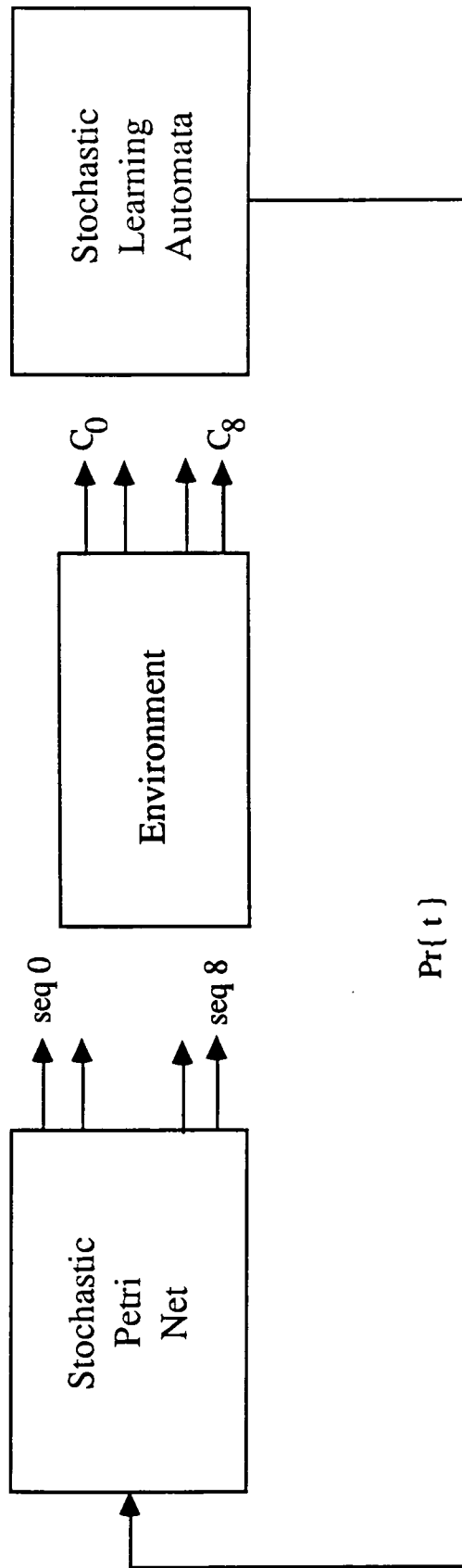


Figure 6.7 – Stochastic Learning Petri Net (SLPN)

Reward Parameter = 0.04									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.9	0.35	0.45	0.2	0.3	0.2	0.3	0.4	0.4

Convergence to Sequence 0:

M0 - Pr(0,1); M1 - Pr(1,2); M0;

	M0	M1			M2	
n	Pr(0,1)	Pr(1,1)	Pr(1,2)	Pr(1,3)	Pr(2,2)	Pr(2,3)
0	1.000000	0.333333	0.333333	0.333333	0.500000	0.500000
600	1.000000	0.174549	0.724323	0.101128	0.496244	0.503756
1200	1.000000	0.099875	0.849360	0.050765	0.470108	0.529892
1800	1.000000	0.066647	0.899482	0.033871	0.460627	0.539373
2400	1.000000	0.050013	0.924569	0.025418	0.455883	0.544117
3000	1.000000	0.040027	0.939630	0.020343	0.453034	0.546966

	M3		M4			M5
n	Pr(3,1)	Pr(3,4)	Pr(4,2)	Pr(4,3)	Pr(4,4)	Pr(5,4)
0	0.500000	0.500000	0.333333	0.333333	0.333333	1.000000
600	0.518667	0.481333	0.277263	0.439687	0.283050	1.000000
1200	0.519332	0.480668	0.250052	0.460671	0.289278	1.000000
1800	0.519554	0.480446	0.240587	0.468098	0.291315	1.000000
2400	0.519666	0.480334	0.235851	0.471814	0.292335	1.000000
3000	0.519732	0.480268	0.233008	0.474045	0.292947	1.000000

Table 6.2 - Optimal Path SLPN : Sequence 0

Reward Parameter = 0.04									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.9	0.45	0.2	0.3	0.2	0.3	0.4	0.4

Convergence to Sequence 1:

M0 - Pr(0,1); M1 - Pr(1,1); M2 - Pr(2,2); M1

	M0	M1			M2	
n	Pr(0,1)	Pr(1,1)	Pr(1,2)	Pr(1,3)	Pr(2,2)	Pr(2,3)
0	1.000000	0.333333	0.333333	0.333333	0.500000	0.500000
600	1.000000	0.550452	0.213781	0.235767	0.709502	0.290498
1200	1.000000	0.728851	0.113456	0.157693	0.840304	0.159696
1800	1.000000	0.818899	0.075722	0.105380	0.893398	0.106602
2400	1.000000	0.864098	0.056823	0.079079	0.920004	0.079996
3000	1.000000	0.891233	0.045477	0.063290	0.935977	0.064023

	M3		M4			M5
n	Pr(3,1)	Pr(3,4)	Pr(4,2)	Pr(4,3)	Pr(4,4)	Pr(5,4)
0	0.500000	0.500000	0.333333	0.333333	0.333333	1.000000
600	0.453162	0.546838	0.367937	0.324951	0.307112	1.000000
1200	0.386330	0.613670	0.384717	0.330612	0.284671	1.000000
1800	0.362558	0.637442	0.390320	0.332502	0.277178	1.000000
2400	0.350662	0.649338	0.393123	0.333448	0.273429	1.000000
3000	0.343521	0.656479	0.394806	0.334016	0.271178	1.000000

Table 6.3 – Optimal Path SLPN : Sequence 1

Reward Parameter = 0.04									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.45	0.9	0.2	0.3	0.2	0.3	0.4	0.4

Convergence to Sequence 2:

M0 - Pr(0,1); M1 - Pr(1,3); M3 - Pr(3,4); M0;

	M0	M1			M2	
n	Pr(0,1)	Pr(1,1)	Pr(1,2)	Pr(1,3)	Pr(2,2)	Pr(2,3)
0	1.000000	0.333333	0.333333	0.333333	0.500000	0.500000
600	1.000000	0.176031	0.124048	0.699921	0.481165	0.518835
1200	1.000000	0.090713	0.067059	0.842229	0.481793	0.518207
1800	1.000000	0.060535	0.044761	0.894704	0.482003	0.517997
2400	1.000000	0.045426	0.033589	0.920984	0.482108	0.517892
3000	1.000000	0.036356	0.026883	0.936761	0.482171	0.517829

	M3		M4			M5
n	Pr(3,1)	Pr(3,4)	Pr(4,2)	Pr(4,3)	Pr(4,4)	Pr(5,4)
0	0.500000	0.500000	0.333333	0.333333	0.333333	1.000000
600	0.199988	0.800012	0.400115	0.285478	0.314407	1.000000
1200	0.103941	0.896059	0.400705	0.273219	0.326076	1.000000
1800	0.069366	0.930634	0.400902	0.269126	0.329972	1.000000
2400	0.052053	0.947947	0.401001	0.267078	0.331921	1.000000
3000	0.041660	0.958340	0.401060	0.265848	0.333092	1.000000

Table 6.4 - Optimal Path SLPN : Sequence 2

Reward Parameter = 0.04									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.2	0.45	0.9	0.3	0.2	0.3	0.4	0.4

Convergence to Sequence 3:

M0 - Pr(0,1); M1 - Pr(1,1); M2 - Pr(2,3); M4 - Pr(4,2); M3

	M0	M1			M2	
n	Pr(0,1)	Pr(1,1)	Pr(1,2)	Pr(1,3)	Pr(2,2)	Pr(2,3)
0	1.000000	0.333333	0.333333	0.333333	0.500000	0.500000
600	1.000000	0.553201	0.191477	0.255323	0.576526	0.423474
1200	1.000000	0.691943	0.124194	0.183863	0.413566	0.586434
1800	1.000000	0.792181	0.083418	0.124401	0.302268	0.697732
2400	1.000000	0.844025	0.062604	0.093371	0.233046	0.766954
3000	1.000000	0.875168	0.050104	0.074728	0.186623	0.813377

	M3		M4			M5
n	Pr(3,1)	Pr(3,4)	Pr(4,2)	Pr(4,3)	Pr(4,4)	Pr(5,4)
0	0.500000	0.500000	0.333333	0.333333	0.333333	1.000000
600	0.572875	0.427125	0.418042	0.274119	0.307839	1.000000
1200	0.613918	0.386082	0.565688	0.198648	0.235663	1.000000
1800	0.632179	0.367821	0.695293	0.134768	0.169939	1.000000
2400	0.641316	0.358864	0.770897	0.101162	0.127941	1.000000
3000	0.646802	0.353198	0.816636	0.080963	0.102400	1.000000

Table 6.5 – Optimal Path SLPN : Sequence 3

Reward Parameter = 0.04									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.2	0.45	0.3	0.9	0.2	0.3	0.4	0.4

Convergence to Sequence 4:

M0 - Pr(0,1); M1 - Pr(1,1); M2 - Pr(2,3); M4 - Pr(4,4); M1;

	M0	M1			M2	
n	Pr(0,1)	Pr(1,1)	Pr(1,2)	Pr(1,3)	Pr(2,2)	Pr(2,3)
0	1.000000	0.333333	0.333333	0.333333	0.500000	0.500000
600	1.000000	0.513904	0.313428	0.172667	0.468298	0.531702
1200	1.000000	0.659712	0.236757	0.103531	0.329785	0.670215
1800	1.000000	0.762728	0.167303	0.069970	0.226465	0.773535
2400	1.000000	0.821845	0.125640	0.052514	0.170000	0.830000
3000	1.000000	0.857416	0.100555	0.042029	0.136058	0.863942

	M3		M4			M5
n	Pr(3,1)	Pr(3,4)	Pr(4,2)	Pr(4,3)	Pr(4,4)	Pr(5,4)
0	0.500000	0.500000	0.333333	0.333333	0.333333	1.000000
600	0.462825	0.537175	0.226089	0.309543	0.464368	1.000000
1200	0.452119	0.547881	0.133066	0.231743	0.635191	1.000000
1800	0.444685	0.555315	0.089236	0.157893	0.752871	1.000000
2400	0.440965	0.559035	0.066969	0.118515	0.814517	1.000000
3000	0.438731	0.561269	0.053597	0.094852	0.851551	1.000000

Table 6.6 – Optimal Path SLPN : Sequence 4

Reward Parameter = 0.04									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.2	0.45	0.3	0.2	0.9	0.3	0.4	0.4

Convergence to Sequence 5:

M0 - Pr(0,1); M1 - Pr(1,3); M3 - Pr(3,1); M4 - Pr(4,2); M5;

	M0	M1			M2	
n	Pr(0,1)	Pr(1,1)	Pr(1,2)	Pr(1,3)	Pr(2,2)	Pr(2,3)
0	1.000000	0.333333	0.333333	0.333333	0.500000	0.500000
600	1.000000	0.191899	0.222948	0.585153	0.500762	0.499238
1200	1.000000	0.103439	0.122696	0.773865	0.490015	0.509985
1800	1.000000	0.069090	0.081975	0.848935	0.486426	0.513574
2400	1.000000	0.051847	0.061516	0.886637	0.484631	0.515369
3000	1.000000	0.041495	0.049234	0.909272	0.483553	0.516447

	M3		M4			M5
n	Pr(3,1)	Pr(3,4)	Pr(4,2)	Pr(4,3)	Pr(4,4)	Pr(5,4)
0	0.500000	0.500000	0.333333	0.333333	0.333333	1.000000
600	0.674278	0.325722	0.501654	0.281078	0.217269	1.000000
1200	0.815019	0.184981	0.686260	0.182217	0.131523	1.000000
1800	0.875323	0.124677	0.787389	0.124113	0.088498	1.000000
2400	0.906430	0.093570	0.840427	0.093156	0.066417	1.000000
3000	0.925113	0.074887	0.872288	0.074556	0.053155	1.000000

Table 6.7 – Optimal Path SLPN : Sequence 5

Reward Parameter = 0.04									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.2	0.45	0.3	0.2	0.3	0.9	0.4	0.4

Convergence to Sequence 6:

M0 - Pr(0,1); M1 - Pr(1,3); M3 - Pr(3,1); M4 - Pr(4,4); M1;

	M0	M1			M2	
n	Pr(0,1)	Pr(1,1)	Pr(1,2)	Pr(1,3)	Pr(2,2)	Pr(2,3)
0	1.000000	0.333333	0.333333	0.333333	0.500000	0.500000
600	1.000000	0.165438	0.290162	0.544400	0.522279	0.477721
1200	1.000000	0.106961	0.168110	0.724929	0.529368	0.470632
1800	1.000000	0.083200	0.112413	0.804387	0.557758	0.442242
2400	1.000000	0.062654	0.084359	0.852986	0.573471	0.426529
3000	1.000000	0.050147	0.067516	0.882338	0.582904	0.417096

	M3		M4			M5
n	Pr(3,1)	Pr(3,4)	Pr(4,2)	Pr(4,3)	Pr(4,4)	Pr(5,4)
0	0.500000	0.500000	0.333333	0.333333	0.333333	1.000000
600	0.693759	0.306241	0.206450	0.192350	0.601200	1.000000
1200	0.821326	0.178674	0.133834	0.108875	0.757291	1.000000
1800	0.880258	0.119742	0.092569	0.072814	0.834617	1.000000
2400	0.910139	0.089861	0.069499	0.054643	0.875859	1.000000
3000	0.928081	0.071919	0.055622	0.043732	0.900645	1.000000

Table 6.8 - Optimal Path SLPN : Sequence 6

Reward Parameter = 0.04									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.2	0.45	0.3	0.2	0.3	0.4	0.9	0.4

Convergence to Sequence 7:

M0 - Pr(0,1); M1 - Pr(1,1); M2 - Pr(2,3); M4 - Pr(4,3); M3;

	M0	M1			M2	
n	Pr(0,1)	Pr(1,1)	Pr(1,2)	Pr(1,3)	Pr(2,2)	Pr(2,3)
0	1.000000	0.333333	0.333333	0.333333	0.500000	0.500000
600	1.000000	0.531733	0.290461	0.177806	0.441565	0.558435
1200	1.000000	0.640113	0.224555	0.135332	0.321635	0.678365
1800	1.000000	0.740396	0.165010	0.094594	0.227745	0.772255
2400	1.000000	0.804660	0.124269	0.071071	0.171196	0.828804
3000	1.000000	0.843651	0.099467	0.056882	0.137020	0.862980

	M3		M4			M5
n	Pr(3,1)	Pr(3,4)	Pr(4,2)	Pr(4,3)	Pr(4,4)	Pr(5,4)
0	0.500000	0.500000	0.333333	0.333333	0.333333	1.000000
600	0.498625	0.501375	0.201754	0.586722	0.211524	1.000000
1200	0.504372	0.495628	0.135323	0.736304	0.128373	1.000000
1800	0.495079	0.504921	0.092126	0.821082	0.086792	1.000000
2400	0.490428	0.509572	0.069171	0.865675	0.065154	1.000000
3000	0.487636	0.512364	0.055361	0.892494	0.052145	1.000000

Table 6.9 – Optimal Path SLPN : Sequence 7

Reward Parameter = 0.04									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.2	0.45	0.3	0.2	0.3	0.4	0.4	0.9

Convergence to Sequence 8:

M0-Pr(0,1); M1-Pr(1,3); M3-Pr(3,1); M4-Pr(4,3); M5-Pr(5,4); M3;

	M0	M1			M2	
n	Pr(0,1)	Pr(1,1)	Pr(1,2)	Pr(1,3)	Pr(2,2)	Pr(2,3)
0	1.000000	0.333333	0.333333	0.333333	0.500000	0.500000
600	1.000000	0.132012	0.212609	0.655378	0.508068	0.491932
1200	1.000000	0.079786	0.113330	0.806884	0.513090	0.486910
1800	1.000000	0.053482	0.075730	0.870788	0.515134	0.484866
2400	1.000000	0.040139	0.056831	0.903029	0.516157	0.483843
3000	1.000000	0.032125	0.045484	0.922391	0.516771	0.483229

	M3		M4			M5
n	Pr(3,1)	Pr(3,4)	Pr(4,2)	Pr(4,3)	Pr(4,4)	Pr(5,4)
0	0.500000	0.500000	0.333333	0.333333	0.333333	1.000000
600	0.741401	0.258599	0.205825	0.568628	0.225547	1.000000
1200	0.848475	0.151525	0.123761	0.700174	0.176065	1.000000
1800	0.897521	0.102479	0.083012	0.792916	0.124072	1.000000
2400	0.923069	0.076931	0.062303	0.844451	0.093246	1.000000
3000	0.938429	0.061571	0.049863	0.875505	0.074631	1.000000

Table 6.10 – Optimal Path SLPN : Sequence 8

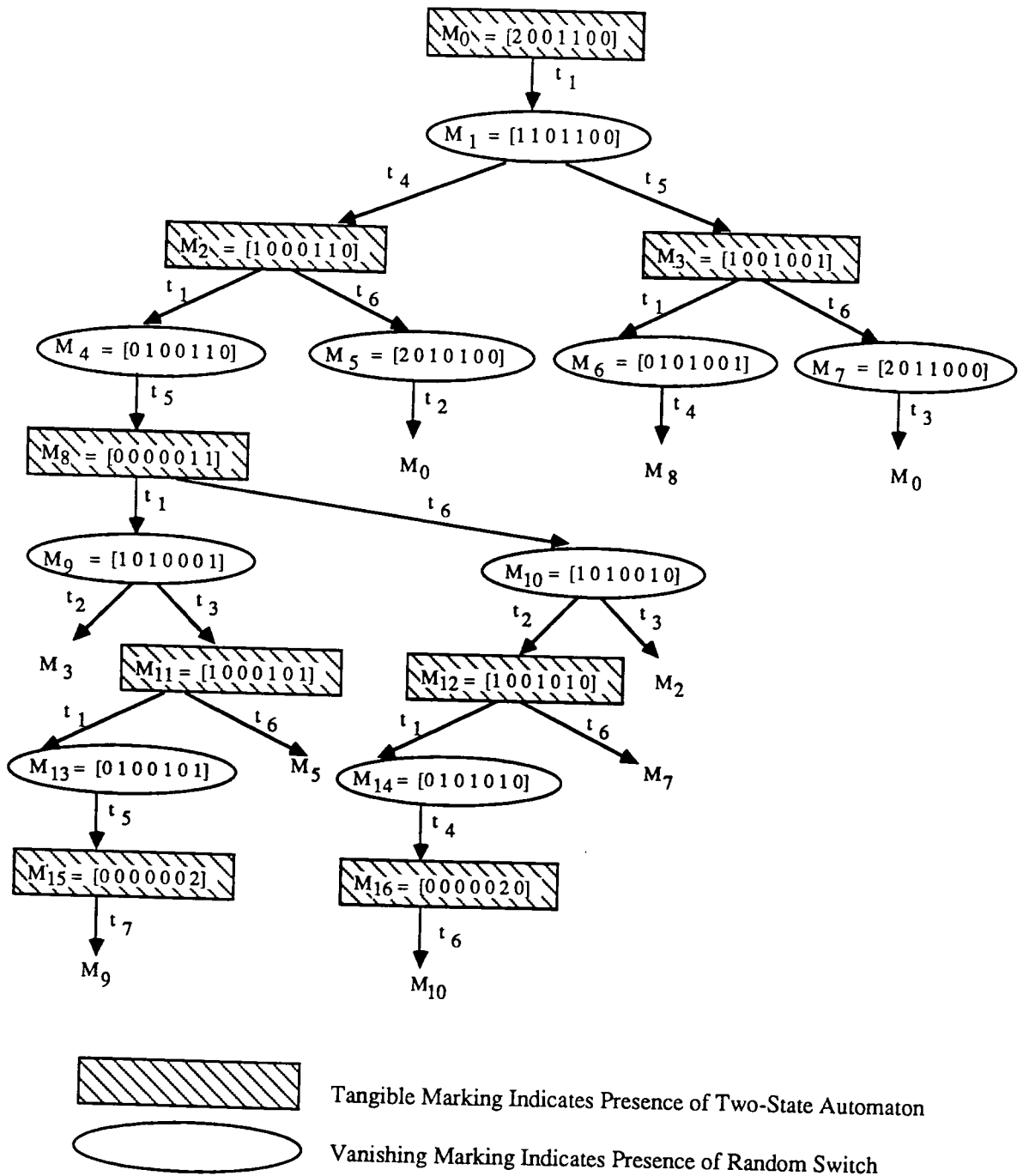


Figure 6.8 – GSPN Reachability Tree : Embedded Stochastic Automata

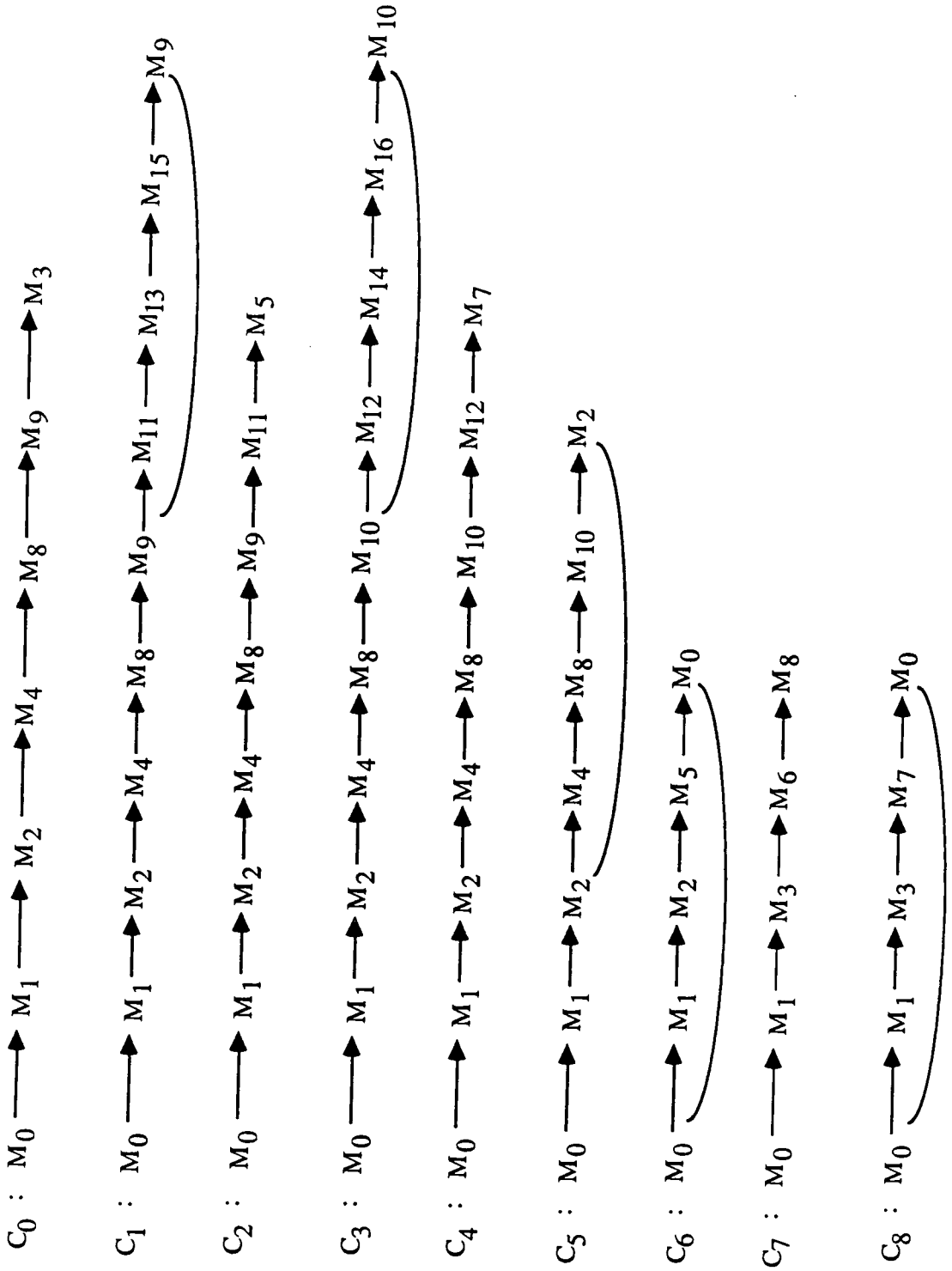


Figure 6.9 – GSPN : Sequence of Decisions/ States

Reward Parameter = 0.1									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.9	0.35	0.45	0.2	0.3	0.2	0.3	0.4	0.4

Convergence to Sequence 0:

M0-Pr(0,1); M2-Pr(2,1); M8-Pr(8,1);

	M0	M2		M3		M8	
n	Pr(0,1)	Pr(2,1)	Pr(2,6)	Pr(3,1)	Pr(3,6)	Pr(8,1)	Pr(8,6)
0	1.000000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000
600	1.000000	0.809068	0.190932	0.416901	0.583099	0.828256	0.171744
1200	1.000000	0.900969	0.099031	0.448418	0.551582	0.910950	0.089050
1800	1.000000	0.933867	0.066133	0.429936	0.570064	0.940533	0.059467
2400	1.000000	0.950372	0.049628	0.400512	0.599488	0.955374	0.044626
3000	1.000000	0.960281	0.039719	0.362137	0.637863	0.964284	0.035716

	M11		M12		M15	M16
n	Pr(11,1)	Pr(11,6)	Pr(12,1)	Pr(12,6)	Pr(15,7)	Pr(16,6)
0	0.500000	0.500000	0.500000	0.500000	1.000000	1.000000
600	0.651463	0.348537	0.500000	0.500000	1.000000	1.000000
1200	0.675787	0.324213	0.500000	0.500000	1.000000	1.000000
1800	0.741825	0.258175	0.500000	0.500000	1.000000	1.000000
2400	0.786258	0.213742	0.500000	0.500000	1.000000	1.000000
3000	0.750739	0.249261	0.500000	0.500000	1.000000	1.000000

Table 6.11 – Optimal Path GSLPN : Sequence 0

Reward Parameter = 0.1									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.9	0.45	0.2	0.3	0.2	0.3	0.4	0.4

Convergence to Sequence 1:

M0-Pr(0,1); M2-Pr(2,1); M8-Pr(8,1); M11-Pr(11,1);

	M0	M2		M3		M8	
n	Pr(0,1)	Pr(2,1)	Pr(2,6)	Pr(3,1)	Pr(3,6)	Pr(8,1)	Pr(8,6)
0	1.000000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000
600	1.000000	0.806212	0.193788	0.415502	0.584498	0.825711	0.174289
1200	1.000000	0.894719	0.105281	0.291938	0.708062	0.905381	0.094619
1800	1.000000	0.929572	0.070428	0.222273	0.777727	0.936706	0.063294
2400	1.000000	0.947142	0.052858	0.173345	0.826655	0.952496	0.047504
3000	1.000000	0.957696	0.042304	0.142028	0.857972	0.961981	0.038019

	M11		M12		M15	M16
n	Pr(11,1)	Pr(11,6)	Pr(12,1)	Pr(12,6)	Pr(15,7)	Pr(16,6)
0	0.500000	0.500000	0.500000	0.500000	1.000000	1.000000
600	0.723412	0.276588	0.500000	0.500000	1.000000	1.000000
1200	0.813575	0.186425	0.500000	0.500000	1.000000	1.000000
1800	0.862112	0.137888	0.500000	0.500000	1.000000	1.000000
2400	0.895455	0.104545	0.500000	0.500000	1.000000	1.000000
3000	0.916280	0.083720	0.500000	0.500000	1.000000	1.000000

Table 6.12 – Optimal Path GSLPN : Sequence 1

Reward Parameter = 0.1									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.45	0.9	0.2	0.3	0.2	0.3	0.4	0.4

Convergence to Sequence 2:

M0-Pr(0,1); M2-Pr(2,1); M8-Pr(8,1); M11-Pr(11,6);

	M0	M2		M3		M8	
n	Pr(0,1)	Pr(2,1)	Pr(2,6)	Pr(3,1)	Pr(3,6)	Pr(8,1)	Pr(8,6)
0	1.000000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000
600	1.000000	0.685376	0.314624	0.626392	0.373608	0.682899	0.317101
1200	1.000000	0.818368	0.181632	0.642078	0.357922	0.791152	0.208848
1800	1.000000	0.877747	0.122253	0.652813	0.347187	0.857948	0.142052
2400	1.000000	0.908209	0.091791	0.711421	0.288579	0.893275	0.106725
3000	1.000000	0.926535	0.073465	0.767056	0.232944	0.914579	0.085421

	M11		M12		M15	M16
n	Pr(11,1)	Pr(11,6)	Pr(12,1)	Pr(12,6)	Pr(15,7)	Pr(16,6)
0	0.500000	0.500000	0.500000	0.500000	1.000000	1.000000
600	0.361052	0.638948	0.453750	0.546250	1.000000	1.000000
1200	0.267933	0.732067	0.451878	0.548122	1.000000	1.000000
1800	0.186095	0.813905	0.451253	0.548747	1.000000	1.000000
2400	0.140398	0.859602	0.450940	0.549060	1.000000	1.000000
3000	0.112463	0.887537	0.450753	0.549247	1.000000	1.000000

Table 6.13 – Optimal Path GSLPN : Sequence 2

Reward Parameter = 0.1									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.2	0.45	0.9	0.3	0.2	0.3	0.4	0.4

Convergence to Sequence 3:

M0-Pr(0,1); M2-Pr(2,1); M8-Pr(8,6); M12-Pr(12,1); M16-Pr(16,6);

n	M0	M2		M3		M8	
	Pr(0,1)	Pr(2,1)	Pr(2,6)	Pr(3,1)	Pr(3,6)	Pr(8,1)	Pr(8,6)
0	1.000000	0.500000	0.333333	0.333333	0.500000	0.500000	0.500000
600	1.000000	0.532724	0.467276	0.593493	0.406507	0.372503	0.627497
1200	1.000000	0.729578	0.270422	0.616164	0.383836	0.237406	0.762594
1800	1.000000	0.818717	0.181283	0.622145	0.377855	0.160715	0.839285
2400	1.000000	0.863946	0.136054	0.625138	0.374862	0.120647	0.879353
3000	1.000000	0.891111	0.108889	0.626935	0.373065	0.096559	0.903441

n	M11		M12		M15	M16
	Pr(11,1)	Pr(11,6)	Pr(12,1)	Pr(12,6)	Pr(15,7)	Pr(16,6)
0	0.500000	0.500000	0.500000	0.500000	1.000000	1.000000
600	0.505083	0.494917	0.570318	0.429682	1.000000	1.000000
1200	0.527504	0.472496	0.657897	0.342103	1.000000	1.000000
1800	0.534990	0.465010	0.763197	0.236803	1.000000	1.000000
2400	0.538736	0.461264	0.822047	0.177953	1.000000	1.000000
3000	0.540985	0.459015	0.857573	0.142427	1.000000	1.000000

Table 6.14 – Optimal Path GSLPN : Sequence 3

Reward Parameter = 0.1									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.2	0.45	0.3	0.9	0.2	0.3	0.4	0.4

Convergence to Sequence 4:

M0-Pr(0,1); M2-Pr(2,1); M8-Pr(8,6); M12-Pr(12,6);

	M0	M2		M3		M8	
n	Pr(0,1)	Pr(2,1)	Pr(2,6)	Pr(3,1)	Pr(3,6)	Pr(8,1)	Pr(8,6)
0	1.000000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000
600	1.000000	0.669857	0.330143	0.384499	0.615501	0.291958	0.708042
1200	1.000000	0.802920	0.197080	0.237181	0.762819	0.202816	0.797184
1800	1.000000	0.866860	0.133140	0.162133	0.837867	0.149552	0.850448
2400	1.000000	0.900061	0.099939	0.122416	0.149552	0.112473	0.887527
3000	1.000000	0.920015	0.079985	0.098275	0.112473	0.090026	0.909974

	M11		M12		M15	M16
n	Pr(11,1)	Pr(11,6)	Pr(12,1)	Pr(12,6)	Pr(15,7)	Pr(16,6)
0	0.500000	0.500000	0.500000	0.500000	1.000000	1.000000
600	0.534500	0.465500	0.365006	0.634994	1.000000	1.000000
1200	0.542237	0.457763	0.224533	0.775467	1.000000	1.000000
1800	0.544820	0.455180	0.153759	0.846241	1.000000	1.000000
2400	0.546113	0.453887	0.115533	0.884467	1.000000	1.000000
3000	0.546889	0.453111	0.092473	0.907527	1.000000	1.000000

Table 6.15 – Optimal Path GSLPN : Sequence 4

Reward Parameter = 0.1									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.2	0.45	0.3	0.2	0.9	0.3	0.4	0.4

Convergence to Sequence 5:

M0-Pr(0,1); M2-Pr(2,1); M8-Pr(8,6);

	M0	M2		M3		M8	
n	Pr(0,1)	Pr(2,1)	Pr(2,6)	Pr(3,1)	Pr(3,6)	Pr(8,1)	Pr(8,6)
0	1.000000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000
600	1.000000	0.722986	0.277014	0.392727	0.607273	0.207815	0.792185
1200	1.000000	0.852646	0.147354	0.239746	0.760254	0.109153	0.890847
1800	1.000000	0.901434	0.098566	0.183716	0.816284	0.072975	0.927025
2400	1.000000	0.926033	0.073967	0.152900	0.847100	0.054763	0.945237
3000	1.000000	0.940802	0.059198	0.127705	0.872295	0.043828	0.956172

	M11		M12		M15	M16
n	Pr(11,1)	Pr(11,6)	Pr(12,1)	Pr(12,6)	Pr(15,7)	Pr(16,6)
0	0.500000	0.500000	0.500000	0.500000	1.000000	1.000000
600	0.500000	0.500000	0.412953	0.587047	1.000000	1.000000
1200	0.500000	0.500000	0.339427	0.660573	1.000000	1.000000
1800	0.500000	0.500000	0.333693	0.666307	1.000000	1.000000
2400	0.500000	0.500000	0.365173	0.634827	1.000000	1.000000
3000	0.500000	0.500000	0.383313	0.616687	1.000000	1.000000

Table 6.16 - Optimal Path GSLPN : Sequence 5

Reward Parameter = 0.1									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.2	0.45	0.3	0.2	0.3	0.9	0.4	0.4

Convergence to Sequence 6:

M0-Pr(0,1); M2-Pr(2,6);

n	M0	M2		M3		M8	
	Pr(0,1)	Pr(2,1)	Pr(2,6)	Pr(3,1)	Pr(3,6)	Pr(8,1)	Pr(8,6)
0	1.000000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000
600	1.000000	0.160091	0.839909	0.770198	0.229802	0.626082	0.373917
1200	1.000000	0.080741	0.919259	0.875818	0.124182	0.630783	0.369217
1800	1.000000	0.053873	0.946127	0.916763	0.083237	0.632353	0.367647
2400	1.000000	0.040427	0.959573	0.937517	0.062483	0.633138	0.366862
3000	1.000000	0.032355	0.967645	0.949991	0.050009	0.633610	0.366390

n	M11		M12		M15	M16
	Pr(11,1)	Pr(11,6)	Pr(12,1)	Pr(12,6)	Pr(15,7)	Pr(16,6)
0	0.500000	0.500000	0.500000	0.500000	1.000000	1.000000
600	0.543250	0.456750	0.500000	0.500000	1.000000	1.000000
1200	0.546619	0.453381	0.500000	0.500000	1.000000	1.000000
1800	0.547744	0.452256	0.500000	0.500000	1.000000	1.000000
2400	0.548307	0.451693	0.500000	0.500000	1.000000	1.000000
3000	0.548645	0.451355	0.500000	0.500000	1.000000	1.000000

Table 6.17 – Optimal Path GSLPN : Sequence 6

Reward Parameter = 0.1									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.2	0.45	0.3	0.2	0.3	0.4	0.9	0.4

Convergence to Sequence 7:

M0-Pr(0,1); M3-Pr(3,1);

	M0	M2		M3		M8	
n	Pr(0,1)	Pr(2,1)	Pr(2,6)	Pr(3,1)	Pr(3,6)	Pr(8,1)	Pr(8,6)
0	1.000000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000
600	1.000000	0.454252	0.545748	0.852835	0.147165	0.600395	0.399605
1200	1.000000	0.628914	0.371086	0.922450	0.077550	0.756201	0.243799
1800	1.000000	0.744904	0.255096	0.948114	0.051886	0.833994	0.166006
2400	1.000000	0.807361	0.192639	0.961063	0.038937	0.874887	0.125113
3000	1.000000	0.845674	0.154326	0.968837	0.031163	0.899801	0.100199

	M11		M12		M15	M16
n	Pr(11,1)	Pr(11,6)	Pr(12,1)	Pr(12,6)	Pr(15,7)	Pr(16,6)
0	0.500000	0.500000	0.500000	0.500000	1.000000	1.000000
600	0.560640	0.439360	0.470833	0.529167	1.000000	1.000000
1200	0.645307	0.354693	0.460434	0.539566	1.000000	1.000000
1800	0.639664	0.360336	0.456962	0.543038	1.000000	1.000000
2400	0.642014	0.357986	0.455224	0.544776	1.000000	1.000000
3000	0.652169	0.347831	0.454181	0.545819	1.000000	1.000000

Table 6.18 – Optimal Path GSLPN : Sequence 7

Reward Parameter = 0.1									
Reward Probability									
i	0	1	2	3	4	5	6	7	8
Ci	0.35	0.2	0.45	0.3	0.2	0.3	0.4	0.4	0.9

Convergence to Sequence 8:

M0-Pr(0,1); M3-Pr(3,6);

	M0	M2		M3		M8	
n	Pr(0,1)	Pr(2,1)	Pr(2,6)	Pr(3,1)	Pr(3,6)	Pr(8,1)	Pr(8,6)
0	1.000000	0.500000	0.500000	0.500000	0.500000	0.500000	0.500000
600	1.000000	0.454252	0.545748	0.1980865	0.809135	0.600395	0.399605
1200	1.000000	0.628914	0.371086	0.098464	0.901536	0.756201	0.243799
1800	1.000000	0.744904	0.255096	0.065804	0.934196	0.833994	0.166006
2400	1.000000	0.807361	0.192639	0.049381	0.950619	0.874887	0.125113
3000	1.000000	0.845674	0.154326	0.039521	0.960479	0.899801	0.100199

	M11		M12		M15	M16
n	Pr(11,1)	Pr(11,6)	Pr(12,1)	Pr(12,6)	Pr(15,7)	Pr(16,6)
0	0.500000	0.500000	0.500000	0.500000	1.000000	1.000000
600	0.560640	0.439360	0.470833	0.529167	1.000000	1.000000
1200	0.645307	0.354693	0.460434	0.539566	1.000000	1.000000
1800	0.639664	0.360336	0.456952	0.543038	1.000000	1.000000
2400	0.642014	0.357986	0.455224	0.544776	1.000000	1.000000
3000	0.652169	0.347831	0.454181	0.545819	1.000000	1.000000

Table 6.19 – Optimal Path GSLPN : Sequence 8

Chapter Seven

Application to Distributed Decision Systems

7.1 Introduction

PNs have been extensively used in the representation and analysis of computing systems and processes. As stated previously, the PN formalism is suitable for representing dynamic processes, particularly when some of the events may occur concurrently. However, recently the use of PNs in the modelling of decision making processes has been proposed, [60], [61]. In the case of modelling decision making organisations, the PN representation considers tokens as items of information or signals which wait to be processed in the places. These places are conditions which must be met before the information held in them can be processed. The transitions are events which execute processes; whereby a process is executed by the firing of a corresponding transition and the flow of tokens represent the flow of information in the process. Thus, PNs represent powerful modelling tools for decision making organisations, since they describe in a precise manner the interactions among decision makers. Several authors have considered the application of PNs in this field of study, [62], [63], [64], [65], [66].

In the discussion that follows a model of the interacting decision maker is presented. The basic model of the interacting decision maker consists of four stages. An application of the hybrid SLPN modelling tool to a specific

two node organisation is considered, [59], [67].

7.2 Model of the Decision Making Process

A basic model of an interacting decision maker appropriate for the study of command and control organisations was introduced by Boettcher and Levis, [68]. Their four-stage model of the human Decision Maker (DM) permits detailed and explicit interactions among organisation members. The decision maker receives an input x from his environment and undergoes a four-stage process, shown in Figure 7.1. The Situation Assessment (SA) and Response Selection (RS) stages are used to model the actual decision making process; while Information Fusion (IF) and Command Interpretation (CI) allow for interaction of the DM with other members of the organisation.

Based on the above discussion, the input x received by the decision maker is processed in the SA stage, this stage operates upon x to produce an assessed situation z . The assessed situation z may be shared with the other members of the organisation; concurrently, the DM may receive the supplementary situation assessment \acute{z} from other parts of the organisation. This information may in turn be combined in the IF stage to yield \bar{z} .

The fused assessed situation, \bar{z} , is processed by one of the algorithms in the RS stage; since in the RS stage possible alternatives of action are evaluated and the output response may be communicated to other team members. The CI stage of the model allows \bar{z} and the input \acute{v} to influence the choice of this algorithm. A command input \acute{v} from the rest of the organisation may be considered to be a command capable, for example, of

restricting options. The RS stage contains algorithms that produce output y in response to the situation assessment \bar{z} and the command inputs.

The internal structure of the four processing stages, is depicted in Figure 7.2 which include the SA, IF, CI and RS stages. Note that the SA stage consists of a set of U algorithms that are capable of producing some situation assessment z . The RS stage also contains a set of V algorithms which are required to produce the final decision response.

7.2.1 Model of an Organisation with a Decision Aid

This sub-section describes the integration of a decision aid within the decision module. The DM module may often be faced with metadecisions, ie. decisions about how to choose whether to use the information provided by an aid and how to use that information, [63]. For example, in conditions of uncertainty, the time constraint is an important factor. Thus, in an emergency situation a decision maker must reach a decision in the order of seconds, at most tens of seconds; because of this an interactive decision aid would not be feasible. On the other hand, the DM may access a decision support system or another form of aid to reach an accurate response. Hence, the aided DM must decide between the following three options, when confronted with a decision aid:

- (1) The user DM ignores (blocks) the information provided by the aid and assesses the situation as trained;
- (2) The user DM assesses the situation as trained and compares the result

with aid information choosing the worst case;

- (3) The user DM relies solely on the aid information.

Consider the structure as illustrated in Figure 7.3, each DM must decide how to choose among the alternatives for addressing the problem. This structure provides a convenient framework for the application of the SLPN model which will be discussed at a later stage.

7.3 Application : Small-scale C³-I System

This section discusses the application of a new class of Petri nets, namely, the Stochastic Learning Petri Nets (SLPN) as a powerful modelling tool for decision making organisations in C³-I systems. Figure 7.4 shows in block diagram form the first model proposed for study. The example consists of a two node organisation; decision module DM1 and a decision module DM2. Each decision module receives signals from the environment and can respond to the environment. The DM module consists of three possible strategies, although the SA stage selects only a single strategy to process the information. As mentioned above the DM must decide between the following three options.

Strategy SA_i : process information without using Decision Support System (DSS);

Strategy IT_i : select a response via an intelligent terminal;

Strategy MF_i : utilise the DSS.

A PN representation for this two node organisation is depicted in Figure

7.5. In the simulation studies, the complexity of the model has been reduced by embedding only the concept of stochastic learning automata in the SA and RS stages for each decision module, excluding the concept of Petri nets, as depicted in Figure 7.6. Thus, the disposition of tokens in the node organisation are not considered. The operation of a single decision module DM1 interacting with a stationary random environment is considered. For each decision module, the corresponding SLPN structure and reachability tree is illustrated in Figure 7.7a and 7.7b, respectively. In this structure, the SLPN concept has been embedded in the SA and RS stages for each decision module.

7.3.1 Performance of Single Decision Module

As previously mentioned, to reduce the complexity of the model the concept of stochastic learning automata has been embedded in the SA and RS stages for each decision module. The decision node contains four learning automata interconnected in the form of a tree structure. The automata are arranged in two levels as shown in Figure 7.6. The hierarchy consists of a single automaton at the first level, and three automata in the second level. The first level automata, situation assessment SA1 consists of three options which are selected with equal initial probability. At this stage, the selected option corresponds to processing information via three possible strategies, as mentioned in the previous sub-section (SA1, IT1, MF1). In the second level which corresponds to the response selection stage, there are two possible alternatives to be chosen with equal initial probability. Thus, from the top to

the lower level automata there exists six possible paths (p_1, p_2, \dots, p_6) which can be selected by DM1. This structure enables the single decision module to select the optimal strategy between six possible strategies.

Considering the structure, Figure 7.6, SA1 acts first choosing either RS11, RS12 or RS13. The action selected by the automaton in the lowest level (response selection RS stage), generates a response from the environment. The action probabilities for the selected path are updated on the basis of this response. Thus the single decision module selects the optimal strategy which corresponds to the optimal path.

7.3.2 Performance of Two Node Organisation

Similar to the previous case, this adopts an identical approach by embedding the stochastic learning automata in the SA and RS stages for each decision module, as depicted in Figure 7.8. Therefore, each decision module contains four learning automata interconnected in the form of a hierarchical system. For decision module DM1, the three options (SA_1, IT_1, MF_1) are selected with equal initial probability; similarly for decision module DM2 (SA_2, IT_2, MF_2) . Also each RS stage has two alternate possibilities which are selected with equal initial probability; thus producing six possible paths for each DM. The strategies associated with decision module DM1 and DM2 are (p_1, p_2, \dots, p_6) and (q_1, q_2, \dots, q_6) respectively. There are 36 (6×6) possible combinations of decision strategies fed to the environment. Consider the structure in Figure 7.8, for each pair of strategies selected by the decision modules the environment responds stochastically to punish/

reward the selection of a particular pair. One pair of decisions is optimum (ie. gives minimum punishment or maximum reward).

7.4 Experimental Results

This sub-section presents results based on a series of experiments which examine the performance of a single decision module and a two node organisation interacting with an uncertain environment. As stated previously, decision modules are in the form of a two level hierarchical system. To simulate these modules, the reward probabilities in the environment were selected from the range [0.2-0.5], except the unique maximum reward probability which was set to 0.9. An L_{RI} scheme was adopted to update action probabilities for the selected path; in the case of the two node organisation, the action probabilities for the optimal strategy pair were updated. Simulations were performed, the results are presented in both table and graph format. For each experiment the reward parameter and reward probabilities are given; the expected values are denoted by a bar eg. $\bar{p}_1(n) = E[p_1(n)]$.

Experiment 1

The first experiment illustrates the operation of a single decision module interacting with a stationary random environment, as shown in Figure 7.6. For this experiment, the objective of a single decision module is such that the optimal strategy is selected from six possible paths. Figure 7.9 displays the route corresponding to the optimal path.

The results are produced in Table 7.1 with their respective reward

parameter and reward probabilities. It is evident that the optimal path is p_1 since the unique maximum reward probability is associated with this action path. Furthermore, the tabulated results show convergence close to unity for this particular path; the corresponding learning curve for the optimal path is shown in Figure 7.9a.

The experiments [2-4] illustrate the learning performance of a two node organisation, as depicted in Figure 7.8. For these experiments, the main objective is such that both decision modules select the optimal pair of decision strategies from 36 (6x6) possible combinations of decision pairs input to the environment. Similar to the previous case, reward probabilities in the range [0.2-0.5] are associated with paths (p_1, p_2, \dots, p_6) and (q_1, q_2, \dots, q_6) for decision modules DM1 and DM2, respectively. However, in this case the unique maximum reward probability which is set to 0.9 exists for each decision module DM1 and DM2. Thus, a single path from the set (p_1, p_2, \dots, p_6) for DM1 is associated with a unique maximum reward probability; and also a single path from the set (q_1, q_2, \dots, q_6) for DM2. The conditions for each experiment are varied by considering the selection of optimal strategy pairs; sudden switch of environmental conditions and by permitting communication between both decision modules at upper and lower levels.

Experiment 2

The simulation results in Table 7.2 examine the learning behaviour of a two node organisation. Note that in the case of a two node organisation, the action paths associated with the unique maximum reward probability converge

close to unity for each decision module DM1 and DM2. The strategy pair selected is clearly indicated in Figure 7.10.

Table 7.2 indicates the value of the reward parameter; the unique maximum reward probability to be employed by the environment and the expected values denoting the convergence to optimal strategy pair. In this case the unique maximum reward probability is associated with path $p_4 \cdot q_2$ for decision module DM1 and DM2, respectively. The results confirm that the coordinated decision strategies selected by each decision module converges close to unity, this is illustrated in Figure 7.10a. Hence, the optimal pair of decisions selected by DM1 and DM2 is $p_4 \cdot q_2$.

Experiment 3

The previous experiment 2 was repeated, with the additional concept of a sudden switch to a different environment. In this experiment, a change in the environment was considered by re-setting the unique maximum reward probability to select an alternate pair of decision strategies. For example, by a repeat of experiment 2, it can be seen that both decision modules converge close to unity by selecting the optimal pair of decision strategies. The sudden switch in the environment is achieved by changing over the unique maximum reward probability, such that an alternate pair of decision strategies may be selected. For this particular experiment, Figure 7.11 indicates the route which may be selected by varying the conditions of the environment: before and after the switch.

This behaviour is best illustrated by analysing the results in Table

[7.3a - 7.3b]; all relevant parameter values are indicated. The simulation results show how fast the structure learns how to converge to the new optimal strategy pair. It is evident from Table 7.3a that both decision modules DM1 and DM2 select the optimal strategy pair $p_1.q_1$; since convergence for this pair is close to unity. The learning curve showing convergence of strategy pair $p_1.q_1$ is represented in Figure 7.12a. However, after introducing a sudden switch of the environment by re-locating the unique maximum reward probability indicated in Table 7.3b, similarly, the structure learns to select the optimal strategy pair shown in Figure 7.12c. In this case the coordinated decision strategy selected is pair $p_3.q_1$; the unique maximum reward probability is associated with this pair. Thus, Figure 7.12b shows a decrease in convergence for path p_1 selected by DM1 and, a rapid increase in convergence close to unity for path $p_3.q_1$ is depicted in Figure 7.12c.

Experiment 4

This final experiment gives an excellent illustration of speeding up the learning process by permitting communication between decision modules DM1 and DM2 (as indicated by dotted lines Figure 7.13). Note that in each of the following experiments an arbitrary value for the stepsize is considered.

(a) First set of results in Table 7.4a permits communication between automata at the top level of the hierarchy for each decision module. To simulate this structure, both automata at the top level (SA1 and SA2) exchange messages so that if each selects action one, then the reward parameter is incremented by stepsize 4. From Table 7.4a, it can be seen that the rate

of convergence for strategy pair $p_1.q_1$ rapidly increases close to unity; since the unique maximum reward probability is associated with this strategy pair.

(b) Second set of results in Table 7.4b enables communication between automata at the top and lower levels of the hierarchy for each decision module. In this case, in addition to automata at the top level (SA1 and SA2) exchanging messages; the lower level automata (RS11 and RS21) also communicate. The same rule is applied, that is, if both automata at the top and lower level select action one, the reward parameter is increased by stepsize 4. Similar to the previous case, the results in Table 7.4b show rapid convergence close to unity for both levels of automata. In comparison to the previous experiment, there is only a fractional increase in rate of convergence by permitting communication between automata at the upper and lower levels.

(c) The third set of results in Table 7.4c illustrates communication between automata at both top and lower levels for each decision module. The same rule is applied, which involves an increment of the reward parameter by stepsize 4, if both automaton select action one. However, in this case the location of the unique maximum reward probability has been changed while all other reward probabilities remain unchanged. It can be seen from Table 7.4c that the unique maximum reward probability is associated with strategy pair $p_3.q_3$ as opposed to $p_1.q_1$ in the previous experiment. The results do not display convergence behaviour for strategy pair $p_3.q_3$ and locking on to the strategy pair $p_1.q_1$ due to the increased stepsize via communication between automata at both levels.

Figures [7.14a - 7.14c] present learning curves, which illustrate conver-

gence behaviour for strategy pair $p_1.q_1$. In each case communication between automata is considered by employing an incremental value of the reward parameter by stepsize 4 (the approach described above); also the case without communication by using reward parameter 0.04. It may be observed that in the former case, the learning curve for strategy pair $p_1.q_1$ rapidly converges close to unity, and both learning curves coincide with each other.

7.10 Conclusion and Summary

This chapter has shown the potential application of a new class of hybrid Petri Net (Stochastic Learning Petri Net) to the modelling of a realistic small-scale distributed decision problem. Although a two node decision model has been described the basic concepts may be extended to more complex scenarios which involve an arbitrary number of nodes in pre-programmed topologies.

In this chapter the potential modelling capability of stochastic learning automata embedded within Petri Nets has been illustrated. The simulation studies have shown the capability of optimum distributed strategies in stochastic environments both for steady-state and switched environments. Initial work has also shown the ability to model communication between adjacent layers for decision models connected in hierarchical layers and the use of confidence communication signals to provide adaptive step sizes to improve convergence rates.

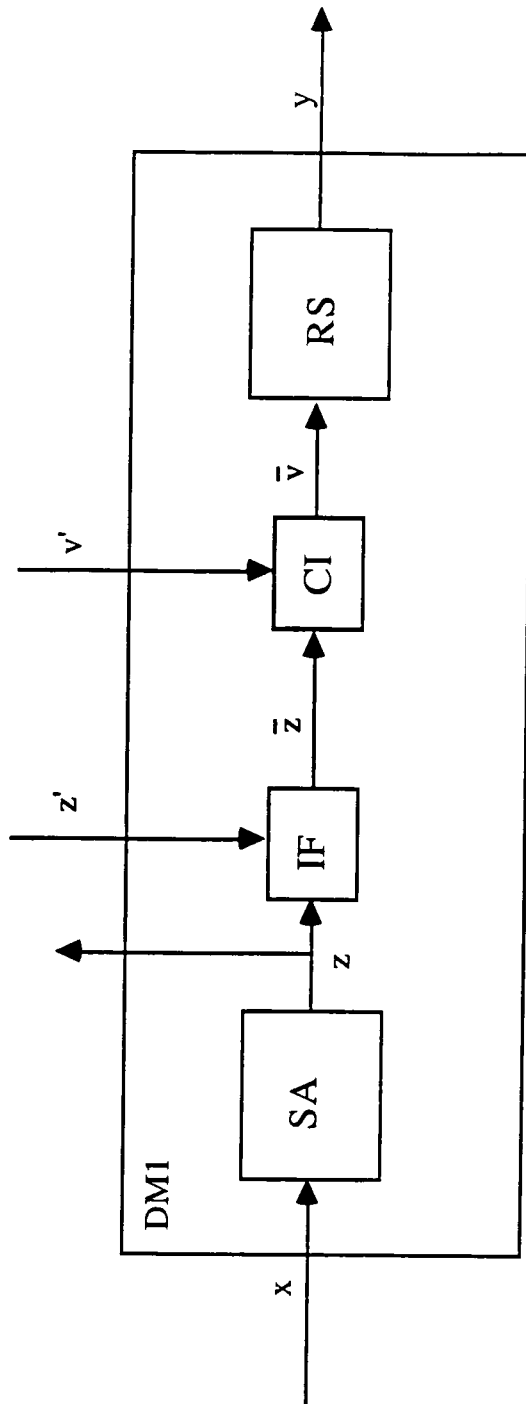


Figure 7.1 – Four-stage Model of Interacting Decision Maker

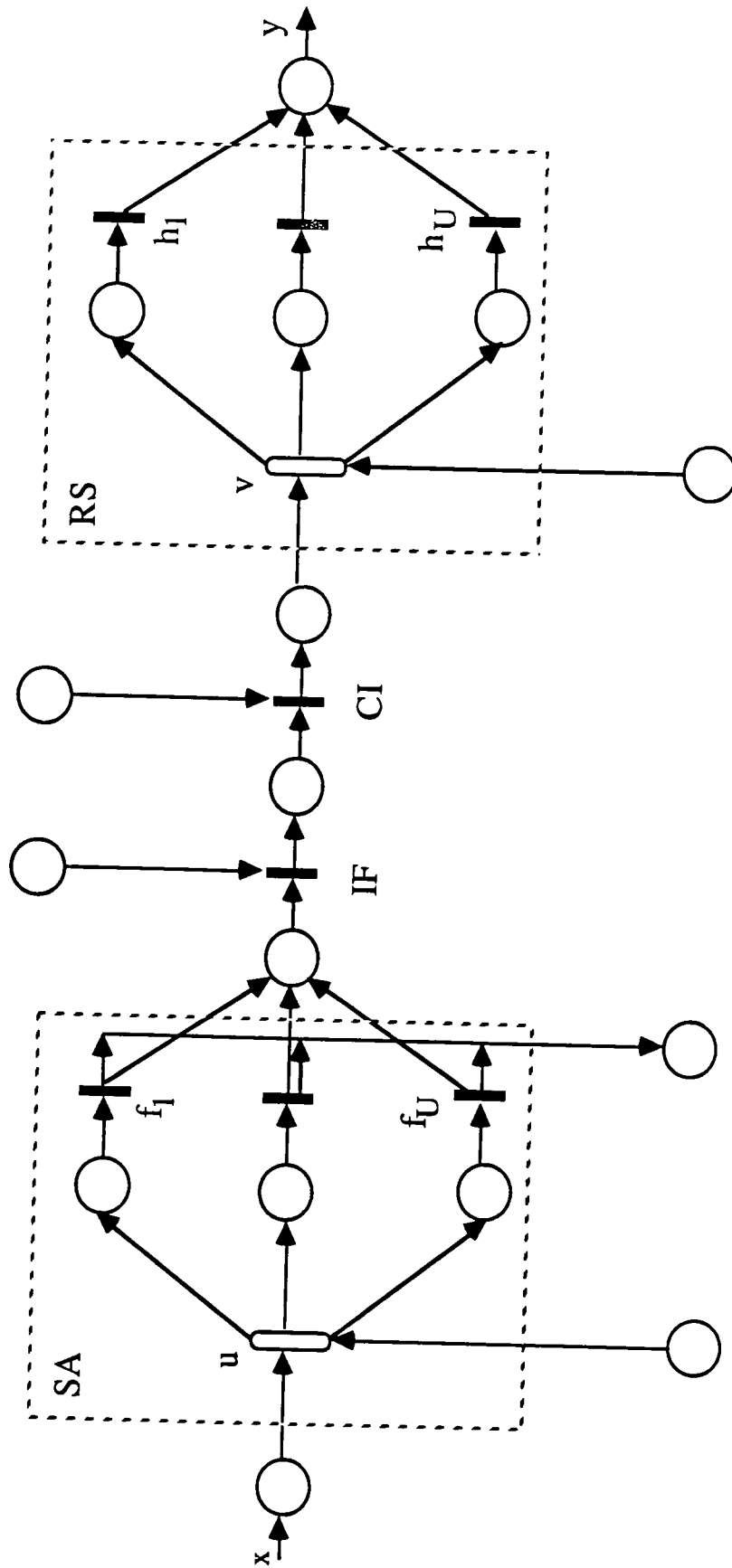


Figure 7.2 – Petri Net Representation of Interacting Decision Maker

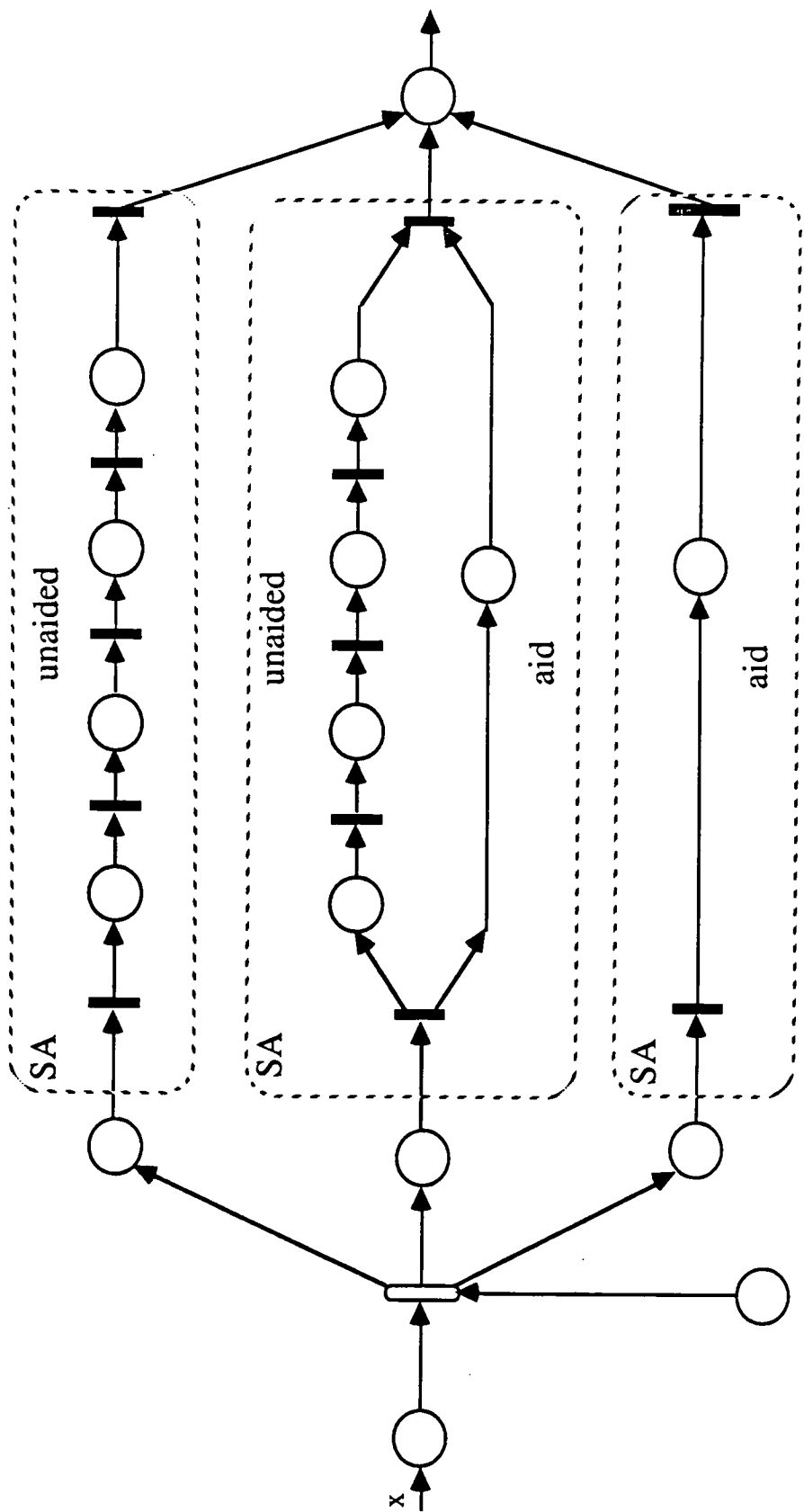


Figure 7.3 - Situation Assessment Module

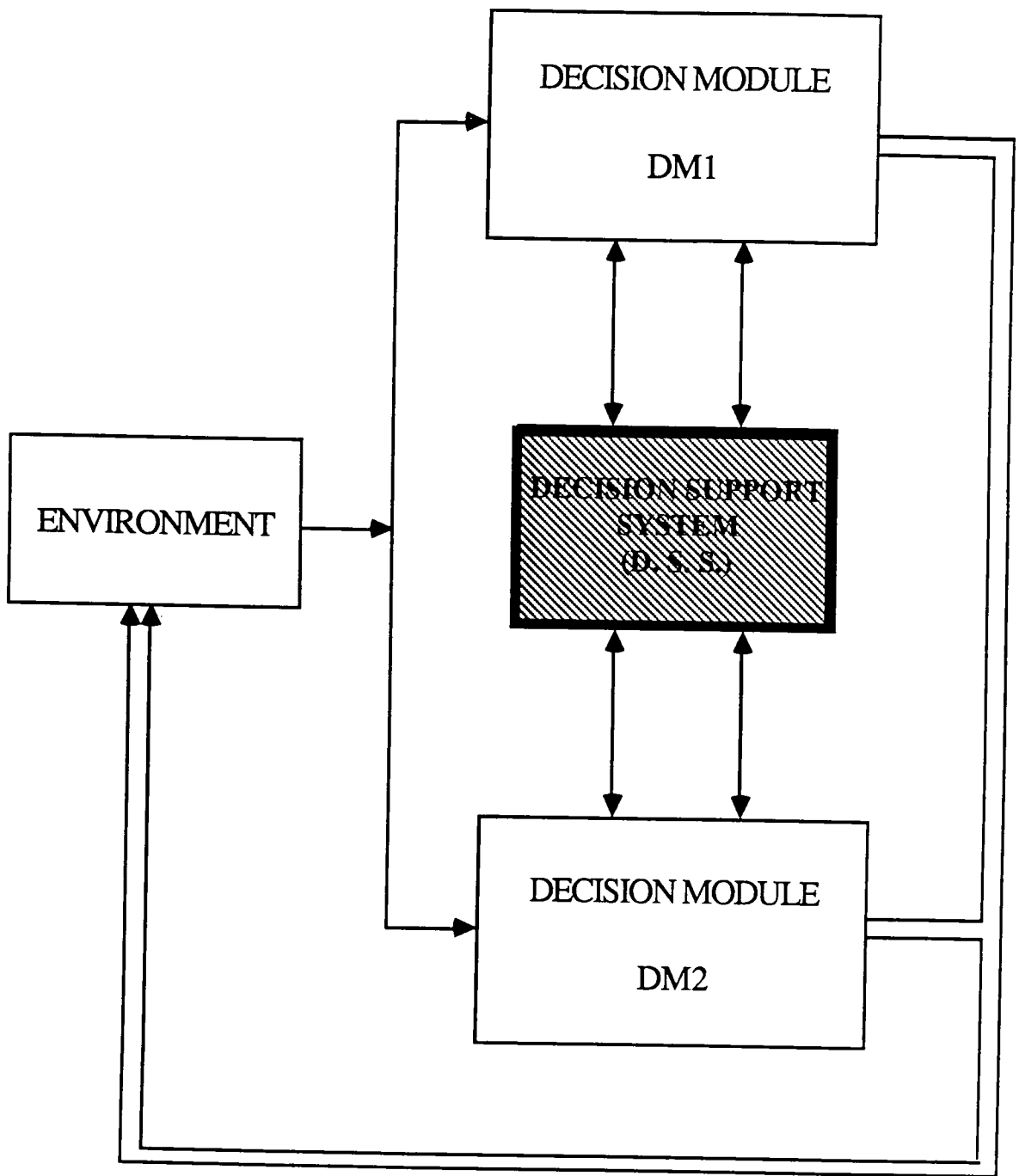


Figure 7.4 - Block Diagram: Two Node Organisation Supported by D.S.S

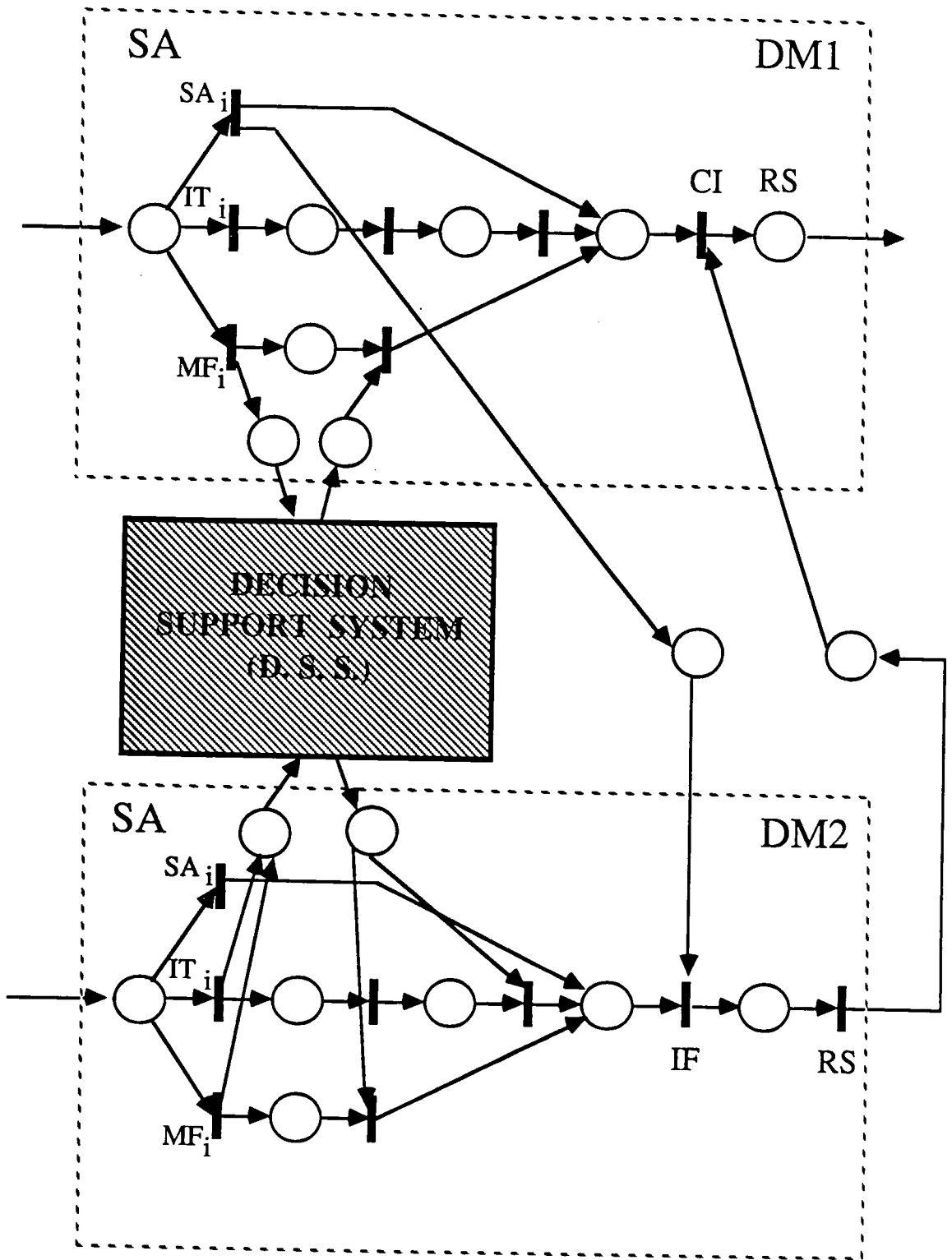


Figure 7.5 - Petri Net: Two Node Organisation Supported by D.S.S

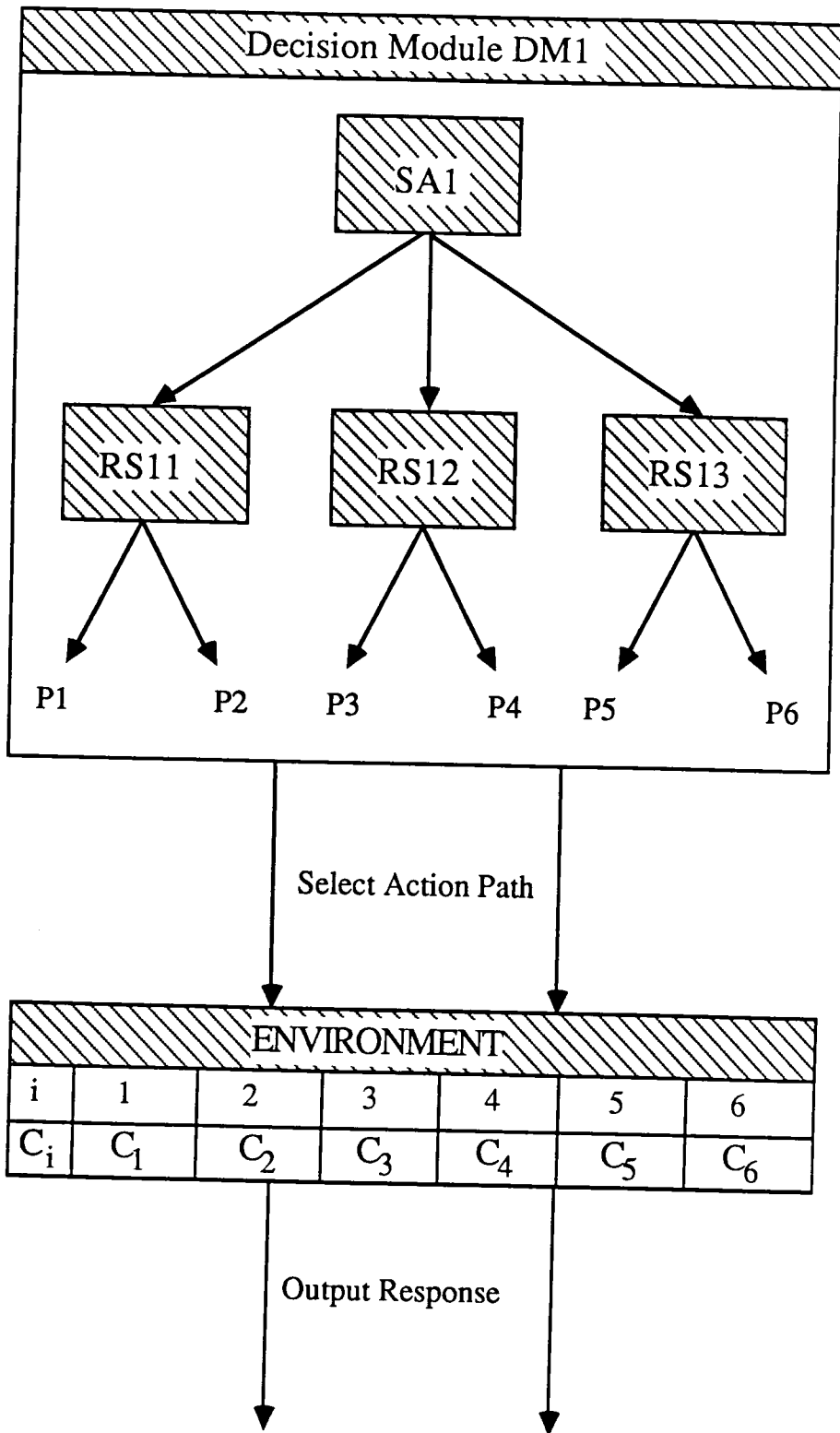


Figure 7.6 – Topology for Simulation: Single Decision Module

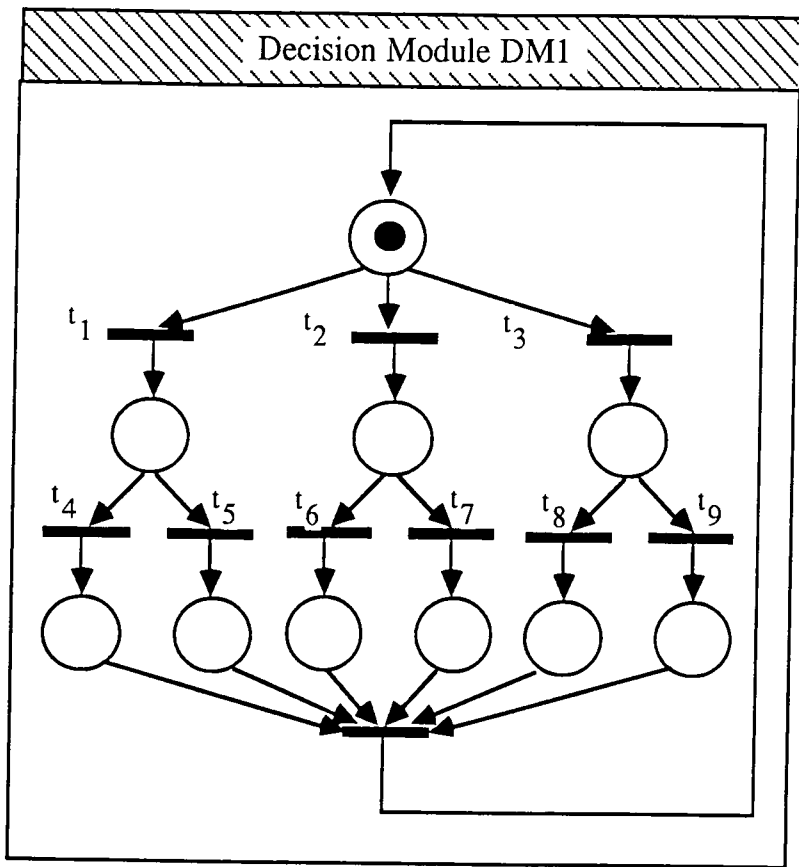
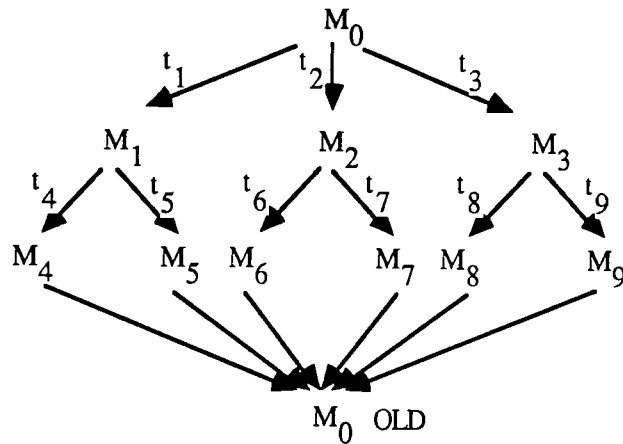


Figure 7.7a – Petri Net Representation Decision Module DM1



Corresponding States of Decision Module DM1 :

$$M_0 = [1000000000]$$

$$M_5 = [0000010000]$$

$$M_1 = [0100000000]$$

$$M_6 = [0000001000]$$

$$M_2 = [0010000000]$$

$$M_7 = [0000000100]$$

$$M_3 = [0001000000]$$

$$M_8 = [0000000010]$$

$$M_4 = [0000100000]$$

$$M_9 = [0000000001]$$

Figure 7.7b – Reachability Tree for Decision Module DM1

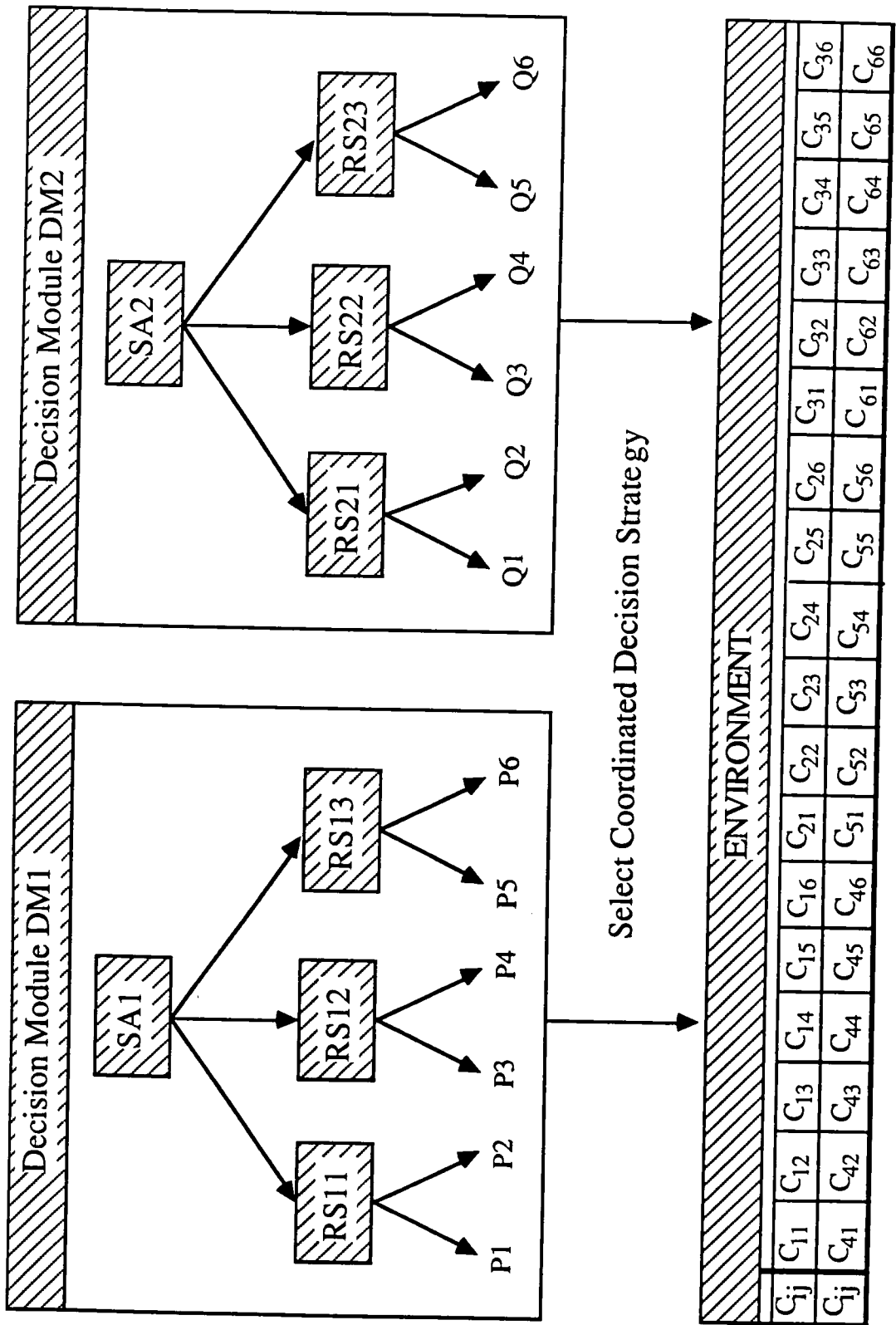


Figure 7.8 – Topology for Simulation: Two Node Organisation

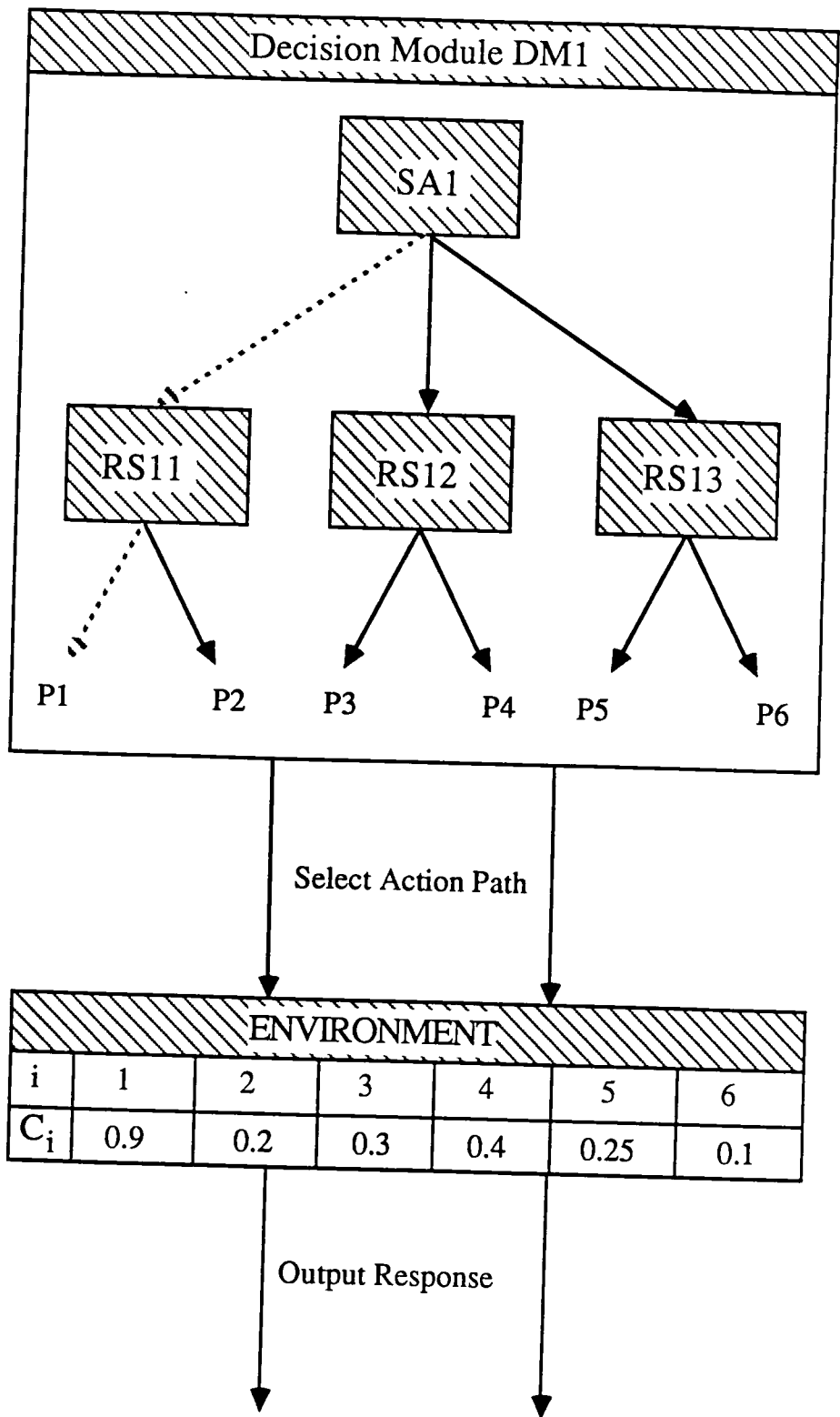


Figure 7.9 – Single Decision Module (Table 7.1)

Reward Parameter = 0.04						
Reward Probability						
i	0	1	2	3	4	5
Ci	0.9	0.2	0.3	0.4	0.25	0.1

Path Probability for Decision Module DM1: Optimal Path P1

n	p1	p2	p3	p4	p5	p6
0	0.166666	0.166666	0.166666	0.166666	0.166666	0.166666
600	0.609142	0.135953	0.080745	0.076972	0.048215	0.048873
1200	0.792024	0.079875	0.040079	0.039205	0.024323	0.024394
1800	0.858710	0.055856	0.026605	0.026250	0.016244	0.016234
2400	0.893030	0.042869	0.019911	0.019730	0.012194	0.012165
3000	0.913935	0.034764	0.015909	0.015805	0.009761	0.009727

Table 7.1 - Simulation of Single Decision Module (Figure 7.9)

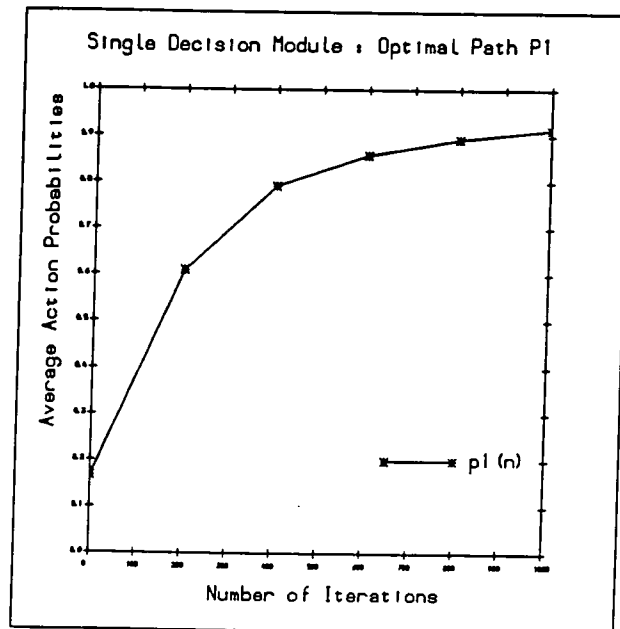


Figure 7.9a - Average Action Path Probability vs Iterations (Table 7.1)

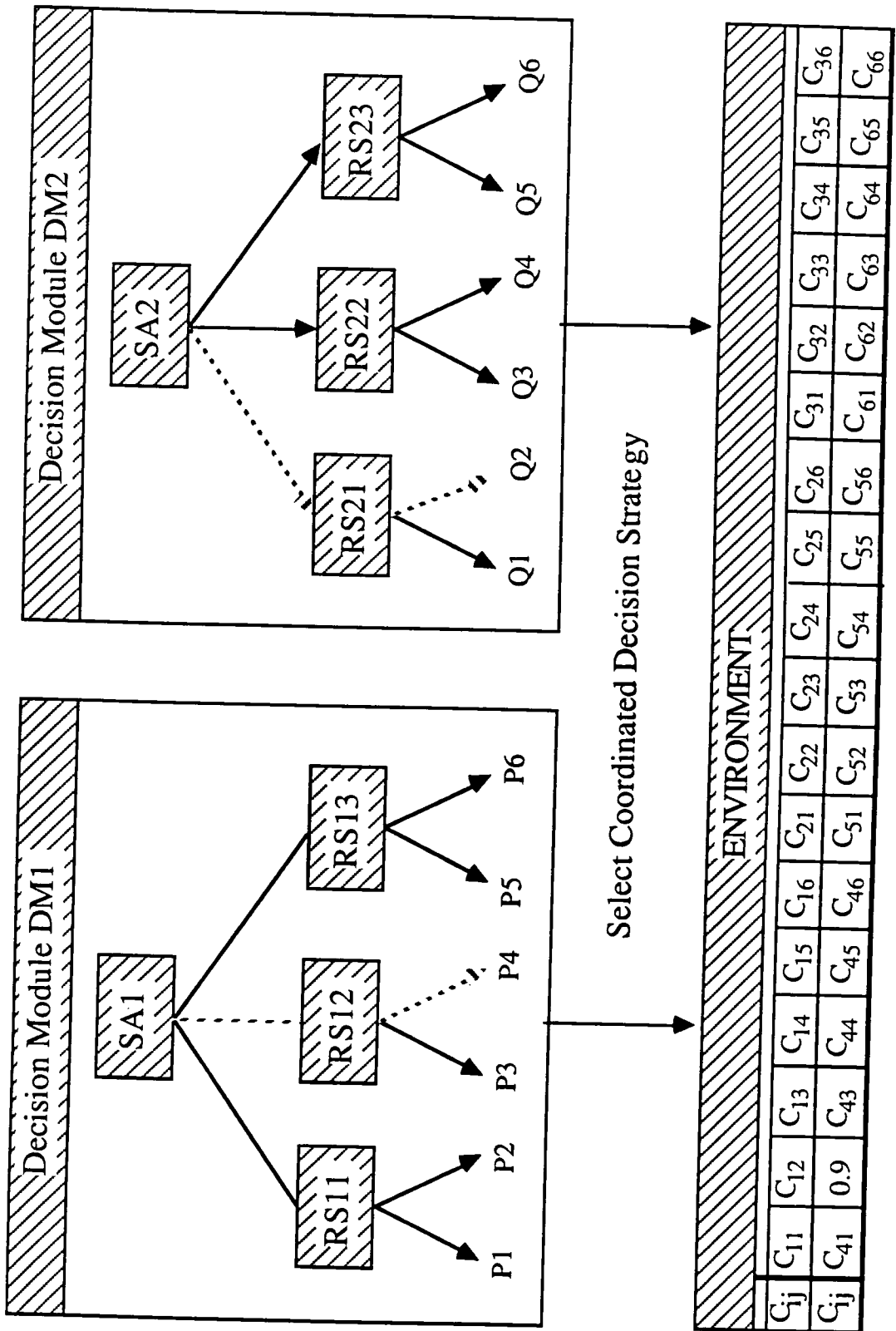


Figure 7.10 – Two Node Organisation (Table 7.2)

Reward Parameter = 0.04
Reward Probability
$C_{42} = 0.9$

Path Probability for Decision Module DM1: Optimal Path P4

n	p1	p2	p3	p4	p5	p6
0	0.166666	0.166666	0.166666	0.166666	0.166666	0.166666
600	0.122752	0.124708	0.173771	0.345529	0.127226	0.106013
1200	0.063909	0.065275	0.128113	0.624448	0.064294	0.053961
1800	0.042560	0.043563	0.094769	0.740271	0.042815	0.036021
2400	0.031903	0.032689	0.074587	0.801693	0.032094	0.027033
3000	0.025514	0.026160	0.061355	0.839669	0.025667	0.021635

Path Probability for Decision Module DM2: Optimal Path Q2

n	q1	q2	q3	q4	q5	q6
0	0.166666	0.166666	0.166666	0.166666	0.166666	0.166666
600	0.166504	0.457269	0.138586	0.130716	0.054399	0.052525
1200	0.108945	0.694528	0.070578	0.070720	0.028651	0.026577
1800	0.078552	0.790430	0.046577	0.047621	0.019225	0.017594
2400	0.062235	0.840602	0.034755	0.035894	0.014465	0.013149
3000	0.049974	0.871415	0.027718	0.028801	0.011595	0.010497

Optimal Strategy Pair P4.Q2

Table 7.2 - Simulation of Two Node Organisation (Figure 7.10)

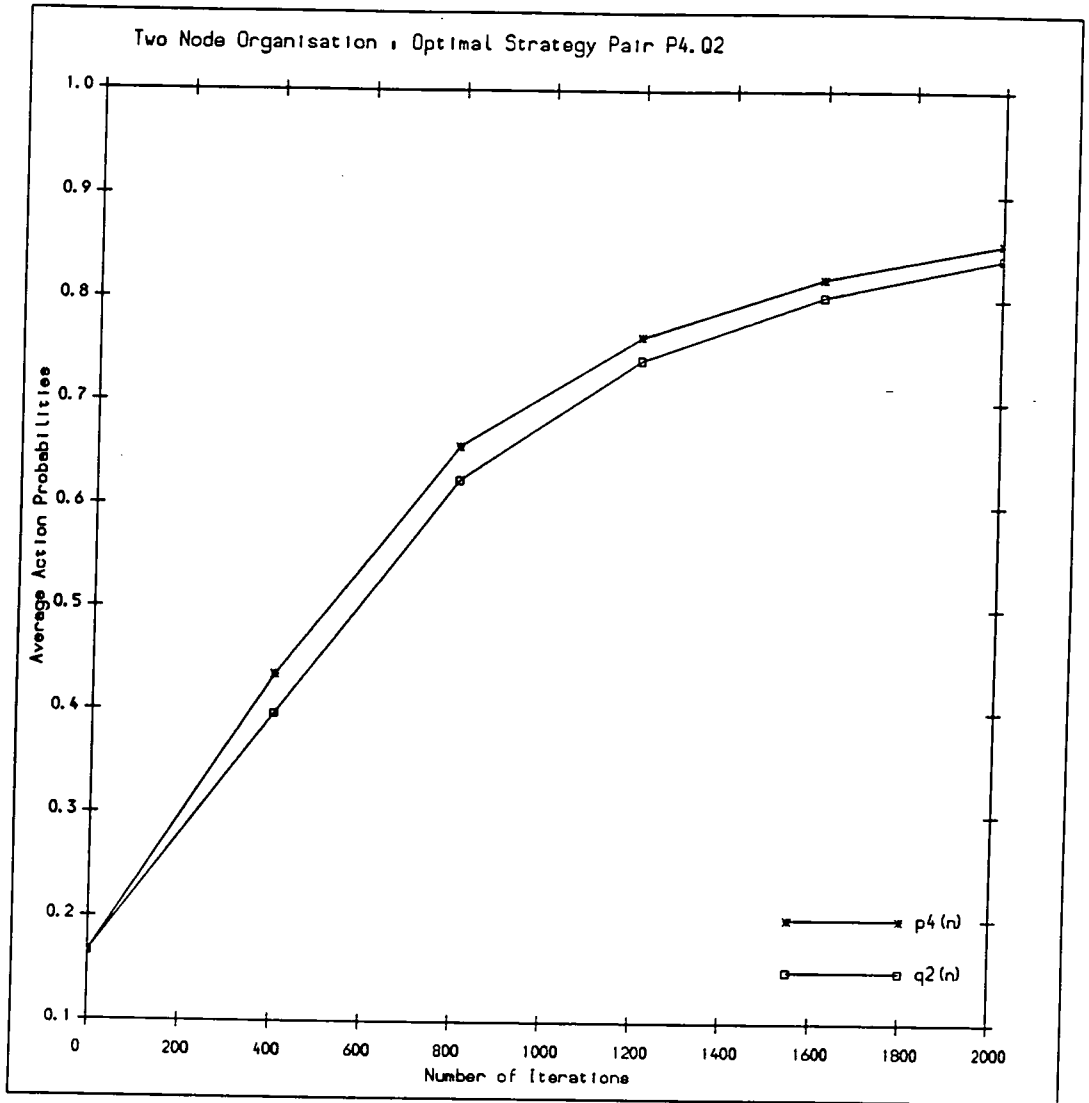
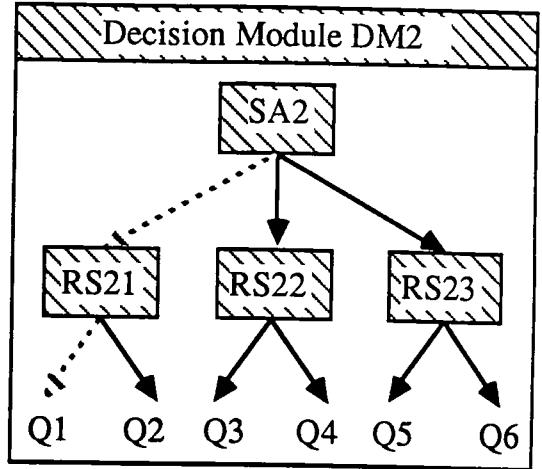
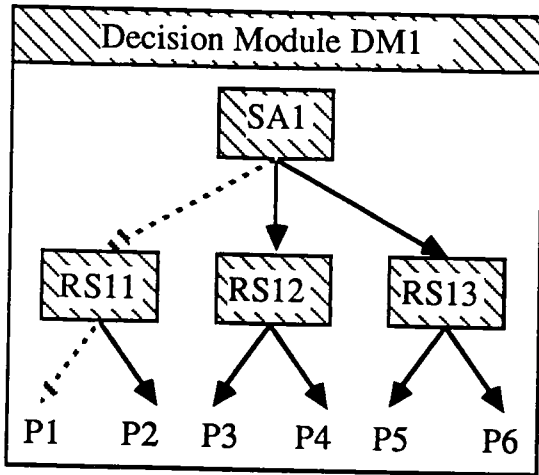
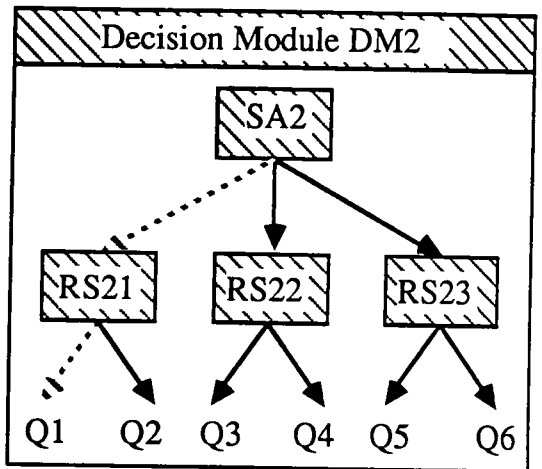
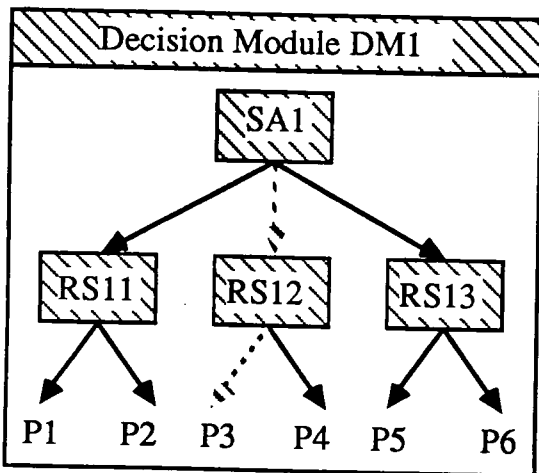


Figure 7.10a – Average Action Path Probability vs Iterations (Table 7.2)



ENVIRONMENT : OPTIMAL STRATEGY PAIR P1.Q1



ENVIRONMENT : OPTIMAL STRATEGY PAIR P3.Q1

Figure 7.11 - Switch of Environment: Before and After Switch

Reward Parameter = 0.04
Reward Probability
$C_{11} = 0.9$

Path Probability for Decision Module DM1: Before Switch P1

n	p1	p2	p3	p4	p5	p6
0	0.166666	0.166666	0.166666	0.166666	0.166666	0.166666
600	0.434715	0.187078	0.128409	0.108408	0.065751	0.075637
1200	0.656245	0.128620	0.081050	0.062218	0.032926	0.038940
1800	0.762995	0.093581	0.054786	0.040727	0.021841	0.026070
2400	0.819308	0.073124	0.041371	0.030263	0.016339	0.019594
3000	0.854036	0.059909	0.033233	0.024075	0.013052	0.015695

Path Probability for Decision Module DM2: Before Switch Q1

n	q1	q2	q3	q4	q5	q6
0	0.166666	0.166666	0.166666	0.166666	0.166666	0.166666
600	0.396374	0.150584	0.177897	0.165583	0.056801	0.052760
1200	0.623670	0.113208	0.116604	0.091556	0.028705	0.026255
1800	0.740127	0.084455	0.080160	0.058616	0.019184	0.017456
2400	0.801726	0.061028	0.061028	0.043054	0.014406	0.013075
3000	0.839764	0.054985	0.049259	0.034007	0.011533	0.010451

(a) Before Switch : Optimal Strategy Pair P1.Q1

Table 7.3 - Simulation of Two Node Organisation (Figure 7.11)

Reward Parameter = 0.04
Reward Probability
$C_{31} = 0.9$

Path Probability for Decision Module DM1: After Switch P3

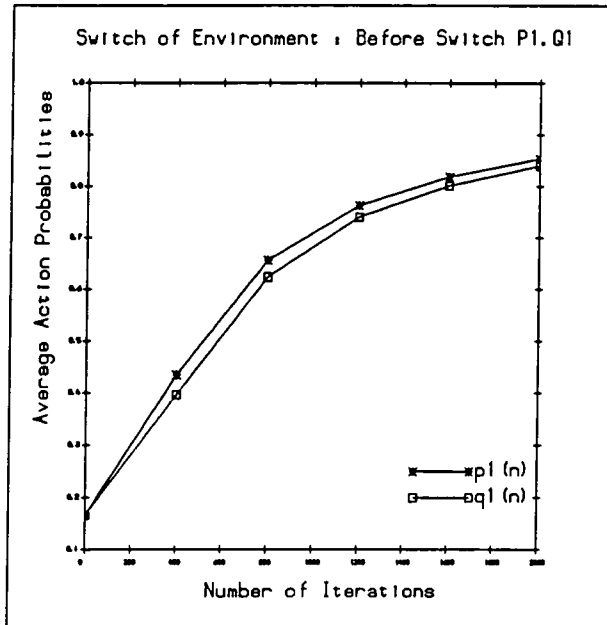
n	p1	p2	p3	p4	p5	p6
0	0.854036	0.059909	0.033233	0.024075	0.013052	0.015695
600	0.583030	0.033474	0.264141	0.102045	0.007646	0.009663
1200	0.302103	0.018049	0.573407	0.097502	0.003883	0.005056
1800	0.201256	0.012179	0.704976	0.075629	0.002574	0.003385
2400	0.150887	0.009189	0.774746	0.060707	0.001925	0.002544
3000	0.120683	0.007378	0.817884	0.050479	0.001537	0.002038

Path Probability for Decision Module DM2: After Switch Q1

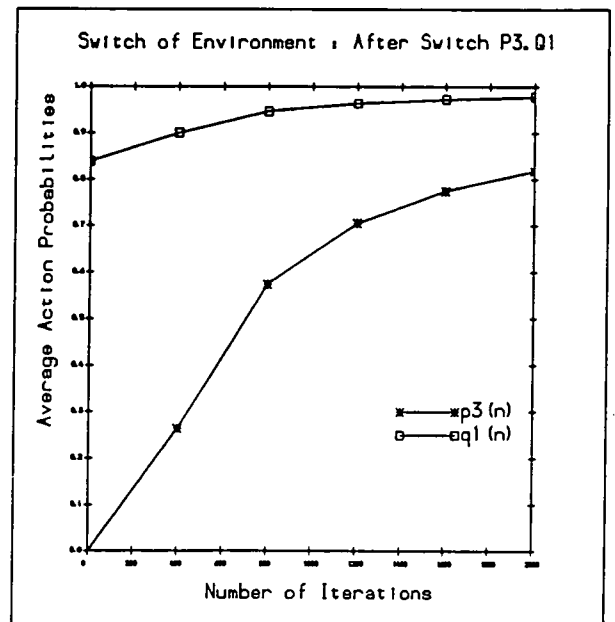
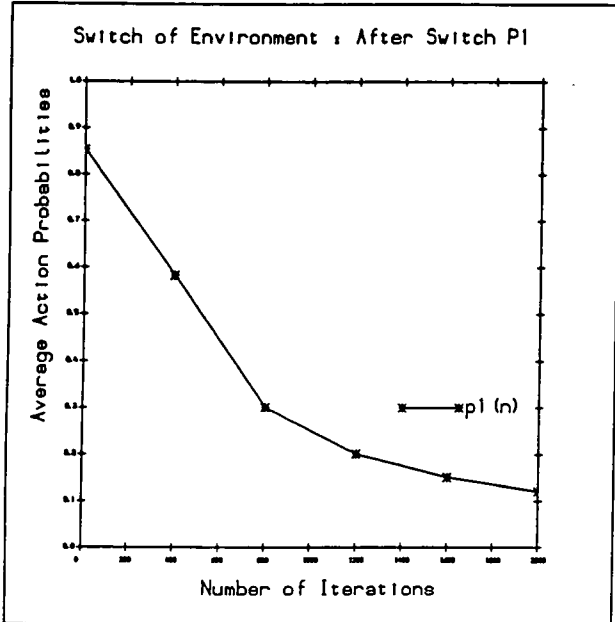
n	q1	q2	q3	q4	q5	q6
0	0.839764	0.054985	0.049259	0.034007	0.011533	0.010451
600	0.900723	0.058943	0.020865	0.015002	0.002343	0.002123
1200	0.946919	0.032772	0.010368	0.007705	0.001172	0.001062
1800	0.964462	0.021999	0.006879	0.005169	0.000782	0.000708
2400	0.973289	0.016556	0.005147	0.003889	0.000586	0.000531
3000	0.978604	0.013272	0.004112	0.003117	0.000469	0.000425

(b) After Switch : Optimal Strategy Pair P3.Q1

Table 7.3 - Simulation of Two Node Organisation (Figure 7.11)



(a) Before Switch P1.Q1 (Table 7.3a)



(a) After Switch P1 (Table 7.3b) (b) After Switch P3.Q1 (Table 7.3b)

Figure 7.12 – Average Action Path Probability vs Iterations

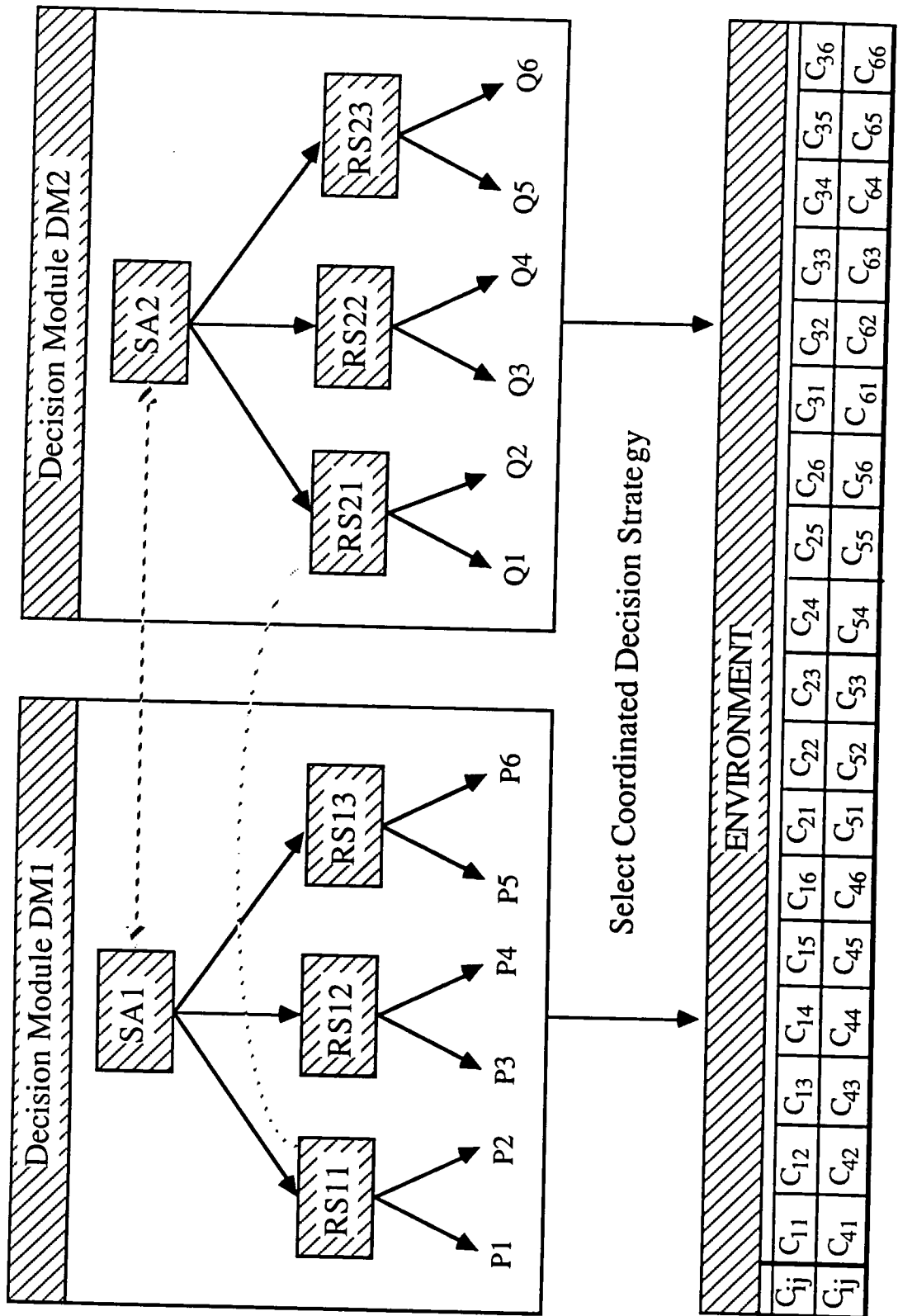


Figure 7.13 - Communication Between Decision Modules

Reward Parameter = 0.04
Increment Reward Probability = 0.16
Reward Probability
$C_{11} = 0.9$

Path Probability for Decision Module DM1: P1

n	p1	p2	p3	p4	p5	p6
0	0.166666	0.166666	0.166666	0.166666	0.166666	0.166666
600	0.838864	0.101717	0.014694	0.014704	0.015153	0.014867
1200	0.917820	0.052470	0.007341	0.007359	0.007583	0.007427
1800	0.944856	0.035337	0.004892	0.004907	0.005057	0.004950
2400	0.958508	0.026637	0.003669	0.003681	0.003793	0.003712
3000	0.966742	0.021374	0.002935	0.002945	0.003035	0.002969

Path Probability for Decision Module DM2: Q1

n	q1	q2	q3	q4	q5	q6
0	0.166666	0.166666	0.166666	0.166666	0.166666	0.166666
600	0.900723	0.058943	0.020865	0.015002	0.002343	0.002123
1200	0.946919	0.032772	0.010368	0.007705	0.001172	0.001062
1800	0.964462	0.021999	0.006879	0.005169	0.000782	0.000708
2400	0.973289	0.016556	0.005147	0.003889	0.000586	0.000531
3000	0.978604	0.013272	0.004112	0.003117	0.000469	0.000425

(a) Top Level Communication : (SA1 and SA2)

Table 7.4 – Communication Between Automata (Figure 7.13)

Reward Parameter = 0.04
Increment Reward Parameter = 0.16
Reward Probability
$C_{11} = 0.9$

Path Probability for Decision Module DM1: P1

n	p1	p2	p3	p4	p5	p6
0	0.166666	0.166666	0.166666	0.166666	0.166666	0.166666
600	0.909035	0.039772	0.012995	0.013038	0.012565	0.012593
1200	0.953980	0.020423	0.006498	0.006519	0.006284	0.006295
1800	0.969201	0.013734	0.004332	0.004346	0.004190	0.004196
2400	0.976856	0.010345	0.003249	0.003259	0.003143	0.003147
3000	0.981463	0.008298	0.002599	0.002607	0.002514	0.002518

Path Probability for Decision Module DM2: Q1

n	q1	q2	q3	q4	q5	q6
0	0.166666	0.166666	0.166666	0.166666	0.166666	0.166666
600	0.903764	0.042945	0.015087	0.015940	0.010836	0.011427
1200	0.951277	0.022077	0.007532	0.007981	0.005418	0.005714
1800	0.967384	0.014852	0.005019	0.005323	0.003612	0.003809
2400	0.975487	0.011189	0.003763	0.003994	0.002709	0.002857
3000	0.980365	0.008976	0.003010	0.003195	0.002167	0.002286

(b) Top and Lower Level Communication: (SA1:SA2) (RS11:RS21)

Table 7.4 – Communication Between Automata (Figure 7.13)

Reward Parameter = 0.04
Increment Reward Parameter = 0.16
Reward Probability
$C_{33} = 0.9$

Path Probability for Decision Module DM1: P3

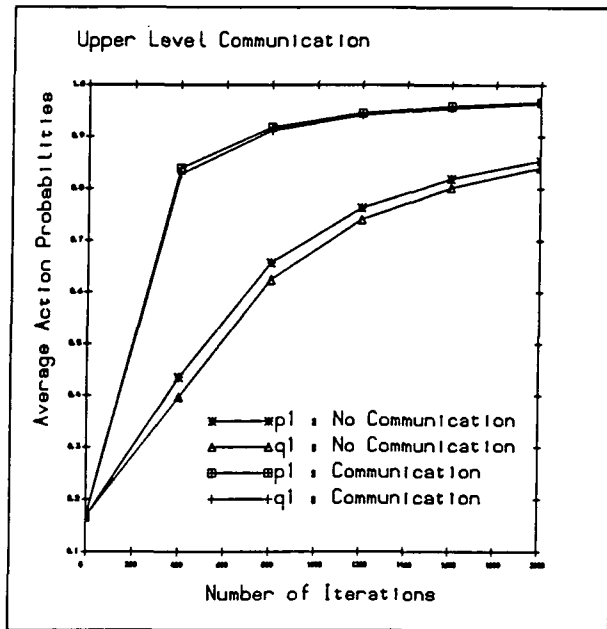
n	p1	p2	p3	p4	p5	p6
0	0.166666	0.166666	0.166666	0.166666	0.166666	0.166666
600	0.802600	0.091126	0.029971	0.026251	0.027005	0.023046
1200	0.898591	0.048272	0.014989	0.013121	0.013563	0.011463
1800	0.931791	0.032783	0.009994	0.008747	0.009055	0.007629
2400	0.948618	0.024813	0.007496	0.006560	0.006796	0.005717
3000	0.958785	0.019959	0.005997	0.005248	0.005439	0.004571

Path Probability for Decision Module DM2: Q3

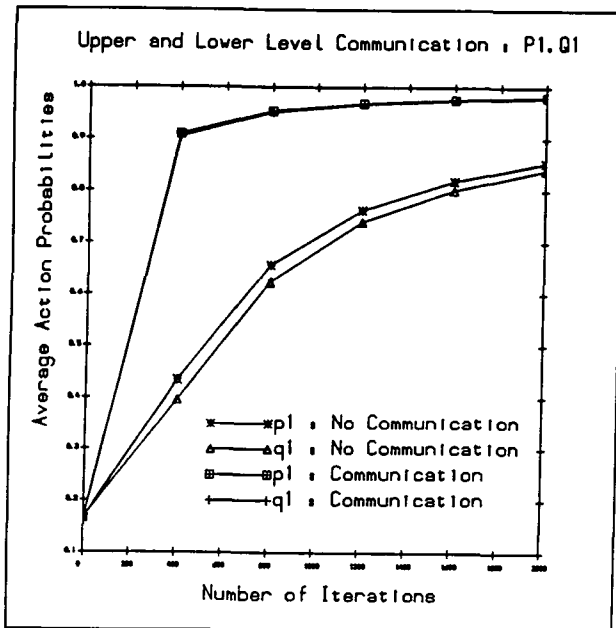
n	q1	q2	q3	q4	q5	q6
0	0.166666	0.166666	0.166666	0.166666	0.166666	0.166666
600	0.798982	0.109165	0.024862	0.027623	0.019942	0.019426
1200	0.896730	0.057343	0.012413	0.013830	0.009964	0.009720
1800	0.930540	0.038842	0.008271	0.009224	0.006641	0.006481
2400	0.947675	0.029361	0.006202	0.006919	0.004980	0.004861
3000	0.958029	0.023599	0.004961	0.005536	0.003984	0.003889

(c) Re-locate Unique Maximum: (SA1:SA2) (RS11:RS21) Communicate

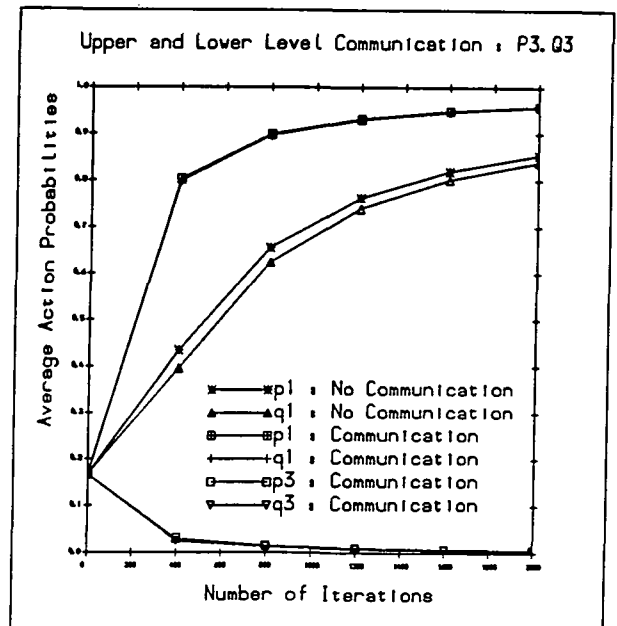
Table 7.4 – Communication Between Automata (Figure 7.13)



(a) Upper Level Path P1.Q1 (Table 7.4a)



(b) Path P1.Q1 (Table 7.4b)



(c) Path P3.Q3 (Table 7.4c)

Figure 7.14 - Average Action Path Probability vs Iterations

Chapter Eight

Conclusion and Recommendations for Future Work

8.1 Conclusion and Summary

This thesis has described the various approaches in developing analytical models for decentralised decision making under uncertainty. It has been emphasised that the core problems are quite profound and have a bearing on many areas that are interesting to a diverse range of disciplines. There are many challenging problems which remain unsolved, and the applications for new results will be widespread. The purpose of this chapter is to summarise and conclude the work that has been completed during the course of this project. In particular, to focus on the contribution made during the research programme and to outline areas of research that appear promising for the future.

The initial phase of research has provided an overview of the field of DAI and considered the importance of coordination in such systems. The thesis is not intended to provide a survey of the entire field of DAI, but rather it focusses on coordination techniques and the motivations for learning in DAI systems. The analysis of the work related to coordinating the problem solving of multiple agents has been regarded as the central problem of DAI research. The survey has highlighted that effective coordination in DAI systems requires three facets to be present: (i) *structure* within which agents can interact in predictable ways; (ii) *flexibility* so that nodes which exist in

dynamically changing environments can deal with incomplete, inaccurate or obsolete information; and (iii) the *knowledge* and *reasoning* capabilities to intelligently use the structure and flexibility. The basic stochastic learning automaton framework deals with the first two points. In particular, the interactive decision making models discussed in Chapter Four, exhibit the features of structure and flexibility. These interconnections provide each model with varying levels of flexibility in their interactions towards agents and the changing environment. However, the discussion has concluded that the basic framework used in isolation is inadequate for the representation of a generalised network. It is necessary to extend the modelling framework, to overcome these limitations. This extension has been addressed in Chapter Six which proposes a hybrid model. The final feature is not a matter of coordination, rather it is the ability to reason about information and predictions when making decisions about its local problem solving. The stochastic learning automaton does not have this characteristic. However, the model exhibits an intelligence capability that use the structure and flexibility in order to adapt to dynamically changing environments.

The survey has also highlighted that there has been limited research dealing with learning in the DAI literature. The implementation of successful learning methods in DAI systems can have significant impacts on the development of distributed decision making models operating in uncertain environments. The thesis proposes learning in a multiagent setting which has been discussed in the context of an AI approach. This methodology is based on the stochastic learning automata which is considered to represent

a promising approach to providing a conceptual framework for modelling of decentralised decision making.

Interactive Automata Model

The previous chapters have addressed the problems of decentralisation and uncertainty. To formalise these ideas a promising framework based on the stochastic learning automaton model has been considered. In particular, the concept has been directed towards modelling highly interactive situations which consider different methods of interconnection of individual decision makers. These models illustrate how decision makers interact with each other and update their decisions using known learning schemes. The behaviour of such models may be explained by using concepts from both stochastic learning theory and game theory. A detailed description has been presented by considering both synchronous and sequential models. For such structures it is important to know what kind of interconnections result in a desirable overall system performance. This can be achieved by analysing the corresponding game structure as interconnections are varied. Whilst such interconnections were appropriate for the representation of interactive models, these models are rather primitive. It is evident that a detailed investigation of the specific interconnections was needed before organisational structures can be designed that promote high quality decentralised decision making performance. The basic framework was not sufficient to illustrate the explicit interaction between decision makers which capture information flow and time delays that are crucial in the modelling of systems. In addition these models were restricted

in modelling flexibility, since they could only be used to model systems that exhibit feedback and hierarchical configurations. It was necessary to propose a more convenient modelling tool to meet these specific requirements and this was accomplished in the next phase of research.

Petri Net and Associated Models

The thesis has defined a high-level mathematical framework based on Petri net methodology. This formalism has presented an abstract, formal graph model useful for representing systems which exhibit concurrent, asynchronous, distributed parallel and/ or stochastic activities. Several recent attempts have considered the potential of Petri nets in the modelling of decision making organisations, [60], [61]. In particular, their work was oriented towards the optimal design of organisations. This optimal design is based on the data flow formations which are used to model in a precise manner the various types of interactions between decision makers as well as interactions between decision aids and systems that support the organisation. Although such work may be necessary for the optimal design of organisations the thesis has considered that a focus on data flow formations alone is not sufficient to guarantee high levels of performance in a distributed decision making organisation. It was also essential to focus on the behaviour of an organisation that operates under uncertainty. This viewpoint introduced a new dimension to existing Petri net theory. The thesis proposed an extension to Petri nets and has developed a new class of modelling/design tools known as *Learning Petri Net* models.

The extension to Petri nets was introduced by embedding the concept

of stochastic learning automata into the model. The intelligence capability incorporated within different forms of Petri nets has greatly enhanced the modelling power of Petri nets. Each variation in the modelling technique has exhibited a data flow formation, a decision making process embedded within the structure and, various types of transitions associate with each model. Clearly, these learning Petri net structures highlight a powerful design tool for the effective representation of distributed decision problems.

The thesis has shown how the use of a SLPN model enables dynamic decision making by controlling the selection of decisions on a probabilistic basis. The model has also illustrated how information can be monitored at each time instant such that probabilistic outcomes of the decisions can be captured to achieve a desirable global performance. It may also be noted that all the proposed hybrid models alter in some way the firing rule of Petri nets, and make it so that the main part of net theory is no longer applicable. These features have emphasised the increased complexity of the hybrid models in comparison to the standard Petri net model.

The learning Petri net models are often constrained to a graphical representation and reachability tree analysis. Many problems regarding the correct operation of these new models can be posed in terms of questions related to the reachability of states in Petri nets. The reachability problem may be studied by finding finite representations for reachability sets. However, to ensure the correct operation of modelling systems many factors must be addressed. These characteristics may be stated in the form of assertions which includes maintaining integrity in concurrent activities, guarantee deadlock free

operation and system must be resilient to failures. Correctness models are used for two purposes: to provide descriptions of systems and to facilitate proofs about system assertions. In simple terms, correctness models attempt to prove some desired characteristics of systems. The issue of proving correct operation has received attention only relatively recently. There seems to be no model which is widely applicable to the broad spectrum of correctness problems. The analysis of correctness used in many systems tend to be relatively informal, however in some cases mathematical models are used. Petri nets are considered to be ideal representatives of correctness models.

The thesis has presented a classical analytical framework which have been developed largely within the framework of mathematics and game theory. These models use the available information (without recourse to experience) to make optimal judgements and decisions. In these models skill/ experience levels are not reflected. The models are not applicable in situations where a decision maker must use skills of sizing up a situation, detecting patterns, imagining how a course of action will be performed, anticipating undesirable consequences and so forth. The strongest disadvantage of analytical models is that they prevent decision makers from taking advantage of their skills in sizing up situations and planning courses of action. In addition such models do not work under time pressure because they take too long. However, when there is enough time, they require much work and lack flexibility for handling rapidly changing conditions. It may be emphasised that analytical decision making is more helpful when there is a conflict to be resolved, especially when conflict involves people with different concerns. The analytical frameworks

are usually a better strategy to be used when an optimal solution is required. Finally, analytical strategies necessary when the problem involves so much computational complexity that alternate strategies would be inadequate. Thus, an interesting exercise to overcome the limitations that exist within analytical frameworks would involve an examination of the internal structure of decision modules. Additional characteristics could be embedded within each module to provide modelling flexibility in operational settings.

Applicability of Models to Realistic Problems

The application of the new modelling tool to a non-trivial example has been considered. This application has demonstrated the modelling power of Stochastic Learning Petri net (SLPN) tool. In particular, the design tool was used to illustrate modelling flexibility and suitability to a realistic distributed decision problem. To demonstrate the versatility of the model different scenarios were contrived to observe the performance of the organisation. The basic model involved a two node organisation interacting with a stochastic environment. Each module communicated with a decision support system to learn the optimum strategy for interaction with the random environment. The scenario illustrated the modelling capability of optimum distributed strategies in stochastic environments for both steady state and switched environments. The ability to model communication between adjacent hierarchical layers and the ability to use confidence communication signals to provide adaptive stepsizes to improve convergence rates has been shown. The new results provided by each experiment have shown some trends that enable general

statements to be made about the behaviour of the organisation under varied circumstances. These models are well suited to represent more complex decision problems. In practice it is much more common that such organisations comprise a large number of decision modules connected in arbitrary topology.

It may be noted that the Generalised Stochastic Learning Petri net (GSLPN) may also be considered for these applications. This approach requires more modelling effort, but results in a much smaller reachability set. It is thus desirable to consider a GSLPN application whose complexity only depends on the combination of transitions. This can be done provided that a careful study of the application domain is considered.

8.2 Recommendations for Future Work

During the course of this research project a modelling tool for decentralised distributed systems based on a hybrid approach has been proposed. The application of this modelling tool to a simple non-trivial example has been considered. These models are capable of dealing with realistic more complex decentralised decision problems. The next phase of research can be extended by using concepts which have originated from the research programme.

8.2.1 Models (Byzantine Generals)

To date the application of the modelling tool to a two node organisation has been considered. This may be extended to observe the performance of an arbitrary number of decision modules connected in a range of topological structures. However, an important component of such large, reconfigurable distributed structures is considered to be *reliability*. Intuitively, reliability is a measure of how well a system can tolerate and recover from failures. Reliability has two aspects, a system is said to be reliable if its output or results are correct. Correctness is an important factor because the system is required to maintain consistency. The presence of failures may temporarily cause an inconsistent state but the recovery algorithms should restore the system to a correct state. The second aspect of reliability has to do with availability. A reliable system should tolerate failures and should be able to continue operation even in a degraded state. The objective is to design a system that can sustain multiple failures and continue to process transactions promptly and correctly.

Analytical models may be used to study unreliable communication between decision modules. An unreliable decentralised system often requires a means by which independent decision modules can arrive at an exact mutual agreement of some kind. In the absence of faults, errors and failures reaching a satisfactory mutual agreement is usually an easy matter. However, in the presence of failed components a decision module can behave in an unpredictable manner; block information from being relayed, alter the information relayed through itself, incorrectly reroute the information and in the worst case it can send conflicting information to different parts of the system. The Byzantine Generals problem is proposed as an appropriate model in resolving this type of failure, [69], [70]. Appendix Four provides a description of the Byzantine Generals problem and several algorithms for dealing with conflicting, inconsistent information in a system.

8.2.2 Modelling Human Factor in C³I Systems

Most studies of command and control have focussed on an organisation which is formed in order to perform a set of tasks that individuals cannot perform alone. The task to be performed by the organisations being considered consist of receiving signals or inputs from one or more sources, processing them and producing outputs which can be actions or signals. A single decision maker cannot perform these tasks alone because of the large amount of information processing required and because of the fast tempo of operations (eg. tactical situations). Such organisations have also neglected the ways in which decision makers diagnose problems, develop solutions and select options.

It is precisely these functions and the limitations of humans as information processors and problem solvers which constitutes a major problem in the development of realistic models of C³I. Thus, it is important to examine the ways in which human performance affects the functioning of C³I systems.

The theory of human decision making involves an interdisciplinary body of knowledge. There exists contrasting schools of thought in this complex area. These include classical analytical theories, naturalistic theories and a class which reside somewhere in between these two opposing approaches to understanding and explaining decision making processes, [7], [71], [72], [73]. Figure 8.1 illustrates that there are certain factors which increase the decision makers tendency to use particular types of decision making strategies, [72].

8.2.3 Automatic Data Fusion

The requirements for an automatic data fusion process are complex and diverse. A distributed data fusion system represents a good platform for integrating several complementary AI technologies. Such a system is capable of providing a robust, maintainable, concurrent processing environment. Additionally, the system would be potentially flexible enough to support the wide variety of processing capabilities which are necessary for the task of data fusion.

The automation of data fusion is not a trivial task by any standards. The fusion process must be able to combine high processing throughput with human like processing faculties. A fusion system must deal with all situations including, drawing upon a wide range of knowledge and experience,

or alternatively a provision to learn from experience must be incorporated within the system. It should also exhibit a means for reasoning with information which can be uncertain and unreliable. In such cases, the technical complexity of an automated fusion system will depend upon the level of processing which is supported. The benefits to be gained from automating various levels of data fusion tasks are significant. These include: the automation of the mundane fusion tasks such as data association and classification; when the influx of reports are high, such systems are potentially capable of much higher processing throughput; a computerised system is less susceptible to errors, and is not fatigued by repetitive, menial tasks; capable of identifying hidden inferences and associations which are typically not identified through human level reasoning.

A simplified model of the fusion process can be described in terms of two levels. The lower level procedures are mostly concerned with mechanistic calculations to determine the classification and state of an object. This involves the derivation of target position and identity, and hence the processing speed will usually be an important consideration. The higher level procedures are concerned with the derivation of threats, patterns of behaviour, and predictions of future intentions from the perceived situation. This level of processing is more abstract and involves sophisticated reasoning, and less mechanistic computation. A basic generic model of data fusion has been described by Waltz and Llinas, [74]. In addition, it is necessary to examine the potential technologies which could form the basis of a simple fusion system. In this context an area worthy of further investigation is the internal structure of the

four-stage decision making process, Figure 7.1. In particular, it is necessary to focus on the information fusion (IF) and command interpretation (CI) stages, to improve system performance and to achieve specified task objectives.

8.2.4 Migration of Control

The notion of control migration seems to intuitively reflect system behaviour. In a classical feedback loop, control refers to the result of processing the externally produced command statements of system requirements and analysing system behaviour in the presence of uncertainty. The migration of control refers to the movement of the control function through the information structure of the system. It is a feature that can be built into large scale systems, if an adequate structure for control can be established. In large scale organisations, control may migrate in an unpredictable manner away from the decision makers who have been assigned specific authority and responsibility to subordinate members of the organisation. The changes in an organisations structure such as, access to decision support systems can change the sensitivity of performance measures to the actions of different decision makers. Moreover, the choice of strategies selected by each decision maker affects which one has the most impact on performance.

The migration of control may be viewed from both positive and negative perspectives. From a positive viewpoint, it is desirable for control to migrate in the event of a failure of a decision making node. However, from a negative perspective control may migrate in an unforeseen or undesirable manner. Migration may occur away from nodes which belong to higher

echelons to the lower echelons. Research in migration of control is still in its primitive stages, and is continuing as part of an effort to understand the dynamics of organisations, [62], [75].

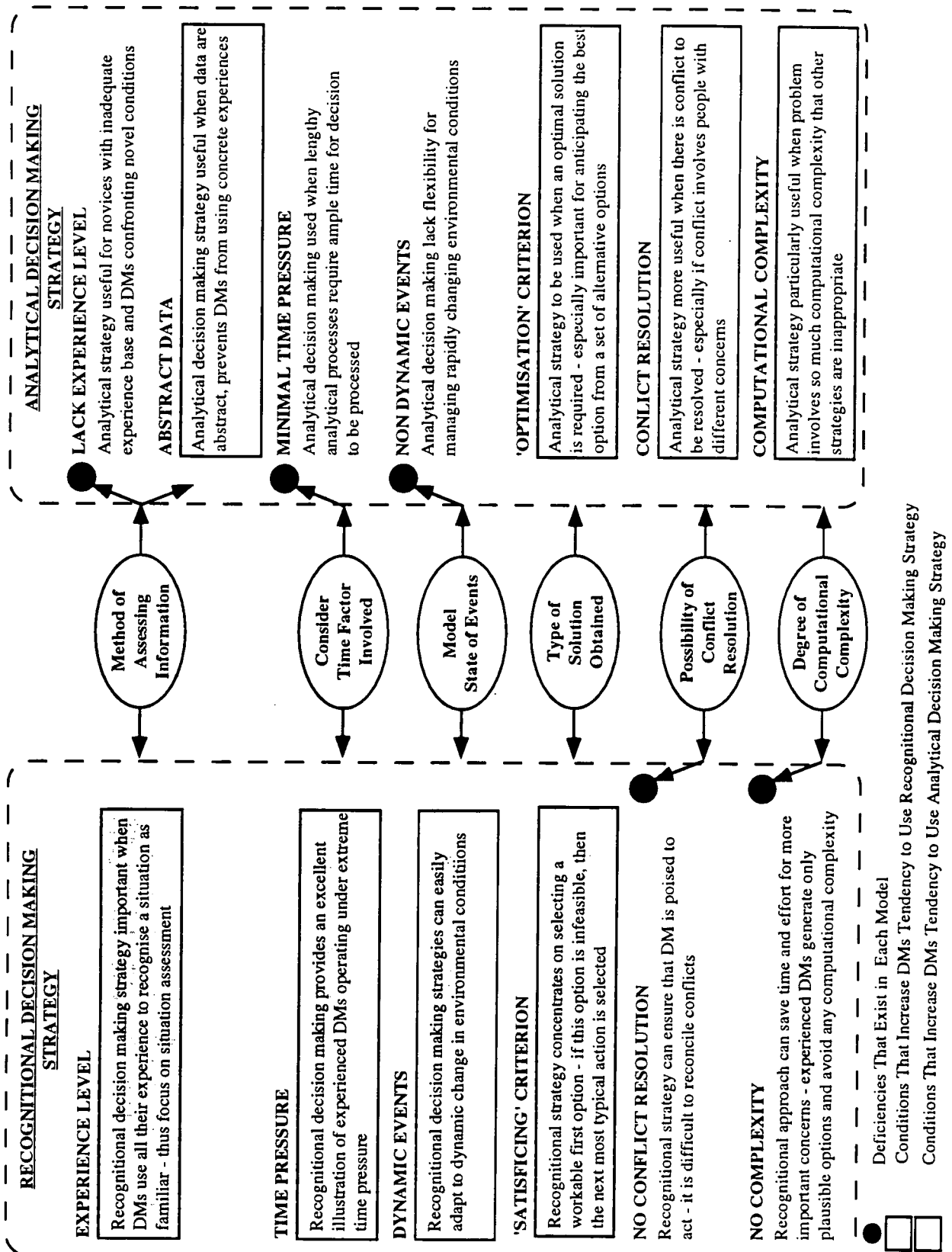


Figure 8.1 - Factors Affect Use of Recognitional/
Analytical Decision Making Strategies

References

- [1] Athans, M.:
'Command and control theory: A challenge to control science',
IEEE Trans. on Automatic Control, 1987, AC-32, No. 4, pp. 286-293.
- [2] Andriole, S. J., and Halpin, S. M.:
'Information technology for command and control',
IEEE Trans. Syst. Man and Cybern., 1986, SMC-16, No. 6, pp. 762-765.
- [3] Stephanou, H. E., and Sage, A. P.:
'Perspectives on imperfect information processing',
IEEE Trans. Syst. Man and Cybern., 1987, SMC-17, No. 5, pp. 780-798.
- [4] Ahmed, Q. F.:
'Decentralised Decision Making in $C^3 - I$ systems',
Int. Res. Report, Univ. of Durham, 1988.
- [5] Sage, A. P.:
'Information systems engineering for distributed decision making',
IEEE Trans. Syst. Man and Cybern., Nov.-Dec. 1987, Vol. SMC-17, No. 6,
pp. 920-936.
- [6] Kleinrock, L.:
'Distributed systems',
Comm. of the ACM, Nov. 1985, Vol. 28, No. 11, pp. 1200-1213.
- [7] Klein, G. A., Orasanu, J., Calderwood, R., and Zsombok, C. E.:
'Decision Making in Action: Models and Methods',
New Jersey: Ablex Publishing Corporation, 1993
- [8] Bond, A. H., and Gasser, L.:
'Readings in Distributed Artificial Intelligence',
Morgan Kaufmann Publishers, Inc., P0 Box 50490, Palo Alto, CA 94403, 1988.
ISBN 0-934613-63-X
- [9] Barr, A. H., Cohen, P. R., and Feigenbaum, E. A.:
'The Handbook of Artificial Intelligence ; Volume IV',
Addison-Wesley Publishing Company, Inc., 1989.
ISBN 0-201-51819-8
- [10] Avouris, N. M., and Gasser, L.:
'Distributed Artificial Intelligence: Theory and Praxis',
Computer and Information Science, Vol. 5
Kluwer Academic Publishers, Inc., P0 Box 17, 3300 AA Dordrecht, The Netherlands, 1992.
ISBN 0-7923-1585-5
- [11] Narendra, K. S., and Thathachar, M. A. L.:
'Learning automata - a survey',
IEEE Trans. Syst. Man and Cybern., 1974, SMC-4, pp. 323-334.
- [12] Narendra, K. S., and Thathachar, M. A. L.:
'Learning Automata - An Introduction',
Prentice Hall, Englewood Cliffs, New Jersey 07632, 1989.
ISBN 0-13-485558-2

- [13] Wheeler, R. M., and Narendra, K. S.:
'Learning models for decentralised decision making',
Automatica, 1985, Vol. 21, No. 4, pp. 479-484.
- [14] Wheeler, R. M.:
'Decentralised learning in games and finite Markov Chains',
Ph.D. Thesis, Dept. of Elect. Eng., Yale Univ., 1985.
- [15] Peterson, J. L.:
'Petri Net Theory and the Modelling of Systems',
Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.
ISBN 0-13-661983-5
- [16] Murata, T.:
'Petri Nets: Properties, Analysis and Applications',
Proc. IEEE, April 1989, Vol. 77, No. 4, pp. 541-580.
- [17] Gasser, L. and Huhns, M.:
'Distributed Artificial Intelligence: Vol.II',
Pitman Publishing, London, 1989.
ISBN 0268-7526
- [18] Huhns, M. N.m Mukhopadhyay, U., Stephens, L. M., and Bonnel, R.D.:
'DAI for document retrieval: The MINDS project',
in Distributed Artificial Intelligence, Pitman, London, 1987, pp.249-284.
- [19] Shaw, M., and Whinston, A.:
'Learning and adaptation in DAI systems',
in Distributed Artificial Intelligence : Vol 2, Pitman, London, 1989, pp.413-429.
- [20] Sian, S. S.:
'Adaptation based on cooperative learning in Multi-Agent systems',
in Decentralised A. I. : Vol 2, Elsevier Science Publishers, B.V, 1991, pp.257-272.
- [21] Gasser, L.:
'Approaches to Coordination',
in Distributed Artificial Intelligence: Theory and Praxis,
Computer and Information Science, Vol. 5
Kluwer Academic Publishers, Inc., PO Box 17, 3300 AA Dordrecht, The Netherlands, 1992, pp. 31-51.
- [22] Jennings, N.:
'The ARCHON system and its applications',
Proc. Int. Working Conf. on Cooperative Knowledge Based Systems, University of Keele, Keele, 1994.
- [23] Jennings, N.:
'Cooperation in Industrial Multi-Agent Systems',
World Scientific Publishing Co. Pte. Ltd, 1994.
ISBN 981-02-1652-1
- [24] Sugawara, N.:
'A cooperative LAN diagnostic and observation expert system',
Proc. International Conf. on Computers, Communications, Scottsdale, AZ, 1990.
- [25] Barr, A. H., Cohen, P. R., and Feigenbaum, E. A.:
'Cooperative distributed problem solving',
in The Handbook of Artificial Intelligence ; Volume IV,

Addison-Wesley Publishing Company, Inc., 1989, pp.85-147.
ISBN 0-201-51819-8

- [26] Smith, R. G.:
'The Contract Net protocol: High level communication and control in distributed problem solver',
IEEE Trans. on Computers, 1980, Vol. C-29, No. 12, pp. 1104-1113.
- [27] Smith, R. G., and Davis, R.:
'Negotiation as a metaphor for distributed problem solving',
in Readings in Distributed Artificial Intelligence,
Morgan Kaufmann Publishers, Inc., 1988, pp. 333-357.
ISBN 0-934613-63-X
- [28] Conry, S. E., Meyer, R. A., and Lesser, V. R.:
'Multistage negotiation in distributed planning',
in Readings in Distributed Artificial Intelligence,
Morgan Kaufmann Publishers, Inc., 1988, pp. 367-385.
ISBN 0-934613-63-X
- [29] Durfee, E. H., Lesser, V. R., and Corkill, D. D.:
'Trends in cooperative distributed problem solving',
IEEE Trans. Knowledge and Data Engineering, 1989, Vol. 1, No. 1, pp.63-83.
- [30] Lesser, V. R., and Corkill, D. D.:
'The distributed vehicle monitoring testbed: a tool for investigating distributed problem solving networks',
AI Magazine, 1983, pp. 15-33.
- [31] Georgeff, M.:
'Communication and interaction in multiagent planning',
in Readings in Distributed Artificial Intelligence,
Morgan Kaufmann Publishers, Inc., 1988, pp. 200-205.
ISBN 0-934613-63-X
- [32] Georgeff, M.:
'A theory of action for multiagent planning',
in Readings in Distributed Artificial Intelligence,
Morgan Kaufmann Publishers, Inc., 1988, pp. 205-210.
ISBN 0-934613-63-X
- [33] Georgeff, M.:
'The representation of events in multiagent domains',
in Readings in Distributed Artificial Intelligence,
Morgan Kaufmann Publishers, Inc., 1988, pp. 210-216.
ISBN 0-934613-63-X
- [34] Cammarata, S., McArthur, D., and Steeb, R.:
'Strategies for cooperation in distributed problem solving',
in Readings in Distributed Artificial Intelligence,
Morgan Kaufmann Publishers, Inc., 1988, pp. 102-106.
ISBN 0-934613-63-X
- [35] Rosenchein, J. S., and Genesereth, M. R.:
'Deals among rational agents',
in Readings in Distributed Artificial Intelligence,
Morgan Kaufmann Publishers, Inc., 1988, pp. 227-235.
ISBN 0-934613-63-X

- [36] Gasser, L.:
 'An Overview of DAI',
in Distributed Artificial Intelligence: Theory and Praxis,
 Computer and Information Science, Vol. 5
 Kluwer Academic Publishers, Inc., P0 Box 17, The Netherlands, 1992, pp. 9-30.
- [37] Durfee, E. H., and Lesser, V. R.:
 'Using partial global plans to coordinate distributed problem solvers',
in Readings in Distributed Artificial Intelligence,
 Morgan Kaufmann Publishers, Inc., 1988, pp. 285-294.
 ISBN 0-934613-63-X
- [38] Jennings, N.:
 'Commitments and conventions: the foundation of coordination in multi-agent systems',
The Knowledge Engineering Review, 1994, Vol. 8, No. 3, pp.223-250.
- [39] Rosenchein, J. S., and Genesereth, M. R.:
 'Communication and cooperation among logic-based agents',
in Readings in Distributed Artificial Intelligence,
 Morgan Kaufmann Publishers, Inc., 1988, pp. 1104-1113.
 ISBN 0-934613-63-X
- [40] Rosenchein, J. S.:
 'Synchronisation of multiagent plans',
in Readings in Distributed Artificial Intelligence,
 Morgan Kaufmann Publishers, Inc., 1988, pp. 187-192.
 ISBN 0-934613-63-X
- [41] Mars, P., Ahmed, Q. F., and Edwards, P.:
 'Applications of artificial intelligence techniques to decentralised decision-making in C³ - MIS'
IEE Third Int. Conf. on Command, Control, Communications and Management Information Systems, May 1989, pp.78-83.
- [42] Ahmed, Q. F.:
 'New Approaches (Stochastic Learning Automata)',
in 'Decentralised Decision Making in C³ - I systems',
 Int. Res. Report, Univ. of Durham, Oct. 1988, pp.72-77.
- [43] Tsetlin, M. L.:
 'Automaton Theory and Modelling of Biological Systems',
 Academic Press, 1973.
- [44] Narendra, K. S., and Thathachar, M. A. L.:
 'Interconnected automata and games',
in, Learning Automata - An Introduction, Prentice Hall, Englewood Cliffs, New Jersey, 1989, pp. 281-357.
- [45] Ahmed, Q. F.:
 'Decentralised Decision Making Models',
 Int. Res. Report, Univ. of Durham, Aug. 1989.
- [46] Thomas, L. C.:
 'Games, Theory and Applications',
 Ellis Horwood Limited, 1986.
 ISBN 0-7458-0142-0

- [47] Ramakrishnan, K. R.:
'Hierarchical systems and cooperative games of learning systems',
Ph.D Thesis, Indian Institute of Science, Bangalore, India, 1982.
- [48] Thathachar, M. A. L., and Ramakrishnan, K. R.:
'A hierarchical system of learning automata',
IEEE Trans. Syst. Man and Cybern., Mar. 1981, Vol. SMC-11,
pp. 236-241.
- [49] Mitchell, B. T., and Kountais, D. I.:
'A reorganisation scheme for a hierarchical system of learning automata',
IEEE Trans. Syst. Man and Cybern., Mar. 1984, Vol. SMC-14, No. 2, pp.
328-334.
- [50] Baba, N.:
'Learning behaviours of hierarchical structure stochastic automata operating in a
general multiteacher environment',
IEEE Trans. Syst. Man and Cybern., July/ Aug. 1985, Vol. SMC-15, pp.
585-587.
- [51] Peterson, J.:
'Petri Nets',
Computing Surveys, Sept. 1977, Vol. 9, No. 3, pp. 223-252.
- [52] Agerwala, T.:
'Putting Petri Nets to Work',
Computer, Dec. 1989, Vol. 12, No.12, pp. 85-94.
- [53] Petri, C. A.:
'Communication with Automata',
Ph.D dissertation, Tech. Rep. RADC-TR-65-377, Rome Air Development Center,
Rome, NY, 1966
- [54] Marsan, M. A., Balbo, G., and Conte, G.:
'Stochastic Petri Nets',
in Performance Models of Multiprocessor Systems
Cambridge, MA: The MIT Press, 1987, pp. 73-95.
ISBN 0-262-01093-3
- [55] Pagnoni, A.:
'Stochastic nets and performance evaluation',
LNCS, Systems, 1977, pp. 460-478.
- [56] Molloy, M. K.:
'Performance analysis using stochastic Petri nets',
IEEE Trans. Computers, Sept. 1982, Vol. C31, No. 31, pp. 93-97.
- [57] Marsan, M. J., Conte, G., and Balbo, G.:
'A class of generalised stochastic Petri nets for the performance evaluation of
multiprocessor systems',
ACM Trans. on Comp. Syst., May 1984, Vol. 2, No. 2, pp. 93-122.
- [58] Ahmed, Q. F.:
'Stochastic Learning Petri Nets',
Int. Res. Report, Univ. of Durham, Mar. 1990.

- [59] Ahmed, Q. F., and Mars, P.:
 'Application of stochastic learning Petri Nets to small-scale distributed decision making organisations',
IMA Int. Conf. on Control: Modelling, Computation, Information, Sept. 1992.
- [60] Levis, A. H.:
 'Information processing and decisionmaking organisations :
 a mathematical description',
Large Scale Systems, 1984, 7, pp. 151-167.
- [61] Tabak, D., and Levis, A. H.:
 'Petri net representation of decision models',
IEEE Trans. Syst. Man, Cybern., Nov.-Dec. 1985, vol. SMC-15, No. 6, pp. 812-818.
- [62] Skulsky, S. L., and Levis, A. H.:
 'Migration of control in distributed intelligence systems',
Proc. IEEE International Symposium on Intelligent Control 1989,
- [63] Weingaertner, S. T., and Levis, A. H.:
 'Analysis of decision aiding in submarine emergency decisionmaking',
Automatica, 1989, vol. 25, No. 3, pp. 349-358.
- [64] Demael, J. J., and Levis, A. H.:
 'On generating variable structure architectures for decision making systems',
Information and Decision Technologies, 1994, Vol. 19, pp.233-255.
- [65] Perdu, D. M., and Levis, A. H.:
 'Petri Net model for evaluation of expert systems in organisations',
Automatica, 1991, Vol. 27, No. 2, pp. 225-237.
- [66] Levis, A. H.:
 'A Coloured Petri Net model of intelligent nodes',
 in *Robotics and Flexible Manufacturing Systems*, 1992, J. C. Gentina and S. G. Tzafestas, Eds., Elsevier Science Publishers B. V., The Netherlands.
- [67] Ahmed, Q. F.:
 'Application of Stochastic Learning Petri Nets to C³I Systems',
 Int. Res. Report, Univ. of Durham, Jan. 1991.
- [68] Boettcher and Levis, A.:
 'Modelling the interacting decision maker with bounded rationality',
IEEE Trans. Syst. Man and Cybern., May-June 1982, Vol. SMC-12, No. 3, pp. 334-345.
- [69] Bhargava, B. K.:
 'The Byzantine Generals',
 in 'Concurrency Control and Reliability in Distributed Systems',
 Van Nostrand Reinhold Company Inc., 1987, pp. 348-369.
 ISBN 0-442-21148-1
- [70] Ahmed, Q. F.:
 'Models (Byzantine Generals)', in 'Decentralised Decision Making in C³ - I Systems',
 Int. Res. Report, Univ. of Durham, Oct. 1988, pp.52-71.

- [71] Klein, G. A., and Zsombok, C. E.:
'Models of skilled decision making',
Proc. Human Factors Society 35th Annual Meeting, 1991, pp. 1363-1366.
- [72] Klein, G. A.:
'Strategies of decision making',
Military Review, May 1989, pp. 56-64.
- [73] Wohl, J. A.:
'Force management decision requirements for air force tactical command and control',
IEEE Trans. Syst. Man and Cybern., 1981, Vol. SMC-11,
pp. 618-639.
- [74] Waltz, E., and Llinas, J.:
'Multisensor Data Fusion',
Artech House, Inc., Norwood, MA, 1990.
ISBN 0-89006-277-3
- [75] Kahne, S.:
'Control Migration: A characteristic of C³ systems',
IEEE Control Systems Magazine, Feb. 1983, Vol. 3, No. 1, pp. 15-19.

Appendix One

Computer Simulation Structure

A1.1 Introduction

This section outlines the general structure of simulations, which illustrate the performance of the interactive decision making models. An algorithm is included to show the basic stochastic model operating in an unknown random environment, and updating action probabilities using Linear Reward/ Inaction scheme. It is possible to modify this Stochastic Automaton Algorithm based on the chart presented in Figure A1.1, to acquire the appropriate synchronous and sequential models.

A1.2 Structure of Simulation

The general structure of a simulation is displayed in the form of a chart in Figure A1.1. The chart shows the main stages in the development of programs for synchronous models. This structure holds for various configurations described in Chapter Three and Four, all simulations have been completed in the 'C' programming language.

A1.3 Stochastic Automaton Algorithm

The algorithm was developed to obtain a closed-loop configuration for the basic stochastic automaton model. The specific steps for simulation of automaton-environment:

Procedures relevant for the Algorithm

STEP 1 Input and Initialisation

```
Set reward    = 0.01    (Reward Parameter)
Set punish    = 0.0     (Punish Parameter)
Set d1       = 0.6     (Penalty Probability for Selecting Action 1)
Set d2       = 0.1     (Penalty Probability for Selecting Action 2)
Set nexpts    = 2       (Total number of experiments)
Set itrial    = 1000    (Total number of iterations/expt)
Set ntrial    = 200     (Average sample path)
Set seedval
srand48(seedval)       (Initialise random number generator, range [0.0, 1.0])
```

STEP 2 For k=0, ..., (nexpts)

```
Set prob1     = 0.5
Set prob2     = (1.0-prob1)
Set β1      = 1
Set α1      = 1
Set flag      = 0
Set sump1     = 0.0
Set sump2     = 0.0

for j=0, ..., (itrial)
  for i=0, ..., (ntrial)
    /* Interconnect automata-environment to form closed loop configuration */
    (i) Set α1 = auto1() (Call function auto1())
    (ii) Set β1 = envir1() (Call function envir1())
    (iii) Evaluate action probabilities: prob1, prob2;
    if (α1=1)
      {Set prob1= lri_prob12() (Call function lri_prob12())
      Set prob2= lri_prob22() (Call function lri_prob22())
      else
      {Set prob2= lri_prob22() (Call function lri_prob22())
      Set prob1= lri_prob12() (Call function lri_prob12())
    /* Store probabilities in array to evaluate expected values */
    Set pr1[i][k] = prob1
    Set pr2[i][k] = prob2
    Compute sump1+=pr1[i][k]
    Compute sump2+=pr2[i][k]
    /* Reset flag */
    if (flag=0)
      inum=0
      Compute prb1[inum][k]+=prob1
      Compute prb2[inum][k]+=prob2
      flag=1
      Compute i++
      Compute j+=ntrial
      Compute inum++
```

```
Compute prb1[inum][k]=sump1/j
Compute prb2[inum][k]=sump2/j
Compute k++
```

STEP 3 Evaluate expected values

```
Reset itrial= 5
(i) For i=0, ..., (itrial)

Set sump1=0.0
Set sump2=0.0

for j=0, ..., (nexpts)

Compute sump1+=prb1[i][j]
Compute sump2+=prb2[i][j]
Compute j++
Compute prb1[i][j]=sump1/j
Compute prb2[i][j]=sump2/j
Compute i++
```

STEP 4 OUTPUT

```
For i=0, ..., (itrial)
Print i, prb1[i][nexpts], prb2[i][nexpts]

(Results are tabulated, action probabilities (p1, p2) at each stage n)
```

By considering the chart shown in Figure A1.1, this algorithm can be modified to implement the various synchronous models. Since synchronous models consider multiple automata-environment pairs then Step 2(i-iii) are composed of the relevant routines, auto1, env1, lri-prob12, lri-prob22, as depicted in Figure A1.2a - Figure A1.2d. Several routines may be interconnected, to achieve the desired configuration for the synchronous and sequential models.

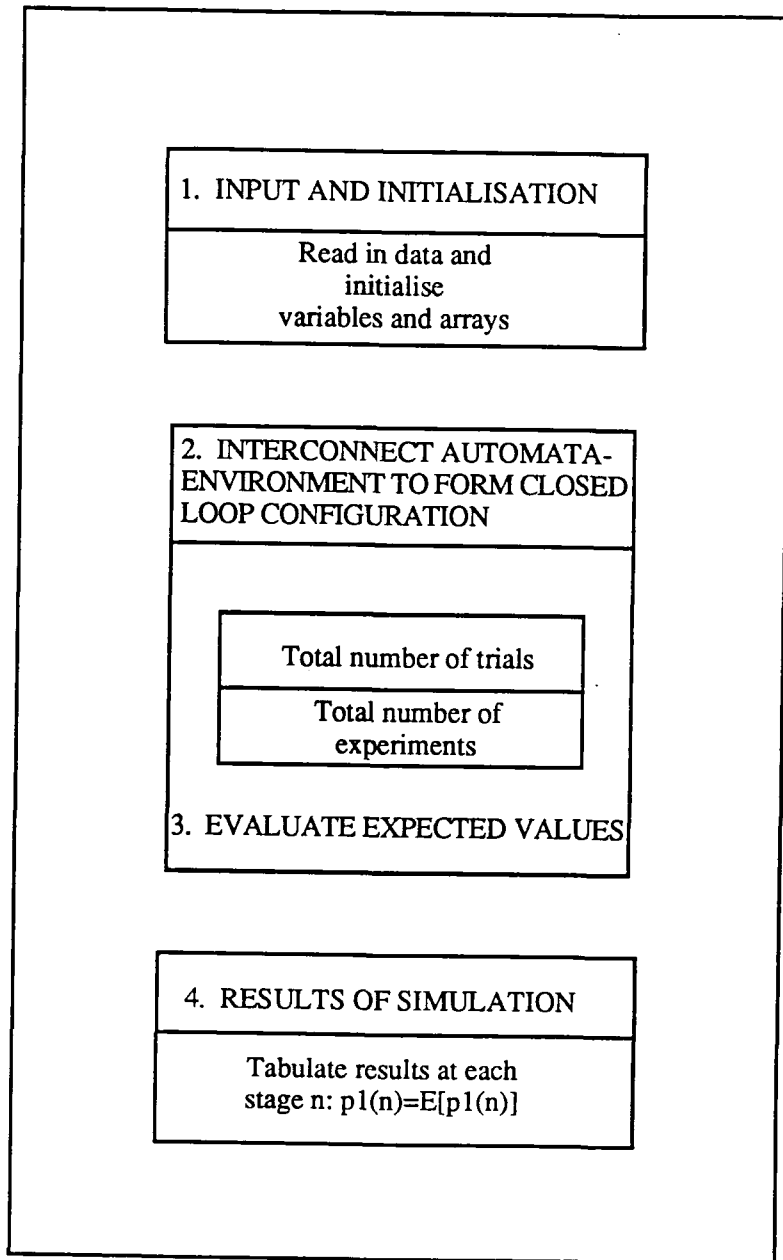


Figure A1.1 – Overall Structure of Program

```

INT AUTO1(ACTION1, PROB1)
*****
C PURPOSE
    INT AUTO1,DETERMINES THE ACTION SELECTED BY THE
    AUTOMATON

C METHOD
    INT AUTO1,GENERATES RANDOM NUMBER AND COMPARES WITH
    CORRESPONDING ACTION PROBABILITY TO GENERATE THE NEXT ACTION
    SELECTED BY THE AUTOMATON.

C HISTORY
    COPYRIGHT (C) 1994 :   Q.F.AHMED. UNIVERSITY OF DURHAM, SCHOOL OF
                          ENGINEERING AND COMPUTER SCIENCE,DURHAM,
                          DH1 3LE.

C ARGUMENT IN
    ACTION1  INTEGER, ACTION SELECTED BY AUTOMATON

    PROB1   REAL,PROBABILITY OF THE ACTION SELECTED BY AUTOMATON

C ARGUMENT OUT
    ACTION1  ON EXIT , CONTAINS THE NEXT ACTION SELECTED BY
    AUTOMATON

INT AUTO1(ACTION1,PROB1 ).
*****
    INTEGER    ACTION1
    REAL       PROB1,RAND

    GENERATE RANDOM NUMBER
    RAND = DRAND48( )
    IF (RAND <= PROB1) THEN
        SET ACTION1 = 1
    ELSE
        SET ACTION1 = 2
    RETURN (ACTION1)

```

(a) Routine Auto1()

Figure A1.2 – Programme Routines

```
INT ENVIR1(ACTION1, PENALTY1, PENALTY2, RESPONSE1)
*****
```

C PURPOSE

INT ENVIR1, DETERMINES THE RESPONSE FROM THE ENVIRONMENT, THE
 RESPONSE1=0 CORRESPONDS TO PUNISH,RESPONSE=1 CORRESPONDS TO A
 REWARD SIGNAL

C METHOD

INT ENVIR1,GENERATES RANDOM NUMBER AND EVALUATES THE
 RESPONSE FROM THE ENVIRONMENT

C HISTORY

COPYRIGHT (C) 1994 : Q.F.AHMED. UNIVERSITY OF DURHAM, SCHOOL OF
 ENGINEERING AND COMPUTER SCIENCE,DURHAM,
 DH1 3LE.

C ARGUMENT IN

ACTION1 INTEGER, ACTION SELECTED BY AUTOMATON

PENALTY1 REAL, PENALTY PROBABILITY CORRESPONDING TO AN
 ACTION(ACTION1) SELECTED BY THE AUTOMATON

PENALTY2 REAL, PENALTY PROBABILITY CORRESPONDING TO AN
 ACTION(ACTION2) SELECTED BY AUTOMATON

RESPONSE1 INTEGER,RESPONSE PROVIDED BY THE ENVIRONMENT

C ARGUMENT OUT

RESPONSE1 ON EXIT, PROVIDES THE RESPONSE FROM THE ENVIRONMENT
 REWARD/PUNISH SIGNAL

```
INT ENVIR1(ACTION1,PENALTY1,PENALTY2,RESPONSE1)
*****
```

```
INTEGER ACTION1,RESPONSE1
REAL PENALTY1,PENALTY2,RAND

GENERATE RANDOM NUMBER
RAND = DRAND48( )
IF (( ACTION1=1) AND (RAND <= PENALTY1)) THEN
  RESPONSE1 = 1
ELSE IF (( ACTION1=1) AND ( RAND > PENALTY1)) THEN
  RESPONSE1=0
ELSE IF (( ACTION1=2) AND ( RAND <= PENALTY2)) THEN
  RESPONSE1=1
ELSE
  RESPONSE1=0
RETURN (REPOSE1)
```

(b) Routine Envir1()

Figure A1.2 – Programme Routines

```

DOUBLE LRI_PROB12(TRIAL,ACTION1,RESPONSE1,PROB1,PROB2,REWARD,PUNISH)
*****
C PURPOSE
    DOUBLE LRI_PROB12, UPDATES THE ACTION PROBABILITY (PROB1)
    USING THE LRI SCHEME

C METHOD
    DOUBLE LRI-PROB12,GENERATES A RANDOM NUMBER TO DETERMINE
    THE UPDATED ACTION PROBABILITY BY USING THE LRI LEARNING
    ALGORITHM

C HISTORY
    COPYRIGHT (C) 1994 : Q.F.AHMED. UNIVERSITY OF DURHAM, SCHOOL OF
    ENGINEERING AND COMPUTER SCIENCE,DURHAM ,
    DH1 3LE.

C ARGUMENT IN
    TRIAL          INTEGER, NUMBER OF ITERATIONS
    ACTION1        INTEGER, ACTION SELECTED BY AUTOMATON
    RESPONSE1      INTEGER, RESPONSE FROM THE ENVIRONMENT
    PROB1          REAL, ACTION PROBABILITY FOR ACTION1
    PROB2          REAL, ACTION PROBABILITY FOR ACTION2
    REWARD         REAL, REWARD PARAMETER
    PUNISH         REAL, PUNISH PARAMETER

C ARGUMENT OUT
    PROB1          ON EXIT , THE UPDATED ACTION PROBABILITY FOR ACTION1

DOUBLE LRI_PROB12(TRIAL,ACTION1,RESPONSE1,PROB1,PROB2,REWARD,PUNISH)
*****
    INTEGER      ACTION1,RESPONSE1
    REAL         PROB1,PROB2,REWARD,PUNISH,RAND

    GENERATE RANDOM NUMBER
    RAND = DRAND48( )
    IF ( TRIAL > 0 ) THEN
    [IF (( RESPONSE = 1) AND (ACTION1 = 1)) THEN
        SET PROB1 = PROB1+REWARD*(1.0-PROB1)
    ELSE IF ((RESPONSE1=1) AND (ACTION=2) THEN
        SET PROB1 = (1.0-REWARD)* PROB1
    ELSE IF ((RESPONSE1=0) AND ((ACTION1=1) OR (ACTION1=2))) THEN
        SET PROB1 = (1.0-PUNISH)* PROB1]
    ELSE
        SET PROB1 = PROB1
    RETURN(PROB1)

```

(c) Routine Lri-prob12()

Figure A1.2 – Programme Routines

```

DOUBLE LRI_PROB22(TRIAL,ACTION1,RESPONSE1,PROB1,PROB2,REWARD,PUNISH)
*****
C PURPOSE
    DOUBLE LRI_PROB22, UPDATES THE ACTION PROBABILITY (PROB2)
    USING THE LRI SCHEME

C METHOD
    DOUBLE LRI-PROB22,GENERATES A RANDOM NUMBER TO DETERMINE
    THE UPDATED ACTION PROBABILITY BY USING THE LRI LEARNING
    ALGORITHM

C HISTORY
    COPYRIGHT (C) 1994 : Q.F.AHMED. UNIVERSITY OF DURHAM, SCHOOL OF
    ENGINEERING AND COMPUTER SCIENCE,DURHAM ,
    DH1 3LE.

C ARGUMENT IN
    TRIAL          INTEGER, NUMBER OF ITERATIONS
    ACTION1        INTEGER, ACTION SELECTED BY AUTOMATON
    RESPONSE1      INTEGER, RESPONSE FROM THE ENVIRONMENT
    PROB1          REAL, ACTION PROBABILITY FOR ACTION1
    PROB2          REAL, ACTION PROBABILITY FOR ACTION2
    REWARD         REAL, REWARD PARAMETER
    PUNISH         REAL, PUNISH PARAMETER

C ARGUMENT OUT
    PROB2          ON EXIT , THE UPDATED ACTION PROBABILITY FOR ACTION1

DOUBLE LRI_PROB12(TRIAL,ACTION1,RESPONSE1,PROB1,PROB2,REWARD,PUNISH)
*****
    INTEGER      ACTION1,RESPONSE1
    REAL         PROB1,PROB2,REWARD,PUNISH,RAND

    GENERATE RANDOM NUMBER
    RAND = DRAND48( )
    IF ( TRIAL > 0 ) THEN
    [IF (( RESPONSE = 1) AND (ACTION1 = 2)) THEN
        SET PROB2 = PROB2+REWARD*(1.0-PROB2)
    ELSE IF ((RESPONSE1=1) AND (ACTION=1) THEN
        SET PROB2 = (1.0-REWARD)* PROB2
    ELSE IF ((RESPONSE1=0) AND ((ACTION1=1) OR (ACTION1=2))) THEN
        SET PROB2 = (1.0-PUNISH)* PROB2]
    ELSE
        SET PROB2 = PROB2
    RETURN(PROB2)

```

(d) Routine Lri-prob22()

Figure A1.2 – Programme Routines

Appendix Two

Game Theoretic Concept

A2.1 Introduction

This Appendix introduces some of the terminology of game theory, providing formal definitions and basic concepts, [45], [46], which are relevant to this thesis. The theory of games originated at the end of the Second World War, such a concept involved modelling problems with two or more decision makers. To date this area of 'applicable mathematics' has continued to be one of the most active branches of research and development.

A2.2 What Is Game Theory?

Game theory is a method for the study of decision making in situations of conflict. It is a theoretical model that deals with human processes in which the individual decision maker is not in complete control of other decision makers entering into the environment. It describes conflicts of interest, cooperation or both between individuals, groups, formal or informal organisations or society. The theoretical models of such conflicts of interests between people or groups of people such as political parties, government organisations, generals engaged in fighting an enemy, a player in a poker game may all be viewed as a *game* situation. As such *game theory* consists of ways of analysing these problems. Game theory is a *normative*, not a *descriptive* theory. That is, it does not describe how actual people make

decisions in situations involving conflicts of interests; but rather it discovers how certain *rational players* can be expected to make decisions in such situations. In simple terms, game theory is not a prescriptive way of how to play a game, but rather it is a set of ideas and techniques for analysing these mathematical models of conflict of interest. The problems in game theory are complex, since it involves decision makers with different goals or objectives. Each individual is in a situation in which there are many possible outcomes with different values to them. The individuals may have some control which influence the outcome, but they do not have complete control over others. An individual must consider how to achieve as much as is possible, taking into account that there are others with different goals from his own and whose actions have an effect on all. Thus, it is necessary to adjust plans not only to his own desires and abilities but also to the desires and abilities of others. In its strict game theoretic sense, a game has the following features:

- (i) Any game consists of more than one decision maker, called a *player*. A player in a game is an autonomous decision making unit.
- (ii) At specified instances, one or more players must make decisions by choosing among a specified set of alternatives. The selected decision determines the resulting *situations* of the game. Thus, a *play* of a game is a sequence of situations.
- (iii) Each situation in turn determines which of the players is to make the next decision (whose 'move' it is) and the range of choices open to him. Note that certain specified situations define the end of the

particular play of the game.

- (iv) An *outcome* of the game may be defined as a situation in which a particular play of a game ends. At the end of each game each player receives a *payoff* eg. win, lose or draw. The payoffs represent gains or losses.
- (v) A *rational player* is one who, having taken into account all the information available to him by the rules of the games, makes his choices in such a way as to maximise the actual or the statistically expected payoff to accrue him in the outcome of the game.

Note that each player has to make decisions at some moves of the game. A *strategy* for a player can be defined generally as a plan of action containing instructions as to what to do in every contingency. Thus having selected a specific strategy it will enable him to adapt to situations that may arise, no matter what the outcomes of the chance events. Two types of strategies are of importance: a *mixed strategy* consists of performing a random experiment each time the game is played in order to choose which strategy to use that time. A strategy which does not involve this random experiment is called a *pure strategy*.

If the sum of the players' payoff is zero no matter what strategy they use, the game is called *zero-sum*. In these games, the players are completely opposed to one another in that, what one wins the other loses. Games that do not exhibit this property are called *non-zero-sum* games. Moreover, a game with two players, where a gain of one player *equals* a loss to the other

is known as *two-person zero-sum game*. In such a game, the outcomes may be expressed in terms of the payoff to one player.

A matrix is usually used to summarize the payoffs to the players whose strategies are given by the rows of the matrix. The definitions of a two-person zero-sum game may be considered by a coin-matching situation in which each of the players A and B selects a head (H) or a tail (T). If the outcomes match (ie. H and H, or T and T), player A wins 1.00 from B. Otherwise, A loses 1.00 to B. In this game each player has two strategies (H or T).

A2.3 Basic Definitions

Many general properties such as strategy dominance, uniqueness of the equilibrium point, and Pareto-optimality have been considered as useful features of an optimal solution, [44], [45]. These properties are relevant to discussions in Chapter Four, and they are described as follows:

Equilibrium Points

In an N-person game strategy N-tuple is said to be an *equilibrium point* if no player has a positive reason for changing his or her strategy, assuming that no other player changes his strategy. The outcome (also called payoff) corresponding to this set of strategies is called an *equilibrium outcome* (payoff). Thus, an equilibrium outcome is one from which neither player can change his strategy without impairing his payoff, assuming that the other player sticks to his strategy.

Dominant Strategy

A strategy dominates another when, independent of the action taken by the other players, the first strategy leads to an outcome as favourable as the second.

Pareto Optimal

In an N-person game, an outcome is said to be *Pareto optimal* if there is no other outcome in which all players simultaneously do better (receive larger payoff). It is possible for a (2x2) game to have one, two, three or four Pareto optimal payoffs.

Consider the following example of two-player games where each player has two strategies. Each game can be represented by a (2x2) matrix D whose elements are of the form (d_{ij}^1, d_{ij}^2) , where d_{ij}^1 is the payoff to player 1 and d_{ij}^2 the payoff to player 2 when they play strategies i and j , respectively. The payoff matrices of the three cases as follows:

$$D_1 = \begin{pmatrix} (0, 0) & (10, 7) \\ (0, 0) & (9, 8) \end{pmatrix} \quad D_3 = \begin{pmatrix} (5, 5) & (0, 10) \\ (10, 0) & (1, 1) \end{pmatrix}$$

$$D_2 = \begin{pmatrix} (10, 10) & (0, 5) \\ (5, 0) & (3, 3) \end{pmatrix} \quad D_4 = \begin{pmatrix} (1, -1) & (2, -2) \\ (3, -3) & (0, 0) \end{pmatrix}$$

Game D_1 , the strategy of the first player (row) and second strategy of the second player (column) are dominant, and (10, 7) is an equilibrium payoff.

It is also Pareto optimal. Game D_2 , $(10, 10)$ and $(3, 3)$ are both equilibrium payoffs, but only $(10, 10)$ is Pareto optimal. Game D_3 , both players have dominant strategies but the resulting outcome $(1, 1)$ which is an equilibrium payoff, is not Pareto optimal (an instance of Prisoner's Dilemma). Game D_4 , a zero-sum game, the payoff $(1, -1)$ corresponds to the first strategy of the two players is an equilibrium payoff and is called a *saddle point*.

Appendix Three

Petri Net Concepts

A3.1 Introduction

This section introduces some of the basic Petri net properties and terms which are normally used in the analysis of Petri nets, [15], [16].

A3.2 Some Petri Net Properties

Boundedness and Safe

A Petri net is said to be *k-bounded* or simply *bounded* if the number of tokens in each place does not exceed a finite number k for any marking reachable from M_0 , ie. the number of tokens in each place is $\leq k$ for all markings in $R(M_0)$. A Petri net is said to be *safe* if the number of tokens in each place is ≤ 1 for all markings in $R(M_0)$.

Boundedness is a very important practical property of Petri nets. For example places in a PN are used to represent buffers and registers for storing intermediate data. If the net is bounded or safe, it is guaranteed that there will be no overflows in the buffers or registers no matter what firing sequence is taken.

Liveness

The concept of liveness is very significant in the modelling of operating systems. Liveness is a property that ensures deadlock-free operation, such that a transition remains potentially fireable in all markings reachable from a given marking.

A Petri net is said to be *live* (or equivalently M_0 is said to be a *live-marking* for N) if, no matter what marking has been reached from M_0 , it is possible to ultimately fire *any* transition of the net by progressing through some further firing sequence. This property is ideal for many systems, but it is impractical and too costly to verify this property for large computer systems. Thus, a number of different levels of liveness have been considered.

A transition t in a Petri net is said to be:

- Dead (L0-Live) if t can never be fired in any firing sequence in $L(M_0)$;
- L1-Live (Potentially Fireable) if t can be fired at least once in some firing sequence in $L(M_0)$;
- L2-Live if, given any positive integer k , t can be fired at least k times in some firing sequence in $L(M_0)$;
- L3-Live if t can appear infinitely, often in some firing sequence in $L(M_0)$;
- L4-Live or Live if t is L1-Live for every marking M in $R(M_0)$.

It is necessary to state precisely the definition being used, since each definition is quite different.

A3.3 Reachability (Coverability) Tree Algorithm

To reduce a tree to finite form, it is necessary to find a means of limiting the new markings (*frontier nodes*) introduced at each step. During the construction of the reachability tree it is possible to find dead markings, ie. markings in which no transition is enabled and these markings are known as *terminal nodes*. In addition the expansion of the tree is stopped when a class of markings are reached that have previously appeared and have been considered, as they represent *duplicate (old)* nodes. No successor of a duplicate node need be considered; all these successors will be produced from the first occurrence of the marking in the tree. A final means to reduce the tree is by using a special symbol ω which can be thought of as infinity. The reachability (coverability) tree may now be precisely stated as follows:

Let ω be a symbol, such that:

$$\omega \pm n = \omega, \omega > n \text{ and } \omega \geq \omega$$

Coverability (Reachability) Tree Algorithm

The coverability tree for a PN is constructed by the algorithm presented in Figure A3.1. By adopting the procedure outlined in Figure A3.1 all frontier nodes which have not been processed by the algorithm are converted to terminal, duplicate or interior nodes. Once all nodes have been classified as terminal, duplicate or interior, the algorithm halts. The coverability tree is

an extremely useful tool for the analysis of PNs. The following outlines some of the properties that can be studied:

- A Petri net is *bounded* and thus $R(M_0)$ is finite if and only if ω does not appear in any node labels in the tree.
- A Petri net is *safe* if and only if only 0's and 1's appear in node labels in the tree.
- A transition t is *dead* if and only if it does not appear as an arc label in the tree.

- STEP 1** Label the initial marking M as the root and tag it 'new'
- STEP 2** While 'new' marking exist do the following:
- (2.1) Select a new marking M
 - (2.2) If M is identical to a marking on the path from the root M to M , then tag M to be 'OLD' and stop processing M (DUPLICATE NODE)
 - (2.3) If no transitions are enabled at M , tag M 'DEAD END' (TERMINAL NODE)
 - (2.4) While there exist enabled transitions at M , for every transition t enabled in M
 - (2.4.1) Obtain the marking M' that results from firing t at M
 - (2.4.2) If there exists a path from the root to M for which a marking M'' exists such that $M' \geq M''$ for each place M for which a marking M'' is coverable, then replace M' by ω for each p such that $M' > M''$
 - (2.4.3) Introduce M' as a node, draw an arc label t from M to M' and tag M' 'new'

Figure A3.1 – Coverability (Reachability) Tree Algorithm

Appendix Four

Models (Byzantine Generals)

A4.1 Introduction

Appendix Four is concerned with possible analytical models which may be used to study unreliable communication between decision modules. The Byzantine Generals problem, [69], [70], is proposed as an appropriate model in resolving this type of failure. The Oral Messages algorithm is presented, which is used to solve the Byzantine Generals problem for $3m+1$ or more generals in the presence of at most m traitors. The algorithm uses the majority function, to select the appropriate value. The Signed Messages algorithm has also been presented. Note that the unforgeable signed messages algorithm provides a solution to the Byzantine Generals problem for any number of generals and possible traitors.

A4.2 Reliable Systems

A method known to implement a reliable system is to use several different 'processors' to compute the same result, and perform a majority vote on their outputs to obtain a single result. The use of majority voting to achieve reliability is based on the assumption that all correctly functioning processors must produce the same output provided they use the same input value. In order for majority voting to yield a reliable system, the following *Interactive Consistency* conditions must be satisfied:

- (1) All nonfaulty processors must use the same input value (so that they produce the same output).
- (2) If the input unit is nonfaulty, then all nonfaulty processors use the value it provides as input (so that they produce the correct output).

Therefore in terms of reliable systems, the fundamental problem is the agreement on a piece of data based on the cooperation among several processors. Several solutions to this problem have been provided in relation to Byzantine Generals analogy rather than computer systems.

A4.3 Byzantine Generals Problem

Any reliable system must be able to cope with the failure of one or more of its components, and also malfunctions that send conflicting information to different parts of the system. It may be defined that a component 'fails' when it completely stops functioning, and the term 'malfunction' is related to a system if it continues to operate but performs one or more operations incorrectly. The Byzantine Generals approach resolves this type of failure, consider the following scenario:

- Several divisions of the Byzantine Army are camped outside an enemy city;
- Each division is commanded by its own general;
- The generals can only communicate by messenger;
- After observing the enemy they must decide upon a common plan of

action;

- Some of the generals may be traitors trying to prevent loyal generals from reaching agreement.

The generals follow an algorithm satisfying certain conditions, whose objectives are to reach agreement and follow a reasonable plan of action. The Byzantine Generals problem is restricted to considering how a commanding general sends an order to his lieutenants, such that the following conditions are fulfilled:

Condition IC1 – All loyal lieutenants obey the same order.

Condition IC2 – If the commanding general is loyal, then every loyal lieutenant obeys the order that he sends.

These are examples of Interactive Consistency conditions. Note that if the commanding general is loyal, then IC1 follows from IC2. However the commander need not necessarily be loyal. For a clear representation of the relationship with reliable systems the following notions are used:

The sender of messages in the Byzantine Generals notation is referred to as the *commander*; in terms of reliable systems it is considered as the *transmitter*, ie. the unit generating the input. A message that the commander sends carry its *value*. The commander sends its value to its *lieutenants* either directly or through other lieutenants called *relays*. A *lieutenant* can be a commander, a receiving lieutenant or a relay according to its function in the network with respect to a given message. A lieutenant is *loyal* if it transfers the messages it has received without altering or eavesdropping on them;

delaying the forwarding, sending conflicting values. A *loyal commander* is a reliable lieutenant that sends the same value to all its receiving lieutenants. It is assumed that a *traitor* is a lieutenant/ relay or a commander that is not loyal. In simple terms, the analogy described can be considered in terms of reliable systems, as follows:

Commander – represents the unit generating the input values, indicating a transmitter.

Lieutenants – represent the processors.

Loyal – relates to nonfaulty (correctly functioning) processors, that is a reliable processor; this implies that a traitor is an unreliable processor.

Therefore in terms of Byzantine Generals the fundamental problem is to find an algorithm to ensure that loyal generals reach agreement. The following sections present algorithms which ensure that loyal generals reach agreement and also guarantee interactive consistency conditions for (n, m) where n is the total number of generals of which it is known that m are traitors.

A4.3.1 Impossibility Results

This defines a formal model which states that, if the generals can send only oral messages, then no solution will work unless more than two-thirds of the generals are loyal. Impossibility Results deals with only three generals in the presence of a single traitor, and proves that it is impossible to assure interactive consistency for $n < (3m + 1)$ with $(m+1)$ rounds of information exchange. An oral message may be defined as follows:

An *Oral Message* – is one whose contents are completely under the control of the sender, so that a traitorous general can transmit any possible message. For simplicity, it is assumed that the only possible decisions that can be taken by a commander are ‘attack’ or ‘retreat’. Figure A4.1a illustrates the case in which the commander is loyal and sends an ‘attack’ order to both Lieutenants, but Lieutenant 2 is a traitor and he reports to Lieutenant 1 that he received a ‘retreat’ order. The receiving Lieutenant 1 has to consider two possibilities: ie. the commander is loyal and Lieutenant 2 is a traitor; or the commander is a traitor and Lieutenant 2 is loyal. For condition IC2 to be satisfied Lieutenant 1 assumes that the commander is loyal and he must obey the order to ‘attack’, which shows that the first case is correct.

Consider another scenario, Figure A4.1b, in which the commander is a traitor and sends an ‘attack’ order to Lieutenant 1 and a ‘retreat’ order to a Lieutenant 2. Similarly, Lieutenant 1 encounters the same problem as above; he does not know who the traitor is, and cannot tell what message the commander actually sent to Lieutenant 2. If the traitor lies consistently, Lieutenant 1 cannot distinguish between these situations. Therefore, whenever Lieutenant 1 receives an ‘attack’ order from the commander he must obey it. By applying a similar argument in the case of Lieutenant 2; if he receives a ‘retreat’ order from the commander, then he must obey it even if Lieutenant 1 tells him that the commander said ‘attack’. Analysing Figure A4.1b –

Lieutenant 1 : obeys 'attack' order; while

Lieutenant 2 : obeys 'retreat' order

Thereby violating condition IC1 – all loyal Lieutenants obey the same order. This proves that no matter what decision the lieutenants make, no solution exists for three generals that work in the presence of a single traitor, formal proof has been included in, [69].

A4.3.2 Solution with Oral Messages

This provides a solution to the Byzantine Generals problem, that works for $(3m+1)$ or more generals in the presence of at most m traitors. In this case an algorithm is presented that acquires an extension of oral messages definition, based on the following assumptions:

Definition of Oral Messages

- (A1) : Every message that is sent is delivered correctly.
- (A2) : The receiver of a message knows who sent it.
- (A3) : The absence of a message can be detected.

Assumptions A1 and A2 prevent a traitor from interfering with the communication between two generals. Since, for assumption A1 a traitor cannot interfere with the messages that are sent; and in assumption A2 a traitor cannot confuse their interaction by introducing erroneous messages. Finally, assumption A3 stops a traitor who tries to prevent a decision being reached by simply not sending messages.

Each general executes some algorithm that involves sending messages to the other generals, it is assumed that a loyal general correctly executes his algorithm. The oral messages algorithm requires that each general be able to send messages directly to every other general. Note that a traitorous commander may decide not to send any order. In such a case, since the lieutenants must obey some order, they require some default order to obey. Hence, *RETREAT* may be considered as this default order. The Oral Messages algorithm is provided in Appendix (A4.4).

The Oral Messages algorithm $OM(m)$ may be defined inductively for all nonnegative integers m , by which the commander sends an order to his $(n-1)$ lieutenants. The procedure consists of an exchange of messages. In the $OM(m)$ algorithm two phases of information exchange are required. For the first phase the lieutenants exchange their private values. In the second round they exchange the results obtained in the first round. If a traitor exists he may lie consistently, or refuse to send messages. For simplicity this algorithm is described in terms of the lieutenants 'obtaining a value' rather than 'obeying an order'.

Figure A4.2a illustrates the messages received by Lieutenant 2 when the commander sends the value v to all three lieutenants and Lieutenant 3 is a traitor, in this case $m = 1$, and $n = 4$. By applying the Oral Messages algorithm, the first phase of Oral Messages algorithm $OM(1)$, the commander sends the order v to all three lieutenants. In the second phase by using the trivial algorithm $OM(0)$, Lieutenant 1 sends the value v to Lieutenant 2; also the traitorous Lieutenant 3 sends Lieutenant 2 some other value x . In the

final phase, the Oral Messages algorithm applies a Majority Function to the input values received by Lieutenant 2. Thus, Lieutenant 2 has $v_1 = v_2 = v$, $v_3 = x$ so that he obtains the correct order majority $v = \text{majority}(v, v, x)$.

Now consider the case when the commander is a traitor. Figure A4.2b shows the values received by the lieutenants if a traitorous commander sends three arbitrary values x , y and z to the lieutenants. Similarly, applying the Oral messages algorithm. The end result indicates that each lieutenant obtains the same value $\text{majority}(x, y, z)$ in the final step of Oral Messages algorithm, regardless of whether or not any of the three values x , y and z are equal.

A4.3.3 Solution with Signed Messages

This solution contradicts Impossibility results, since it restricts the traitors ability to lie, by allowing the generals to send unforgeable signed messages. By introducing this restriction, the Byzantine Generals problem becomes easier to resolve. In addition to the assumptions A1-A3 the following may be included:

Assumption (A4.a) :

A loyal general's signature cannot be forged, and any alteration of the contents of his signed messages can be detected.

Assumption (A4.b) :

Anyone can verify the authenticity of a general's signature.

In the signed messages algorithm, the commander signs the order that he

wants to send to each of his lieutenants. Each receiving lieutenant then adds his signature to that order and sends it to the other lieutenants, who add their signatures and send it to others, and so on. The receiver of the forwarded order can determine the true value sent by the commander, as well as the true identity of the sender. It is necessary that all loyal lieutenants receive exactly the same list of values, say v_1, \dots, v_q , or else they may obtain different values in the final step. The notation used in the algorithm is outlined as follows, let $v_1 : 0$ denote the value signed by General 0. Thus, $v_1 : 0 : L_i$ denotes the value v_1 signed by General 0, and then that value $v_1 : 0$ signed by Lieutenant L_i . Let General 0 be the commander.

Figure A4.3 depicts the case for three generals, when the commander is a traitor – illustrating Algorithm SM(1): In the first phase of signed messages, the commander is a traitor, sending an ‘attack’ order to Lieutenant 1 and ‘retreat’ order to Lieutenant 2. For the second phase of signed messages: Both lieutenants receive their orders, add their signature to that order and send it to each other.

The algorithm guarantees agreement as defined by conditions IC1 and IC2, even if there are very few loyal lieutenants. Observe here that unlike Impossibility Results, the lieutenants know the commander is a traitor because his signature appears on two different orders, and assumption A4 states that only he could have generated those signatures. It is shown that with unforgeable signed messages the problem is solvable for any number of generals and possible traitors.

A4.4 Byzantine Generals Algorithm

Oral Messages Algorithm

Algorithm OM(0); m=0

- (1) The commander sends his value to every lieutenant.
- (2) Each lieutenant uses the value he receives from the commander or uses the value RETREAT if he receives no value.

Algorithm OM(m); m > 0

- (1) The commander sends his value to every lieutenant.
- (2) For each i , let v_i be the value lieutenant i receives from the commander; or else be RETREAT if he receives no value. Lieutenant i acts as the commander in *Algorithm OM(m-1)* to send the value v_i to each of the $(n-2)$ lieutenants.
- (3) For each i , and each $j \neq i$, let v_j be the value Lieutenant i received from Lieutenant j in step (2) (using *Algorithm OM(m-1)*) or else RETREAT if he received no such value. Lieutenant i uses the value $\text{majority}(v_1, \dots, v_{n-1})$.

Once exchange of information is completed, the algorithm assumes a function *majority* which is used by each lieutenant for deciding what the value is, given a set of received values. The function must have the property that if a majority of the values v_i equal v , then $\text{majority}(v_1, \dots, v_{n-1})$ equals v . However, if a majority value among the v_i does not exist, a default value such as RETREAT is used.

Signed Messages Algorithm

Algorithm SM(m)

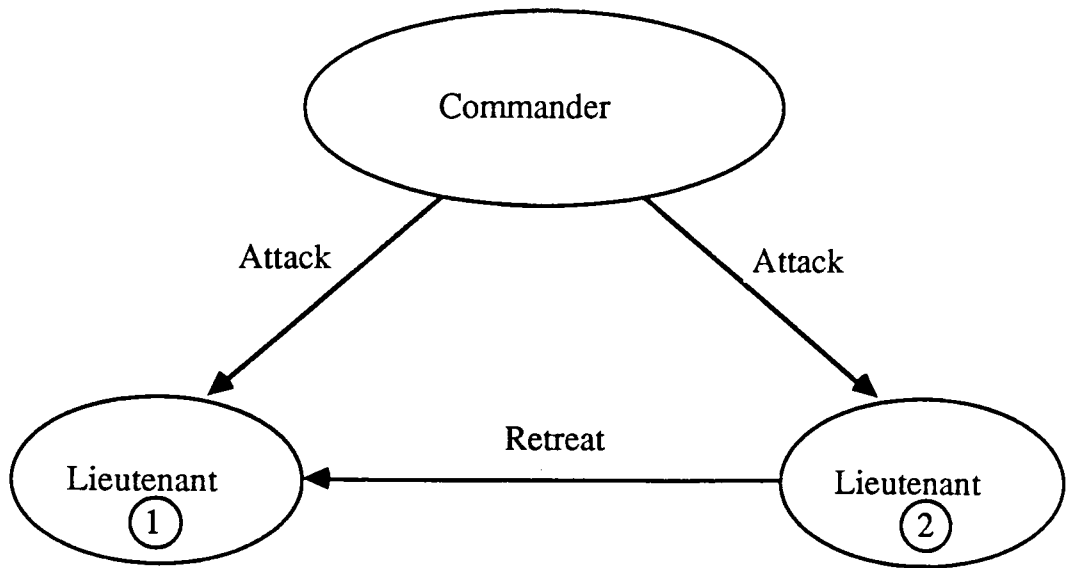
Initially $V_i = 0$.

- (1) The commander signs and sends his value to every lieutenant.
- (2) For each i :
 - (a) If Lieutenant i receives a message of the form $v : 0$ from the commander at phase 0, and he has not yet received any order, then
 - (i) he lets V_i equal v ;
 - (ii) he sends the message $v : 0 : i$ to every other lieutenant.
 - (b) If Lieutenant i receives a message of the form $v : 0 : L_1 : \dots : L_k$ at k , $1 \leq k \leq m$, V_i contains at most one value, v is not in the set V_i , and the signatures belong to the different lieutenants, then:
 - (i) he adds v to V_i ;
 - (ii) if $k < m$, then he sends the message $v : 0 : L_1 : \dots : L_k : i$ to every lieutenant other than L_1, \dots, L_k .

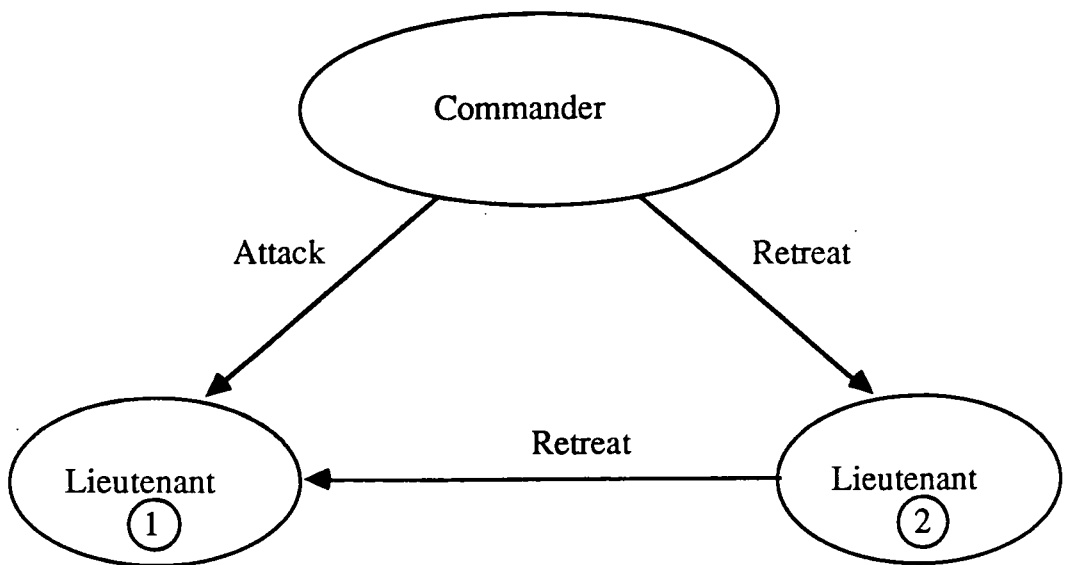
- (3) For each i :

When Lieutenant i will receive no more messages, at the end of phase m he obeys the order $choice(V_i)$.

For this particular algorithm $(m+1)$ rounds of information exchange are required. In summary in step (2) lieutenant L_i ignores any message containing an order v that is already in the set V_i , and accepts at most two different orders originated by the commander. In addition lieutenant L_i ignores any messages that do not have the correct form, followed by a string of different signatures.

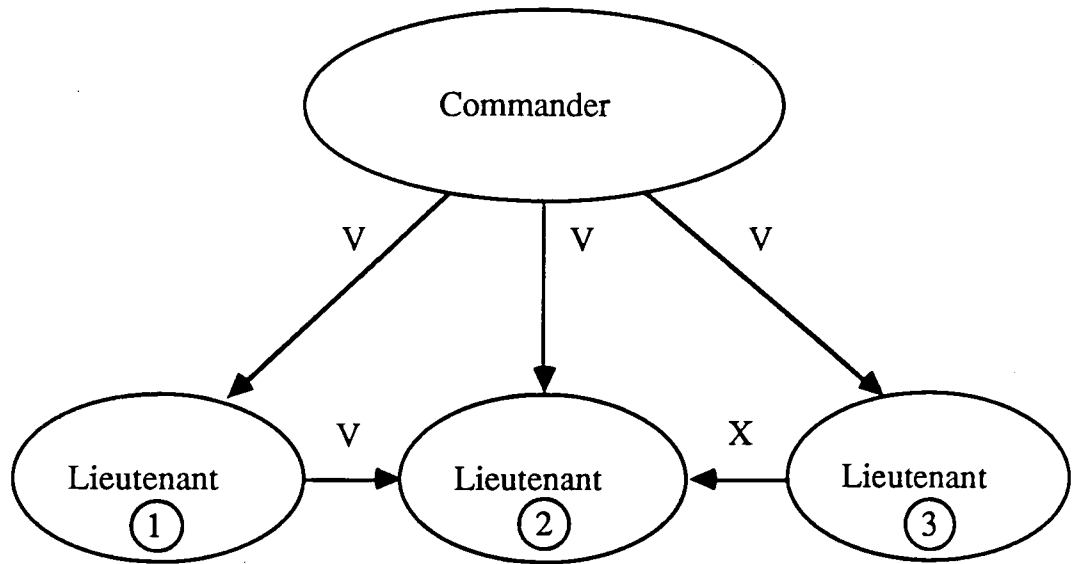


(a) Lieutenant 2 a traitor

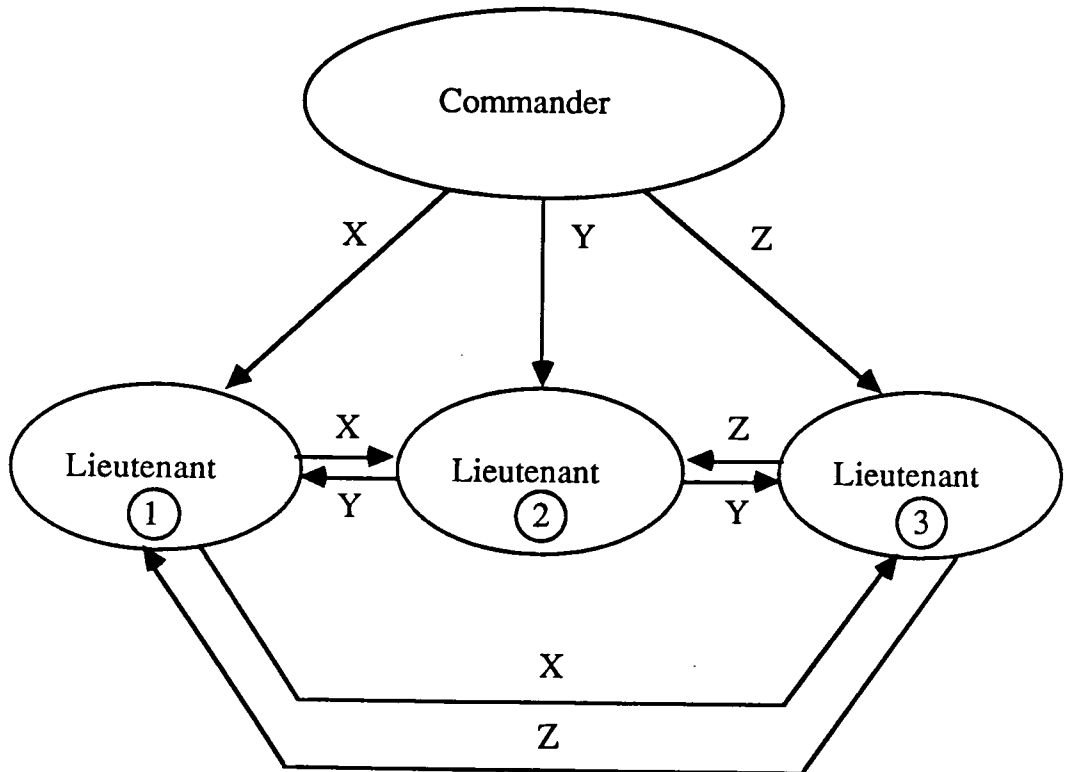


(b) Commander a traitor

Figure A4.1 - Impossibility Results



(a) Lieutenant 3 a traitor



(b) Commander a traitor

Figure A4.2 - Solution with Oral Messages

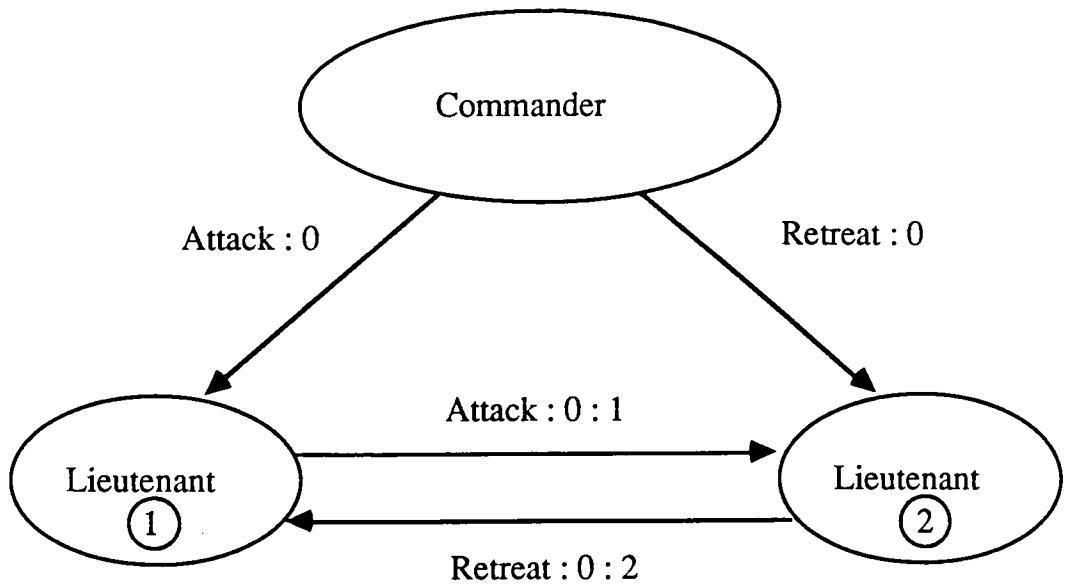


Figure A4.3 - Solution with Signed Messages - Commander a traitor

Appendix Five

Publications

APPLICATIONS OF ARTIFICIAL INTELLIGENCE TECHNIQUES TO DECENTRALISED DECISION MAKING IN C³-MIS

P. Mars, Q. F. Ahmed and P. Edwards

IEE Third International Conference on Command,
Control, Communications and Management Information Systems

Bournemouth, UK., 2nd-4th May 1989

APPLICATION OF STOCHASTIC LEARNING PETRI NETS TO SMALL-SCALE DISTRIBUTED DECISION MAKING ORGANISATIONS

Q. F. Ahmed and P. Mars

IMA Sixth International Conference on Control:

Modelling, Computation, Information

Manchester, UK., 2nd-4th September 1992

APPLICATION OF ARTIFICIAL INTELLIGENCE TECHNIQUES TO DECENTRALISED DECISION MAKING IN C³-MIS

P. Mars

Q.F.Ahmed

P. Edwards

University of Durham,UK

University of Durham,UK

British Aerospace, Warton, UK

1. INTRODUCTION

The analysis and design of complex, survivable, and responsive C³-MIS requires novel advances in the area of distributed decision making under uncertainty. In particular, systems engineering tools are needed for describing, decomposing and analysing such systems which must meet very demanding performance, survivability and response specifications. This paper is a first step in a research effort to develop the required theoretical and algorithmic tools for the systematic analysis and design of C³-MIS systems. Such systems are characterised by a high degree of complexity. Key features are a distribution of decision making processes amongst several decision making 'agents', the need for reliable operation in the presence of multiple failures, and inevitable interaction of humans with computer based decision support systems, (1),(2),(3).

At present analysis and synthesis studies for C³-MIS architectures tends to be performed in an ad-hoc manner. It is essential to develop quantitative methodologies, theories and algorithms relevant to C³-MIS architectures. Totally centralised hierarchical structures although efficient in resource allocation are highly vulnerable and introduce unacceptable delays. At the other extreme autonomous systems involving minimal delays are inefficient in the utilisation of scarce resources. Clearly the design compromise is in the use of distributed systems architectures combining distributed agents with communications capability.

In this paper we consider the use of a stochastic learning automata approach for both the adaptive control and modelling of decentralised decision making in C³-MIS.

2. BASIC THEORY

An extensive literature and a well established mathematical foundation now exists for stochastic learning automata. Early work in the context of mathematical psychology, (4), (5), (6), (7), was followed by major research efforts in both Russia, (8), and the USA, (9), (10). Hardware implementations and applications in process control and communication networks has also been considered, (11).

In general, a learning automaton may be defined as an element which interacts with a random environment in such a manner as to

improve a specified overall performance by changing its action probabilities dependent on responses received from the environment, Figure 1 shows the basic model. An automaton is a quintuple $(\beta, \psi, \alpha, F, G)$ where $\beta = (0,1)$ is the input set (output from the environment), $\psi = (\psi_1, \psi_2, \dots, \psi_r)$ is a finite state, $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_r)$ the output action set (inputs to the environment), $F: \psi \times \beta \rightarrow \psi$ is a state transition mapping and $G: \psi \rightarrow \alpha$ is the output mapping.

We restrict our attention to variable structure automata described by the triple (β, T, α) . Here T denotes the rule by which the automaton updates the probability of selecting certain actions. At stage n assuming r actions each selected with probability $p_i(n) [i = 1, \dots, r]$ we have:

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

A binary random environment is defined by a finite set of inputs $\alpha = [\alpha_1, \dots, \alpha_r]$ (outputs from the automaton), an output set $\beta = [0,1]$ and a set of penalty probabilities $c = [c_1, c_2, \dots, c_r]$. The output $\beta(n) = 0$ at stage n is called a favourable response (success) and $\beta(n) = 1$ an unfavourable response (failure). The penalty probabilities are defined as:

$$c_i = \Pr[\beta(n) = 1 / \alpha(n) = \alpha_i] \dots \dots \dots (2)$$

Apart from binary environment models other possible environments have included Q models (finite number of outputs) and S models (continuous outputs in range 0 to 1). In practice the choice of environmental models is obviously dictated by the particular application. If the penalty probabilities from the environment do not depend on stage number n, the environment is classified as stationary; otherwise the environment is non-stationary. Important convergence results have been proved for both types of environment, (12).

The convergence characteristics of learning automata are dependent on the properties of the algorithm used in the updating scheme. A performance measure that has been extensively used is the updated penalty that the automaton receives from the environment defined as:

$$M(n) = E[\beta(n)/p(n)] \dots \dots \dots (3)$$

Assuming a stationary environment and an automata selecting actions with equal probability the average received probability

is M_0 where

$$M_0 = (c_1 + c_2 + \dots + c_r)/r \dots \dots \dots (4)$$

A learning automaton is said to be expedient if:

$$\lim_{n \rightarrow \infty} E[M(n)] < M_0 \dots \dots \dots (5)$$

and optimal if:

$$\lim_{n \rightarrow \infty} E[M(n)] = \min_i [c_i] \dots \dots \dots (6)$$

Both linear and non-linear forms of updating algorithms have been considered. The most widely used are the class of linear algorithms which include linear reward/penalty (L_{RP}), linear reward/ ϵ penalty (L_{R-p}), and linear reward/inaction schemes (L_{RI}). For the L_{RP} scheme if an automaton tries an action α_i which results in success $p_i(n)$ is increased and all other $p_j(n) (j \neq i)$ are decreased. Similarly if action α_i produces a penalty response $p_i(n)$ is decreased and all other $p_j(n)$ modified to preserve the probability measure. An L_{RI} scheme ignores penalty responses from the environment and L_{R-p} only involves small changes in $p_i(n)$ compared with changes based on success.

In practice for adaptive optimisation problems the L_{R-p} scheme is preferred. The retention of a small element of penalty avoids the possibility of locking on uniquely to one specific action. For r actions and a binary environment the learning algorithm for L_{R-p} is described as follows:

For $\alpha(n) = \alpha_i$ and $\beta(n) = 0$ (reward)

$$p_i(n+1) = p_i(n) + a[1 - p_i(n)] \dots \dots (7)$$

$$p_j(n+1) = (1-a)p_j(n) \dots \dots \dots (8)$$

For $\alpha(n) = \alpha_i$ and $\beta(n) = 1$ (penalty)

$$p_i(n+1) = (1-b)p_i(n) \dots \dots \dots (9)$$

$$p_j(n+1) = \frac{b}{r-1} + (1-b)p_j(n) \dots \dots (10)$$

Here learning parameters a and b are both within the range $(0,1)$. In the case of L_{RP} automata $a = b$ and for L_{RI} $b = 0$.

So far we have considered models of a single decision maker (an automaton) interacting with an uncertain environment. We now extend the discussion to consider multiple decision makers interacting with an uncertain environment. It is considered that such models will provide a first basis for developing analytical models of C^3 -MIS systems. Of particular interest is that the outstanding practical application of learning automata to date is in adaptive control of routing information in circuit and packet switched networks. This is a classic example of decentralised control in an uncertain environment, (13).

3. DECENTRALISED CONTROL IN COMMUNICATION NETWORKS

The next decade will witness an increasing need for new and sophisticated methods for the optimal utilisation of capacity in communication networks. The importance of the study of routing and flow control is rapidly increasing. Future integrated services digital networks will incorporate a spectrum of traffic ranging from simple transaction measurements (100 bits) to multimegabit messages associated with colour facsimile. The use of learning algorithms is considered to represent a highly promising approach to the adaptive control of such complex systems. Recently advances have been made in applying these principles to the problem of adaptive routing in communication networks. Based on established theory, these applications show promise of practical solutions to the complex problems of routing and flow control and provide incentive for further exploration of learning techniques. Initial research considered circuit switched networks (telephone networks) in which learning algorithms at the network nodes update their strategies for routing traffic on the basis of success or failure in completing calls (14), (15). Recent research has focussed on packet-switched networks with learning automata schemes proposed for both virtual call and datagram networks. Packets are routed by automata selecting suitable outgoing links, the delay experienced by a packet being feedback to update the future selection strategy (10-18).

From a practical standpoint, the simplicity of the feedback as well as the updating schemes which exploit existing control mechanisms and protocols make the learning approach a practically viable alternative for routing in both circuit and packet-switched communication networks.

Circuit-Switched Networks

In a previous paper (14) simulation studies of telephone traffic routing in simple networks was considered. Specifically it was shown that a Linear Reward Inaction (L_{RI}) automaton scheme, when used in a simple network for call routing, performs at least as well as the optimum Fixed Rule (FR). The L_{RI} and Linear Reward Penalty (L_{R-p}) schemes were compared to FR. It was concluded that both routing strategies always perform as well as the optimum FR while in simulations requiring mixed routing strategies they give superior performance. An internal report (19) has investigated dynamic routing of fully connected circuit switched networks. The routing policy used is Least Busy Alternative (LBA) with Trunk Reservation (TR). It was concluded that LBA with TR is as good as FR yet with the advantage of flexibility and spreading out of local overload. Subsequent research has compared LBA with random routing

(RL), fixed routing (FR) learning automata and a dynamic alternative routing strategy (DAR), (20), for a five node fully connected network (21). It has been shown that DAR and L_{R-I} algorithms provide the best dynamic routing strategy but under conditions of network failure (e.g. link failure) the additional intelligence associated with the learning algorithms leads to significantly improved performance.

Packet-Switched Networks

Although adaptive or dynamic routing in packet networks is undoubtedly needed under network failure conditions (e.g. link or node failure) controversy exists on whether dynamic routing should be used under normal operating conditions. Recent work has shown that dynamic routing can in fact reduce throughput or increase delay as the network load is increased. Dynamic routing only improves network performance over an intermediate traffic range. This is intuitively obvious since as the network load increases less spare capacity is available. It should be noted that this possible increase throughput at moderate loads may of course defer the entry of the network into a high load condition. There is clearly a need for adaptive control of the dynamic routing mechanism such that at high loads the routing strategy reverts to minimum resource routing (i.e. fixed paths). In general a deterministic strategy is the best for balanced traffic but a dynamic strategy is essential for unbalanced and chaotic conditions.

The development of realistic analytical models for dynamic routing in packet networks is notoriously difficult. The fundamental problem is that route selections are by definition state dependant which negates the mathematically convenient property of separability. Recent work has used the theory of stochastic learning automata to calculate mean routing probabilities (22). These probabilities may be used as an approximation for link loadings and thus aid network dimensioning.

In a classic paper the optimal static routing problem was formulated as a convex programming problem in the space of routing variables (23). Necessary and sufficient conditions were determined for the problem solution. The basic result is that optimal static routing is obtained by an equalisation of the differential delays observed by a node on outgoing paths. Previous work has demonstrated that learning automata [type L_{R-p}] reach a steady state condition such that delays (as opposed to differential delays) are equalised (17).

Recently a decentralised non-linear technique has been described which permits a computation of the equilibrium solution for learning automata under steady state conditions. By

appropriate modification the recursion can be used to compute the system optimal (equalisation of differential delays) routing pattern (24). This is an important contribution which in addition to providing a cost effective alternative to simulation provides a benchmark which may be used in comparison studies of adaptive routing schemes. In addition, using a 10 node network previously studied by the present authors group no significant difference in delay performance was obtained between the automata and the optimal routing strategy. This important result confirms previous conclusions that for practical networks above a threshold of complexity the automata performance is virtually optimal. For small (3 node) asymmetric networks the performance of the automata is sub-optimal. There could be some averaging process involved such that above a given level of system complexity equalisation of delays is virtually equivalent to equalisation of differential delays. Clearly further research is needed in this area but at this stage the viability of the learning approach has been confirmed and for a general class of networks the automata provide close to an optimal strategy.

4. MODELS OF DECENTRALISED DECISION MAKING

Game theoretic issues provide a fundamental basis for the study of decentralised decision processes. Figure 2 shows the basic multiple automata game. With N automata $A_i (i = 1, \dots, N)$ interacting through a stationary¹ environment. At each stage n the automata select one of their actions and this determines the distribution of the random process involved. It should be noted that in contrast to the usual game-theoretic formulation, no player is aware of the other players, the actions selected by or the responses from the environment to other players. Extensive research has considered two-person zero sum games when the game matrix is unknown. For the identical payoff game optimal strategies are the same for the individuals and the group. In this case important convergence properties have been proved, (9), (10), (25). However in practice the merging of individual and group rationality is difficult. For the present work we seek models in which decision makers are not autonomous and their decisions affect each other. In this area initial work has involved synchronous models in which the time instants for automata actions and updates are synchronised, and sequential models which are asynchronous. Some of the simple synchronised models can be analysed by game theoretic concepts, (26), (27). The sequential models are more realistic in the practical sense and the power of this approach has been illustrated by a demonstration of the optimal control of a Markov chain with unknown transition and reward probabilities, (28).

It is considered that the stochastic learning automata approach will provide the fundamental

framework for a model of decentralised decision making in C^3 -MIS. As the research progresses additional layers of sophistication can be incorporated within the basic model. Initially work has considered simple topologies of synchronous, sequential and hybrid (synchronous/sequential) systems and future analytical work will be supported by simulations using a SUN-based interactive distributed decision model, (29). This simulator will be modelled and have sufficient flexibility to permit choices of topology, learning algorithms, level of communication reliability and selection of environmental models.

5. CONCLUSIONS

This paper has considered the application of stochastic learning automata to the problems of adaptive control and decentralised decision making in C^3 -MIS. The advantages of applying learning automata to either circuit or packet-switched networks may be summarised as follows:

- (i) Learning automata are based on simple principles and a well established mathematical theory.
- (ii) They are computationally attractive, i.e. only simple arithmetic operations are involved.
- (iii) Automata can be used individually without being dependent on other automata.
- (iv) They are cost-effective since they require minimum alteration to existing protocols.

Although at an early stage the study of learning automata as distributed "agents" in decentralised decision making is considered to represent a promising approach to providing a conceptual framework for modelling decision making in complex C^3 -MIS.

6. REFERENCES

1. Athans, M., 1987, "Command and control theory: A challenge to control science", IEEE Trans.on Automatic Control, AC-32, No.4, 286-293.
2. Andriole, S.J., and Halpin, S.M., 1986, "Information technology for command and control", IEEE Trans.Syst.Man and Cybern., SMC-16, No.6, 762-765.
3. Stephenson, H.E., and Sage, A.P., 1987, "Perspectives on imperfect information processing", IEEE Trans.Syst.Man and Cybern., SMC-17, No.5, 780-798.
4. Bush, R.R., and Mosteller, F., 1958, "Stochastic models for learning", John Wiley.
5. Atkinson, R.C., Bower, G.H., and Crothers, E.J., 1985, "An introduction to mathematical learning theory", John Wiley.
6. Iosifescu, M., and Theodorescu, R., 1969, "Random processes and learning", Springer.
7. Norman, M.F., 1972, "Markov processes and learning models", Academic Press.
8. Tsetlin, M.L., 1973, "Automaton theory and modelling of biological systems", 1973, Academic Press.
9. Narendra, K.S., and Thathachar, M.A.L., 1974, "Learning automata - a survey", IEEE Trans.Syst.Man and Cybern, SMC-4, 323-334.
10. Narendra, K.S., 1977, "Special volume on learning automata", J.Cybern and Inf.Sci., Vol.1, No.2.
11. Mars, P., and Poppelbaum, W.J., 1981, "Stochastic and deterministic averaging processors", IEE Research Monograph, Peter Peregrinus.
12. Srikantakumar, P.R., and Narendra, K.S., 1982, "A learning model for routing in telephone networks", SIAM.J.Control and Optimiz.
13. Mars, P., and Narendra, K.S., 1987, "Routing flow control and learning2 algorithms", IEE First Int.Conf. on Telecom Networks", 78-83.
14. Narendra, K.S., and Mars, P., 1983, "The use of learning algorithms in telephone traffic routing - a methodology", Automatica, 19, 5, 495-502.
15. Narendra, K.S., and Mars, P., 1981, "A study of telephone traffic routing using learning algorithms", IEEE Conf. on Comm., Denver, Colorado.
16. Chrystall, M.S., and Mars, P., 1982, "Simulation study of switched circuit communication networks using learning automata routing", Trans.IMACS, XXIV, 281-287.
17. Mars, P., and Chrystall, M.S., 1981, "Adaptive routing in computer communications networks", Proc.IEEE Int.Telecommunications Conf., New Orleans, A3.2.1-A3.2.7.

18. Mars, P., Narendra, K.S., and Chrystall, M.S., 1983, "Learning automata control of computer communication networks", Proc. of 3rd Yale Workshop on Applications of Adaptive Systems Theory, 114-119.
19. Ng, W.Y., 1985, "Dynamic routing in circuit-switched networks", Cambridge Univ. (Dept. of Pure Maths and Statistics) Project Report.
20. Kelly, F.B., 1985, "Blocking probabilities in large circuit-switched networks", Adv. Appl. Probability 18.
21. Eshragh, N., 1986, "Simulation studies of telephone traffic routing", Internal Research Report, School of Eng. and Applied Science, Univ. of Durham. 22. Whitehead, M.J., 1983, "An analytical model of a class of adaptive virtual call routing procedures, BTTJ, 1, 2, 28-36.
23. Gallager, R.G., 1977, "A minimum delay2 routing algorithm using distributed computation", IEEE Trans. on Comm., COM-25,1.
24. Mason, L.G., 1985, "Equilibrium flows, routing patterns and algorithms for store-and-forward networks", J. Large Scale Systems, 8.
25. Narendra, K.S., and Wheeler, R.M., 1983, "An N-player sequential stochastic game with identical payoffs", IEEE Trans. Syst. Man and Cybern., SMC-13, No. 1154.
26. Wheeler, R.M., and Narendra, K.S., 1985, "Learning models for decentralised decision making", Automation, Vol. 21, No. 4, 479-484.
27. Wheeler, R.M., 1985, "Decentralised learning in games and finite Markov Chains", Ph.D. Thesis, Dept. of Elect. Eng., Yale Univ.
28. Wheeler, R.M., and Narendra, K.S., 1986, "Decentralised learning in finite Markov Chains", IEEE Trans. Aut. Control, AC-31, No. 6, 519-526.
29. Ahmed, Q.F., 1988, "Decentralised decision making in C^3 -I Systems, Int. Res. Report, Univ. of Durham.

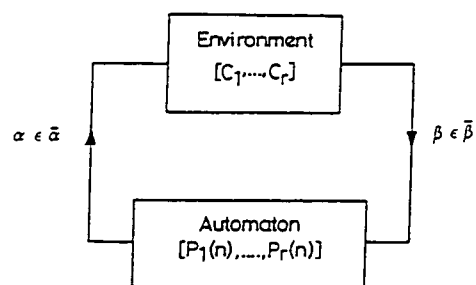


Figure 1. - Stochastic Learning Automaton Model

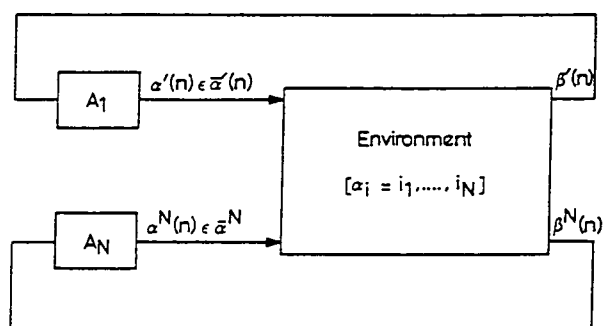


Figure 2. - Automata Game Schematic

Application of Stochastic Learning Petri Nets to Small-Scale Distributed Decision Making Organisations

Q F Ahmed and P Mars
School of Engineering and Computer Science
University of Durham
South Road
Durham

Abstract

A new class of Petri nets, namely, *Stochastic Learning Petri nets* are introduced as a powerful modelling tool for decision making organisations in complex systems. This extension to stochastic Petri nets has developed a model which has the additional feature of an embedded stochastic learning automata. This novel idea provides an artificial intelligence (AI) based decision making process embedded within Petri nets. An example application of this modelling tool is presented to demonstrate the impact that the use of an AI technique embedded within Petri nets can have on the performance of decision making organisations.

1 Introduction

This paper presents new developments in the study of distributed decision making in C³I (Command/Control/Communication/Intelligent) systems. The analysis and design of complex, survivable and responsive C³I systems requires novel advances in the area of distributed decision making under uncertainty. It is clear that the present largely qualitative design approaches adopted in C³I need to be replaced by a systematic quantitative design methodology. [1], [2].

The paper indicates the potential of an AI approach based on stochastic learning automata which provides a conceptual framework for modelling decision making in complex systems. An extensive literature and well established mathematical foundation now exists for stochastic learning automata, [3], [4]. In general, a learning automaton may be defined as an element which interacts with a random environment in such a manner so as to improve a specified overall performance by changing its action probabilities dependent on responses received from the environment. However, this approach is limited in modelling flexibility particularly for arbitrary topologies of decision models. It is essential that additional layers of sophistication are incorporated within the ba-

sic model. Thus, a quantitative framework based on Petri net methodology (PN) is proposed, [5]. The PN formalism presents an abstract, formal graph model useful for representing dynamic processes. In particular they provide a powerful means for the description and analysis of systems that are characterised as being concurrent, asynchronous, distributed and/or stochastic. Several authors have considered the use of PNs in the modelling on decision making processes, [6], [7], [8]. However, in such representations existing models do not exhibit the intelligence capability needed to provide effective decision models for C³I systems in stochastic environments.

The purpose of this paper is to introduce appropriate algorithmic tools for the systematic analysis and design of complex systems. Hence, a new class of PN are proposed and an application domain is considered.

2 Stochastic Learning Petri Nets (SLPN)

SLPNs are obtained by embedding the concept of stochastic learning automata into the model. A formal definition of a SLPN is thus the following:

$$SLPN = (P, T, A, M, \lambda, M_x) \quad (1)$$

where (P, T, A, M, λ) is the stochastic Petri net (SPN) underlying the model, [9]. The components may be described as: $P = (p_1, p_2, \dots, p_n)$, a finite set of places; $T = (t_1, t_2, \dots, t_m)$, a finite set of transitions; $A \subseteq \{P \times T\} \cup \{T \times P\}$, a set of input/output functions; $M = (m'_1, m'_2, \dots, m'_n)$, a state (marking) of PN; $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_m)$, a set of firing rates associated with transitions. M_x indicates the presence of two/three state stochastic learning automata. An automaton may be defined as a sextuple $(\beta, \psi, \alpha, p, F, G)$ where $\beta = (0, 1)$ is the input set to the automaton; $\psi = (\psi_1, \psi_2, \dots, \psi_n)$ is a finite state set; $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_r)$ is the output set from the automaton and each action is selected with probability $p = (p_1, p_2, \dots, p_r)$; $F: \psi \times \alpha$ is a state transition

mapping and $G : \psi \rightarrow \alpha$ is the output mapping. A P-model environment is characterised by a binary input set to the automaton $\beta = (0, 1)$, where $\beta = 0$ is known as a favourable response (success) and $\beta = 1$ an unfavourable response (failure).

At stage n , $p(n+1) = T[p(n), \alpha(n), \beta(n)]$, where T denotes the rule by which the automaton updates the probability of selecting the actions. Both linear and non-linear forms of the updating algorithm T have been considered. The most widely used are the class of linear algorithms which include linear reward/penalty (L_{RP}), linear reward/ ϵ penalty ($L_{R\epsilon P}$) and linear reward/inaction (L_{RI}), [4].

2.1 Model of SLPN

Consider the SPN model depicted in Figure 1. By analysis of the reachability tree in Figure 2, it is evident that the SPN model may exhibit one of six different states, depending on the transition that fires. Several transitions may be simultaneously enabled by a particular marking. Assume that H is the set of enabled transitions, then a transition t_i ($i \in H$) fires with probability:

$$Pr\{t_i\} = \frac{\lambda}{\sum_{k \in H} \lambda_k} \quad (2)$$

as stated previously, λ is the firing rate associated with PN transitions. Thus, the different states of a SPN define probability ratios which correspond to the firing of each transition. In any state, the sum of probability ratios is always equal to unity. For example, consider state $M_1 = [1100]$; the enabled transitions are t_1 , t_2 and t_3 and their respective firing probabilities may be defined as follows:

$$Pr\{t_1\} = \frac{\alpha}{(\alpha + \beta + \gamma)} \quad (3)$$

$$Pr\{t_2\} = \frac{\beta}{(\alpha + \beta + \gamma)} \quad (4)$$

$$Pr\{t_3\} = \frac{\gamma}{(\alpha + \beta + \gamma)} \quad (5)$$

Thus,

$$Pr\{t_1\} + Pr\{t_2\} + Pr\{t_3\} = 1 \quad (6)$$

The concept of a stochastic automaton may be introduced to select probabilistically the transition that fires. A transition selected in a particular marking corresponds to an action selected by an automaton. The firing of the chosen transition determines the next state (marking) of the system, by modifying the token distribution. In the tree representation of the SPN, Figure 2, there exists both two-state and three-state automata. Consider the following cases:

Two-state Automaton It is clear that state $M_2[0200]$ and state $M_3[1011]$ represent a two-state automaton, as depicted in Figure 2. The SPN with marking M_2 enables transitions t_2 and t_3 , since tokens are present in the input places (p_2). Each transition has an equal initial probability of being selected. The firing of t_2 , determines the next state of SPN to be M_0 ; the firing of t_3 , determines that the next state is M_4 . The firing probabilities for each transition is given:

$$Pr\{t_2\} = \frac{\beta}{(\beta + \gamma)}, Pr\{t_3\} = \frac{\gamma}{(\beta + \gamma)} \quad (7)$$

Similarly,

$$Pr\{t_2\} + Pr\{t_3\} = 1 \quad (8)$$

This concept also applies to state M_3 .

Three-state Automaton Clearly, the states M_1 and M_4 correspond to a three-state automaton. It is shown that the transitions t_1 , t_2 and t_3 are enabled; each transition has an equal initial probability of being selected. The possibility of firing t_1 , determines the next state is M_2 ; the firing of t_2 determines the next state of the SPN to be M_0 ; finally, if t_3 is selected by the automaton then the state transfers to M_3 . A similar concept also applies to state M_4 .

Note that the transition firing probabilities in each state M_0 and M_5 is always equal to unity. Since in state M_0 , the only transition that is enabled is t_1 ,

$$Pr\{t_1\} = \frac{\alpha}{\alpha} = 1 \quad (9)$$

Thus, it must fire with probability one. Similarly, in state M_5 the only transition that is enabled is t_4 , so it must also fire with probability equal to unity.

Hierarchical System of Automata The reachability tree may now be considered as a simple hierarchical system of automata; each state corresponding to an automaton. It may be noted that in a hierarchy each action has a unique path connecting it to the automaton (state) that has been selected previously, or to an automaton at the top level (state M_0). From the tree structure of Figure 2, it is possible to define nine unique paths which may be considered as sequence of states/decisions. To introduce the concept of an environment into this model, each sequence of states is associated with a reward probability.

The operation of this hierarchical learning system is as follows. At any instant the first level automaton, state M_0 selects an action (fires t_1). This activates an automaton in the second level which fires a transition

from its current transition probability distribution. This in turn activates, automata in the next level and so on. However, if a particular sequence of decisions corresponding to a unique path has been reached; the environment in turn generates a reward/ punish signal as its reaction. The reaction of the environment is used to update the transition probabilities for the various levels of automata in the selected path. This process repeats until all the probabilities in one path become close to unity from the top level (M_0) to the lowest level (M_5). Such a system may be considered as a SLPN model; structure is shown in Figure 3.

3 Simulation Results : SLPN

This section presents a computer simulation result for the SLPN model. The reward parameter is indicated; and $Pr(i, j)$ denote the transition firing probabilities, where i represents the state of the system and j provides the notation for the transition that fires. For example, consider the notation for state M_1 firing transition t_3 ; the transition firing probability is $Pr(1, 3)$. Expected values are denoted by the expression $Pr(i, j) = E[Pr(i, j)]$. In the simulation study the hierarchical system in Figure 2 was examined. To simulate this SLPN, all of the reward probabilities in the environment were in the range [0.2 - 0.45] except the unique maximum reward probability which was set to 0.9. An L_{RI} updating scheme was used to update action probabilities for the selected path.

Table 1 provides the reward probabilities of the environment which are used for simulation. Note that the unique maximum reward probability is associated with the selected sequence of decisions. Consider Table 1 which illustrates the convergence to the unique maximum reward probability, such that sequence 1 is selected from the reachability tree. This sequence represents the path $M_0 - Pr(0, 1); M_1 - Pr(1, 1); M_2 - Pr(2, 2); M_1$. In this case, transition probability vector in state M_0 is equal to unity; since t_1 must always fire with probability equal to one. Also the convergence of transition probability $Pr(1, 1)$ in the three-state automaton M_1 ; and $Pr(2, 2)$ in two-state automaton M_2 show that the optimal path selected is sequence 1, which has the unique maximum reward probability.

Similarly, the learning performance can be observed for all sequence of states of the SLPN model. In each case the transition probability vectors that converge close to unity, correspond to the sequence of decisions associated with the unique maximum reward probability.

4 Application : Small-scale C³-I System

The following sub-section presents the model of the interacting organisation member. An application of the SLPN to a specific two decision maker organisation is examined. A series of experiments are performed to observe the learning behaviour of the organisation.

4.1 Model of the Decision Making Process

A four stage model on the decision maker has been developed, [7], that permits the detailed and explicit specification of the interactions among organisation members. The internal structure of the four processing stages, is depicted in Figure 4. This shows that a decision maker receives an input signal x , from its environment and undergoes a four stage process. The first and last of these stages, situation assessment (SA) and response selection (RS), model the actual decision making process while information fusion (IF) and command interpretation (CI) allow for interaction of the decision module (DM) with other members of the organisation. The SA stage consists of a set of U algorithms that are capable of producing some situation assessment z . The RS stage also contains set of V algorithms which are required to produce the final decision response. This information may in turn be combined in the IF stage to yield \bar{z} . The fused assessed situation, \bar{z} , is processed by one of the algorithms in the RS stage. The CI stage of the model allows \bar{z} and the input \hat{v} to influence the choice of this algorithm; \hat{v} may be considered to be a command capable of restricting options. The RS stage contains algorithms that produce output y in response to the situation assessment \bar{z} and the command inputs.

4.2 Two Node Distributed Organisation

Figure 5 shows in Petri net form the first model proposed for study. The example consists of a two node organisation: a submarine decision module DM1 and a surface ship decision module DM2. Each DM receive signals from the environment and can respond to the environment. The DM module consists of three possible strategies, although the SA stage selects only a single strategy to process the information. For example, the DM must decide between the following three options:

Strategy SA; process information without using Decision Support System (DSS);

Strategy IT; select a response via an intelligent terminal;

Strategy MF; utilise the DSS.

4.3 Performance of Two Node Organisation

Figure 6 demonstrates the application of the SLPN approach to examine the behaviour of the two node organisation. An approach has been adopted by embedding the concept of SLPN in the SA and RS stages for the decision module. Therefore each DM contains four learning automata interconnected in the form of a tree structure. As illustrated in Figure 6 the automata are arranged in two levels. The hierarchy consists of a single automaton at the first level, and three automata in the second level. For decision module DM1, the three options (SA_1 , IT_1 , MF_1) are selected with equal initial probability; similarly for DM2 (SA_2 , IT_2 , MF_2). Also each RS stage has two alternate possibilities which are selected with equal initial probability; thus producing six possible paths for each DM. The strategies associated with decision module DM1 and DM2 are (p_1, p_2, \dots, p_6) and (q_1, q_2, \dots, q_6) respectively. There are 36 (6x6) possible combinations of decision strategies fed to the environment. Considering this structure, Figure 6, for each pair of strategies selected by the decision modules the environment responds stochastically to punish/reward the selection of a particular pair. One pair of decisions is optimum (ie. gives minimum punishment or maximum reward).

4.4 Experimental Results

The following experiments [1-3] illustrate the learning performance of a two node organisation, as depicted in Figure 6. For these experiments, the main objective is such that both decision modules select the optimal pair of decision strategies from 36 (6x6) possible combinations of decision pairs input to the environment. As stated previously, decision modules are in the form of a two level hierarchical system. To simulate these modules, the reward probabilities in the range [0.2-0.5] are associated with paths (p_1, p_2, \dots, p_6) and (q_1, q_2, \dots, q_6) for decision modules DM1 and DM2, respectively. However, in this case the unique maximum reward probability which is set to 0.9 exists for each DM1 and DM2. Thus, a single path from the set (p_1, p_2, \dots, p_6) for DM1 is associated with a unique maximum reward probability; and also a single path from the set (q_1, q_2, \dots, q_6) for DM2. An L_{RI} scheme was adopted to update action probabilities for the optimal strategy pair were updated. The conditions for each experiment are varied by considering the selection of optimal strategy pairs; sudden switch of environmental conditions and by permitting communication between both decision

modules at upper and lower levels. The reward parameter and reward probabilities are given; the expected values are denoted by $p_1 = E[p_1(n)]$.

Experiment 1

The simulation results in Table 2 demonstrate the learning behaviour of a two node organisation. The table indicates the value of the reward parameter; the unique maximum reward probability to be employed by the environment and the expected values denoting the convergence to optimal strategy pair. In this case the unique maximum reward probability is associated with path $p_4.q_2$ for decision module DM1 and DM2, respectively. The results confirm that the coordinated decision strategies selected by each decision module converges close to unity. Hence, the optimal pair of decisions selected by DM1 and DM2 is $p_4.q_2$.

Experiment 2

The previous experiment 1 was repeated, with the additional concept of a sudden switch to a different environment. By repeating experiment 2, it can be seen that both decision modules converge close to unity by selecting the optimal pair of decision strategies. The sudden switch in the environment is achieved by re-locating the unique maximum reward probability, such that an alternate pair of decision strategies may be selected.

This behaviour is best illustrated by analysing the results in Table [3a - 3b]; all relevant parameter values are indicated. The simulation results show how fast the structure learns convergence to the new optimal strategy pair. It is evident from Table 3a that both decision modules DM1 and DM2 select the optimal strategy pair $p_1.q_1$; and convergence for this pair is close to unity. However, after introducing a sudden switch of the environment the coordinated decision strategy pair $p_3.q_1$ is selected. Thus, Table 3b shows a decrease in convergence for path p_1 selected by DM1 and a rapid increase in convergence close to unity for strategy pair $p_3.q_1$.

Experiment 3

This final experiment gives an excellent illustration of speeding up the learning process by permitting communication between decision modules DM1 and DM2 (as indicated by dotted lines Figure 6). Note that in each of the following experiments an arbitrary value for the stepsize is considered.

First set of results in Table 4a illustrates communication between automata at the top level of the hierarchy for each decision module. To simulate this structure, both automata at the top level (SA_1 and SA_2) exchange messages such that if each selects action one, then the reward parameter is incremented by stepsize 4. From Table 4a, it can be seen that the convergence rate for strategy pair $p_1.q_1$ rapidly in-

creases close to unity; since the unique maximum reward probability is associated with this strategy pair.

Second set of results in Table 4b exemplifies communication between automata at the top and lower levels of the hierarchy for each decision module. The same rule is applied, that is, if both automata at the top and lower level select action one, the reward parameter is increased by stepsize 4. Similar to the previous case, the results in Table 4b show rapid convergence close to unity for both levels of automata. In comparison to the previous experiment, there is only a fractional increase in convergence rate by permitting communication between upper and lower levels automata.

5 Conclusion

This paper has defined a high-level quantitative framework based on Petri net methodology. It has proposed a new class of Petri net modelling tool for an effective representation of decision models. This approach has enhanced the modelling power of Petri nets. The modelling technique has exhibited a data flow formation, and an AI decision making process embedded within the net. The application of the modelling tool to a non-trivial example has been considered. This has illustrated the modelling flexibility and suitability to a realistic distributed decision problem.

References

- [1] M. Athans. Command and control theory: A challenge to control science. *IEEE Trans. Aut. Control*, AC-32(4):286-293, 1987.
- [2] A. P. Sage. Information systems engineering for distributed decision making. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-17(6):920-936, Nov.-Dec. 1987.
- [3] K. S. Narendra and M. A. L. Thathachar. Learning automata — a survey. *IEEE Trans. on Systems, Man and Cybernetics*, 4(4):323-334, July 1974.
- [4] K. S. Narendra. *Learning Automata - An Introduction*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1989.
- [5] J. L. Peterson. *Petri Net Theory and the Modelling of Systems*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.
- [6] D. Tabak and A. H. Levis. Petri net representation of decision models. *IEEE Trans. on Systems,*

Man and Cybernetics, SMC-15(6):812-818, Nov.-Dec. 1985.

- [7] A. H. Levis. Information processing and decision making organisations: a mathematical description. *Large Scale Systems*, 7:151-167, Nov.-Dec. 1984.
- [8] S. L. Skulsky and A. H. Levis. Migration of control in distributed intelligence systems. *Proc. IEEE Int. Symposium on Intelligent Control*, pages 74-81, Sept. 1989.
- [9] G. Marsan M. A., Balbo and G. Conte. *Performance Models of Multiprocessor Systems*. Cambridge, MA: The MIT Press, 1987.

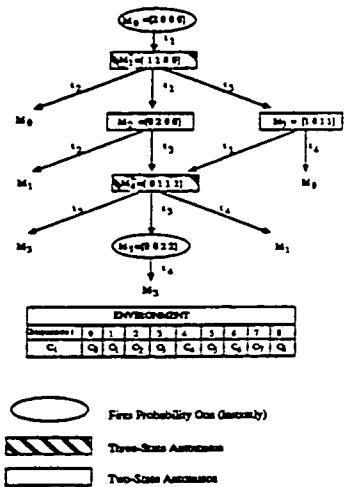
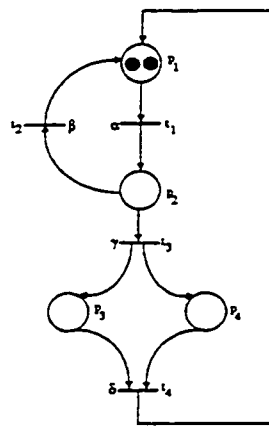


Figure 1 - Stochastic Petri Net

Figure 2 - Reachability Tree

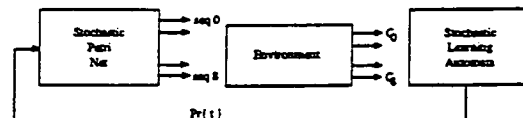


Figure 3 - Stochastic Learning Petri Net (SLPN)

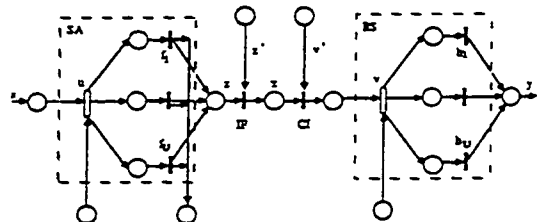


Figure 4- PN Model of Interacting Decision Maker

